```
INITL    CSECT
BEGIN    SAVE  (14,12),,
         BALR  R3,0
         USING BR3,R3
BR3      L     R4,BR4
```

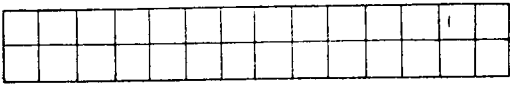
Д. СТЭБЛИ

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ В СИСТЕМЕ/360


ИЗДАТЕЛЬСТВО
МИР

LOGICAL PROGRAMMING WITH SYSTEM/360

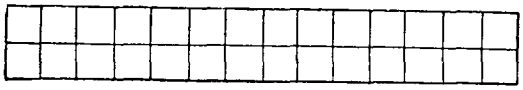
D. STABLY



NEW YORK

1970

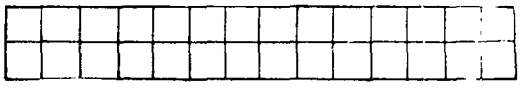
Д. СТЭБЛИ



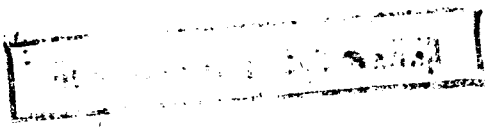
ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ В СИСТЕМЕ/360

ПЕРЕВОД С АНГЛИЙСКОГО
А. П. Гагарина, В. Г. Меркулова и О. Ф. Мясина

ПОД РЕДАКЦИЕЙ
Л. Д. Райкова и М. Р. Шура-Буры



ИЗДАТЕЛЬСТВО «МИР»
Москва 1974



УДК 681.142.2

Учебник по базовому языку программирования ЭВМ третьего поколения — языку Ассемблера. Содержащиеся в нем данные непосредственно применимы к программированию на ЭВМ третьего поколения, которые разработаны в нашей стране и составляют основу вычислительной техники СССР и стран социалистического содружества

Умение программировать на уровне базового языка — важное условие эффективного использования современных машин. Независимо от уровня подготовки будущего пользователя и круга задач, которые ему придется решать, ему полезно пройти курс обучения программированию на языках типа языка Ассемблера.

Книга снабжена большим количеством упражнений, отражающих разнообразные практические ситуации. Она может быть использована как для самостоятельного изучения, так и в качестве пособия для преподавателей курсов программирования.

Редакция литературы по математическим наукам

С $\frac{20204-038}{041\,(01)-74}$ 38—74

Предисловие редакторов перевода

В последние годы у нас издано несколько переводных книг, посвященных электронным вычислительным машинам Системы/360 фирмы IBM. В них с различной степенью детальности рассматриваются вопросы логической структуры и математического обеспечения. Читатели проявили большой интерес к этим изданиям, разошедшимся многотысячными тиражами.

Развитие вычислительных систем третьего поколения, родоначальником которых была Система/360, характеризуется отчасти стихийным процессом «стандартизации», который затрагивает и логическую структуру ЭВМ, и принципы построения операционных систем. Так, широко распространенные в Европе ЭВМ фирм Siemens и ICL имеют системы команд, практически совпадающие с системой команд ЭВМ Системы/360. Близки по основным принципам построения и предоставляемым возможностям и некоторые операционные системы ЭВМ фирм IBM, Siemens и ICL. Эти практически установившиеся соглашения и стандарты оказали определенное влияние на логическую структуру и математическое обеспечение производимой в Советском Союзе Единой системы электронных вычислительных машин (ЕС ЭВМ). Поэтому большой интерес к литературе по Системе/360 объясняется не только тем, что фирма IBM является крупнейшим в капиталистическом мире разработчиком и производителем ЭВМ, а идеи, заложенные в Системе/360, отражают современный уровень в этой области, но и возможностью практического применения части материала, содержащегося в переводных изданиях по Системе/360.

Книга представляет собой учебник по программированию на языке Ассемблера, адресованный прежде всего начинающим. Автор ограничился изложением методов программирования с использованием непривилегированных команд стандартного набора и команд десятичной арифметики. Сведения о командах транслятора и макрокомандах сообщаются лишь в объеме, необходимом для понимания основного материала. Отдельные главы содержат краткое описание машинных команд, вполне достаточное для первоначального изучения, и примеры их применения. Избранный способ изложения позволяет читателю

сосредоточить свое внимание в первую очередь на методах программирования. Желающим изучить систему команд ЭВМ Системы/360 в более полном объеме можно порекомендовать обратиться к книге «Вычислительная система IBM/360. Принципы работы», «Сов. радио», 1968.

Последние главы содержат ориентированный на операционную систему OS обзор по принципам организации данных и методам доступа к ним. Более детальное изложение этих вопросов можно найти в книге «Операционная система IBM/360. Супервизор и управление данными», «Сов. радио», 1973.

Большое количество примеров и обзорных упражнений позволяет надеяться, что книга будет полезна как изучающим программирование для ЭВМ третьего поколения, так и преподавателям.

Переводчики в основном придерживались терминологии книги К. Джермейна «Программирование на IBM/360», «Мир», 1971, 1-е изд., 1973, 2-е изд., практически совпадающей с терминологией, принятой в документации на ЕС ЭВМ. При переводе названия команд на английском языке были приведены в соответствие с официальными документами фирмы IBM. Без специальных оговорок исправлены мелкие неточности и опечатки. Главы 1—8 переведены О. Ф. Мясиным, 9—12 — В. Г. Меркуловым, 13—20 — А. П. Гагариным, который, кроме того, выполнил перевод глоссария и ответов к упражнениям.

Л. Д. Райков, М. Р. Шура-Бура

Предисловие

Немного найдется областей знаний, которые подобно программированию непосредственно служат такому широкому и разнообразному кругу других профессий. Цель настоящей книги — дать доступное изложение идей программирования для Системы/360. Из языков программирования, используемых в Системе/360, для этого наилучшим образом подходит язык Ассемблера, что обусловлено следующими причинами:

1. Язык Ассемблера непосредственно отражает основные принципы построения Системы/360.

2. Независимо от использования других языков программирования, на Ассемблере приходится программировать почти на каждой установке Системы/360.

3. Трансляторы с некоторых языков высокого уровня, например с PL/1, позволяют получать листинг на языке Ассемблера. Этот листинг содержит сгенерированные коды предложений исходной программы и может быть полезен при анализе и интерпретации программы.

Прежде чем писать книгу, нужно было решить, для какого читателя она предназначается. После консультаций со специалистами по обучению в области систем обработки данных было решено, что книга не должна быть ориентирована на какую-либо специализированную программу обучения. Поэтому она написана так, что может быть использована в учебных программах университетов, технических институтов, колледжей и курсов по подготовке программистов. Предварительная проверка эффективности книги подтвердила ее применимость для широкого круга обучающихся.

Предполагается, что читатель имеет среднее образование и практические навыки составления блок-схем программ.

Первые главы книги содержат общее описание аппаратуры Системы/360 и некоторые замечания о применяемой терминологии. Эти сведения создают основу для понимания принципов построения Системы/360, языка Ассемблера и методов программирования. Последовательность изложения материала следующая: системы счисления (с основаниями 2, 10, 16), представление данных, отдельные предложения языка Ассемблера и

методы программирования, в которых каждое из них находит применение.

Последние главы содержат обзор типов организации данных и методов доступа к ним. Эти структуры обсуждаются в связи с программами управления данными Системы/360, включая последовательную, индексно-последовательную и прямую организацию и соответствующие методы доступа.

В конце книги для удобства читателей приведен словарь некоторых встречающихся в тексте специальных терминов.

Я считаю, что как для преподавателя, так и для студента важно уметь правильно оценить достигнутые в процессе обучения успехи, поэтому любой хороший учебник должен включать обзорные упражнения. В отличие от письменных экзаменов и контрольных опросов они должны выявлять любое непонимание изучаемого материала, даже в самой начальной стадии обучения. Упражнения должны не столько проверять усвоение общих принципов, сколько подтверждать способность учащегося ориентироваться в самых мелких деталях изучаемого предмета.

Разработка и совершенствование используемых в тексте упражнений и примеров — естественный результат моей работы в области больших вычислительных систем и их программного обеспечения, области, в которой каждому специалисту приходится быть одновременно и учителем и учеником.

Эти упражнения построены так, что позволяют проверить понимание выполнения каждой отдельной команды языка Ассемблера. Детальное изучение команд поможет добиться высокой эффективности программирования.

Помимо изучения отдельных команд очень важно приобрести практические навыки составления программ. Как хороший учебник немыслим без обзорных упражнений, так и успех любого курса программирования невозможен без кодирования практических программ. К сожалению, на пути решения этой задачи немало трудностей.

Что такое «правильное» написание программы? Или в более общей постановке, что такое оптимальная последовательность команд, образующих конкретную программу?

Хорошо известно высказывание: если перед сотней программистов поставить одну и ту же задачу, то они напишут сто разных программ. Многие из этих программ будут весьма похожи по содержанию, но это в значительной степени определяется сходством методики обучения программистов. Представляется поэтому, что практическое обучение программированию должно вестись под руководством преподавателя, использующего в своей работе эту книгу. Каждый преподаватель сформулирует собственные критерии правильности и эффективности применения учащимися возможностей языка Ассемблера. В соответствии с

этим подходом в книгу не были включены примеры законченных практических программ.

Большая гибкость, присущая Системе/360, и огромный диапазон возможных конфигураций аппаратуры и программного обеспечения не позволили рассмотреть все вопросы, представляющие интерес. За рамками книги остались команды управления вводом-выводом, макрокоманды, язык управления заданиями. Не вошли в книгу и другие способы задания команд или параметров, которые целиком зависят от конкретной операционной системы или конфигурации аппаратуры. Несмотря на очевидную важность всей этой информации, ее исключение из текста имеет основания. Конфигурации установок на вычислительных центрах отличаются друг от друга не в меньшей степени, чем конкретные тексты программ для решения одной и той же задачи, написанные разными программистами. Приходится исходить из того, что необходимые сведения об операциях ввода-вывода, управлении данными и описании наборов данных учащийся может получить от инструкторов вычислительного центра, на котором ему придется работать.

Д. Стэбли

ЧАСТЬ I

Введение в Систему/360

Глава 1

Программист и ЭВМ

А. ВВЕДЕНИЕ, АДРЕСОВАННОЕ ПРОГРАММИСТУ

Современные электронные вычислительные машины, по всей вероятности, самое непонятное средство помощи человеку, которое когда-либо было разработано. Внушающие суеверие одним, ненавидимые другими, они до настоящего времени являются объектом юмористических и серьезных нападок, вызванных их огромными возможностями и предполагаемыми недостатками.

Когда „средний гражданин" получает ежемесячный счет от местного отделения кредитного общества и обнаруживает ошибку — он видит виновника в ЭВМ; автоматизированная система бухгалтерских расчетов удерживает слишком много из заработной платы всех служащих — и ЭВМ приобретает скверную репутацию за свои прегрешения. Это повторяется снова и снова: ракета, которая отклонилась от расчетной траектории, счет за покупку, который автоматически выписывается миллиону клиентов вместо ста, строки нагромождений тарабарщины, которые внезапно появляются среди подборок последних известий в вечерних газетах, — все это приписывается шалостям капризной машины.

К несчастью, вероятно, пройдут многие годы, прежде чем наступит эра всеобщего общественного признания и понимания основных концепций ЭВМ и их возможностей. До той поры, однако, этот чарующий лабиринт электронного оборудования обречен переносить презрение и град каламбуров тех, кто, по их мнению, пострадал от причуд его мерцающих ламп и блоков памяти. Когда-то, возможно, ответственность за все неуловимые огрехи вычислительной машины будет наконец принята теми, кто в действительности порождает эти ошибки, — самими людьми.

Программист должен почти столь же тщательно изучать человеческую природу, как и природу вычислительной машины, которой предписано выполнять его приказания. Программист

может предпринимать лицемерные попытки переложить ответственность за аварийное окончание программы на «машинную неисправность», машина как следствие этого становится «козлом отпущения» для многих ошибок программиста, и она сама не может защитить себя перед администратором, ответственным за обработку данных. Было бы чрезвычайно интересно узнать тот бесконечно малый процент ошибок обработки, за которые действительно несет ответственность сама машина, и сравнить его с общим количеством тех ошибок, в которых ее обвиняют.

Постепенно программист добивается справедливого признания как художник в своем искусстве. Программа в буквальном смысле может явиться предметом истинного искусства, настоящим шедевром использования логических процессов, происходящих в человеческом мозгу, — или, напротив, она может представлять собой кучу «мусора» (термин, который часто используется при обработке данных), монумент из ложных концепций и неприменимых методов. С тех пор как программирование на ЭВМ было по-настоящему признано, оно предоставляет программисту богатые потенциальные возможности. В том, как программист использует эти возможности, проявляется различие между профессиональным программистом и ординарным «кодировщиком».

Повсюду в этой книге к программисту или к лицу, изучающему программирование, обращаются в мужском роде. Это всего лишь средство выражения, не содержащее каких-либо «намеков» женской половине программистского мира, которая, кстати, доказала, что она принадлежит к обширному семейству энтузиасток ЭВМ, в равной степени искушенных, а иногда даже более сведущих, чем их коллеги мужчины.

Б. СИСТЕМА/360 — ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА ТРЕТЬЕГО ПОКОЛЕНИЯ

Современные концепции обработки данных, возможно, зародились с появлением перфокарточного оборудования, с помощью которого эти карты хранились, сортировались и располагались в соответствии с требуемой последовательностью благодаря наличию пробивок в поле карт. Этот тип оборудования на самом деле предшествовал рождению того, чему суждено было стать аппаратурой вычислительных машин первого поколения. Термин *поколение* в применении к ЭВМ часто определяется как период времени, в течение которого внутренняя электронная аппаратура машины базировалась на определенных принципах электроники и электронном оборудовании. В этом смысле вычислительными машинами первого поколения были машины, в кото-

рых для обеспечения питания и обработки информации применялись вакуумные электронные лампы. Эти предшественники сегодняшних ЭВМ были в буквальном смысле «печами», выделявшими тепло в таких количествах, что было трудно поддерживать нормальную температуру в машинных залах.

Применение транзисторов привело к созданию ЭВМ второго поколения, обладавших большей памятью и большим быстродействием логических схем и, несмотря на это, требовавших меньших машинных залов по сравнению с их старшими сестрами. По мере того как электронная промышленность прогрессировала в разработке все меньших и меньших компонентов, микроэлектроника стала реальностью. Электронные приборы на твердом теле были миниатюризированы до такой степени, что электронная логическая схема занимала площадь, меньшую площади куска сахара; были созданы такие маленькие «чип-транзисторы», что 50 000 штук таких транзисторов могли поместиться в обычном наперстке. Эти компоненты представляют собой тот тип электронного оборудования, которое было использовано при проектировании и производстве современных вычислительных машин — машин третьего поколения.

Существует несколько определений понятия поколений ЭВМ в зависимости от того, кто производит анализ этого термина. Некоторые говорят, что третье поколение вычислительных машин представляет собой в действительности удачное соединение машины для научных расчетов и машины для коммерческого применения в единую модульную систему. Такой подход к определению ЭВМ третьего поколения можно объяснить тем, что перед появлением этих машин нередко системы обработки информации ориентировались на определенный тип машины, приспособленный либо для научных, либо для коммерческих применений. В этом отношении представляется, что Система/360 удовлетворяет потребностям как научных исследований, так и коммерческих расчетов.

По мере повышения компактности компонентов новых ЭВМ возрастали внутренние скорости обработки информации. Электрическим импульсам и электронным потокам данных требовалось проходить все меньшие расстояния между логическими секциями машины, благодаря чему достигалась такая производительность, которая становилась все более и более труднодоступной для понимания. Такие термины, как *наносекунды* и *пикосекунды*, представляющие соответственно миллиардные и триллионные доли секунды, стали теперь связываться с определенными внутренними функциями машины. Потребовалось такое периферийное оборудование вычислительных систем, которое могло бы выполнять передачу данных со скоростями, не снижающими существенно производительность обработки. Было бы нереально

ожидать, что скорость работы периферийного оборудования будет физически соответствовать скорости самой машины; в таком оборудовании механическое перемещение является неотъемлемой частью его рабочего режима, и следовательно, от этого устройства нельзя ожидать такой же производительности, как и от устройства обработки данных, скорость которого зависит только от быстродействия электронных схем. Однако удалось разработать внешние запоминающие устройства (ВЗУ) на магнитных дисках, которые обеспечили скорость передачи данных, равную 312 000 символов в секунду, и ВЗУ на магнитных барабанах со скоростью передачи 1 200 000 символов в секунду.

Одно из важнейших достоинств Системы/360 — положенная в основу ее проектирования концепция «модульности», позволяющая наращивать центральный процессор и добавлять периферийное оборудование по мере роста потребностей пользователя. Например, предположим, что некий отдел обработки данных имеет вычислительную установку Системы/360 в следующей конфигурации:

центральный процессор модели 50 с 256К байтов основной памяти (262144 байта);
пультовая пишущая машинка;
устройство ввода-вывода перфокарт;
построчно печатающее устройство;
4 накопителя на магнитной ленте (НМЛ) (7-дорожечная лента с плотностью записи 800 байтов на дюйм);
2 запоминающих устройства с прямым доступом типа 2311;
необходимый комплект устройств управления.

Когда загруженность вычислительного центра увеличивается, руководство отдела обработки данных принимает решение о заказе дополнительных устройств прямого доступа типа 2314. Кроме того, в ближайшем будущем предполагается реализовать систему мультипрограммирования, и поэтому желательно иметь еще 256К байтов основной памяти и еще одно построчно печатающее устройство. Далее руководство компании решает увеличить объем вывода на магнитные ленты, поэтому оно отказывается от двух 7-дорожечных НМЛ и заказывает шесть 9-дорожечных НМЛ с плотностью записи 1600 байтов на дюйм.

После всех этих изменений конфигурация вычислительной системы приняла следующий вид:

центральный процессор модели 50 с 512К байтов (524288 байтов) основной памяти;
пультовая пишущая машинка;
устройство ввода-вывода перфокарт;
2 построчно печатающих устройства;

2 накопителя на магнитной ленте (7-дорожечная магнитная лента с плотностью записи 800 байтов на дюйм);

6 накопителей на магнитной ленте (9-дорожечная магнитная лента с плотностью записи 1600 байтов на дюйм);

1 запоминающее устройство с прямым доступом типа 2314 (емкостью 232 000 000 байтов);

2 устройства с прямым доступом типа 2311;

необходимый комплект устройств управления.

Исходная система путем добавления отдельных модулей превратилась теперь в довольно мощную систему, имеющую производительность, способную удовлетворить требованиям многих установок для обработки данных. Возможность наращивать мощность вычислительной системы подобным образом чрезвычайно желательна для удовлетворения постоянно растущих потребностей автоматизации систем в экономике и разработки научно-исследовательских проектов.

Как уже говорилось, устройство центрального процессора (ЦП) вычислительной системы (рис. 1.1 и 1.2) представляет собой хорошо спланированный лабиринт из электронных схем, неспособный к какому-либо самостоятельному действию.

Для того чтобы активизировать эти схемы и заставить их работать заранее предопределенным образом, ЭВМ должна быть

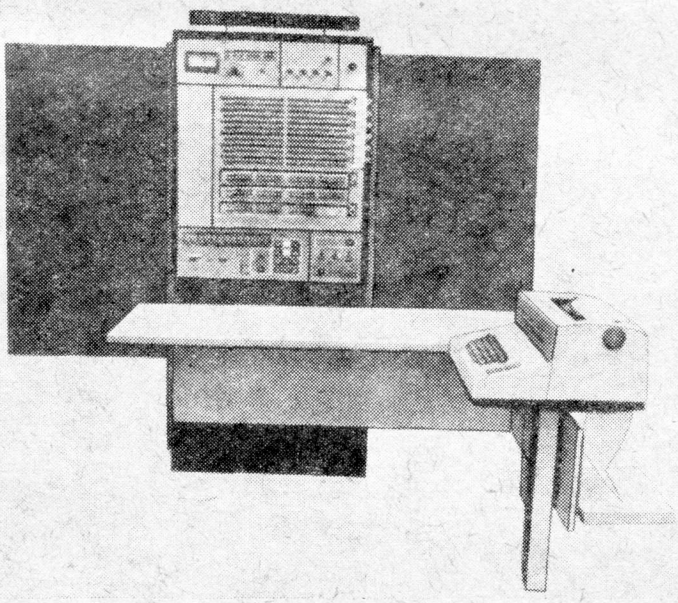


Рис. 1.1. Модель 65 Системы/360 (показаны панель управления устройства ЦП и консольная пишущая машинка).

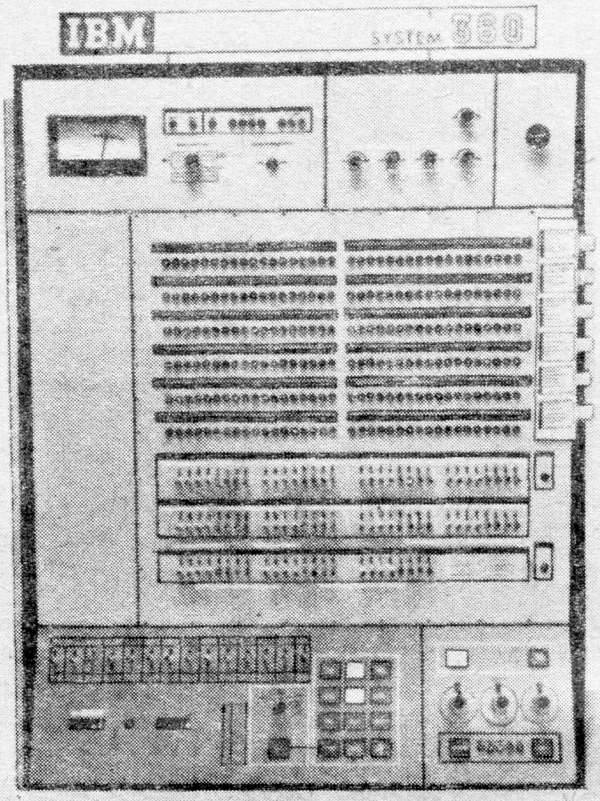


Рис. 1.2. Панель управления устройства центрального процессора модели 65 Системы/360.

снабжена главной управляющей программой, часто называемой *супервизором* или *монитором*. Можно считать, что супервизор так же, как и вычислительная система, построен по модульному принципу, так как он состоит из набора обязательных и набора выбираемых по желанию пользователя модулей и управляющих программ. Полный состав конкретной управляющей программы, вообще говоря, зависит от следующих факторов:

1) операционной системы, к которой он относится;
2) управляющих модулей, которые требуются для этой операционной системы;
3) конфигурации аппаратуры данной вычислительной системы;
4) выбора конкретных программ из имеющегося набора программ управления данными.

Полный пакет супервизора формируется в процессе объединения и выборки программ, называемом *генерацией системы.* Генерация системы на используемой установке необходима для того, чтобы готовая система оказалась в состоянии удовлетворить всем требованиям данной конфигурации ее аппаратуры. Каждую из операционных систем можно генерировать многими различными способами. Эти модификации осуществляются посредством включения или исключения тех или иных параметров во время генерации системы. С помощью специальных программ, поставляемых фирмой-изготовителем ЭВМ, избранные пользователем программы и модули для системы, которую желательно получить, определяются, обрабатываются на вычислительной машине и накапливаются на некотором носителе информации в виде готового супервизора. Результат этой работы представляет собой главную управляющую программу, которая загружается в ЦП вычислительной машины и повседневно осуществляет управление прогоном программ пользователя.

Супервизоры операционных систем разного типа — Базовой операционной системы (Basic Operating System, BOS), Ленточной операционной системы (Tape Operating System, TOS), Дисковой операционной системы (Disk Operating System, DOS) и Операционной системы (Operating System, OS) — значительно отличаются друг от друга. Можно считать, что каждая из этих систем спроектирована для вычислительного оборудования, обладающего различными физическими свойствами и возможностями, хотя во многих случаях для конкретной конфигурации вычислительной системы имеется некоторая свобода выбора из перечисленных операционных систем.

Предполагается, что Базовая операционная система предназначена для относительно малых вычислительных установок из семейства Системы/360, но может выполнять все необходимые функции, которые требуются для достижения максимальной производительности, которая может быть обеспечена вычислительной системой этого класса.

Дисковая операционная система применима к некоторому диапазону вычислительных систем, но она оказывается наиболее подходящей для таких вычислительных систем, которые используют преимущественно запоминающие устройства с прямым доступом. Конфигурация вычислительной системы, работающей под управлением Дисковой операционной системы, может включать накопители на магнитной ленте.

Ленточная операционная система используется на установках среднего размера. Она ориентирована на применение НМЛ и не обеспечивает работу устройств с прямым доступом.

Самой большой и наиболее гибкой является полная Операционная система, которая практически применима ко всем

вычислительным машинам Системы/360 от средних до самых больших. Эта система предоставляет богатый выбор вспомогательных программ, методов доступа, систем организации файлов и потенциальные возможности для комплексированных конфигураций аппаратуры. В зависимости от выбора характеристик в процессе генерации Супервизор OS может занимать объем от 30000 байтов до более чем 120000 байтов основной памяти. OS является наиболее мощной операционной системой и может одновременно обеспечивать работу всего разнообразия периферийного оборудования — накопителей на магнитных лентах, барабанов, печатающих устройств, дисплеев, телекоммуникационного оборудования и связанных с машиной терминалов.

Каждая из операционных систем использует управляющие программы, которые отчасти отличаются от программ других операционных систем. Даже конкретный метод доступа, хотя он и может быть реализован более чем одной операционной системой, активизируется операторами макрокоманд, в каких-то деталях отличающихся друг от друга в разных операционных системах. Именно поэтому данная книга не преследует цели объяснить макрокоманды ввода-вывода, язык управления заданиями или функциональные системные макрокоманды. Невозможно предсказать конкретные конфигурации аппаратуры и соответствующие модификации операционной системы вычислительных установок, при работе которых данная книга могла бы быть использована. Тем более невозможно объяснить и оценить все эти модификации в рамках одной книги. Учебный центр, университет, колледж или индустриальный комплекс, который сочтет эту книгу ценной для обучения своих программистов, должен, следовательно, нести ответственность за обеспечение их дополнительной информацией, содержащей, например, описания применимых операторов языка управления заданиями, справочные данные по устройствам и их адресации, а также макрокоманды, допустимые для конкретной операционной системы, применяемые языки и их уровни и описания методов доступа.

B. ТЕРМИНОЛОГИЯ

Изучение программирования в Системе/360 сопровождается применением массы новых слов и терминов, которые могут вообще оказаться незнакомыми обучающемуся программисту. Некоторые слова могут быть заимствованы из обиходного языка, и все же, используясь в связи с программированием на ЭВМ, они приобретают абсолютно иной смысл. Например, „барабан", „слово", „канал", „память", „переключатель" — все эти слова являются обычными существительными, доступными пониманию всех людей, которые незнакомы с вычислительными системами. Когда эти же самые термины используются в связи с Систе-

мой/360, они интерпретируются совершенно по-иному. Упоминание о барабане у человека, который не знаком с вычислительной техникой, мысленно вызывает образ ударного музыкального инструмента; тем не менее для программиста это есть ничто иное, как запоминающее устройство с прямым доступом! В конце книги приведен словарь терминов из области вычислительной техники. Он содержит большинство терминов и определений, которые могут пригодиться учащемуся.

Некоторые термины, применяемые в связи с Системой/360 и с другими ЭВМ, требуют дополнительного объяснения и обсуждения. Это относится к терминам „hardware" и „software" в применении их к вычислительным системам. Под hardware (аппаратной частью вычислительной машины) понимается любое физическое устройство или компонент оборудования, являющиеся частью конфигурации вычислительной системы в целом. Устройство центрального процессора, устройства с прямым доступом, дисплеи, кабели, лентопротяжные механизмы, устройства управления, построчно печатающие устройства, пультовые пишущие машинки и другие устройства рассматриваются как hardware. Термин „software" (программная часть) связан со вспомогательными программами ввода-вывода или с некоторым набором управляющих программ, которые нужны вычислительной машине для того, чтобы воздействовать необходимым образом на программы пользователя. Хотя термин „software" часто относят непосредственно к специализированным управляющим программам, которые поставляются изготовителем ЭВМ, он в равной степени применим к специализированным программам, написанным программистами в процессе работы на установке и обеспечивающим необходимый сервис для данной операционной системы или для пользователя. К последнему типу может быть отнесена учетная программа, которая накапливает значения времени прогона для отдельных заданий в течение дня, записывает это время в специальный набор данных и печатает сообщение о машинном времени, затраченном на выполнение каждого из этих заданий. Термин „software" применим и к программам, которые доступны всем пользователям и служат для выполнения определенной функции внутри их рабочих программ. Они могут представлять собой набор программ, активизируемых «вызовом» со стороны пользовательской программы для выполнения, например, специальных арифметических вычислений на основе параметрических значений, переданных «вызывающей» программой, с последующей передачей результатов вычислений этой программе.

Если в процессе чтения книги вам встретятся термины или выражения, недостаточно полно разъясненные ранее, рекомендуем обращаться к терминологическому словарю в конце книги.

Глава 2

Вычислительные системы

А. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Для того чтобы программист мог привязать логику выполнения своей программы к конкретной ЭВМ, он должен задавать информацию таким способом, чтобы ее могли «понять» управляющие программы ЭВМ. Общей основой этого понимания между человеком и машиной является язык программирования, который используется для описания логики выполнения программы.

Язык программирования в действительности не дает возможности непосредственной связи с ЭВМ. Он является только связанной с программистом частью двухступенчатого процесса обработки, который будет определять действие, подлежащее выполнению. Используя свойственные языку параметры и ограничения, программист описывает задачу с помощью предложений в кодах этого языка. Тем самым он определяет последовательность шагов, которая должна быть оттранслирована в определенное представление в виде команд, понимаемых внутренней структурой машины, на которой предполагается решать эту задачу. Трансляция выполняется *ассемблером* или *компилятором* с данного языка.

Компилятор является частью содержащегося в операционной системе набора управляющих программ [1]). Каждый язык или уровень языка требует отдельного компилятора для трансляции программы, записанной в кодах исходного языка. Операционная система любой вычислительной установки должна содержать компиляторы для всех языков или уровней языка, которые предполагают использовать на данной установке.

Предложения исходного языка вместе с управляющими перфокартами, необходимыми для идентификации компилятора, вводятся в вычислительную систему, где затем производится их обработка. Как только вся информация на исходном языке введена в рабочую область компилятора, сам компилятор начинает идентифицировать каждое предложение и связывать его с функцией, которая должна быть выполнена вычислительной машиной

[1]) В операционных системах фирмы IBM DOS/360 и OS/360 компиляторы принято относить к классу обрабатывающих программ. — *Прим. ред.*

во время прогона программы. После того как предложение связано со специфической функцией, компилятор транслирует это действие в одну или более машинных команд. Результат трансляции всех исходных предложений представляет собой *объектный модуль*, т. е. программу, которая должна быть связана с системными управляющими программами и выполнена.

Объектный модуль может быть набит на перфокарты как *объектная колода* или может быть записан в виде набора данных как часть *библиотеки программ*. После завершения работы компилятора выдается распечатка (листинг), содержащая предложения исходного языка, из которых был скомпилирован объектный модуль. Эта распечатка обычно выявляет любые ошибки кодирования, а для некоторых компиляторов содержит предупреждения, относящиеся к ошибочным условиям, которые могут возникнуть в процессе выполнения объектной программы.

Языки программирования обычно делят на *языки низкого уровня* и *языки высокого уровня*. Язык низкого уровня столь тесно связан с действительным машинным языком, что функционально почти один к одному совпадает с ним. В Системе/360 язык Ассемблера считается языком низкого уровня.

Язык высокого уровня обычно называют *операторным* языком. Здесь код каждого оператора может генерировать ряд машинных команд. Оператор языка высокого уровня может иметь следующее содержание: «Если A равно B, то выполнять C до тех пор, пока Y не станет равен Z.»

Можно привести много веских аргументов как в пользу языков низкого, так и высокого уровня, но пользователи очень часто делают свой выбор без достаточно обоснованного анализа преимуществ одного языка перед другим. Почти в каждой системе обработки информации есть вполне определенное место и необходимость в обоих типах языков. По мнению автора, программисту лучше начинать с изучения языка низкого уровня, так как такой язык имеет непосредственную связь с принципами действия самой ЭВМ. Есть признаки того, что к этому выводу приходят постепенно также и другие специалисты, о чем свидетельствует тяготение многих центров по обработке информации к обучению программистов языкам низкого уровня. После того как обучающийся программист овладевает приемами практического использования языка низкого уровня, одновременно воспринимая принципы действия той машины, на которой он работает, любое последующее обучение, требующееся для овладения языками более высокого уровня, значительно облегчается.

Как уже упоминалось, язык Ассемблера является широко используемым языком низкого уровня для Системы/360. Какие же языки высокого уровня, или операторные языки, могут использоваться с ЭВМ этой серии? Вместо того чтобы давать полный

список таких языков, достаточно отметить, что, пожалуй, наиболее широко используемыми языками являются PL/1, COBOL, FORTRAN и ALGOL. Наименование COBOL — сокращенное название от **CO**mmon **B**usiness **O**riented **L**anguage (универсальный язык, ориентированный на деловые применения), FORTRAN — от **FOR**mula **TRAN**slation (трансляция формул), ALGOL — от **ALGO**rithm **L**anguage (алгоритмический язык) и PL/1, язык, разработанный фирмой IBM, — от **P**rogramming **L**anguage **1** (язык программирования 1). Язык COBOL, как говорит само его название, обычно используется для деловых и коммерческих применений, FORTRAN и ALGOL очень удобны для научных и инженерных применений, а PL/1 обладает многими характеристиками предыдущих трех языков.

Языки высокого уровня, используемые в настоящее время, такие, как четыре упомянутых выше языка, постоянно совершенствуются. Несомненно, время от времени будут появляться новые языки этого типа, так же как и новые версии существующих языков.

Нет жестких стандартов для определения типа языка программирования, который следует применять в системе обработки информации. Почти единственной характеристикой, которую можно предсказать и ею оперировать, является возможное различие в языках высокого уровня, существующее между научными или инженерными применениями, с одной стороны, и коммерческими или деловыми — с другой. Определенные языки программирования высокого уровня четко ориентированы на особый тип применения, но это не ставит препятствий на пути их использования для других целей. Язык Ассемблера как язык низкого уровня обладает той особенностью, что его можно использовать для любого вида программируемых задач — коммерческих, научных, задач исследования ЭВМ или усовершенствования их математического обеспечения.

Б. ПЕРИФЕРИЙНОЕ ОБОРУДОВАНИЕ

Неотъемлемой частью любой вычислительной системы является особого типа аппаратура, которая должна обеспечивать ввод данных для работающих программ и в свою очередь выводить результаты обработки. Нет необходимости в том, чтобы функции ввода и вывода выполнялись одним и тем же устройством: бывает, что устройство одного типа обеспечивает ввод данных, а другого — их вывод. Это оборудование может сильно варьироваться по типу и конфигурации; структура каждой вычислительной системы приспособлена к потребностям данного пользователя. Аппаратура ввода-вывода может состоять из нескольких устройств одного и того же типа или может включать

в себя много устройств различных типов. Каждое устройство соединяется с устройством центрального процессора либо непосредственно, либо через устройство управления, которое взаимодействует с ЦП.

Прочитав на следующих страницах описания устройств, вы обнаружите там достаточно информации, чтобы познакомиться с общими физическими характеристиками и свойствами каждого из них и общими методами их применения в вычислительной системе. В силу того что данная книга ориентирована на обучение программированию, эти описания не слишком углубляются в детали механики и электроники внешних устройств.

1. Накопители на магнитной ленте (НМЛ)

Использование магнитной ленты для хранения и поиска данных является общепринятым методом выполнения этих функций. Хотя в настоящее время НМЛ брошен вызов со стороны запоминающих устройств с прямым доступом, применяемых в вычислительных системах 3-го поколения, они остаются носителями информации для операций ввода-вывода, обладающими максимальной емкостью.

Магнитная лента состоит из тонкой пластиковой пленочной основы с оксидным покрытием. Это покрытие воспринимает электрические импульсы, которые намагничивают группы оксидных атомов, называемых иногда логически *битами*. Расположение этих групп представляет *расположение битов* данных, которые записаны на ленте и останутся в такой конфигурации до тех пор, пока не будут изменены последующей перезаписью. Биты, записанные на поверхности ленты, располагаются рядами поперек лицевой стороны оксидной поверхности от края до края. В одних накопителях на магнитной ленте каждый ряд битов представляет один символ, в других — для представления символа требуется два ряда битов. Эти ряды битов располагаются вдоль ленты на различном расстоянии друг от друга для разных моделей накопителей на магнитной ленте. Величина расстояния между последовательными рядами битов определяет *плотность* записи на магнитную ленту. Система/360 работает с магнитными лентами, записанными с плотностью 800 BPI и 1600 BPI [1]). BPI (bits per inch — битов на дюйм) представляет размерность плотности в битах или рядах битов, приходящихся на дюйм длины магнитной ленты. Лента, записанная с плотностью 1600 BPI, имеет 1600 рядов намагниченных битов данных на каждый дюйм длины ленты, содержащий данные. Сокращение

[1]) 32 и 64 импульса на миллиметр соответственно. — *Прим. ред.*

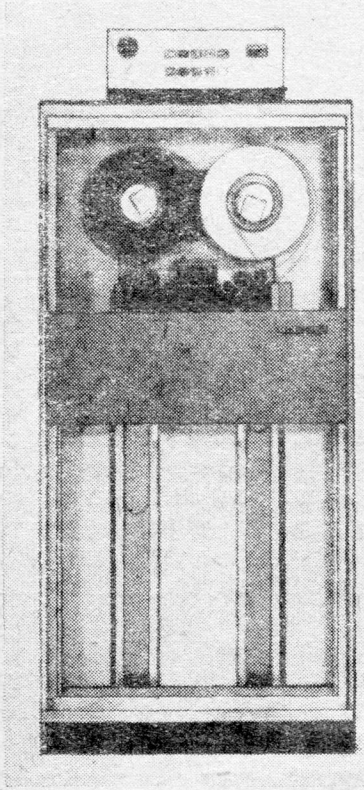


Рис. 2.1. Накопитель на магнитной ленте 2401.

BPI трактуется как байты на дюйм, если один ряд битов данных рассматривается как отдельный байт данных.

Магнитная лента обычно не содержит бесконечной цепи смежных рядов битов данных. После *записи* логического блока данных на поверхность ленты система пропускает часть поверхности ленты, прежде чем писать следующий логический блок данных. Та часть ленты, которая не используется, называется *промежутком между записями* (IRG, inter-record gap). Помимо прочих управляющих функций, этот промежуток используется системой как логическое и физическое средство разделения блоков данных, хранящихся на ленте.

Система/360 предоставляет широкий диапазон конфигураций и моделей накопителей на магнитной ленте (рис. 2.1). Некоторые устройства работают только с одной определенной плотностью записи на магнитную ленту, другие — с двумя величинами плотности. Скорость, с которой магнитная лента обрабатывается накопителями, измеряется количеством K байтов в секунду (KBS, K bytes per second), где K равно 1024. Накопитель, который обрабатывает магнитную ленту со скоростью 90 KBS, на самом деле работает со скоростью 92160 байтов в секунду. Некоторые накопители обрабатывают данные со скоростью 15 KBS, другие могут обрабатывать магнитную ленту со скоростью 180 KBS.

Магнитная лента представляет собой сравнительно дешевое средство хранения больших массивов данных. Боббины с магнитной лентой можно хранить в специальных архивах до тех пор, пока в них не возникнет необходимость. Одним из способов практического применения магнитной ленты является использование ее в качестве носителя для вывода информации; данные, которые необходимо напечатать как результат работы программы или системы, записываются вместо этого на магнитную

ленту. Распечатка этих данных может затем быть выполнена в более подходящий момент времени на другой вычислительной или автономной печатающей системе.

2. Устройства ввода-вывода перфокарт

Перфокарты, по всей вероятности, наиболее старый носитель информации для ввода-вывода из всех применяемых в настоящее время с вычислительными машинами. Задолго до появления магнитной ленты, устройств с прямым доступом и дисплеев перфокарты использовались для систем хранения данных так же, как и в их сегодняшних применениях при первоначальном вводе данных, вводе программ на исходном языке, объектных модулей проблемной программы и управляющих операторов для решения задачи. В Системе/360 используются гибкие стандартные перфокарты, имеющие размер $3^1/_4 \times 7^3/_8$ дюймов, причем каждая перфокарта содержит 80 колонок. Каждая колонка состоит из 12 вертикальных позиций, которые могут содержать небольшие прямоугольные отверстия, пробитые в карте. Комбинация отверстий или одиночное отверстие, пробитое в вертикальной колонке, интерпретируется системой как данные, представленные в шестнадцатеричном или двоичном коде или коде EBCDIC.

Считывающая часть комбинированного устройства ввода-вывода перфокарт (рис. 2.2) передает данные центральному процессору, «нащупывая» отверстия, содержащиеся на каждой карте, по мере того как она продвигается в механизме считывания. Собственно считывание («ощупывание») отверстий обычно выполняется небольшими проволочными щетками или фотоэлектрическими устройствами. В механизмах щеточного типа карта продвигается под набором из 80 щеток, по одной щетке на каждую колонку на карте. При наличии отверстия щетка войдет в соприкосновение с платой, находящейся под перфокартой, и отверстие в данной позиции перфокарты регистрируется аппаратурой. Информация о каждой вертикальной колонке интерпретируется затем управляющими программами системы. В считывающих устройствах, использующих светочувствительные элементы, источник света выполняет функции щеток, а фотоэлектрические ячейки — платы. При продвижении перфокарт свет через отверстия попадает на светочувствительные ячейки, и таким образом осуществляется регистрация отверстий в перфокарте.

Перфорационная часть комбинированного устройства обычно выполняет процедуру, с точки зрения логики работы обратную процедуре считывания, но работает медленнее из-за необходимости пробивки требуемых отверстий в перфокартах. Данные, которые должны быть набиты на перфокарту, преобразуются в

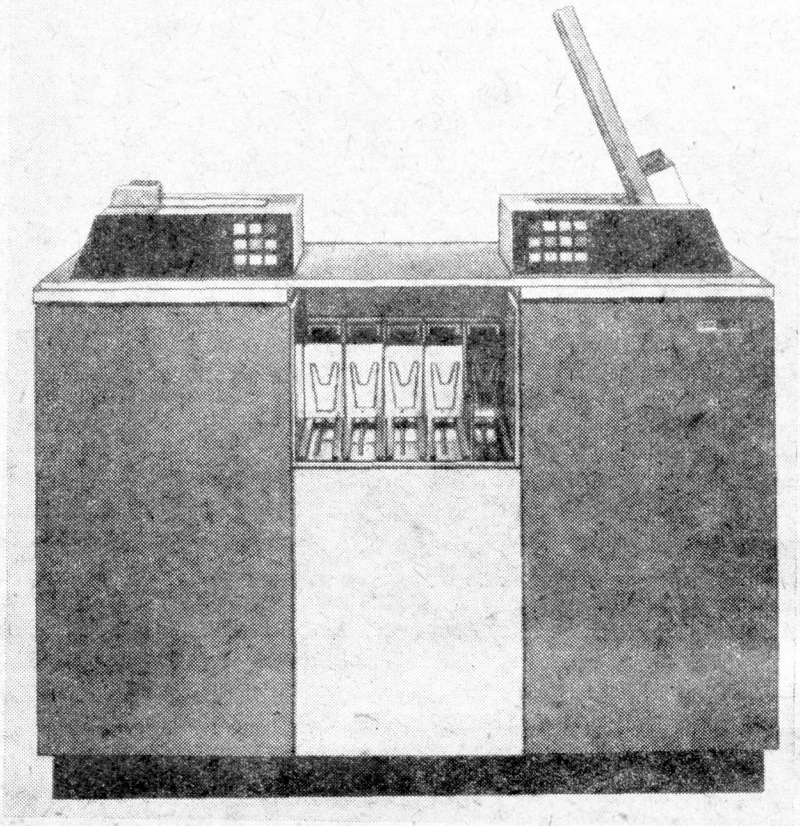


Рис. 2.2. Устройство ввода-вывода перфокарт 2540.

соответствующий код и передаются на устройство пробивки. Перфорирующие головки в процессе прохождения под ними перфокарты активизируются электронными схемами в соответствии с позициями отверстий, которые должны быть пробиты в вертикальных колонках карты. Карты, которые перфорируются как выходные результаты одной задачи, впоследствии часто используются в качестве входных данных для другой задачи или операции.

3. Печатающие устройства

Печатающие устройства являются средством визуального представления данных, выводимых из ЭВМ, в форме отпечатанного документа с несколькими копиями или без них. Термин *печатающее устройство* (printer) в подавляющем большинстве

случаев относится к устройству, которое используется исключительно для вывода данных. При некоторых обстоятельствах пишущая машинка, способная выполнять как операции ввода, так и операции вывода, может рассматриваться как печатающее устройство, но из-за относительно низкой скорости она представляет собой очень дорогое средство вывода данных на печать. Истинное печатающее устройство обычно представляет собой быстродействующее устройство печати реечного или цепного типа.

Печатающее устройство *реечного типа* содержит повторяющиеся сегменты печатаемых символов, вмонтированные в рейку, которая перемещается попеременно слева направо и справа налево вдоль печатаемой строки. Каждая позиция печати имеет молоточек, который перемещает выбранный сегмент рейки вперед по направлению к печатающей ленте, и после соприкосновения с ней отображает этот символьный сегмент на бумаге. В процессе такого челночнообразного движения молоточек каждой печатной позиции активизируется, когда сегмент рейки, проходящий перед ней, соответствует тому символу, который предопределен управляющей программой. При удачном расположении сегментов на рейке многие позиции могут быть отпечатаны одновременно.

Печатающие устройства *цепного типа* (рис. 2.3) выполняют в основном те же функции. От других печатающих устройств они отличаются характеристиками механизма протяжки.

Печатающая цепь состоит из последовательности звеньев, образующих непрерывную цепь. На каждом звене расположены две литеры. Эта цепь, укрепленная на двух шестернях, перемещается горизонтально в одном направлении с высокой скоростью. Аналогично устройству реечного типа каждая позиция печати представлена молоточком, который активизируется, когда выбранный для печати символ располагается напротив него. Однако печать символа производится не так, как в устройствах реечного типа. Молоточек подает бумагу в направлении звена печатающей цепи.

Существует еще одна разновидность цепной печати, отличающаяся тем, что сегменты, каждый из которых содержит три литеры, механически не связаны друг с другом. Сегменты последовательно смонтированы внутри дорожки, в которой они свободно скользят, приводимые в движение с помощью ведущей шестеренки на одном конце и направляющей шестеренки на другом. Сегменты содержат зубья, которые входят в зацепление с ведущей шестеренкой для их перемещения вдоль дорожки. Печать каждого символа в принципе происходит точно так же, как и при обычной цепной печати, но имеются некоторые отличия в физических характеристиках механизма молоточков.

Скорость работы некоторых печатающих устройств, например 1100 строк в минуту, зачастую производит на зрителей большее впечатление, чем внутреннее функционирование самой вычислительной машины. Печатающее устройство выполняет механическую работу, которая гораздо легче может быть воспринята и понята неспециалистом, чем функционирование электроники процессора, так как ему легче сопоставить скорость печатающего устройства с реальными физическими процессами.

Печатающие устройства обычно используют непрерывную бумажную ленту стандартной ширины, которая перфорируется в местах сгиба. Это облегчает хранение и позволяет в случае необходимости оторвать часть ленты. Длина каждой из предполагаемых печатных страниц определяется расстоянием от одного ряда перфорации до следующего. Многие фирмы в настоящее время выпускают бумагу для печати на ЭВМ в виде специальных бланков, бирок, накладных, чеков и т. д.

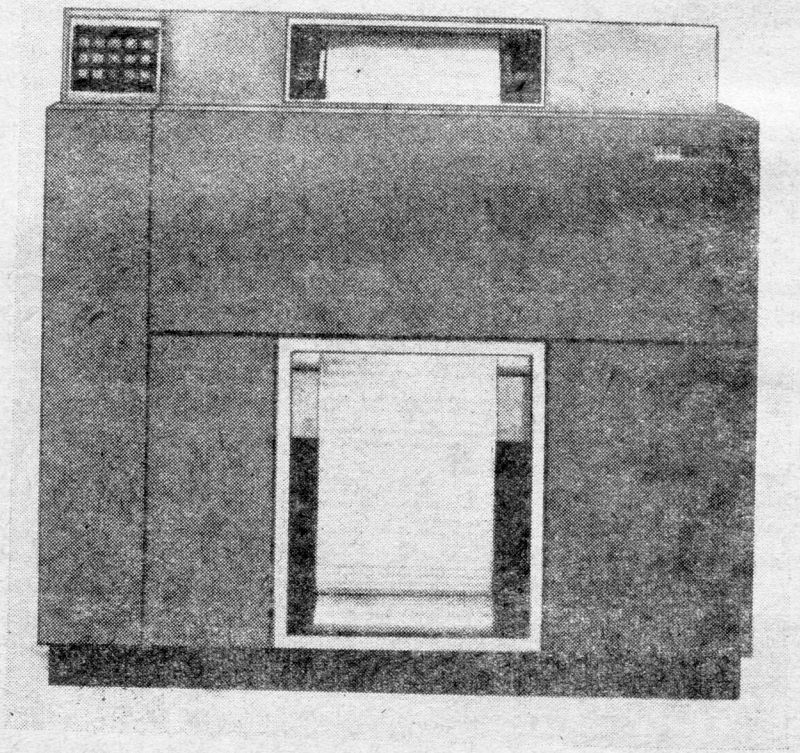


Рис. 2.3. Печатающее устройство 1403.

4. Запоминающие устройства с прямым доступом

Диски, барабаны и запоминающие устройства на магнитных картах представляют собой три класса устройств, которые обычно относят к устройствам с прямым доступом. Их называют так потому, что они дают возможность операционной системе прямо обращаться к любой записи, блоку или области данных, не обрабатывая всю предыдущую информацию внутри тома или набора данных. Большинство устройств с прямым доступом применяется для выполнения одной или более из следующих функций:

1) запоминания больших массивов данных;

2) запоминания данных, которые необходимо отыскивать методом, отличным от последовательной обработки;

3) запоминания данных, которые должны оставаться активно доступными для программы или системы.

Устройства с прямым доступом по методу, сходному с применяемым для магнитной ленты, записывают данные на оксидную поверхность. Однако для каждого из трех упомянутых классов устройств физические свойства запоминающей среды существенно отличаются друг от друга. Запоминающие устройства на дисках состоят из одной или более круглых пластин — *дисков*, напоминающих граммофонные пластинки, где данные хранятся на *дорожках*.

Подобно тому как диск напоминает граммофонную пластинку, дорожки можно себе представить похожими на канавки записи на этой пластинке. Но дорожки не направлены по спирали к центру диска, они концентричны. Каждая дорожка — один полный круговой путь, начинающийся и кончающийся у точки индексного маркера. Набор таких сменных пластин называют *пакетом дисков*.

Каждый диск имеет дорожки на верхней и нижней поверхностях, число дорожек на каждой поверхности зависит от типа устройства, для которого этот пакет дисков предназначен. Например, устройства с прямым доступом фирмы IBM как типа 2311 (рис. 2.4), так и 2314 (рис. 2.5) используют пакеты дисков, содержащие по 200 адресуемых дорожек на одну поверхность; диски устройства фирмы IBM типа 2302 содержат 492 адресуемые дорожки, но они выполняются стационарными и не допускают смены.

Цилиндр — это термин, который используется для описания всех адресуемых дорожек, располагающихся по вертикали одна под другой. Например, если внешней дорожке на поверхности верхнего диска присвоить номер 0, следующей дорожке в направлении к центру — 1, следующей — 2 и т. д., то цилиндр образуется всеми дорожками, находящимися в той же самой

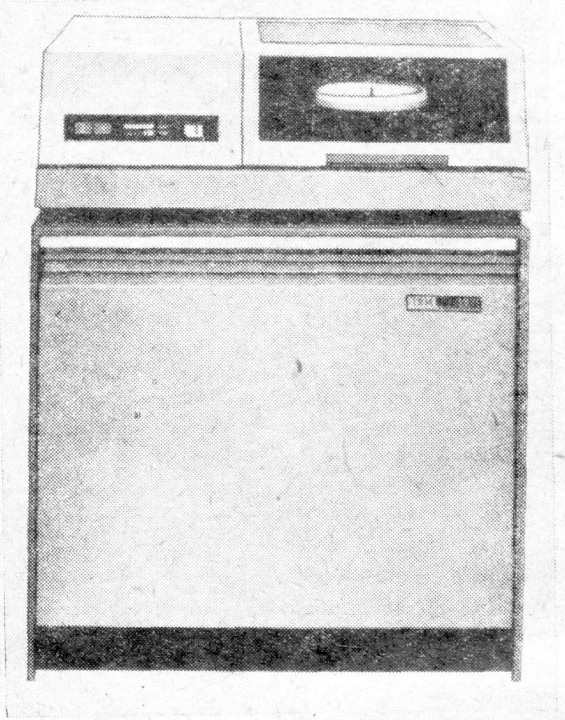


Рис. 2.4. Запоминающее устройство с прямым доступом 2311.

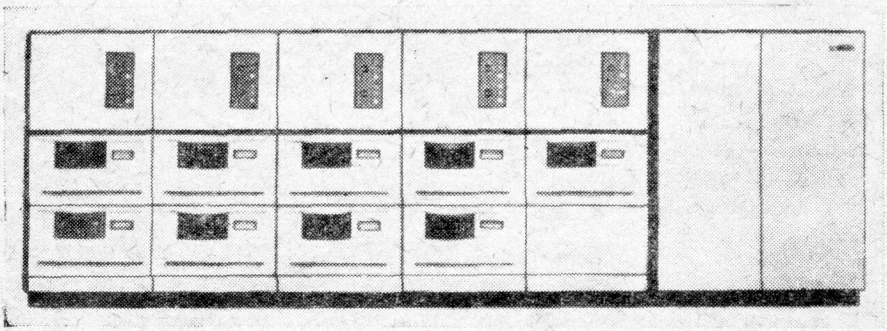


Рис. 2.5. Запоминающее устройство с прямым доступом 2314.

относительной позиции на всех дисковых поверхностях. Первый цилиндр образуют внешние дорожки на всех поверхностях всех дисков этого пакета; второй цилиндр — все дорожки с номером 1 на всех дисках и т. д.

Чтение или запись данных на устройствах дисковой памяти выполняется головками чтения-записи, установленными на подвижном механизме доступа. Каждая используемая поверхность имеет головку (или головки) чтения-записи, которая перемещается механизмом доступа к той дорожке, где должна производиться операция записи или считывания. Устройства с прямым доступом типа 2311 и 2314 имеют один механизм доступа — если одна из головок установлена на некоторую дорожку, все остальные головки располагаются по образующей этого цилиндра. Устройство фирмы IBM типа 2302 имеет два механизма доступа на один модуль; один из них осуществляет доступ к внешней половине дорожек на всех поверхностях, а второй — к внутренней половине дорожек.

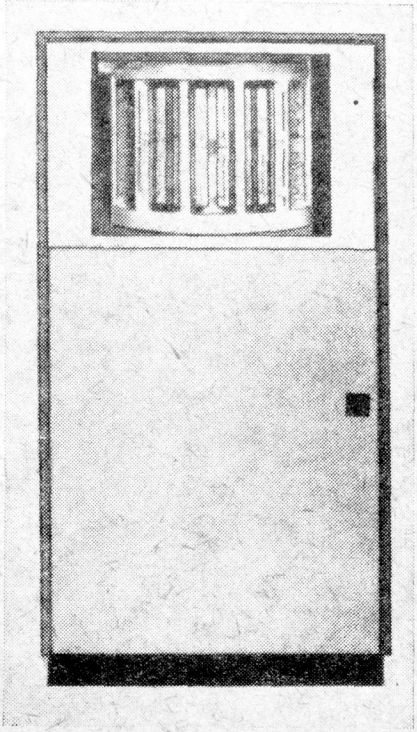


Рис. 2.6. Барабан 2303.

Запоминающее устройство на магнитном барабане имеет в своем составе один вертикально установленный барабан. В отличие от запоминающих устройств на магнитных дисках оно имеет одну записывающую поверхность, образуемую внешней поверхностью цилиндрической части самого барабана. Дорожки расположены в горизонтальной плоскости вокруг барабана, и каждая дорожка имеет свою собственную неподвижную головку чтения-записи. Если нужно сделать запись на конкретную дорожку или считать с нее, то выбирается соответствующая головка и производится необходимая операция. Закрепление отдельной головки за каждой дорожкой уменьшает время, затрачиваемое на доступ к данным, хранящимся на этой дорожке, так как при такой конструкции нет необходимости ждать, пока механизм доступа передвинет головку в позицию, соответствующую заданной дорожке (рис. 2.6).

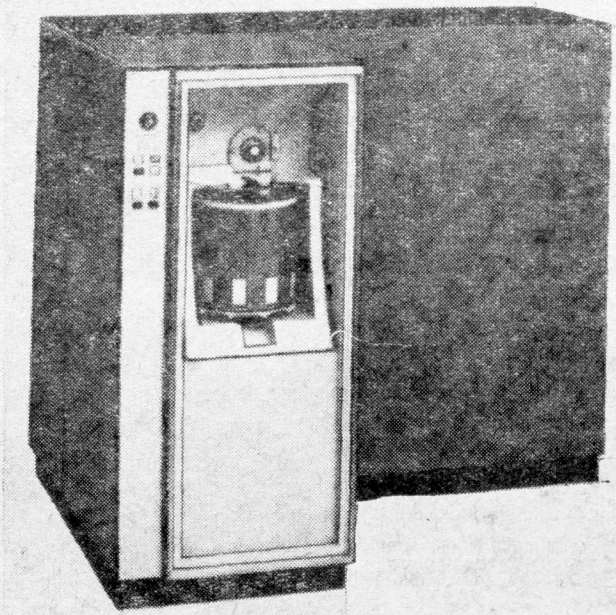


Рис. 2.7. Накопитель на магнитных картах 2321.

Барабан фирмы IBM типа 2303 производит запись данных последовательно таким же способом, что и запоминающие устройства на магнитных дисках. Данные записываются одновременно только на одну физическую или логическую дорожку. При этом обеспечивается скорость чтения-записи приблизительно 300000 байтов в секунду. Однако если говорить о барабане фирмы IBM типа 2301, то одна его логическая дорожка в действительности состоит из четырех физических дорожек; данные записываются на четыре физические дорожки как логические группы из 4 битов, по 1 биту на каждую дорожку. Эта особенность барабана типа 2301 обеспечивает ему возможность выполнения операций чтения-записи со скоростью, приблизительно в четыре раза большей, чем скорость устройства 2303, или приблизительно равной 1,2 миллиона байтов в секунду.

Накопитель на магнитных картах фирмы IBM типа 2321 (рис. 2.7) совмещает в себе свойства НМЛ и устройства с прямым доступом. Данные хранятся на оксидной поверхности карт, имеющих размеры приблизительно $2^1/_4$ дюйма в ширину и 13 дюймов в длину. Эта карта напоминает обычную магнитную ленту, но основа, на которую нанесено покрытие, приблизительно в три раза тоньше. Карты организованы в подъячейки внутри

ячеек, причем каждая подъячейка содержит десять карт, а каждая ячейка — 20 подъячеек, или 200 карт. Как и в случае других устройств с прямым доступом, данные хранятся на дорожках. Имеется 100 дорожек по 2000 байтов на каждую карту, или 200 000 байтов на одну карту. В целом все устройство прямого доступа на магнитных картах состоит из десяти подвижных блоков-ячеек, расположенных по окружности. Этот блок может вращаться для того, чтобы занять соответствующее положение по отношению к механизму доступа.

Когда обращаются к отдельной карте, то механизм вращения перемещает массив ячеек до тех пор, пока соответствующая подъячейка не окажется под механизмом доступа. Этот механизм выдвигает соответствующую карту из подъячейки и наворачивает ее на барабан, который в свою очередь пропускает карту под набором головок чтения-записи. Затем эта карта возвращается на соответствующую позицию внутри той подъячейки, из которой она была извлечена. Хотя накопитель на магнитных картах не позволяет обращаться к данным так же быстро, как другие устройства с прямым доступом, он является превосходным средством хранения больших массивов данных, так как дает возможность запомнить 400 миллионов байтов информации.

5. Дисплеи

Для коммерческих применений вычислительных машин относительно новыми являются устройства визуального отображения (дисплеи), использующие электронно-лучевые трубки (ЭЛТ). Эти устройства содержат ЭЛТ, весьма похожие на трубки обычных телевизионных приемников, но оснащены специальным устройством управления, которое обеспечивает их взаимодействие с центральным процессором ЭВМ. В Системе/360 имеется два основных типа дисплеев: IBM 2260 и IBM 2250. Каждое из этих устройств выпускается в нескольких вариантах, но принцип их действия, вообще говоря, одинаков. Устройства управления в значительно большей степени отличаются друг от друга в зависимости от размера, количества оборудования и производительности дисплея. Данные передаются от ЭВМ к устройству управления или посылаются с клавиатуры дисплея и запоминаются в *буферах*, которые работают по методу, известному как *буферизация на линиях задержки*. Данные, хранящиеся в этих буферах, отображаются на экране ЭЛТ и остаются там до тех пор, пока не будут заменены проблемной программой или по запросу с клавиатуры дисплея.

Устройство IBM 2260 (рис. 2.8) имеет экран для представления алфавитно-цифровой информации и клавиатуру для ввода-

вывода данных и может использоваться в качестве *локального* или *удаленного* терминала.

При локальном присоединении операции осуществляются посредством прямой связи между устройством управления и каналом ЦП; режим удаленного терминала заключается в том, что устройство дисплея и устройство управления передают и воспринимают данные через линии связи, например телефонные кабели, и взаимодействуют с ЦП через устройства управления этими линиями — мультиплексоры передачи данных. Это устройство чрезвычайно удобно для применения в информационно-поисковых системах оперативного редактирования наборов данных и даже внесения изменений непосредственно в программу в ходе решения задачи вместо ввода данных с перфокарт.

Устройство IBM 2250 (рис. 2.9) имеет в своем составе так называемый *растровый* экран. Свечение сетки экрана может быть использовано для представления символьных данных, ри-

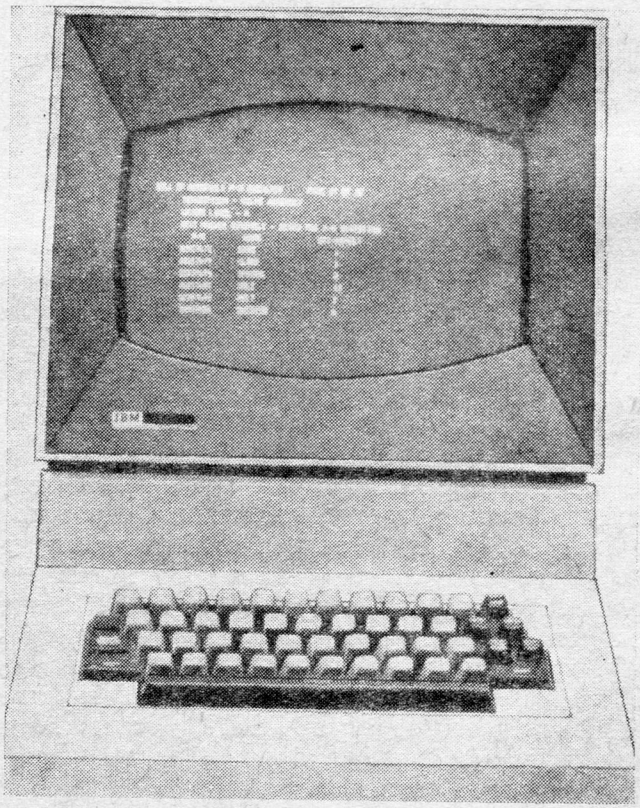


Рис. 2.8. Устройство IBM 2260.

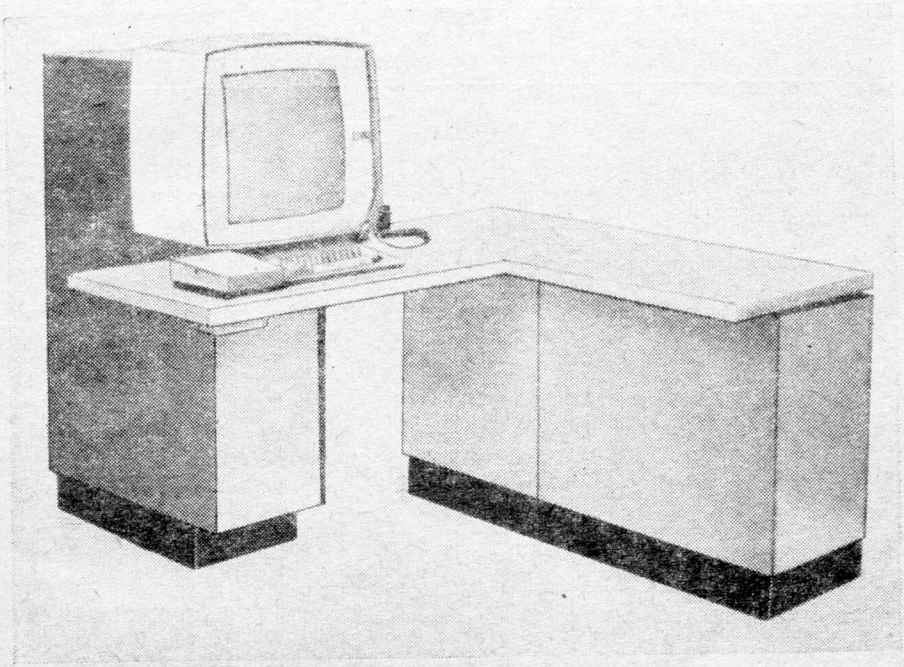


Рис. 2.9. Устройство IBM 2250.

сунков, графиков, чертежей и т. п. Помимо стандартной символьной клавиатуры, это устройство имеет функциональную клавиатуру, состоящую из группы клавиш, каждая из которых может быть запрограммирована для выполнения определенной функции или специальной подпрограммы, связанной с решаемой задачей. Для внесения изменений в данные на экране можно использовать световое перо. Пусть, например, на экране высвечена схема электронного устройства, извлеченная из запоминающего устройства с прямым доступом; используя световое перо, можно исправить или заменить компоненты этого чертежа, после чего скорректированная схема может быть возвращена в запоминающее устройство.

Хотя дисплеи описанного выше типа не работают со скоростью, с какой мы обычно связываем функционирование ЭВМ, они предоставляют проектировщику массу возможностей. Тот существенный факт, что эти устройства предназначены для применения в оперативном режиме, устраняет необходимость в их сверхвысокой производительности, так как они должны работать с такой скоростью, чтобы дать возможность оператору считывать, воспринимать и выдавать данные.

ЧАСТЬ II

Введение в язык Ассемблера

Глава 3

Представление чисел и системы счисления

В этой книге нам придется иметь дело с пятью типами представления чисел. Четыре из них выражают системы счисления, которые, возможно, незнакомы обучающемуся программисту, — шестнадцатеричная, двоичная, с фиксированной точкой и десятичная в упакованном формате. Пятый является стандартным представлением цифр в коде EBCDIC[1]).

Описания и объяснения, приведенные в данном разделе, носят вводный характер. Их цель — дать читателю знания о структуре чисел Системы/360 и языка Ассемблера, достаточные для понимания логики работы системы. Более детальные сведения будут сообщаться по мере необходимости.

А. ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ ВЕЛИЧИН

Эта схема представления чисел основана на применении всего двух числовых значений — 0 (нуль) и 1 (единица). Любая из этих двух цифр, написанная отдельно, выражает соответствующее значение: нуль — для цифры 0, единица — для цифры 1. Однако, когда некоторая последовательность этих цифр, например 10010101, интерпретируется как величина, каждая последующая цифровая позиция, расположенная слева, понимается как представляющая величину, в два раза большую, чем цифровая позиция, расположенная справа от нее. Значение величины подсчитывается путем сложения всех величин, представляющих значащие (равные единице) цифровые позиции. Такое

---

[1]) Стандартный в рамках фирмы IBM восьмибитный код для представления букв, цифр, знаков и управляющих символов, в совокупности составляющих алфавит вычислительной системы. — *Прим. ред.*

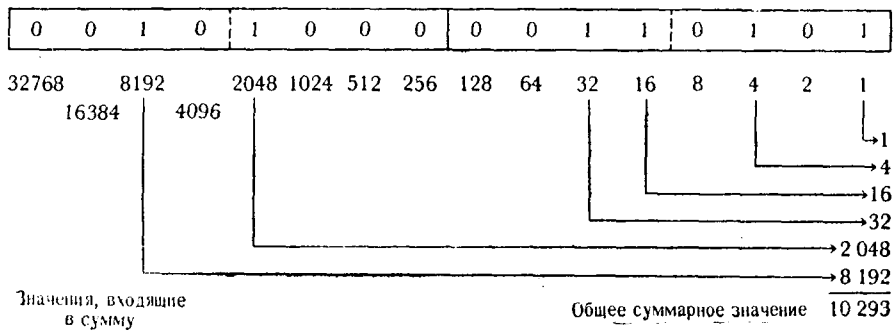
представление известно как двоичное представление по степеням двойки, и, по существу, на нем основана двоичная система счисления. Последовательные двоичные позиции (биты) любого полубайта, байта или совокупности байтов можно рассматривать как двоичное представление по степеням двойки, вычисляя значения соответствующих двоичных чисел.

Ниже показаны потенциальные значения последовательности двоичных позиций.

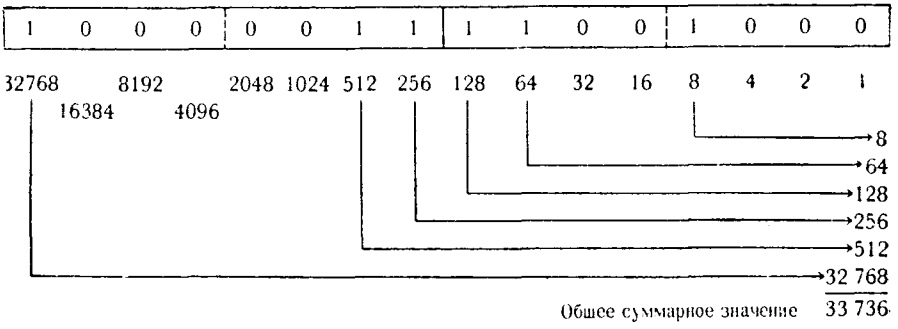
Потенциальные значения двоичных разрядов

На этой иллюстрации представлено 16 двоичных позиций, которые содержатся внутри двухбайтового поля. Как было установлено, потенциальное значение любой отдельной двоичной позиции не считается эффективным, если эта позиция не занята единицей. Следовательно, абсолютное значение числа, представленного на иллюстрации, можно считать равным нулю, так как в этом поле нет ни одного значащего двоичного разряда. Используя то же самое двухбайтовое поле, но содержащее различные двоичные конфигурации, рассмотрим на примерах накопление значащих двоичных разрядов в суммарное значение.

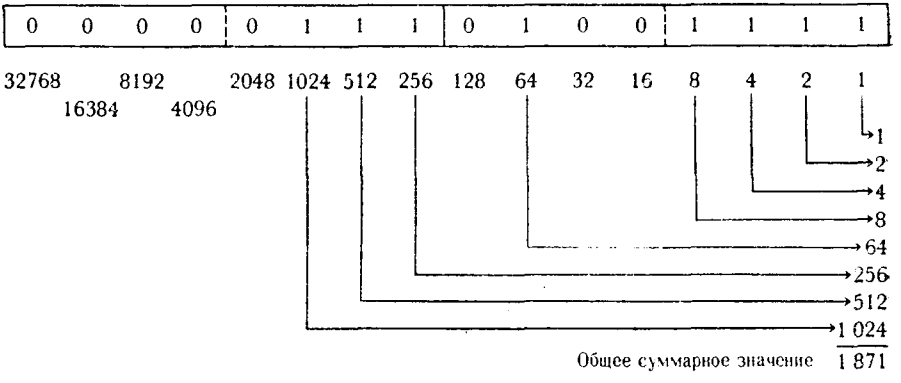
Пример 1.



Пример 2.



Пример 3.



Максимально возможное значение двоичного числа, размещенного в некотором поле, ограничено числом двоичных позиций поля. Эти значения для различных длин поля с приращением по половине байта даны ниже.

Длина поля в байтах	Максимальное значение двоичного числа
1/2	15
1	255
1 1/2	4095
2	65535
2 1/2	1048575
3	16277215
3 1/2	268435455
4	4294967295
4 1/2	68719476735
5	1099511627775

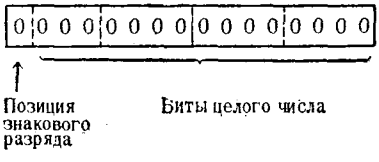
Возможное значение любой одиночной двоичной позиции внутри 32-битового поля можно определить с помощью таблицы, приведенной в части V этой книги. Каждый последующий элемент этой таблицы представляет относительную двоичную позицию двоичного поля справа налево.

Значение, выраженное в рассмотренном выше «истинно» двоичном представлении, не имеет определенного знака; знак величины должен, следовательно, считаться положительным. В силу этого такое представление ограничено само по себе как средство выражения чисел. С другой стороны, оно является основой для системы представления чисел с фиксированной точкой.

Б. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ С ФИКСИРОВАННОЙ ТОЧКОЙ

Операции с фиксированной точкой используются операционной системой при работе с адресами, смещениями и при манипуляциях с общими регистрами. С помощью команд с фиксированной точкой языка Ассемблера они доступны программисту как средство выполнения компактных и удобных арифметических операций. Представление чисел с фиксированной точкой базируется на двоичном представлении величин с дополнительной возможностью выражения как отрицательных, так и положительных значений.

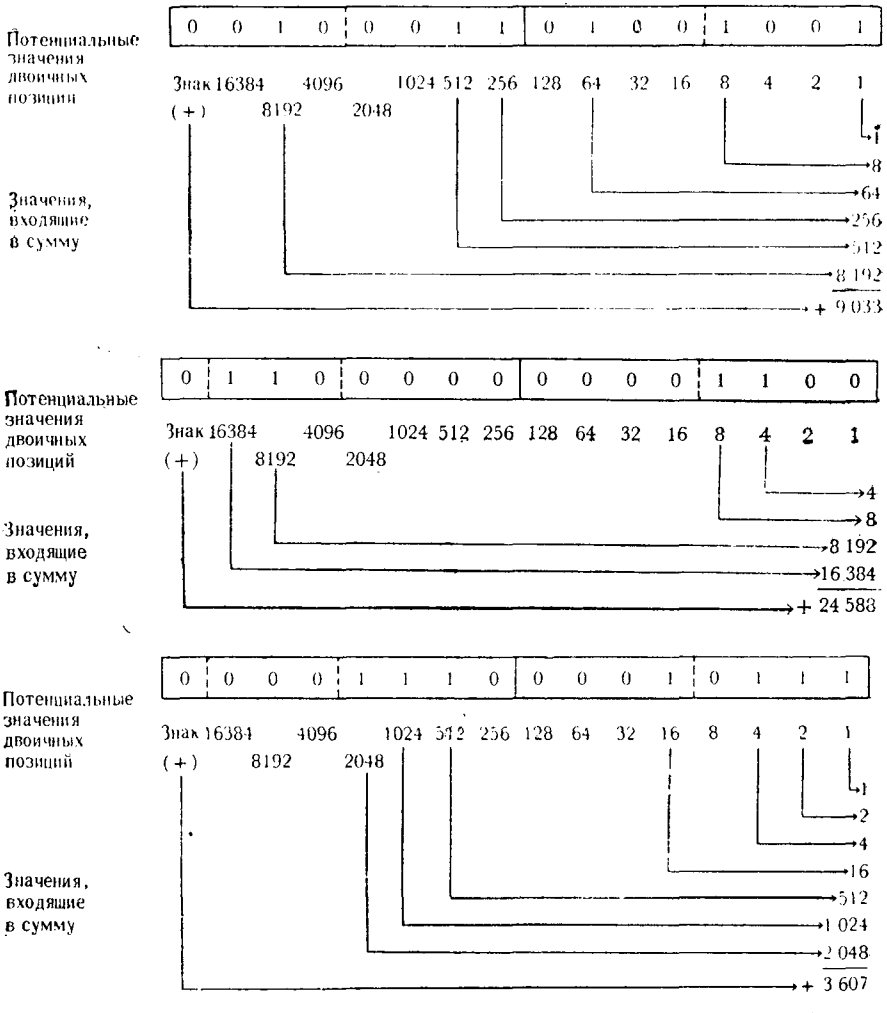
Знак величины с фиксированной точкой определяется самым левым (старшим) битом поля. Все остальные биты служат для представления целого числа. На следующем примере показано 16-разрядное поле и в нем 1 бит знака и 15 битов, используемых для представления целого числа (поле целого числа).



В связи с тем что числа с фиксированной точкой в зависимости от их значения, вообще говоря, могут быть представлены в форматах полуслова или полного слова, количество битов для представления целого числа зависит от размера поля. Величина с фиксированной точкой размером в полуслово содержит 1 знаковый бит и 15-битовое поле целого числа; величина с фиксированной точкой размером в полное слово содержит 1 знаковый бит и 31-битовое поле целого числа.

Положительное число с фиксированной точкой выражается в описанном выше двоичном представлении с нулевым знаковым битом. Значащие разряды поля целого числа представляют значения, вычисляемые как 2 в степени, определяемой позицией

разряда внутри поля целого числа. Сумма этих эффективных значений и дает значение положительной величины с фиксированной точкой. Проиллюстрируем сказанное на следующих ниже примерах, использующих величины с фиксированной точкой длиной в полуслово. Форма представления чисел в этих примерах аналогична применявшейся при обсуждении двоичного представления.



Отрицательная величина с фиксированной точкой выражается как целочисленная часть длиной 31 или 15 битов, записанная в виде дополнения до двух, с единицей в знаковом разряде. При такой записи в качестве «значащей» цифры поля целого числа следует рассматривать нуль как противоположный по

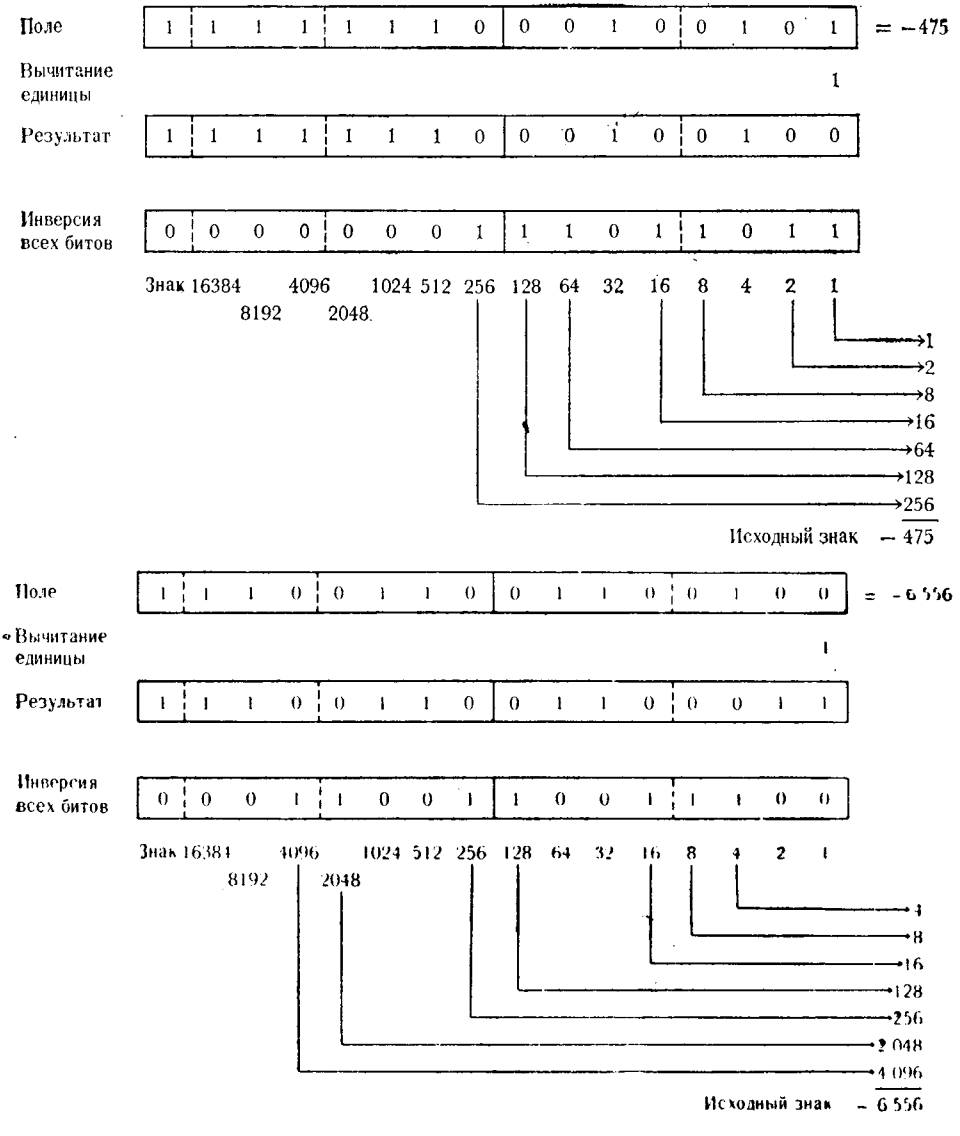
значению знаковому биту. Значение отрицательной величины с фиксированной точкой можно вычислить с помощью следующих действий:

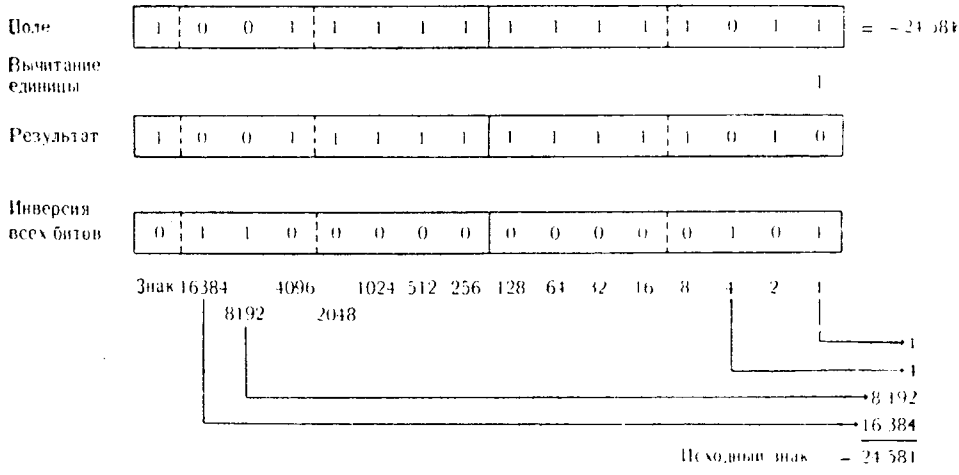
1) вычесть единицу из младшего (самого правого) бита поля;

2) инвертировать содержимое всех позиций;

3) сложить величины всех значащих разрядов.

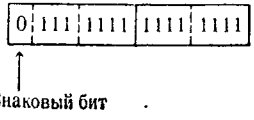
Этот процесс показан на следующих иллюстрациях.



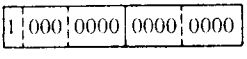


Максимальные положительные и отрицательные числа, которые могут находиться внутри полуслова или полного слова, и их двоичная конфигурация показаны ниже.

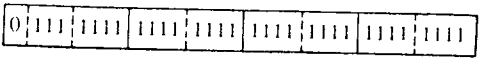
Максимальное положительное значение числа с фиксированной точкой длиной в полуслово = +32 767



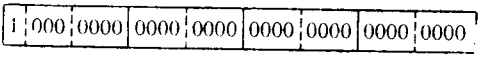
Максимальное отрицательное значение числа с фиксированной точкой длиной в полуслово = —32 768



Максимальное положительное значение чисал с фиксированной точкой длиной в слово = +2 147 483 647



Максимальное отрицательное значение числа с фиксированной точкой длиной в слово = —2 147 483 648



Отметим, что максимальное положительное значение числа с фиксированной точкой длиной в слово (+2147483647) меньше, чем максимальное двоичное число без знака, представимое пол-

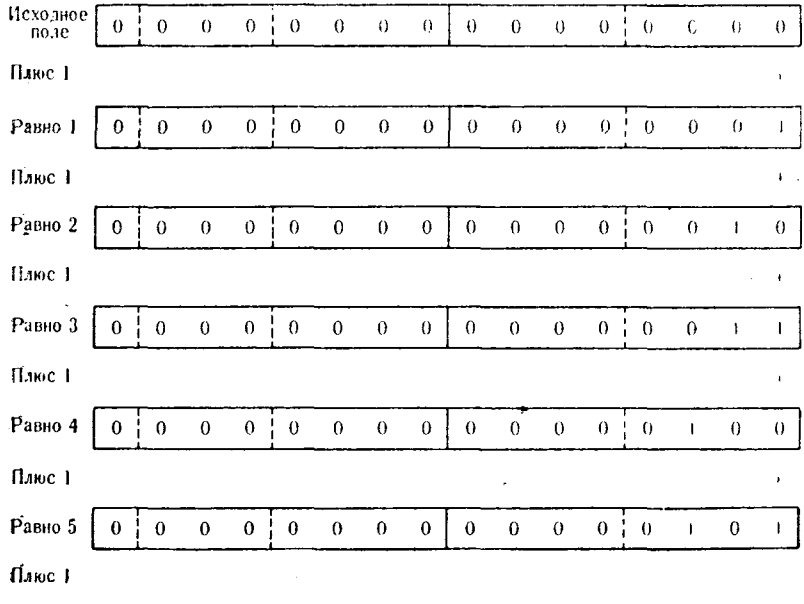
ным словом (+4294967295). Это объясняется тем, что величина с фиксированной точкой может использовать только 31 бит поля в качестве поля целого числа, в то время как чисто двоичное представление величины может использовать все 32 двоичные позиции.

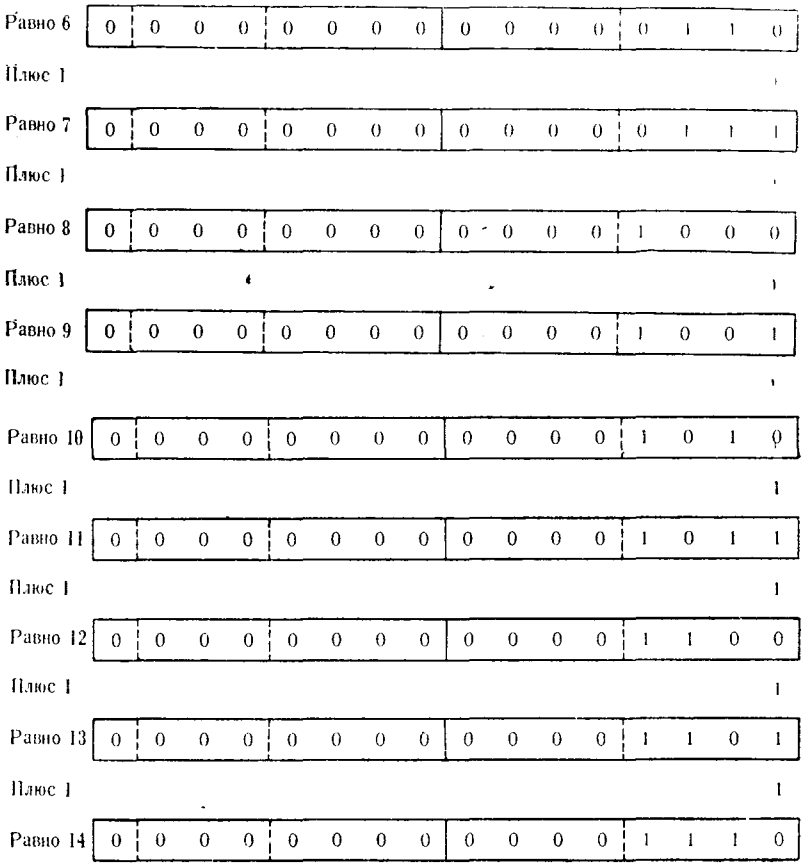
Сложение величин с фиксированной точкой, как и чисто двоично представленных величин, основано на следующей таблице сложения:

Таблица 3.1

Поле А		Поле В		Сумма
0	плюс	0	равно	0
0	плюс	1	равно	1
1	плюс	0	равно	1
1	плюс	1	равно	0 с переносом единицы в более старший разряд или другое поле

Следующая иллюстрация представляет последовательность сложений величины +1 с числом с фиксированной точкой длиной в полуслово:

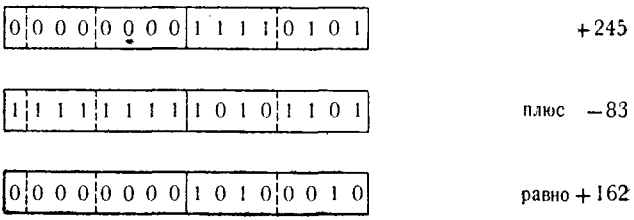




Вычитание чисел с фиксированной точкой выполняется системой посредством, во-первых, поразрядной инверсии вычитаемого, чтобы получить дополнение вычитаемого, и, во-вторых, сложения уменьшаемого с дополнением вычитаемого (см. примеры 1, 2 и 3).

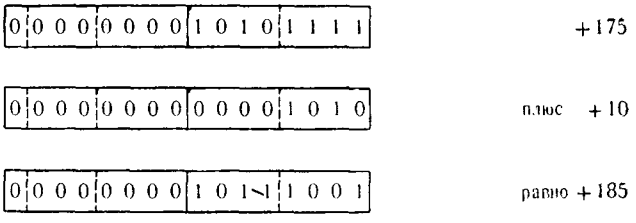
Пример 1.

+245минус+ 83 = +162 или +245 плюс −83 = +162



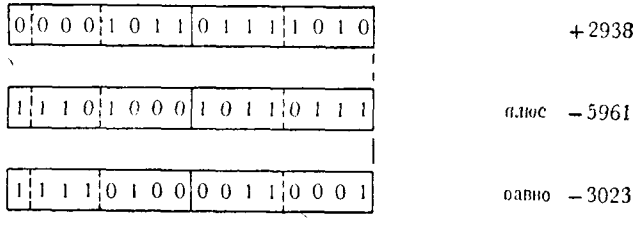
Пример 2.

+175минус−10 = +185 или +175 плюс +10 = +185



Пример 3.

+2938минус+5961 = −3023 или +2938 плюс −5 961 = −3023



Заметим, что в примере 2 отрицательную величину необходимо было вычесть из положительной. Вычитаемое (отрицательная величина) было сначала инвертировано до его дополнения, в результате чего получилась двоичная форма представления положительной величины. Затем ее прибавили к первой величине (уменьшаемому) и была получена правильная разность.

В. ШЕСТНАДЦАТЕРИЧНОЕ ПРЕДСТАВЛЕНИЕ

Шестнадцатеричная система счисления очень широко используется при программировании в Системе/360. Это численная структура «по основанию 16», в которой каждые четыре разряда двоичной конфигурации данных или величин представлены одной шестнадцатеричной цифрой. *Дамп* основной памяти, т. е. вывод на печать содержимого определенных областей основной памяти в заранее выбранный момент времени по указанию программиста или полученный автоматически в результате ненормального окончания обработки, осуществляется в шестнадцатеричном формате. В этом формате представлены адреса и команды в ассемблер-листинге исходной программы и многие другие функциональные величины в Системе/360. Даже величины с фиксированной точкой и чисто двоичные величины

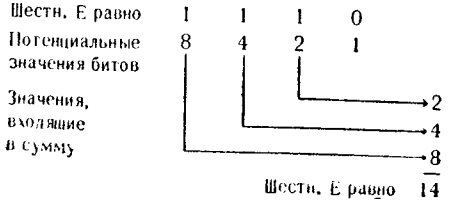
значительно легче воспринимаются, если они записаны в шестнадцатеричной форме, а не в двоичной. Поэтому очень важно, чтобы программист основательно усвоил принципы шестнадцатеричной системы счисления и шестнадцатеричных арифметических операций.

Для представления шестнадцатеричных цифр используются цифры от 0 до 9 и буквы латинского алфавита от A до F. Конфигурация каждой из этих шестнадцатеричных цифр показана в табл. 3.2.

Таблица 3.2

Десятичное число	Шестнадцатеричное число	Двоичная конфигурация
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

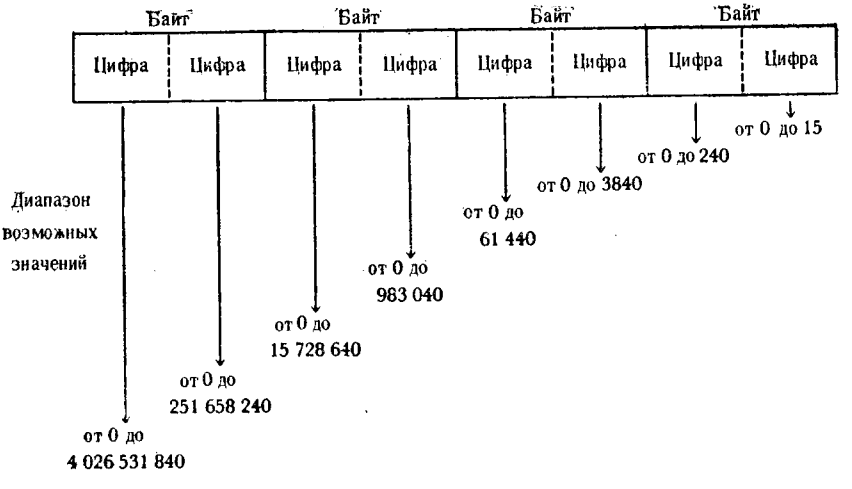
Десятичное значение любой шестнадцатеричной цифры непосредственно связано со значением ее двоичной конфигурации, рассматриваемой как двоичное число без знака. Поясним это на примере шестнадцатеричной цифры E и ее двоичной конфигурации.



Отдельная шестнадцатеричная цифра может, следовательно, представлять любую десятичную величину внутри диапазона чисел от 0 до 15. Однако случаи, когда шестнадцатеричная цифра используется как отдельная величина, редки. Значительно чаще для представления величины или множества данных используются последовательности шестнадцатеричных цифр. Составное шестнадцатеричное поле, т. е. поле, которое состоит более чем из одной шестнадцатеричной цифры, может представлять различного рода символьные или числовые данные. Необходимо знать подразумеваемую конструкцию поля и тип величины, которая предположительно находится в этом поле, прежде чем пытаться анализировать ее значение. В следующих разделах будет установлена соответствующая связь между шестнадцатеричными форматами полей и типом данных, которые представлены этими полями.

Когда шестнадцатеричные цифры используются для представления десятичного значения двоичной величины или величины с фиксированной точкой, каждая шестнадцатеричная цифра рассматривается как единица приращения по основанию 16. Самая правая шестнадцатеричная цифра имеет максимальное значение 15, следующая более старшая цифра имеет максимальное значение 240, следующая — 3840 и т. д. Каждая последующая шестнадцатеричная цифра, считая справа налево, имеет потенциально максимальное значение, которое в 16 раз больше, чем потенциально максимальное значение цифры, стоящей справа от нее.

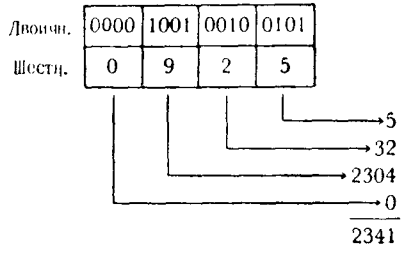
Потенциальный диапазон каждой шестнадцатеричной цифры четырехбайтового двоичного поля показан на следующей иллюстрации:



Для интерпретации двоичных величин, которые состоят более чем из одной шестнадцатеричной цифры, рекомендуется пользоваться приведенной в части V таблицей шестнадцатеричноно-десятичных преобразований. Она предоставляет удобный метод преобразования шестнадцатеричных цифр с учетом их позиций в числе в соответствующие десятичные эквиваленты.

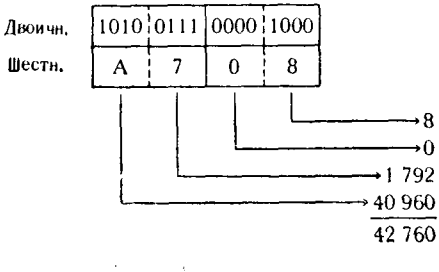
В следующих примерах приведены двоичные величины, изображенные как в двоичном, так и в шестнадцатеричном формате, и представляемые ими десятичные величины, полученные с помощью таблицы шестнадцатеричноно-десятичных преобразований.

Пример 1.

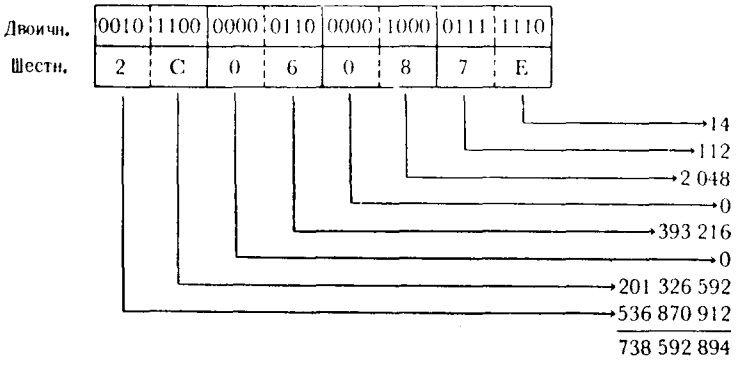


При использовании таблицы применяется стандартная процедура, начиная с правой шестнадцатеричной цифры. Самая правая шестнадцатеричная цифра 5 принимает свое собственное значение. Следующая шестнадцатеричная цифра (2) имеет значение, которое определяется при просмотре соответствующей колонки внутри таблицы: 0 = 0, 1 = 16, 2 = 32 и т. д. Значение, определяемое следующей более старшей шестнадцатеричной цифрой (9), вычисляется с помощью соответствующей колонки таблицы: 0 = 0, 1 = 256, 2 = 512, 3 = 768 и т. д.

Пример 2.



Пример 3.



Вполне возможно, что время от времени программисту может понадобиться осуществить преобразование чисел из десятичной системы в шестнадцатеричную. Использование таблицы шестнадцатерично-десятичных преобразований существенно облегчает выполнение этой задачи.

Десятичная величина, которая должна быть преобразована в шестнадцатеричную, используется для нахождения диапазона значений в таблице преобразований. В таблице отыскивается значение, равное преобразуемой величине, или, если такового не находится, берется ближайшее меньшее значение. Соответствующая шестнадцатеричная цифра записывается, а табличная величина вычитается из десятичного значения. Затем исследуется остаток и процесс повторяется. Детальный пример применения этого алгоритма приведен ниже.

Десятичная величина, которая должна быть преобразована, равна 15931.

Шаг а 15931
Шаг б 12288 = 3
Шаг в 3643
Шаг г 3584 = E
Шаг д 59
Шаг е 48 = 3
Шаг ж 11
11 = B
Шаг з Шестнадцатеричное значение = 3E3B

Шаг а. Записывается десятичное число, которое должно быть преобразовано.

Шаг б. Производится просмотр таблицы для того, чтобы найти величину, которая равна 15931 или является ближайшей меньшей по отношению к ней величиной. Наиболее близкой величиной в таблице является 12288 (в колонке 4 справа), так что это число записывается для вычитания из преобразуемой десятичной величины. Шестнадцатеричная цифра 3 представляет значение 12288 в этом столбце таблицы, и она поэтому тоже записывается.

Шаг в. Производится вычитание табличного значения из исходной величины, чем определяется разность по отношению к исходной величине, которая еще должна преобразовываться.

Шаг г. Таблица снова просматривается, на этот раз для того, чтобы найти значение, ближайшее к 3643, но не большее этого числа. Вторичный и все последующие просмотры следует всегда делать в столбце значений, который находится на один правее столбца, где было найдено последнее значение. В данном примере величина 3584 найдена в столбце, примыкающем непосредственно справа к столбцу, в котором была найдена величина 12288. Как самая близкая к 3643 табличная величина 3584 записывается вместе с ее шестнадцатеричным представлением Е.

Шаг д. Величина 3584 вычитается из разности, получившейся на предыдущем шаге, что приводит к новой разности, а именно к 59. Это значение предстоит обрабатывать при помощи таблицы на следующем шаге.

Шаг е. Отыскивается величина, ближайшая, но не большая чем 59, в следующем справа столбце; она равна 48. Значение 48 и его шестнадцатеричное представление 3 записываются.

Шаг ж. Последнее значение (48) вычитается из последнего остатка (59), в результате чего получается новый остаток. В силу того что в процессе поиска мы дошли до самого правого столбца таблицы, то новая разность не должна превышать величину 15, в противном случае следует искать ошибку, допущенную в процессе преобразования. В рассматриваемом примере в таблице отыскивается точное значение 11 и записывается его шестнадцатеричное представление.

Шаг з. Шестнадцатеричные цифры записываются в порядке их получения слева направо как составное шестнадцатеричное выражение. В результате получается шестнадцатеричная величина 3Е3В, которая имеет десятичное значение 15931.

При описании этих шагов упоминалось, что каждый раз после нахождения величины в некотором столбце таблицы последующий поиск следует начинать в соседнем столбце справа. Необходимо особо отметить, что если во время последующего поиска в этом столбце не найдено удовлетворительное значение, то записывается шестнадцатеричная цифра 0 и поиск продолжается в следующем соседнем столбце справа. Если для каждого столбца, в котором не найдено удовлетворительного значения, не записывать шестнадцатеричного нуля, то результирующее шестнадцатеричное число будет ошибочным. Примеры, следующие ниже, иллюстрируют процедуру записи шестнадцатеричных нулей во время выполнения преобразования.

Пример 1. Преобразуемое десятичное значение равно 47621.

```
47621
45056 = B   (найдено в столбце 4 справа)
 2565
 2560 = A   (найдено в столбце 3)
    5
      = 0   (нет сравнимой величины в столбце 2)
    5 = 5   (найдено в столбце 1)
```

Результирующее шестнадцатеричное значение BA05.

Пример 2. Преобразуемое десятичное значение равно 36908

```
36908
36864 = 9   (найдено в столбце 4)
   44
      = 0   (нет сравнимой величины в столбце 3)
   32 = 2   (найдено в столбце 2)
   12
   12 = C   (найдено в столбце 1)
```

Результирующее шестнадцатеричное значение 902C.

Пример 3. Преобразуемое десятичное значение равно 11863712.

```
11863712
11534336 = B   (найдено в столбце 6 справа)
  329376
  327680 = 5   (найдено в столбце 5)
    1696
       = 0     (нет сравнимой величины в столбце 4)
    1536 = 6   (найдено в столбце 2)
     160
     160 = A   (найдено в столбце 2)
       0 = 0   (найдено в столбце 1)
```

Результирующее шестнадцатеричное значение B506A0.

Из этих примеров видна важность введения шестнадцатеричного нуля в тех случаях, когда величина не отыскивается в каждом последующем столбце. Если нулевая цифра не была бы записана в примере 1, то результирующая шестнадцатеричная конфигурация выглядела бы как BA5 — что соответствует десятичному значению 2981. Если бы нули не были введены в примере 3, то результирующая шестнадцатеричная конфигурация выглядела бы как B56A, что соответствует десятичному значению 46442. Отсюда можно вывести общее правило: как только обнаружен столбец, содержащий первое значение, которое необходимо вычитать, каждый следующий за ним справа столбец должен быть представлен либо значащей шестнадцатеричной цифрой, либо шестнадцатеричным нулем. Если общее десятичное значение было сведено к нулевой разности до того, как были просмотрены все столбцы, стоящие справа, то каждый из оставшихся столбцов должен быть представлен нулевыми цифрами.

Пусть десятичное значение, которое необходимо преобразовать, равно 720896.

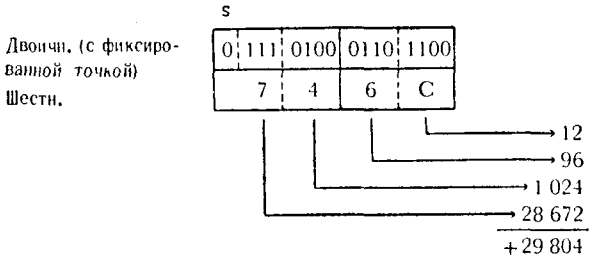
```
720896
720896 = B   (найдено в столбце 5 справа)
     0 = 0   (найдено в колонке 4)
     0 = 0   (найдено в колонке 3)
     0 = 0   (найдено в колонке 2)
     0 = 0   (найдено в колонке 1)
```

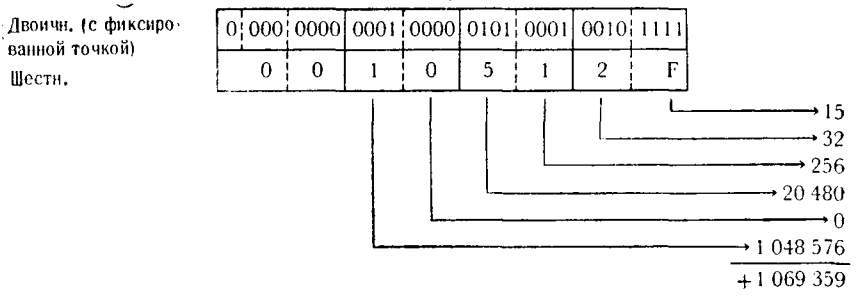
Результирующее шестнадцатеричное значение B0000.

Величины с фиксированной точкой, выраженные в шестнадцатеричном формате, несколько отличаются от аналогично выраженных двоичных величин тем, что значение знакового бита включается в старшую шестнадцатеричную цифру. Анализируя ее, можно сразу же определить, является ли величина с фикси-

рованной точкой положительной или отрицательной. Так как положительная величина будет содержать нуль в самом старшем бите самой старшей шестнадцатеричной цифры, то положительная величина с фиксированной точкой в качестве старшей шестнадцатеричной цифры всегда будет иметь цифру, меньшую или равную 7. Напротив, отрицательная величина с фиксированной точкой всегда будет иметь старшую шестнадцатеричную цифру, равную или большую 8.

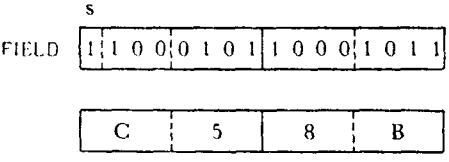
Получение десятичных эквивалентов с положительным знаком положительных величин с фиксированной точкой путем интерпретации шестнадцатеричных цифр показано в следующих примерах:



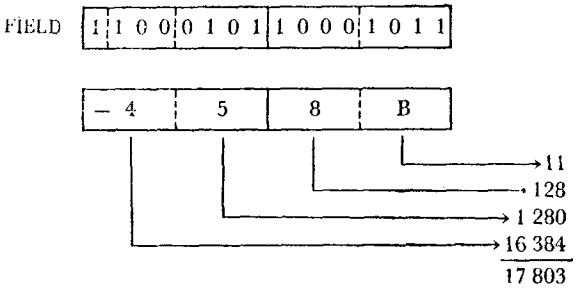


Как видно из приведенных примеров, нет очевидной разницы между процессами интерпретации из шестнадцатеричных полей двоичных величин без знака и положительных чисел с фиксированной точкой. Это связано с тем, что положительные числа с фиксированной точкой всегда содержат нулевой знаковый бит. Отрицательная величина с фиксированной точкой всегда будет содержать единицу в знаковом бите, который не рассматривается как часть целого числа. Поэтому, прежде чем преобразовывать отрицательные величины с фиксированной точкой из шестнадцатеричного представления в десятичное, необходимо модифицировать представление старшей шестнадцатеричной цифры. Так как отрицательная величина этого типа представлена в дополнительном коде, то описанная выше интерпретация значений шестнадцатеричных цифр не будет представлять действи-

тельного отрицательного значения поля. Эту проблему можно обойти, преобразуя шестнадцатеричные цифры отрицательной величины в десятичные эквиваленты этих цифр и вычитая затем это десятичное значение из максимальной отрицательной величины с фиксированной точкой, которая может содержаться в поле равной длины. Например, пусть отрицательная величина, которую необходимо интерпретировать, расположена в ячейке памяти длиной в полуслово и имеет значение, равное — 14965. Знаковый бит будем обозначать буквой s (sign).



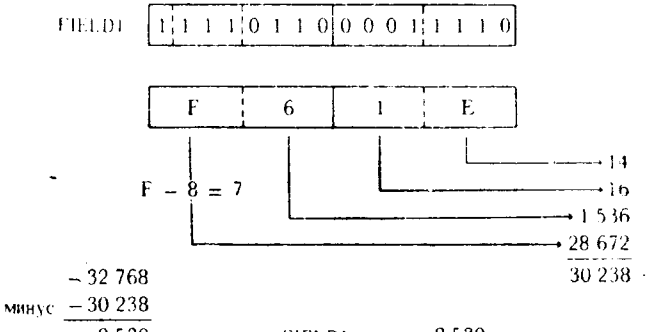
В силу того что старшая шестнадцатеричная цифра больше 7, величина отрицательна. Для того чтобы преобразовать целочисленное значение этой величины, необходимо исключить единичный бит знака из старшей шестнадцатеричной цифры; С минус 8 (шестнадцатеричное значение бита знака) равняется 4. Теперь шестнадцатеричные цифры могут быть преобразованы следующим образом:



Когда десятичный эквивалент дополнительного кода отрицательной величины с фиксированной точкой определен с помощью таблицы преобразования, его следует вычесть из максимальной отрицательной величины с фиксированной точкой, представленной полем длиной в полуслово.

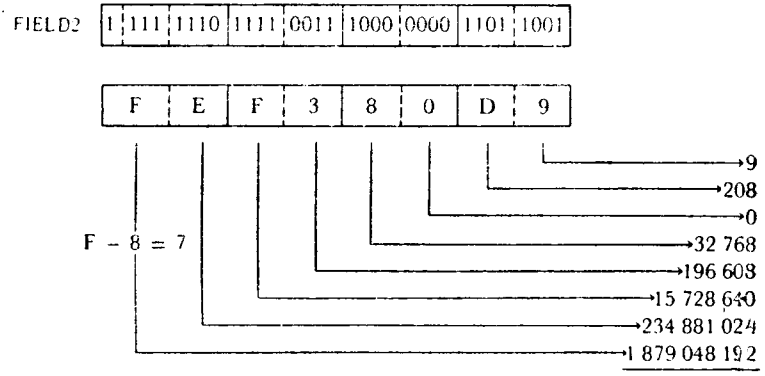
Максимальная отрицательная величина с фиксированной точкой	32768
Вычесть десятичный эквивалент дополнительного кода поля FIELD	17803
Действительное отрицательное значение с фиксированной точкой в поле FIELD	14965

Этот же самый процесс используется в следующих примерах:



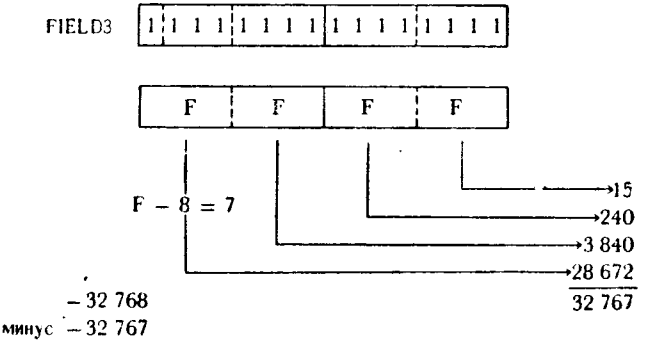
минус – 30 238

2 530 или значение FIELD1 равно – 2 530



минус – 2 129 887 449

– 17 596 199 или значение FIELD2 равно – 17 596 199



минус – 32 767

– 1 или значение FIELD3 равно – 1

По мере увеличения объема работ с шестнадцатеричными цифрами и величинами программист получит возможность оценить все удобства использования таблицы преобразований из шестнадцатеричного представления в десятичное. Постоянно используя таблицу, он настолько привыкает к ней, что некоторые ее части, вероятно, запомнит наизусть. Вдобавок к таблице имеются процедуры, относящиеся к сложению и вычитанию шестнадцатеричных чисел, которые сослужат хорошую службу программисту при определении перемещаемых адресов, факторов смещения и приращений в таблицах. Правила сложения шестнадцатеричных цифр связаны с величиной основания этой системы счисления — числом 16. Это видно из следующего примера:

$$\begin{array}{r} F \\ +\,A \\ \hline ? \end{array} \quad \text{или} \quad \begin{array}{r} F\,(15) \\ +\,A\,(10) \\ \hline ?\,(25) \end{array} \quad \text{или} \quad \begin{array}{r} F\,(15) \\ A\,(10) \\ \hline 19\,(25) \end{array}$$

(25 минус 16 равно 9 с переносом 1).

Тщательно изучите следующие примеры сложения многозначных шестнадцатеричных чисел:

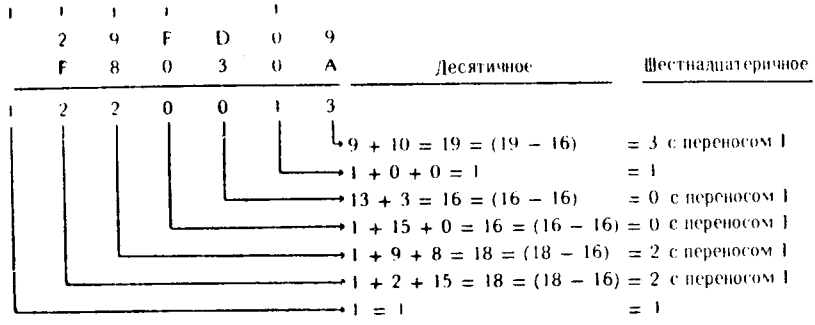
Пример 1. Чему равна сумма двух шестнадцатеричных чисел 0C3 и 175?

```
      1
      0   C   3
плюс  1   7   5        Десятичное                  Шестнадцатеричное
      ---------
      2   3   8
              └──→ 5 + 3 = 8                       = 8
          └──────→ 12 + 7 = 19 = 19 − 16           = 3  с переносом 1
      └──────────→ 1 + 0 + 1 = 2                   = 2
```

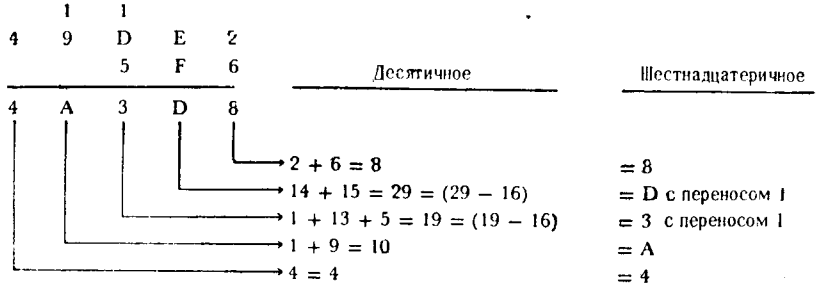
Пример 2. Чему равна сумма двух шестнадцатеричных чисел 3F268 и 91433?

```
      1
      3   F   2   6   8
плюс  9   1   4   3   3        Десятичное              Шестнадцатеричное
      -------------
      D   0   6   9   B
                  └──→ 8 + 3 = 11                  = B
              └──────→ 6 + 3 =  9                  = 9
          └──────────→ 2 + 4 =  6                  = 6
      └──────────────→ 15 + 1 = 16 = (16 − 16)     = 0  с переносом 1
  └──────────────────→ 1 + 3 + 9 = 13              = D
```

Пример 3. Чему равна сумма двух шестнадцатеричных чисел 29FD09 и F8030A?



Пример 4. Чему равна сумма двух шестнадцатеричных чисел 49DE2 и 5F6?



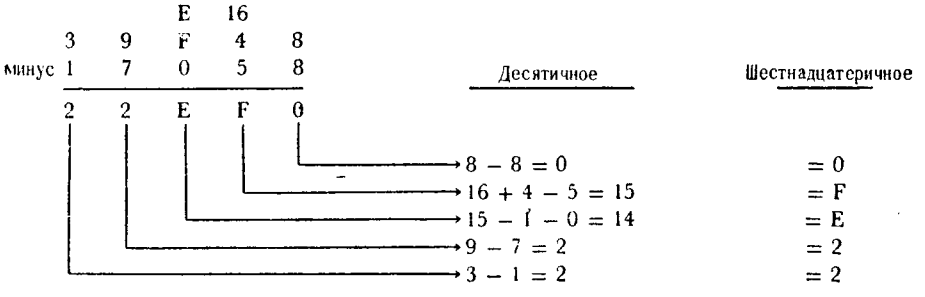
Вычитание шестнадцатеричных цифр производится по общим правилам вычитания, которые применимы к любой системе счисления. Например, когда из следующей более старшей шестнадцатеричной цифры производится «заем» единицы, то это эквивалентно добавлению значения 16 в столбец, для которого производится заем

минус	D	8	*или*	минус	C	16	*или*	минус	C	
	1	9			Ø	8			Ø	24
	?				1	9			1	9
					B	F			B	F

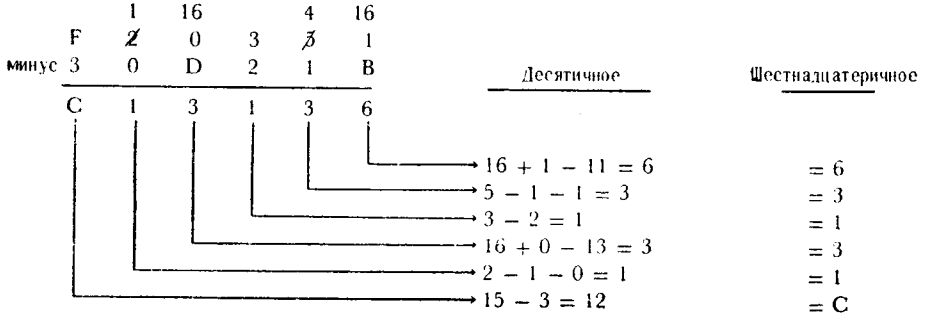
(12 − 1 = 11 = B) (24 − 9 = 15 = F)

Следующие примеры иллюстрируют процесс использования заемов при вычитании шестнадцатеричных чисел.

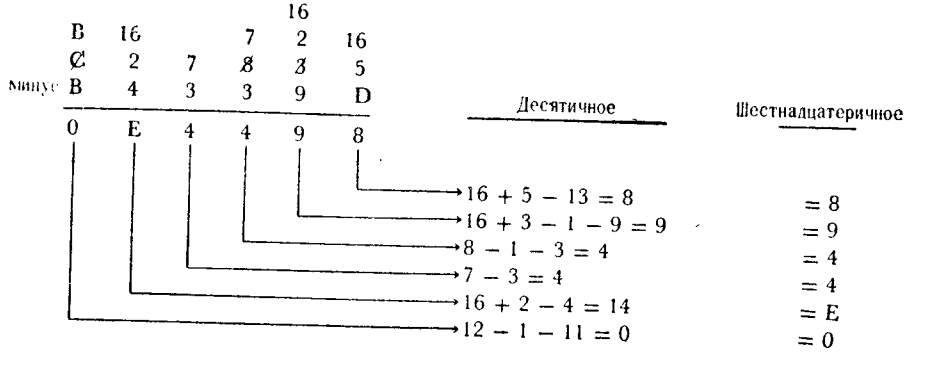
Пример 1. Чему равна разность двух шестнадцатеричных чисел 39F48 и 17058?



Пример 2. Чему равна разность двух шестнадцатеричных чисел F20351 и 30D21B?



Пример 3. Чему равна разность двух шестнадцатеричных чисел C27835 и B4339D?.



Понятно, что обучающиеся программисты часто относятся к шестнадцатеричной системе с некоторым опасением. Это происходит прежде всего потому, что большинство систем счисления имеет чисто цифровую структуру и не использует символы алфавита, как это имеет место в шестнадцатеричной системе.

Как только программист привыкнет связывать с шестнадцатеричными цифрами, выражаемыми буквами, их численный смысл, как, например, A = 10, B = 11, C = 12 и т. д., многие опасения, связанные с шестнадцатеричными конфигурациями, исчезнут сами собой.

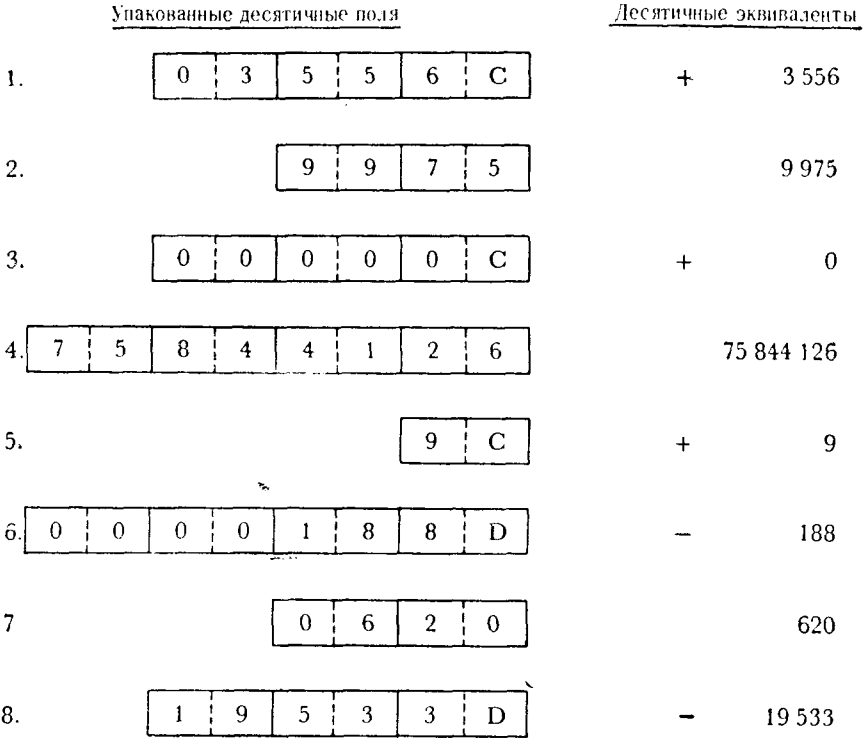
Помимо применения в шестнадцатеричной системе счисления, шестнадцатеричные числа используются для представления упакованных десятичных цифр и для определения полубайтовых конфигураций алфавитно-цифровых символов кода EBCDIC. Структура упакованной десятичной системы будет описана в этой главе; шестнадцатеричные конфигурации алфавитно-цифровых символов кода EBCDIC рассматриваются по всему тексту книги.

В данном месте читателю следует самостоятельно оценить полученные им знания принципов выполнения вычислений с шестнадцатеричными числами. Если представленные в данном разделе объяснения не обеспечили все же достаточного навыка при выполнении шестнадцатерично-десятичных преобразований и выполнении таких простых шестнадцатеричных арифметических операций, как вычитание и сложение, то настоятельно рекомендуется заново прочитать весь этот раздел.

Г. ДЕСЯТИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ С УПАКОВАННЫМ ФОРМАТОМ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ

Упакованная десятичная система счисления, как мы будем называть ее для краткости, использует такое представление внутри поля значений, которое обычно предполагает наличие одной «знаковой» цифры и одной или более «численных» цифр, причем все они представлены в шестнадцатеричном виде. В данном применении каждая шестнадцатеричная цифра рассматривается как самоопределенная величина. Так как упакованные десятичные цифры представляются шестнадцатеричными цифрами, то каждая упакованная десятичная цифра занимает только половину байта. Конфигурация упакованной десятичной величины может быть представлена со знаком или без него, но она должна иметь знак, если используется в операциях упакованной десятичной арифметики или преобразуется в формат с фиксированной точкой. В десятичной упакованной величине без знака каждый полубайт представлен шестнадцатеричной «численной» цифрой (от 0 до 9); в величине со знаком все полубайты, за исключением самого младшего (самого правого) полубайта, представлены шестнадцатеричными «численными» цифрами. Знак упакованного десятичного поля всегда должен находиться в младшем полубайте и представлять собой одну из шестнадцатеричных «буквенных» цифр (от A до F).

Стандартный положительный (+) знак для упакованной десятичной величины — это шестнадцатеричные C или F, стандартный отрицательный (—) знак — это шестнадцатеричное D. Ниже приведены примеры упакованных десятичных величин со знаком, величин без знака и эквивалентных им десятичных величин.



Как видно из этих примеров, десятичная конфигурация упакованного десятичного поля может рассматриваться сама в качестве десятичного эквивалента упакованной десятичной величины. Поля 1, 3 и 5 представляют собой величины с положительным знаком, поля 6 и 8 — величины с отрицательным знаком, а поля 2, 4 и 7 — величины без знака, относительно которых подразумевается, что они положительные. Последние три поля в таком формате нельзя использовать в арифметических операциях над упакованными десятичными числами.

Д. СИМВОЛЬНОЕ ПРЕДСТАВЛЕНИЕ ЧИСЕЛ

Цифры кода EBCDIC (Extended Binary Coded Decimal Interchange Code — расширенный двоично-кодированный десятичный код для обмена информацией) служат стандартным символьным

представлением цифр. Группу этих цифр можно рассматривать как число или просто набор символов, который, возможно, служит для образования идентификатора. Любой символ кода EBCDIC состоит из восьми двоичных позиций — одного полного байта. Четыре левых бита образуют зонную часть символа; правые четыре бита — его числовую часть. Числовые символы могут быть со знаком или без знака. Цифра без знака кода EBCDIC предполагается положительной величиной (+) или нейтральной цифрой; цифра со знаком кода EBCDIC представляет положительную (+) или отрицательную (—) величину в зависимости от конфигурации знака.

Таблица 3.3

Один байт	Поразрядная конфигурация	Цифра EBCDIC без знака
F0	1111 0000	0
F1	1111 0001	1
F2	1111 0010	2
F3	1111 0011	3
F4	1111 0100	4
F5	1111 0101	5
F6	1111 0110	6
F7	1111 0111	7
F8	1111 1000	8
F9	1111 1001	9

Таблица 3.4

Положительные зонные десятичные цифры				Отрицательные зонные десятичные цифры			
Значение	Шестн.	Поразрядная конфигурация	Символы EBCDIC	Значение	Шестн.	Поразрядная конфигурация	Символы EBCDIC
+0	C0	1100 0000	(пробел)	—0	D0	1101 0000	(пробел)
+1	C1	1100 0001	A	—1	D1	1101 0001	J
+2	C2	1100 0010	B	—2	D2	1101 0010	K
+3	C3	1100 0011	C	—3	D3	1101 0011	L
+4	C4	1100 0100	D	—4	D4	1101 0100	M
+5	C5	1100 0101	E	—5	D5	1101 0101	N
+6	C6	1100 0110	F	—6	D6	1101 0110	O
+7	C7	1100 0111	G	—7	D7	1101 0111	P
+8	C8	1100 1000	H	—8	D8	1101 1000	Q
+9	C9	1100 1001	I	—9	D9	1101 1001	R

Цифры без знака кода EBCDIC от 0 до 9 представляются набором шестнадцатеричных цифр от F0 до F9 (табл. 3.3).

Конфигурация цифр со знаком кода EBCDIC называется *зонной десятичной*. Только правый байт величины со знаком, выраженной цифрами EBCDIC, будет содержать зонную десятичную цифру. Положительные и отрицательные зонные десятичные цифры представлены в табл. 3.4.

Как видно из этих таблиц, цифра EBCDIC со знаком (зонная десятичная цифра) соответствует стандартным буквенным символам EBCDIC для всех цифр, кроме нуля. Нуль со знаком, отрицательный или положительный, образует конфигурацию, для которой нет символьного представления.

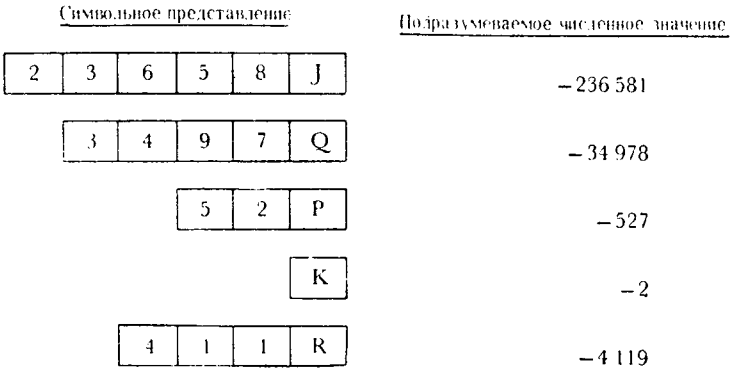
Числовые поля без знака в коде EBCDIC:

Символьное представление	Подразумеваемое численное значение
2 3 6 5 8 1	+236 581
3 9 4 7 8	+39 478
5 2 7	+527
2	+2
4 1 1 9	+4 119

Положительные числовые поля в коде EBCDIC:

Символьное представление	Подразумеваемое численное значение
2 3 6 5 8 A	+236 581
3 9 4 7 H	+39 478
5 2 G	+527
B	+2
4 1 1 I	+4 119

Отрицательные числовые поля в коде EBCDIC:



Упражнения

1. Двоичная система счисления образована с помощью двух цифр. Этими цифрами являются _____ и _____.

2. Считается, что старшая (левая) двоичная позиция величины, выраженной в формате с фиксированной точкой, содержит _____. Все остальные биты образуют _____.

3. Целочисленная часть полного слова состоит из правых _____ битов.

4. Система счисления с основанием 16, которая используется в Системе/360, относится к _____ представлению.

5. Алфавитно-цифровые символы кода EBCDIC представляются кодами из _____ битов.

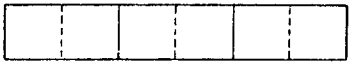
6. Шестнадцатеричная цифра C представляет десятичную величину (по основанию 10), равную _____.

7. Упакованные десятичные цифры графически представляются как синонимы _____ представления чисел.

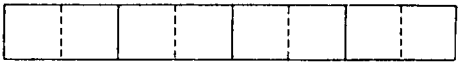
8. Символ EBCDIC, который представляет численную величину со знаком (положительную или отрицательную), называется _____.

9. Изобразите поразрядную конфигурацию следующих *двоичных* величин:

а. 48763

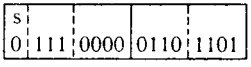


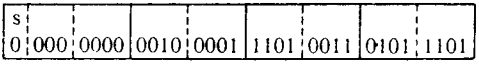
б. 219465

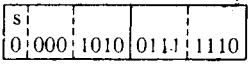


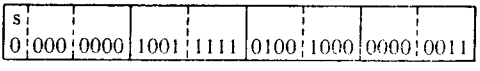
в. 11557

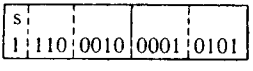
10. Определите значение *с фиксированной точкой* каждого из следующих полей и выразите их в десятичной форме:

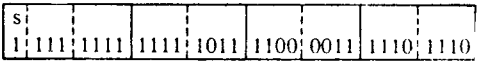
а. ———— 

б. ———— 

в. ———— 

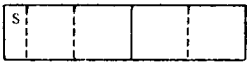
г. ———— 

д. ———— 

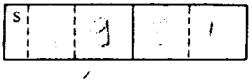
е. ———— 

11. Достройте поразрядную конфигурацию для следующих величин *с фиксированной точкой*:

а. Величина длиной в полуслово, равная +26715



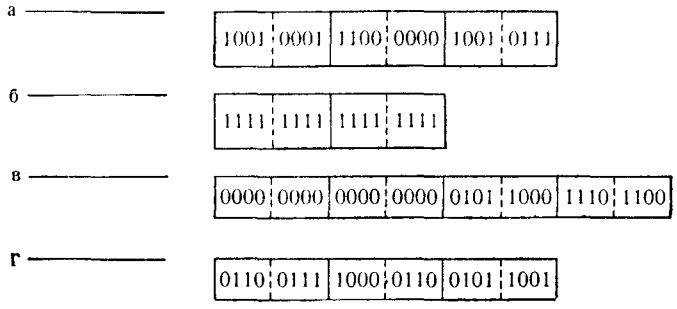
б. Величина длиной в полуслово, равная – 8901



в. Величина длиной в слово, равная + 14211536

г. Величина длиной в слово, равная – 397827

12. Определите значение каждого из полей, представленных в двоичном виде, и укажите его величину в десятичной форме:

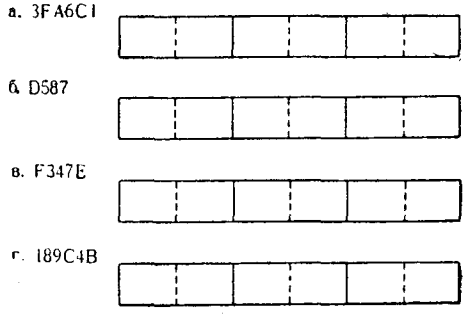


13. Сложите каждый из следующих ниже наборов шестнадцатеричных чисел, выражая сумму каждого набора в шестнадцатеричном и в десятичном (по основанию 10) виде.

	а.	б.	в.	г.
	2CA1	59F	600C	A0FF
	3029	3082	514D	2B33
			2953	05C2
Шестн.				
Десятичн.				

	д.	е.	ж.	з.
	0C00	6FF6		
	31D6	2A1D	A37	2001
	281C	2177	6C9F	300F
Шестн.				
Десятичн.				

14. Заполните поразрядную конфигурацию для следующих наборов шестнадцатеричных цифр:



15. Выполните вычитание шестнадцатеричных чисел, выражая в каждом случае разность в виде шестнадцатеричной и положительной десятичной величины:

	а.	б.	в.	г.
	2C3356	3A29	A9F	7B73D
	1D0F7	B36	34B5	42ACF
Шестн.	_____	_____	_____	_____
Десятичн.	_____	_____	_____	_____

16. Укажите десятичные значения величин, представленных в *упакованных десятичных полях*:

а. _____

4	0	0	C

б. _____

0	0	3	0	9	7	0	D

в. _____

0	0	0	0	6	5	0	0	2	C

г. _____

0	8	7	0	0	D

д. _____

1	0	7	8	6	0	3	9

17. Определите цифровые символьные значения следующих представлений символов в зонном десятичном формате:

а. _____

3	0	2	2	5	7	0	6	4	R

б _____

0	0	7	0	3	M

в. _____

5	5	8	A

г. _____

4	0	0	6	7	9	9	P

Глава 4

Форматы команд языка Ассемблера

Язык Ассемблера Системы/360 предоставляет средства кодирования программ, которые дают возможность программисту управлять отдельными логическими действиями программы на максимальном уровне детализации. Ассемблер называют языком низкого уровня, т. е. языком, структура и взаимосвязи предложений которого достаточно близки к уровню языка машинных команд. Хотя отдельная команда на машинном языке может вызывать в ЭВМ выполнение нескольких внутренних функций, внутримашинный язык представляет наинизший уровень команд, которые можно закодировать для выполнения на вычислительной машине.

Отдельная *команда* языка Ассемблера, служащая для указания ЭВМ на необходимость выполнения некоторой операции, порождает команду на языке машины вместе с определенными, неявно или явно заданными параметрами, относящимися к этой команде. Если программист попытался бы программировать непосредственно на языке машины, то ему потребовалось бы постоянно заботиться о выравнивании границ, распределении регистров и присваивании значений, смещениях, длинах, адресах и многих других факторах, что вскоре показалось бы ему делом крайне трудоемким. Однако компилятор с языка Ассемблера интерпретирует отдельные предложения таким образом, что берет на себя выполнение многих из этих функций. В обязанности программиста все еще входит распределение регистров и наблюдение за их содержимым, но далеко не в такой степени, как это требуется при кодировании на машинном языке.

При программировании на языке Ассемблера используются так называемые *макрокоманды*, входящие в состав операционной системы. Макрокоманда представляет собой отдельный оператор, в качестве операндов которого задаются параметры из набора, допустимого для данной макрокоманды. Одна макрокоманда *генерирует* последовательности команд языка Ассемблера и макрокодов, служащие для выполнения запрошенных программистом услуг, причем программист избавлен от необходимости заниматься кодированием сложных последовательно-

стей команд. Термин «генерировать» в том смысле, в котором он здесь употребляется, может отчасти ввести в заблуждение. В действительности макрокоманда сама по себе не создает какую-то уникальную последовательность кодов. Все связанные между собой команды, которые относятся к каждой макрокоманде, уже находятся в библиотеке операционной системы, если включение этих макрокоманд было оговорено при создании системы. Использование макрокоманды в проблемной программе приводит к обращению к этой библиотеке для обеспечения системы командами, необходимыми для выполнения задачи, запрошенной данной макрокомандой. Как наименование, так и функции макрокоманд могут в значительной степени изменяться для различных методов доступа и операционных систем. В данной книге не ставится цель изучения макрокоманд, и они используются только в некоторых примерах кодирования программ. Но и в этих случаях использование таких макрокоманд будет обсуждаться лишь в самом общем виде.

Команды языка Ассемблера, описанные в данной книге, входят в стандартный набор команд и в набор команд десятичной арифметики. Команды с плавающей точкой языка Ассемблера здесь не рассматриваются. По мнению автора, арифметические операции с плавающей точкой языка Ассемблера не нужны большинству программистов на этом этапе их обучения; сложность применения команд с плавающей точкой сама по себе приводит к обращению к языкам высокого уровня типа PL/1 или FORTRAN. Это мнение можно оспаривать, но именно оно послужило основой при написании данной части книги.

По мере изучения языка Ассемблера и знакомства с аргументацией и примерами, приведенными в данной книге, вы будете приобретать лучшее представление о гибкости и широких возможностях Системы/360. Изучая команды, вы лучше поймете детали процессов, которые используются управляющими программами вычислительной системы при выполнении этих команд.

Имеется два варианта формата команд. Первый представляет собой способ записи команды программистом. Второй формат, в который компилятор преобразует команду, написанную программистом, — формат машинной команды. Хотя коды этих двух форматов тесно связаны между собой, программисту в первую очередь нужно быть хорошо знакомым с форматом кодирования операторов команд. Начав писать проблемные программы, он найдет полезным и знание формата этих команд на машинном языке, что поможет ему при анализе и интерпретации ошибок и решении вопросов, возникающих при кодировании.

А. ФОРМАТ КОДИРОВАНИЯ ПРЕДЛОЖЕНИЙ

Предложение, закодированное программистом, может состоять из трех частей:

1) метки или символа;
2) мнемонического кода операции;
3) операндов в количестве от одного до трех на один оператор.

Из них необходимыми для написания правильного предложения являются только код операции и операнды.

1. Метка или символ

Метку называют еще символом, этикеткой, именем или идентификатором в зависимости от привычек программиста. Однако безотносительно к этому разнообразию метка представляет собой имя, которое используется для идентификации адреса предложения, определенной области, константы или начальной точки программы или подпрограммы. В каких бы целях не использовалась метка, компилятор связывает ее с адресом этих данных внутри программы. Каждая ссылка к этой метке, таким образом, равносильна указанию действительного адреса, к которому относится метка. Метка, следовательно, является просто удобным способом ссылки к определенному адресу секции или части проблемной программы. Если не требуется никаких ссылок, то и в использовании меток нет необходимости.

Конфигурация метки имеет определенные ограничения. Число символов, составляющих метку, зависит от операционной системы. При использовании полной Операционной системы (OS) метка может содержать от одного до восьми символов. Хотя эти символы могут представлять смесь из букв и цифр, первый символ обязательно должен быть буквенным. Метка не должна содержать специальных символов или внутренних пробелов. В следующем списке представлены правильные метки:

SKIP	PACKRT1
K23456	R
PTS32R	BRANCH
ADD	MOVE
M3M3M3	MVO

Заметим, что последняя метка (MVO) является также кодом операции для команды Move with Offset (Пересылка со сдвигом.) При размещении в соответствующем месте бланка кодирования эта метка будет рассматриваться как правильная. Любая алфавитно-цифровая комбинация, которую может составить

программист, является правильной, если при ее составлении были учтены соответствующие ограничения. Так как метка используется для обращения к некоторому отдельному адресу внутри проблемной программы, понятно, что для обращения к двум различным адресам внутри программы одинаковые метки применяться не должны.

2. Код операции

Код операции является мнемоническим представлением команды, которая должна быть выполнена в результате реализации данного предложения. Он представляет действие, которое компилятор должен транслировать в команду на машинном языке, тем самым точно сообщая объектной программе, какое действие должно быть выполнено. Код операции обычно представляет одиночную команду на языке Ассемблера, хотя он может также представлять макрокоманду. В большинстве случаев оказывается, что мнемоника кода операции тесно связана с описательным именем той команды, которую она представляет. Это видно из следующих примеров:

AR	**A**dd **R**egisters	(Сложение)
CL	**C**ompare **L**ogical	(Сравнение кодов)
EDMK	**ED**it and **M**ark	(Отредактировать и отметить)
CVD	**C**on**V**ert to **D**ecimal	(Преобразование в десятичную)
LPR	**L**oad **P**ositive **R**egister	(Загрузка положительная)
STM	**ST**ore **M**ulipte	(Запись в память групповая)
DP	**D**ivide **P**acked	(Деление десятичное)
EX	**EX**ecute	(Выполнить)

3. Операнды

Хотя оператору макрокоманды вовсе необязательно иметь какие-либо операнды, любое одиночное предложение языка Ассемблера может содержать от 1 до 3 операндов. Каждая команда имеет определенное число операндов, которые необходимо указать в предложении. Операнды сообщают проблемной программе адреса полей или данных, которые будут участвовать в процессе выполнения команды с данным кодом операции. Например, команда MVC (Пересылка символов) требует указания

значений двух операндов в предложении: операнда, идентифицирующего область, из которой должны пересылаться данные, и операнда, идентифицирующего поле, в которое должны быть помещены эти данные. Код операции для каждой команды определяет использование и применение адресов, выработанных с помощью операндов. В большинстве случаев величина, определенная вторым операндом, будет использована для изменения содержимого первого операнда. Из этого правила имеются исключения, так что программисту следует хорошо разбираться в действиях, осуществляемых при выполнении каждой команды.

В этой книге при описании команд языка Ассемблера форматы операндов будут представлены как состоящие из отдельных элементов, например R_1, $D_2(X_2, B_2)$. Здесь R_1 представляет первый операнд полного предложения, а $D_2(X_2, B_2)$ — второй операнд. Скобки показывают, что элементы, стоящие в них, могут быть использованы для дальнейшей модификации адреса, выработанного частью D_2 операнда.

Ниже следует полный список всех элементов, составляющих форматы операндов:

B Представляет *базовый регистр* (регистр базы), используемый для операнда, в котором он задан.
D Указывает *смещение*; действительный или символический адрес либо величина.
I *Непосредственный* или самоопределенный символ или величина.
L Явный указатель *длины* для операнда, в котором он задан.
M Значение *маски*, определенное либо как литерал, либо как константа.
R *Общий регистр*, используемый в качестве операнда, или метка, которая приравнена номеру общего регистра.
X Индекс-регистр (регистр индекса).

Для того чтобы указать связь этих символов с тем операндом, к которому они относятся, используются цифры, идентифицирующие компонент предложения:

цифры 1 вместе с элементом формата предложения идентифицирует этот элемент как первый операнд или его часть;

цифра 2 вместе с элементом формата идентифицирует этот элемент как второй операнд или его часть;

цифра 3 вместе с элементом формата идентифицирует этот элемент как третий операнд. Это значение используется только с групповыми регистровыми операндами, указывая конечный регистр в последовательности общих регистров. В операторе команды этого типа операнды располагаются в следующем порядке: первый операнд, затем третий и затем второй.

На рис. 4.1 приведены обозначения компонентов и операндов для нескольких вариантов кодирования некоторых команд языка Ассемблера.

Форматы предложений могут быть сгруппированы следующим образом:

1. Формат RR:

$$R_1,R_2$$

Оба операнда находятся в общих регистрах.

$$M_1,R_2$$

Первый операнд — значение маски; второй операнд — в общем регистре.

2. Формат RS:

$$R_1,R_3,D_2(B_2)$$

R_1 и R_3 указывают общие регистры; $D_2(B_2)$ задает смещение или адрес памяти либо в виде абсолютной величины, либо в символическом виде.

$$R_1,D_2(B_2)$$

R_1 указывает на общий регистр; $D_2(B_2)$ задает смещение или адрес памяти либо в виде абсолютной величины, либо в символическом виде.

3. Формат RX:

$$R_1,D_2(X_2,B_2)$$

R_1 указывает на общий регистр; $D_2(X_2,B_2)$ задает смещение или адрес памяти либо в виде абсолютной величины, либо в символическом виде.

$$M_1,D_2(X_2,B_2)$$

M_1 указывает значение маски; $D_2(X_2,B_2)$ задает смещение или адрес памяти либо в виде абсолютной величины, либо в символическом виде.

4. Формат SI:

$$D_1(B_1),I_2$$

$D_1(B_1)$ задает смещение или адрес памяти либо в виде абсолютной величины, либо в символическом виде; I_2 задает самоопределенный (*непосредственный*) символ, величину или ссылку.

5. Формат SS:

$$D_1(L,B_1),D_2(B_2)$$

Оба операнда задают смещения или адреса памяти либо в виде абсолютных величин, либо в символическом виде.

$$D_1(L_1,B_1),D_2(L_2,B_2)$$

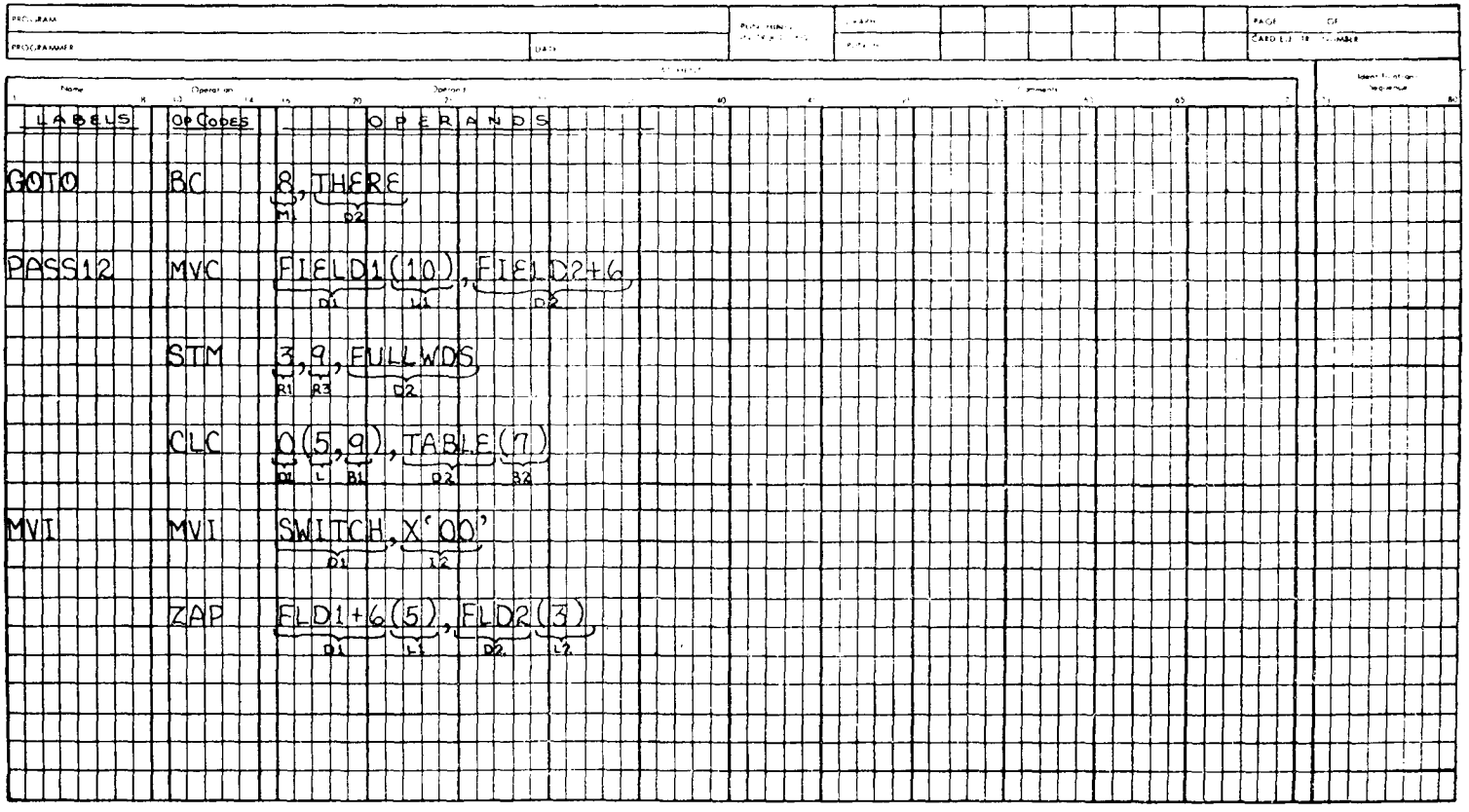

IBM
IBM System 360 Assembler Coding Form
LABELS
OPCODES
OPERANDS
GOTO
BC
8,THERE
M1
D2
PASS12
MVC
FIELD1(10),FIELD2+6
D1
L1
D2
STM
3,9,FULLWDS
R1
R3
D2
CLC
0(5,9),TABLE(7)
D1
L
B1
D2
B2
MVI
MVI
SWITCH,X'00'
D1
I2
ZAP
FLD1+6(5),FLD2(3)
D1
L1
D2
L2

Рис. 4.1.

Оба операнда задают смещения или адреса памяти либо в виде абсолютных величин, либо в символическом виде.

Компоненты операндов, заключенные в скобки, являются необязательными — иными словами, их можно определить, задав при кодировании, или они могут подразумеваться и в этом случае не кодируются. Например, если длина поля AREA была определена равной 6 байтам, а предложения были закодированы следующим образом:

```
MVC     AREA(4),DATA
MVC     AREA,DATA
```

то первая команда будет пересылать 4 байта из поля DATA в поле AREA, так как в ней явно определен указатель длины 4. Вторая команда перешлет 6 байтов данных, так как неявная длина поля AREA равна 6 байтам.

Б. ПРАВИЛА ЗАПОЛНЕНИЯ БЛАНКА КОДИРОВАНИЯ

Продолжая обсуждение форматов предложений, пора познакомиться с установившимися практическими соглашениями о кодировании программ. Хотя компилятор с языка Ассемблера допускает определенную свободу записи предложений, программисту целесообразно придерживаться ясной, стандартизованной формы кодирования, при которой метки, коды операций и первые операнды почти всегда выравниваются по начальным точкам, предложение за предложением.

Поясним назначение колонок стандартного восьмидесятиколонного бланка для кодирования программ.

Колонка 1. Метка всегда должна начинаться именно в этой колонке. Ее длина может быть равна восьми позициям, если программа написана для Операционной Системы (OS).

Колонка 10. С этой колонки следует начинать запись кода операции. Метка всегда должна отделяться от кода операции пробелом, и такое соглашение гарантирует по крайней мере один пробел между кодом операции и меткой даже в случае метки максимальной длины.

Колонка 16. Здесь следует размещать первый символ первого операнда. Это обеспечивает по крайней мере один пробел между кодом операции и операндами. При использовании макрокоманд код операции может быть длиннее, чем обычный код операции команд языка Ассемблера; поэтому для обеспечения пробела перед первым символом первого операнда операнд в этом случае должен начинаться в колонке с большим номером.

IBM

IBM System/360 Assembler Coding Form

X28-6509-2 U/M040
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE OF | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT				Identification-Sequence
Name	Operation	Operand	Comments	
START	MVC	FIELD1,DATA		0001
	STM	2,13,SAVEAREA		0002
	L	5,FWD1		0003
BRANCHED	BAL	6,FINDIT		0004
	ZAP	PACKFLD1,=PL1'0'		0005
	BCT	5,BRANCHED		0006
*				0007
*				0008
*		FINDIT LOOKUP ROUTINE		0009
*				0010
FINDIT	LA	9,10	LOAD BASE REGISTER	0011
	CLI	0(9),C'S'		0012
	BCR	8,6		0013
	MVI	0(10),C'X'		0014
	BCR	15,6		0015
*				0016

Рис. 4.2.

Соблюдение перечисленных соглашений облегчает чтение и анализ листинга, распечатанного компилятором. На рис. 4.2 представлен ряд предложений, закодированных на бланке в соответствии с принятыми соглашениями для последующей перфорации.

Как видно из рисунка, все метки, коды операций и операнды соответственно выравнены по своим начальным позициям. В этом примере используются некоторые новые возможности, предусмотренные для удобства программиста. Одна из них касается двух способов записи *комментариев*, при которых текст комментария не оказывает влияния на интерпретацию компилятором команд программы. В средней части примера есть четыре строки, которые содержат в колонке 1 звездочки *. Символ * указывает компилятору, что эти строки являются «строками комментариев», которые должны быть включены в текст листинга компиляции, но не являются частью объектного модуля. В данном примере они используются для разделения двух программ, а также для размещения наименования второй программы, записанного в виде комментария-заголовка. Программист может написать на такой строке любую информацию и может ввести такие строки в любое место, куда пожелает, — даже между двумя последовательными строками программы, нисколько не воздействуя при этом на ее работу.

Второй способ записи комментариев показан на строке с меткой FINDT (строка 0011). Слова LOAD BASE REGISTER, которые записаны после операндов этой команды, отделены от последнего операнда несколькими пробелами. Комментарий, который должен быть записан на той же самой строке, что и предложение, может занимать все колонки бланка кодирования до 71-й включительно, но должен отделяться от операндов по крайней мере одним пробелом.

Используя эти способы записи комментариев, программист может детально документировать логику своей программы, отделять тексты соседних подпрограмм, давать общие заголовки и пояснения к программам и при желании краткое описание назначения каждой команды. Хорошая документация исходной программы очень желательна и во многих случаях является стандартным компонентом систем обработки данных.

Колонки с 73-й по 80-ю бланка кодирования используются для нумерации отдельных предложений или строк бланка. Номера или идентификаторы перфорируются на картах с исходными кодами и одновременно с перфорацией самих предложений. Перфокарты с предложениями на исходном языке содержат теперь информацию, которая позволит программисту заново расположить перфокарты в нужной последовательности, если колода карт рассыпется, или заменить любое исходное предло-

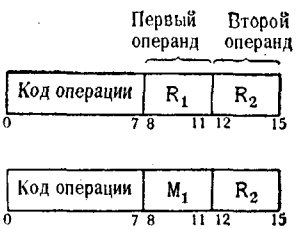
жение путем простого перебора последовательности номеров колоды перфокарт исходной программы до тех пор, пока не встретится нужная карта. Последовательные номера перфокарт исходной колоды печатаются на листинге, получаемом в результате компиляции, так что программист может легко отыскать любое исходное предложение.

В. ФОРМАТЫ КОМАНД НА МАШИННОМ ЯЗЫКЕ

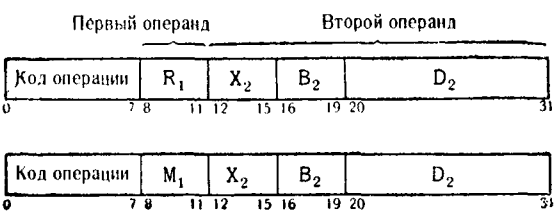
Знакомство с форматом команд на машинном языке, полученных на выходе компилятора с языка Ассемблера, помогает при анализе ошибок в программе и интерпретации *дампов*. Под дампом понимают распечатку в шестнадцатеричном формате содержимого областей основной памяти, занятых проблемной программой и связанными с ней программами Супервизора системы. Дамп основной памяти обычно производится всякий раз, когда при выполнении проблемной программы встречаются различного рода нарушения нормального режима выполнения или противоречивость данных, приводящие к прерываниям. Такая распечатка основной памяти может быть затребована программистом или выполняется автоматически при возникновении определенных ошибочных ситуаций. Именно в это время программисту может понадобиться разобраться в форматах команд на машинном языке, чтобы проследить за ходом выполнения команд программы непосредственно перед ее прерыванием.

Форматы машинных команд, сгруппированные по типам команд, представлены ниже.

1. Формат RR

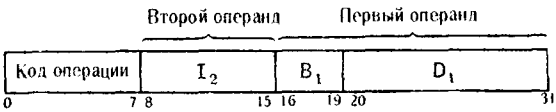


2. Формат RX

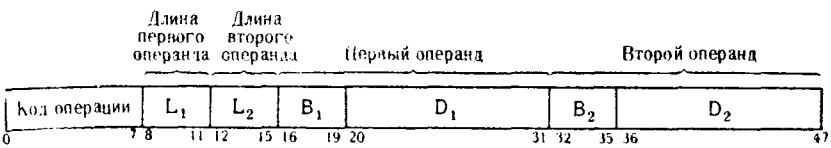


3. Формат RS

4. Формат SI



5. Формат SS



Числа под каждым примером формата указывают относительное положение (в номерах двоичных разрядов) соответствующих частей формата команды на машинном языке.

Для описания этих форматов используются точно такие же коды элементов, как и для форматов предложений. На примере одной команды формата RX рассмотрим переход от предложения языка Ассемблера к машинной команде. Предложение:

LA 9,83

Формат этого предложения на машинном языке выглядит следующим образом:

4 1 9 0 0 0 5 3

Связь между этими форматами иллюстрируется ниже:

Код операции	R_1	X_2	B_2	D_2			
4 1	9	0	0	0	5	3	Шестн.
0100 0001	1001	0000	0000	0000	0101	0011	Двоичн.
L A	9	0	0	+83			Действительное

Г. ПРИЗНАК РЕЗУЛЬТАТА

Признак результата представляется двумя битовыми позициями (34 и 35) Слова состояния программы (PSW — Program Status Word). Он представляет собой, по сути дела, своеобразный индикатор, который используется для фиксации условия,

выработанного в результате выполнения предыдущего предложения. Программист имеет возможность принимать в программе логические решения, основанные на конфигурации признака результата. При обсуждении отдельных команд мы увидим, что не все команды языка Ассемблера влияют на конфигурацию признака результата.

Вот некоторые из вопросов, на которые может дать ответ признак результата:

1. Был ли результат сравнения «больше», «меньше» или «равно»?
2. Был ли результат равен нулю?
3. Был ли результат положительной или отрицательной величиной?
4. Не произошло ли переполнение?

Есть команды, которые проверяют признак результата и затем реагируют в соответствии с параметрами, закодированными в этом предложении. Позднее в этой книге будет сообщена детальная информация о формировании и использовании признака результата. В гл. 8 будут даны подробные сведения о поразрядной конфигурации признака результата в PSW и способы кодирования команд с целью обеспечения перехода к другой секции проблемной программы.

Д. РЕГИСТРЫ И ИХ ПРИМЕНЕНИЕ

Система/360 имеет набор внутренних областей памяти, называемых регистрами. Шестнадцать *общих регистров*, которым присвоены последовательные номера от 0 до 15, являются неотъемлемой частью системы. Дополнительно пользователь может иметь четыре регистра, называемых *регистрами с плавающей точкой*, каждый из которых по числу битовых позиций имеет длину, в два раза большую, чем общий регистр. Так как регистры с плавающей точкой не являются стандартным оборудованием для всех моделей и в силу того, что операции с плавающей точкой в этой книге не рассматриваются, при всех дальнейших ссылках к регистрам будут подразумеваться общие регистры.

Общие регистры имеют длину в одно полное слово — 4 байта или 32 бита. Они расположены подряд от нулевого до 15-го аналогично цепочке полных слов. Описание различных применений регистров будет даваться по мере надобности на протяжении всей книги, но основные сведения об их использовании следует дать уже сейчас.

Важнейшим назначением регистра в языке Ассемблера является установление *базы* для обращений к группе операторов команд. Когда регистр используется в этих целях, его называют

базовым регистром или *регистром базы*. Система требует, чтобы по крайней мере каждой группе из 4096 байтов объектных кодов, выработанных из исходных кодов, был приписан базовый регистр. Это относится к областям, занятым константами, литералами и рабочими полями программ. Базовый регистр создает некоторую разновидность связи между машинными командами, находящимися внутри той области из 4096 байтов, которой был приписан данный базовый регистр. Если ожидается, что программа должна генерировать более чем 4096 байтов объектного модуля, то необходимо отвести такое количество базовых регистров, которое оказалось бы достаточным, чтобы охватить все байты программы. При назначении регистров обычно ориентируются на приращения в 4096 байтов.

Существует несколько способов назначения базовых регистров. Если программист имеет возможность оценить объем объектных кодов, вырабатываемых предложениями программы, то он может назначать базовые регистры по мере надобности в процессе кодирования программы. Это может привести к недостаточно эффективному использованию регистров, потому что почти невозможно определить точное место внутри программы, где окажется необходимым назначить новый базовый регистр. Наиболее удобным способом назначения базовых регистров является указание цепочки базовых регистров в начале проблемной программы. Грубая оценка объема программы — это все, что необходимо знать для инициализации нескольких базовых регистров в этой точке программы. Они назначаются таким образом, что каждые 4096 байтов, связанных с отдельным базовым регистром, присоединяются к следующим 4096 байтам. Применение этого метода описано в разделе «Инициализация программы». Так как назначение базовых регистров производится в одной точке программы, то этот метод позволяет без труда осуществлять введение дополнительных или удаление базовых регистров.

При назначении базовых регистров следует соблюдать осторожность. Операционные системы используют некоторые из общих регистров для целей связи между программами, изменяя при этом ранее содержавшуюся в них информацию. Например, в процессе работы программ под управлением OS система почти наверняка будет использовать общие регистры 0, 1, 13, 14, 15. Кроме того, команда Ассемблера «Перекодировать и проверить» будет разрушать любое предыдущее содержимое общего регистра 2. С учетом сказанного, можно без опасения принять, что общие регистры 3—12 находятся в полном распоряжении программиста, работающего с OS, для их использования либо в качестве базовых регистров, либо в качестве управляющих регистров в соответствии с предложениями.

Применение регистров как части предложения определено при описании каждой команды, к которой это относится. Существуют, однако, основные правила, которые регламентируют использование регистров. Данные, загружаемые в регистр из ячейки памяти, должны выбираться из области памяти, которая выравнена по *границе полного слова.* Границей полного слова является любой адрес основной памяти внутри области проблемной программы, который делится без остатка на 4. Данные, помещаемые из общего регистра в память, должны размещаться на границе полного слова.

Для выполнения арифметических операций с фиксированной точкой требуются регистры. Хотя величины, которые используются в операциях с фиксированной точкой, могут располагаться в основной памяти, по крайней мере одна из них должна быть загружена в регистр. В зависимости от типа команды с фиксированной точкой при ее выполнении могут использоваться общие регистры 1, 2 или 3.

Е. ФОРМАТ ДАННЫХ

Ясно, что при специальном кодировании или преобразованиях информации при работе проблемной программы возможны и различные формы организации данных. Ниже будут рассмотрены три основные формы *логической* организации данных: символьные данные в коде EBCDIC, упакованные десятичные числовые данные и арифметические данные с фиксированной точкой. Эти формы являются выражением логических концепций форматов данных, хотя данные в любой из этих форм могут рассматриваться как поразрядная двоичная конфигурация или конфигурация из шестнадцатеричных цифр. Однако, как правило, существует общая взаимосвязь между концепцией форматов данных и интерпретацией входящих в них конфигураций.

Специальные коды или преобразованная информация, выработанные проблемной программой, также должны рассматриваться в терминах одного из форматов или конфигураций данных. Для определения этих форматов нет жестких стандартов, так как в каждом случае формат выбирается программистом, исходя из задачи облегчения процессов декодирования данных или обратного их преобразования в стандартную форму.

1. Символьные данные

Символьные данные, о которых идет речь в этой книге, представляют собой алфавитно-цифровые или специальные символы, кодируемые восемью двоичными позициями, и включают такие графические символы, как числа, буквы или знаки. Это буквы

латинского алфавита от A до Z, цифры от 0 до 9 и знаки, такие, как), (, #, ?, ', $, / и т. д. Полный перечень символов, из которых формируются символьные данные, можно найти в таблице кода EBCDIC в части V этой книги. Сокращение EBCDIC означает «Extended Binary-Coded Decimal Interchange Code» (расширенный двоично-кодированный десятичный код для обмена информацией). Он представляет собой стандартный набор кодов символов, в котором каждому символу соответствует своя конфигурация битов, перенумерованных от 0 до 7 слева направо. Отсюда ясно, что отдельный графический символ EBCDIC занимает один байт.

Используя сочетание буквы F и цифры 4 в качестве примера символа EBCDIC, конфигурацию, включающую эти символы, можно представить тремя способами.

	1 байт		1 байт	
Символьн.	F		4	
Шестн.	C	6	F	4
Двоичн.	1100	0110	1111	0100

Знание поразрядной двоичной конфигурации символьных данных требуется весьма редко, гораздо чаще оказывается полезным помнить вид композиции шестнадцатеричных цифр, составляющих алфавитно-цифровые символы, например при анализе дампов основной памяти, содержащих неоттранслированные шестнадцатеричные цифры, представляющие содержимое оперативной памяти.

Как уже упоминалось, каждому стандартному графическому символу кода EBCDIC соответствует своя восьмибитная конфигурация. Несмотря на то что байт может содержать 256 различных конфигураций, код EBCDIC содержит лишь 63 графических символа. Остальные 193 двоичные конфигурации не представляют символов в том смысле, который подразумевается при определении символьного формата данных.

2. Упакованные десятичные данные

Упакованные десятичные данные по форме совпадают с шестнадцатеричной конфигурацией цифр: каждая упакованная десятичная цифра занимает четыре двоичных разряда, или один полубайт. Сами данные состоят из упакованных десятичных цифр (от 0 до 9) и цифр, предназначенных для обозначения знака (алфавитных шестнадцатеричных цифр от A до F). Каждая упакованная десятичная величина с правильно указанным знаком имеет одну цифру для указания знака, которая должна нахо-

диться в младшем (правом) полубайте поля, занимаемого этой величиной. Остальные упакованные десятичные цифры, представляющие саму величину, должны быть только «числовыми» цифрами (от 0 до 9). Можно сформировать такую упакованную десятичную величину, в которой все цифровые позиции представлены числовыми цифрами, но этот тип упакованных десятичных чисел не может использоваться в десятичных арифметических операциях. В командах десятичной арифметики можно использовать только упакованные десятичные величины с правильно указанными знаками.

Так как упакованные десятичные величины имеют шестнадцатеричные конфигурации своих цифр, то их легко распознать при анализе дампов основной памяти, что иллюстрируется примерами, приведенными в этом и последующих разделах.

Можно рассматривать упакованный десятичный формат как компактный способ выражения общепринятых символьно форматизованных десятичных величин. Каждый байт упакованных десятичных цифр будет содержать две числовые цифры, за исключением младшего байта, который обычно будет содержать одну числовую цифру и одну цифру для изображения знака числа. Примеры десятичных величин вместе с соответствующими им упакованными форматами длиной 4 байта приведены в табл. 4.1. Хотя длины полей этих величин в каждом примере равны 4 байтам, достаточно иметь длину, равную такому количеству байтов, чтобы вместить крайнюю левую значащую (ненулевую) упакованную десятичную цифру. Заметим, что знаковая цифра для показанных на иллюстрации величин различна для аналогичных положительных или отрицательных конфигураций. Правильный знак плюс можно представить шестнадцатеричными «буквенными» цифрами A, C, E или F; правильный отрицательный знак может быть представлен в виде D или B. Обычным представлением знака, вырабатываемым для упакованного положительного числа, будет C, а для отрицательного — D.

Назначение упакованных десятичных полей заключается в обеспечении выполнения операций десятичной арифметики, сжатии числовых данных при хранении массивов и в использовании команд редактирования для вывода числовых данных на графические устройства вывода (печатающие устройства, дисплеи или пишущие машинки). При формировании упакованной десятичной величины из символа в коде EBCDIC левые 4 бита символов EBCDIC отбрасываются. Когда эта упакованная десятичная величина распаковывается в восьмиразрядный символьный формат EBCDIC, все символьные байты, за исключением младшего байта, дополняются слева 4 битами, соответствующими зонной части символов цифр, шестнадцатеричным F. Поэтому бесполезно пытаться перевести буквенные символы EBCDIC в упакован-

Таблица 4.1

Десятичная величина	Эквивалентный упакованный десятичный формат величины
+254	0 0 0 0 2 5 4 C
+395 116	0 3 9 5 1 1 6 C
+0	0 0 0 0 0 0 0 C
+1	0 0 0 0 0 0 1 C
−45 926	0 0 4 5 9 2 6 D
−0	0 0 0 0 0 0 0 D
+0096	0 0 0 0 0 9 6 C
+7 134 298	7 1 3 4 2 9 8 C
−0	0 0 0 0 0 0 0 B
+3 655	0 0 0 3 6 5 5 A
+116 644	0 1 1 6 6 4 4 E
+00882	0 0 0 0 8 8 2 F

ный десятичный формат, так как при последующей распаковке этих данных они неизбежно потеряют свой буквенный признак и предстанут как символы чисел.

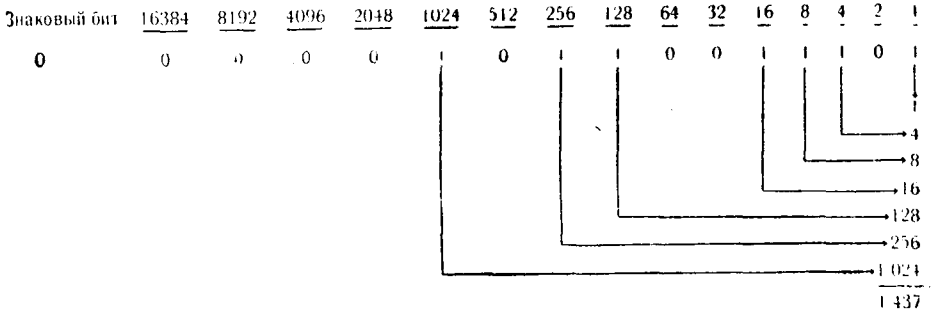
Функции упакованных десятичных величин и способы их преобразования подробно рассматриваются в разделах «Операции десятичной арифметики» и «Процедуры преобразования форматов упакованных десятичных чисел и чисел с фиксированной точкой», а также в ряде других разделов.

3. Данные с фиксированной точкой

Данные с фиксированной точкой можно рассматривать как двоичные арифметические величины, которые содержатся в полусловах, словах или регистрах. Сами по себе эти данные используются не только для арифметических операций, но и могут выступать в качестве адресов, приращений или абсолютных величин. Величину с фиксированной точкой можно рассматривать как двоичную конфигурацию целого числа, в которой для получения значения целого числа производится суммирование значений битовых позиций с весами, определяемыми степенями двойки, равными номерам позиций. Знак величины с фиксированной точкой представлен крайней левой (старшей) позицией поля или регистра. Отрицательные целые числа содержат единицу в старшей позиции, положительные — нуль.

В табл. 4.2 представлены примеры, иллюстрирующие логическую структуру целых чисел со знаком.

На примере последнего полуслова табл. 4.2 (+1437) можно показать образование суммы степеней двойки для значащих разрядов этого полуслова:



Детальную информацию об интерпретации величин с фиксированной точкой, преобразовании отрицательных величин с фиксированной точкой и шестнадцатеричных вычислениях величин с фиксированной точкой можно найти в следующих разделах:

«Арифметические действия над числами с фиксированной точкой»;

«Процедуры преобразования форматов упакованных десятичных чисел и чисел с фиксированной точкой»;

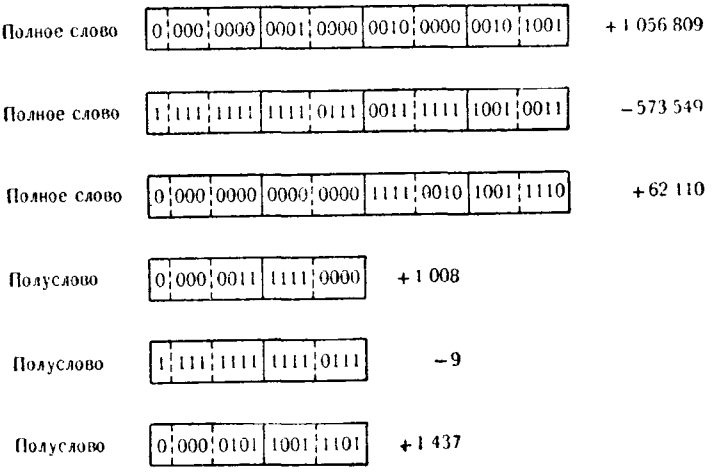
«Таблица шестнадцатерично-десятичных преобразований»;

«Таблица „степеней двойки"»;

«Сравнение данных».

Существует также прямая и ясная связь между шестнадцатеричной (арифметическое основание 16) и двоичной (арифметическое основание 2) конфигурациями величин с фиксирован-

Таблица 4.2

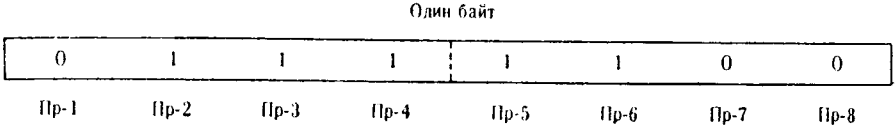


ной точкой. Используя таблицу шестнадцатерично-десятичных преобразований (часть V), из шестнадцатеричной конфигурации величины с фиксированной точкой можно непосредственно получить значение числа. Этот тип преобразования величин чрезвычайно полезен; величины с фиксированной точкой и адреса, формируемые проблемной программой, доступны только в шестнадцатеричном виде, если только программист не захочет получить поразрядную двоичную конфигурацию и затем подсчитать сумму степеней двойки для значащих разрядов. При этом программист несомненно обнаружит, что подсчет значения величины с фиксированной точкой по ее шестнадцатеричному представлению значительно более удобен.

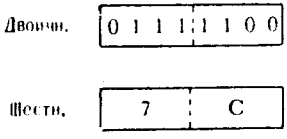
4. Специальные данные

Специальные данные сами по себе не представляются особым форматом, они скорее являются интерпретацией информации, которая может иметь любую из трех логических конфигураций — символьную, упакованную десятичную или с фиксированной точкой. Весьма часто информация выражается с помощью самой простейшей из всех возможных конфигураций, а именно в двоичном формате, при этом внешне она может выглядеть как правильное упакованное десятичное поле или символ EBCDIC. Эту ситуацию можно легко проиллюстрировать следующим примером: предположим, что программист зарезервировал байт памяти, в котором он намеревается содержать восемь переключателей, причем каждый бит этого байта используется как индикатор или двоичный переключатель. Пусть дво-

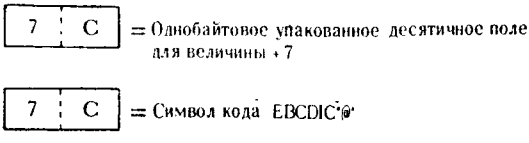
ичные позиции, взятые слева направо, соответствуют переключателям с номерами от 1 до 8. Пусть далее значение 1 переключателя соответствует выполнению условия «да», значение же 0 соответствует условию «нет». Вполне возможно, обстоятельства могут сложиться таким образом, что условие «да» одновременно выполняется в переключателях 2, 3, 4, 5 и 6. В этом случае поразрядная конфигурация байта выглядит следующим образом:



Эта двоичная конфигурация также может быть представлена шестнадцатеричной конфигурацией 7С, как это показано ниже:



Это представление может также соответствовать двум другим правильным форматам данных



Однако для программиста реальная конфигурация этих данных — состояния двоичных переключателей. Если бы программист использовал для кодирования шестнадцатеричные цифры, то вполне вероятно, что эти закодированные данные могли бы иметь вид упакованного десятичного поля или символьных данных EBCDIC.

Стоит еще раз напомнить, что свободное обращение с шестнадцатеричными конфигурациями программисту совершенно необходимо. Данные любого типа легко могут быть интерпретированы с помощью их шестнадцатеричного представления.

1. Две шестнадцатеричные цифры могут представлять символ EBCDIC.
2. Две или более шестнадцатеричных цифр могут представлять упакованное десятичное поле.

3. Шестнадцатеричные цифры, образующие величину с фиксированной точкой, могут быть быстро преобразованы в комбинацию из знакового бита и целого числа.

4. Двоичные значения легко отличить от шестнадцатеричных цифр, которые они образуют.

Упражнения [1])

1. Общий регистр имеет емкость____ байта, или одно____.

2. Код операции языка Ассемблера вместе с подходящими операндами образуют _____команды.

3. Имя, которое использовалось для обозначения оператора команды, определенной области, константы или начала программы, называется _____.

4. Программирующему на языке Ассемблера доступны _____ общих регистров.

5. Первый символ метки должен быть _____.

6. Какие номера присваиваются общим регистрам?_____

7. Байт состоит из _____ битов.

8. Мнемоническое представление команды на языке Ассемблера называется _____.

9. Упакованные десятичные цифры имеют такую же конфигурацию, как и _____ цифры.

10. Обычные двоичные величины рассматриваются как _____, даже если они не имеют знака.

11. Граница полного слова находится по тому адресу основной памяти, который делится без остатка на _____.

12. Предложение языка Ассемблера может состоять из метки, кода операции и от 1 до 3 _____.

13. Величинам с фиксированной точкой можно приписать _____ или _____ знак.

14. Граница полуслова находится по тому адресу основной памяти, который делится без остатка на _____.

15. Первый символ метки для определения любой части программы должен находиться в _____ колонке бланка кодирования.

---

[1]) Упражнения с 21-го и далее относятся к материалу, рассматриваемому в последующих главах. — *Прим. ред.*

16. Положительное полуслово с фиксированной точкой имеет максимальное значение, равное _____, отрицательное — _____.

17. Предложение формата RR имеет такой указатель длины, что команда воздействует на _____ байта данных.

18. Строка комментариев может быть указана компилятору с языка Ассемблера на бланке кодирования символом _____ в первой колонке этой строки.

19. _____ указатель длины можно иногда использовать, чтобы изменить или гарантировать количество байтов данных, которые обрабатываются командой.

20. Формат RR предложения показывает, что оба операнда этой команды находятся в _____.

21. Кодами операций, которые используются для формирования полей данных и рабочих областей, являются _____ и _____.

22. Предложение DS очищает (не очищает) область памяти, которую он определяет.

23. Метка области, которая определена предложениями DS или DC, адресует первый (старший) байт этой области. (Верно) (Неверно).

24. Область, определенная следующим предложением, содержит шесть 50-байтовых полей, каждое из которых однозначно адресуется меткой WORKFLD. (Верно) (Неверно).

```
WORKFLD     DS     6CL50
```

25. Какое общее количество байтов памяти резервируется следующими предложениями?

```
DATAFLDS    DS     0CL54
DATA1       DS     CL20
DATA2       DS     0CL34
SUBDATA1    DS     CL15
SUBDATA2    DS     CL12
SUBDATA3    DS     CL7
DATA3       DS     CL6
```

26. Будет ли выравнено в следующем наборе предложений поле RECNO по границам полуслова, полного слова или двой-

ного слова? (Укажите только наибольшую из всех возможных границ.)

```
                DS    0D
FIELDS          DS    CL6
IDNO            DS    CL5
CONSTIN         DS    CL1
RECNO           DS    CL4
SUBRES          DS    CL15
```

27. Какое количество байтов резервируется с помощью каждого из следующих предложений?

а. _____ б. _____ в. _____

```
а. SAVEFLD      DS    9H
б. WORDTABL     DS    15F
в. PACKAREA     DS    0D
```

28. Отметьте каждое из следующих наименований, которое можно использовать как метку:

_____ SETDATA1	_____ DC	_____ ADDFIELDS
_____ R6SET	_____ INTR'1'	_____ G
_____ 5TBARD	_____ BACK IN	_____ M3M265911
_____ FLDA1	_____ INITIATES	_____ 360SYS
_____ Z32MSG	_____ Y	_____ ORG

29. Максимальная длина константы типа C (символьной) _____ байтов.

30. Какое количество байтов памяти будет зарезервировано для следующей константы? _____

```
XFIELD    DC    XL3'5296874C'
```

31. Правильный «пробел» имеет шестнадцатеричную конфигурацию X'_____'.

32. _____ байтов памяти будет зарезервировано для константы

```
TITLE     DC    C'WAGES && TAXES'
```

33. Вставьте символы EBCDIC, соответствующие следующему предложению DC:

```
                LINEND    DC    CL8'START
LINEND          |   |   |   |   |   |   |   |
```

34. Общее количество _____ байтов памяти будет зарезервировано для следующей константы:

```
STATUSFLD    DC    15CL3'INTO'
```

35. Укажите в шестнадцатеричном виде содержимое поля, занятого следующей константой:

```
VALUESET     DC    3XL2'5C'
```

VALUESET 

36. Определите двоичную конфигурацию, соответствующую следующей константе:

```
BITSETS      DC    3BL1'100010001'
```

37. Определите шестнадцатеричную конфигурацию следующей константы:

```
SETHALF      DC    H'15999'
```

SETHALF 

38. Максимальная длина константы типа Z (десятичная в зонном формате) равна _____ байтам.

39. Укажите в шестнадцатеричном виде содержимое константы:

```
PAKFLDS      DC    2PL3'15'
```

PAKFLDS 

Глава 5

Задание предложения

Способность правильно интерпретировать логические задачи и применять соответствующие команды языка Ассемблера не единственное условие эффективности программирования. Созданный алгоритм программист должен закодировать таким образом, чтобы в нем мог без труда разобраться другой программист с аналогичным опытом и степенью тренированности. Точно так же, как коды операций языка Ассемблера содержат мнемонику, облегчающую их интерпретацию и понимание механизма выполнения, метки и символы, которые используются в предложении, должны быть связаны с функциями программы или задачи. Сравним следующие предложения:

```
PT111      CLC    CMP1(3,8),R45
LOOKUP1    CLC    TABLE(3,8),KEY
```

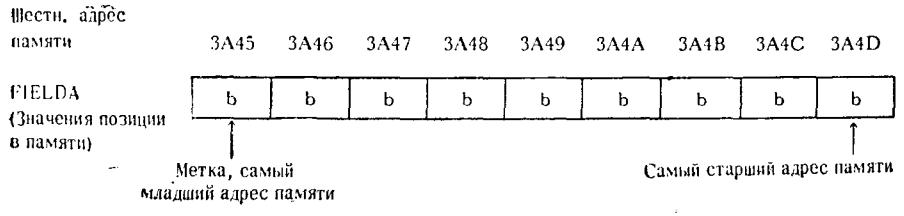
С точки зрения выполняемой задачи эти предложения идентичны. Первое предложение само по себе не дает никаких сведений о его применении. Второе ясно указывает, что является частью программы просмотра таблицы и производит сравнение ее трехбайтового сегмента с ключом. Ценность этого логического метода присвоения наименований, прямо указывающих на применение поименованных объектов, со временем возрастает. Если некоторая программа применялась в течение года и затем в связи с необходимостью учесть новые условия в нее понадобилось внести изменения, то даже автору придется тщательно просмотреть текст своей программы и заново с ней ознакомиться. Вполне возможно, что автор этой программы больше не является ответственным за ее сопровождение. В этом случае разумное присвоение наименований меткам и полям особенно важно. Удачный выбор меток в совокупности с достаточно полными комментариями на бланке кодирования может резко уменьшить время, затрачиваемое на отладку и сопровождение программы. Всякий раз при кодировании программы придерживайтесь хорошего правила: считайте, что вам может быть поручено внести в нее изменение через 2 года, пишите комментарии и присваивайте наименования так, чтобы разобраться в программе по прошествии этого времени.

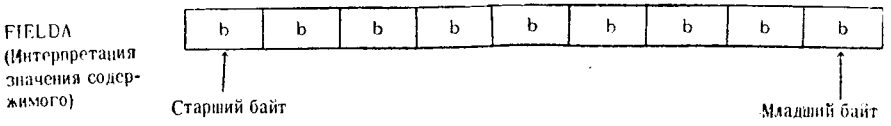
Присвоение индивидуальных имен программам, подпрограммам, областям и константам является одним из способов, с помощью которого программист в процессе кодирования предложений получает возможность осуществлять над ними наиболее удобный и гибкий контроль. Существуют другие функции, которые также предоставляют программисту некоторый диапазон возможностей при кодировании любой заданной команды. Они связаны с методами указания операндов, включающими элементы адресации, указания длины и самоопределенных величин. Хотя в результате применения составных или комплексных операндов должны формироваться конкретные адреса или необходимые величины, эти цели могут быть достигнуты разными способами.

А. АДРЕСАЦИЯ

Несмотря на то что имеется несколько способов формирования любого заданного адреса внутри проблемной программы, сейчас будет рассмотрен только самый простой и непосредственный.

Один из методов формирования адреса для любой заданной точки в программе заключается в том, что эта точка «помечается» меткой или символом. Метки можно использовать для идентификации адреса константы, определенной области памяти, программы или подпрограммы, таблицы или любого конкретного предложения. Сама по себе метка всегда будет вырабатывать адрес, который указывает на самый левый байт поля или команды, представляющий самый младший адрес в области памяти, идентифицированной меткой. Если метка относится к полю данных, то этот принцип может привести к некоторому недоразумению — адрес будет указывать на самый левый байт поля, который в действительности имеет самый младший адрес памяти для этого поля, но в то же самое время этот байт внутри поля может рассматриваться как старший. Очевидно, что имеются два типа значений адреса, которые надо четко различать между собой, — значение адреса памяти и значение содержимого внутри самого поля. Для любого заданного поля это можно представить следующим образом:





Любое обращение с помощью немодифицированной метки будет отсылать к младшему адресу в памяти поля или величины, указанной этой меткой. Если адрес метки «модифицирован», например FIELDA + 5, то это следует понимать таким образом, что вырабатываемый адрес указывает на ту точку, которая в памяти сдвинута в сторону больших адресов по отношению к метке на столько байтов, сколько их указано в модификаторе.

Настройка адреса обычно применяется для локализации определенной непомеченной позиции в памяти внутри поля данных или для организации пошагового поиска в последовательности сегментов данных. Ее можно использовать для модификации указателя длины в команде, хотя это рекомендуется делать только после приобретения некоторого навыка в программировании.

На рис. 5.1 показан один пример настройки адреса в программе. В этом примере некоторое пятибайтовое поле содержит пять отдельных позиций символов. Если в каждой из этих позиций обнаруживаются определенные конфигурации, то происходит переход к соответствующей подпрограмме, обработка информации и затем снова возврат к основной программе. Здесь предполагается, что поле FIELD5 содержит данные из входного потока.

Оператор первой команды сравнивает первый байт поля FIELD5 с символом L; если результат сравнения неудовлетворителен, то происходит переход к следующей команде сравнения, CHEK2. Каждый из операторов сравнения производит проверку по адресу, который на 1 байт больше, чем адресованный предыдущим оператором сравнения — FIELD5, FIELD5+1, FIELD5+2, FIELD5+3, FIELD5+4, — после чего все пять байтов поля FIELD5 оказываются проверенными. Если байт, адресованный командой сравнения, не содержит указанного в ней символа, то происходит переход к следующему оператору сравнения. Если сравнение оказалось удачным (произошло совпадение), то происходит переход к команде Branch and Link — BAL (Переход с возвратом). Выполнение команды BAL приведет к переходу к строго определенной подпрограмме, выполнению задачи и затем к возврату к той команде, которая следует сразу же за оператором BAL. Настройка адресов демонстрируется следующим примером:

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M 040
Printed in U.S.A.

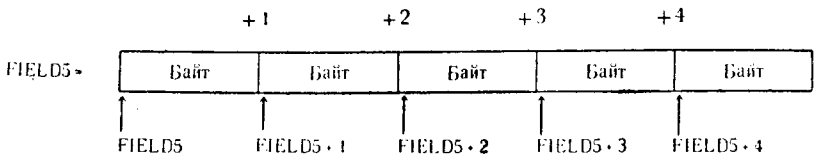
PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				00000 1
CHEKFLD5	CLI	FIELD5,C'L'		00000 2
	BC	7,CHEK2		00000 3
	BAL	9,RTNEL		00000 4
CHEK2	CLI	FIELD5+1,C'A'		00000 5
	BC	7,CHEK3		00000 6
	BAL	9,RTNEA		00000 7
CHEK3	CLI	FIELD5+2,C'M'		00000 8
	BC	7,CHEK4		00000 9
	BAL	9,RTNEM		00001 0
CHEK4	CLI	FIELD5+3,C'P'		00001 1
	BC	7,CHEK5		00001 2
	BAL	9,RTNEP		00001 3
CHEK5	CLI	FIELD5+4,C'S'		00001 4
	BC	7,CHEKDWN		00001 5
	BAL	9,RTNES		00001 6
CHEKDWN	BC	15,ENDCHEK		00001 7
*				00001 8
*				00001 9
FIELD5	DC	CL5' '		00002 0
*				00002 1

Рис. 5.1.



Такая настройка адресов не всегда пригодна для практического использования, особенно при пошаговом поиске в последовательности сегментов данных или таблице значений. Для применений такого рода более эффективной оказывается загрузка адреса первого сегмента в общий регистр и затем прибавление величины (равной длине сегмента) к этому регистру всякий раз, когда в соответствии с алгоритмом программы необходимо проверить следующий сегмент. Значение регистра, содержащего адрес сегмента, используется затем как базовый регистр с операндом, задающим нулевое смещение.

Предположим, что проблемная программа содержит очень большую таблицу, в которой необходимо отыскать определенные величины. Таблица составлена из многих сегментов, каждый из которых имеет длину 10 байтов. Таблицу можно определить внутри программы, как показано на рис. 5.2.

Величина, которую необходимо отыскать в таблице, содержится в третьем и четвертом байтах каждого сегмента таблицы. Кодирование начальной части программы может быть выполнено несколькими способами. На рис. 5.3 показаны два способа кодирования.

Предложение 001 (пример 1) загружает адрес *первого* байта таблицы TABLE в общий регистр 9.

Предложение 001 (пример 2) загружает адрес *третьего* байта TABLE в общий регистр 9.

Предложение 002 (пример 1) сравнивает третий байт текущего сегмента таблицы с литералом. Этот третий байт адресуется смещением «0 + 2», добавленным к текущему адресу TABLE, содержащемуся в общем регистре 9. Число 2 в выражении (2,9) означает, что сравнение должно производиться для полей длиной 2 байта.

Предложение 002 (пример 2) выполняет то же самое сравнение. В силу того что адрес, загруженный в общий регистр 9, равен в данном случае TABLE+2, нет надобности задавать фактор смещения отличным от нуля.

Предложение 003 для обоих примеров будет вызывать переход к PASSBAL при несовпадении сравниваемых в предложении 002 величин.

Предложение 004 для обоих примеров — команда Branch and Link (Переход с возвратом) к предполагаемой подпрограмме.

IBM

IBM System/360 Assembler Coding Form

X28-6508-2 U/M060
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE | OF
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
TABLE	DC	CL30'A1039246708R2653942211S2321477745'		290
	DC	CL30'G7162793241V5156347089Y7719256841'		291
	DC	CL30'E6338472081L7538934201A2769568891'		292
	DC	CL30'B9240 143061C0294869501A1467038101'		293
	DC	CL30'R7597892121B5606311401T8720507381'		294
	DC	CL30'P6470911166F2298411589H0516203181'		295
	DC	etc.		296
	DC	etc.		297
	DC	etc.		298
	DC	etc.		299
				300

Рис. 5.2

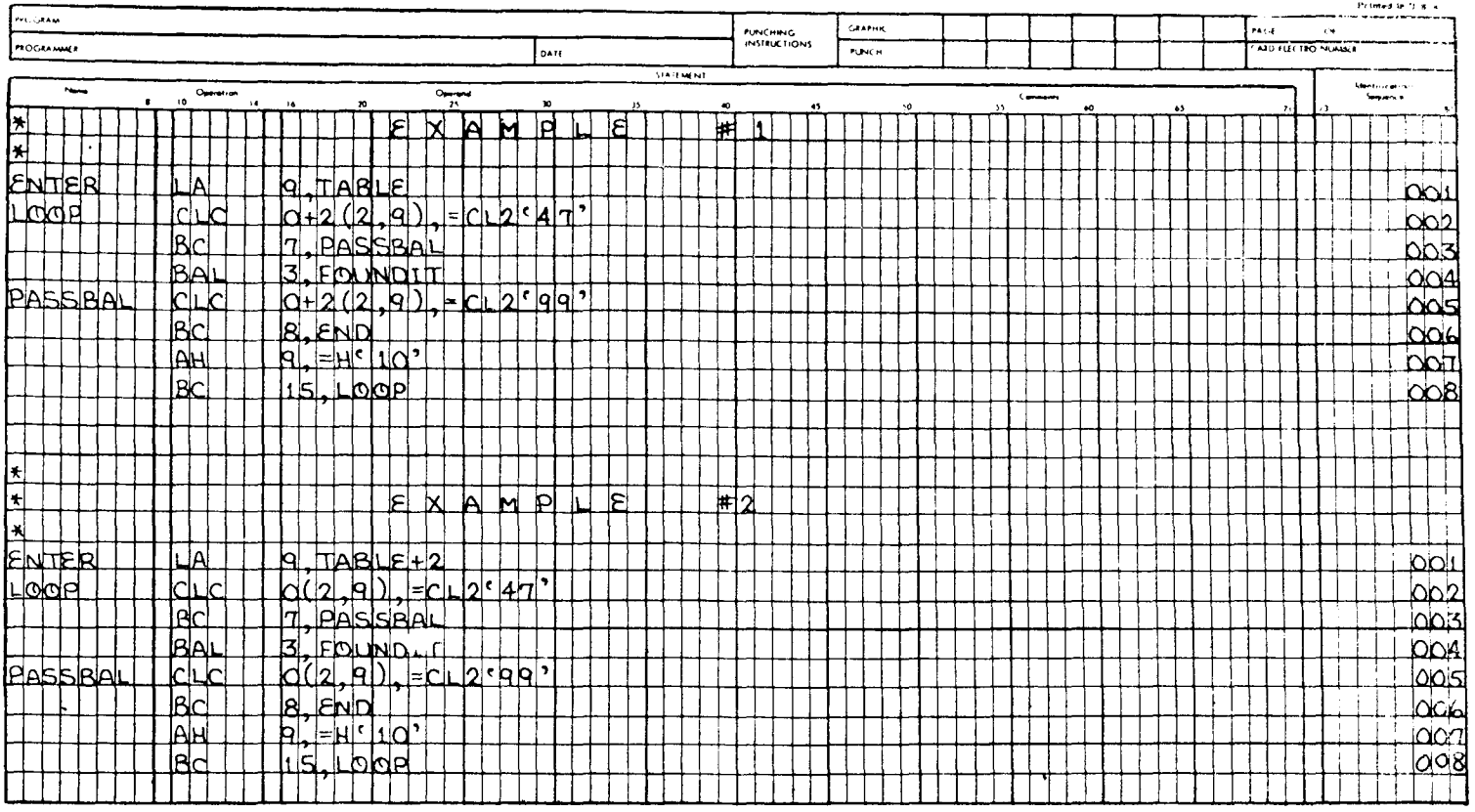

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
*
EXAMPLE #1
ENTER LA 9,TABLE 001
LOOP CLC 0+2(2,9),=CL2'47' 002
BC 7,PASSBAL 003
BAL 3,FOUNDIT 004
PASSBAL CLC 0+2(2,9),=CL2'99' 005
BC 8,END 006
AH 9,=H'10' 007
BC 15,LOOP 008
*
EXAMPLE #2
ENTER LA 9,TABLE+2 001
LOOP CLC 0(2,9),=CL2'47' 002
BC 7,PASSBAL 003
BAL 3,FOUND... 004
PASSBAL CLC 0(2,9),=CL2'99' 005
BC 8,END 006
AH 9,=H'10' 007
BC 15,LOOP 008

Рис. 5.3.

Предложением 005 для обоих примеров является другое сравнение тех же самых байтов из TABLE, которые используются в предложении 002. В этом предложении первый операнд совпадает с операндом, используемым в предложении 002 соответствующего примера.

Предложением 006 для обоих примеров является предложение Branch on Condition (Условный переход), который проверяет результаты выполнения предложения 005 и решает, имеет ли место переход.

Предложение 007 добавляет значение длины одного сегмента таблицы к содержимому регистра 9. При первом проходе через программу LOOP регистр 9 будет содержать адрес, равный TABLE+10 в примере 1 и TABLE+12 в примере 2. При втором проходе через LOOP регистр 9 будет содержать адрес, соответствующий TABLE+20 для примера 1, TABLE+22 для примера 2 и т. д.

Предложение 008 для обоих примеров — *безусловный переход* к первой команде циклической программы LOOP.

Обе приведенные программы решают идентичные задачи. Единственное различие между ними заключается в значении адреса, которое было загружено в общий регистр 9 предложением 001, и в последующем использовании этого адреса в предложениях 002 и 005.

Как показано в этих примерах, содержимое регистра — указателя адреса можно увеличить прибавлением величины с фиксированной точкой длиной в полуслово к содержимому этого регистра. Величиной, которую необходимо было прибавить в этом случае, была длина сегмента таблицы, а именно +10. Вместо предложений 001 и 007 можно использовать любой из наборов предложений, приведенных на рис. 5.4, причем конечный результат будет одним и тем же.

Набор предложений A загружает адрес TABLE в общий регистр 9. Затем он фактически утверждает: «Загрузить действительное значение адреса, равное сумме +10, и адреса, находящегося в настоящее время в регистре 9, в регистр 9». Если этот общий регистр сначала содержал адрес, TABLE, равный 30563, то команда увеличения содержимого регистра 9 работает следующим образом:

+ 10 плюс регистр 9(30563) = регистр 9(30573).

Набор предложений B на своем первом шаге также загружает адрес TABLE в общий регистр 9. Затем он загружает абсолютное значение адреса +10 в общий регистр 4. Увеличение значения в регистре 9 выполняется с помощью команды Add Register (Сложение), которая прибавляет значение приращения из регистра 4 (+10) к значению адреса в общем регистре 9.

IBM

IBM System/360 Assembler Coding Form

X28-6509-5 U/M01-1
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments
*A			
ENTER	LA	9,TABLE	(REPLACES STATEMENT 001)
	LA	9,10(9)	(REPLACES STATEMENT 007)
*B			
ENTER	LA	9,TABLE	(REPLACES
	LA	4,10	STATEMENT 001)
	AR	9,4	(REPLACES STATEMENT 007)
*C			
ENTER	LA	9,TABLE	(REPLACES STATEMENT 001)
	A	9,INCRMNT	(REPLACES STATEMENT 007)
*			
INCRMNT	DC	F'10'	(FULLWORD CONSTANT OF +10)

Рис. 5.4.

Набор предложений, названный C, определяет константу с фиксированной точкой длиной в полное слово, равную +10. После того как общий регистр 9 загружается адресом TABLE, программа увеличивает этот адрес с помощью команды с фиксированной точкой Add (Сложение), прибавляя константу +10 длиной в полное слово к величине, находящейся в регистре 9.

Б. УКАЗАТЕЛЬ ДЛИНЫ

Указатель длины рассматривается как управляющая величина, определяющая количество байтов, на которые должно воздействовать предложение. Одни команды позволяют программисту выборочно использовать явные либо неявные указатели длины, между тем другие имеют фиксированный указатель длины. В каждой команде стандартного набора команд языка Ассемблера, за исключением команд десятичной арифметики, имеется один указатель длины. Команды десятичной арифметики над упакованными десятичными числами имеют два указателя длины — по одному на каждый операнд. Весьма важно хорошо знать характеристики указателей длины каждой из команд языка Ассемблера.

1. Фиксированные указатели длины

Считается, что те команды, в формат предложений которых не входит компонент L, показывающий, что для этой команды может быть определен указатель длины, имеют фиксированный указатель длины. Это означает, что такие предложения внутренне содержат фактор длины, который нельзя изменить во время кодирования этого предложения. Действительное количество байтов или полных слов, определяемых фиксированным указателем длины, различно для разных команд. К командам этой категории относятся все команды форматов RR, RX, RS и SI. Большинство «регистровых» команд, таких, как CLR, AR, LTR, SR и т. п., будет воздействовать только на 4 байта (регистр, или полное слово) данных. Команды умножения и деления с фиксированной точкой будут воздействовать на пару регистров. Команды Load Multiple (Загрузка групповая) и Store Multiple (Запись в память групповая) будут воздействовать на такое количество полных слов и регистров, которое соответствует диапазону значений, определенному операндами R_1 и R_3. Остальные команды воздействуют только на один байт данных — это такие команды, как CLI, STC, IC и MVI.

Любая попытка задать указатель длины для команды, которая содержит фиксированный указатель длины, будет

приводить или к ошибке при работе компилирующих программ, или к неправильной интерпретации этой величины; компилятор будет рассматривать указатель длины как номер регистра базы или индекс-регистра, если заданная величина находилась в пределах от 1 до 15.

2. Неявный указатель длины

Если конкретный формат команды языка Ассемблера предусматривает возможность задания указателя длины, то от программиста зависит, нужно ли его кодировать. Если программист решает не указывать длину, то она считается заданной неявно. В операторе с неявно заданной длиной последняя истолковывается как длина поля, адресованного первым операндом предложения.

Рассмотрим следующий набор предложений:

```
RECVR     DC      CL6'ЬЬЬЬЬЬ'
SEND1     DC      CL4'ABCD'
SEND2     DC      CL4'EFGH'
*
MOVEIT    MVC     RECVR,SEND1
```

Выполнение команды, помеченной MOVEIT, приведет к пересылке 6 байтов данных (ABCDEF) в поле с именем RESVR. Неявная длина для выполнения этого предложения равна длине первого операнда, которая в свою очередь равна 6 байтам. В то время, когда выполняется это предложение, система начинает пересылать эти байты данных, начиная с адреса, заданного вторым операндом, и продолжает пересылку до тех пор, пока неявно выраженная длина первого операнда целиком заполнится. В результате этого будут пересланы 4 байта данных, находящихся в SEND1, и вслед за ними — 2 первых байта поля SEND2.

Противоположный результат будет иметь место в примере, где поле второго операнда длиннее поля первого.

```
RECVR     DC      CL3'ЬЬЬ'
SENDR     DC      CL'139B2'
*
MOVEDATA  MVC     RECVR,SENDR
```

В результате действия этого предложения только первые 3 байта поля SENDR (139) будут пересланы в RECVR. Так как неявно выраженная длина поля первого операнда определяет

количество байтов данных, которые должны быть пересланы, выполнение операции пересылки прекращается, как только длина этого операнда окажется исчерпанной. Заметим, что если используется настройка адреса, как, например, ...MVC RECVR+ +2,SENDR..., то, как и ранее, действует неявно выраженная длина поля, принимающего информацию. 3 байта данных, начиная с SENDR, будут пересылаться в 3 байта данных, начиная с RECVR+2.

Команды десятичной арифметики выделяются в том смысле, что оба операнда этих команд могут рассматриваться как имеющие неявно заданную длину.

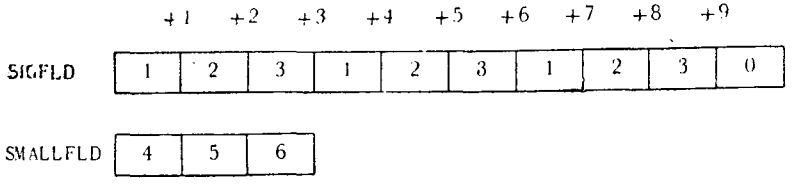
В силу этого выполнение команд десятичной арифметики в упакованном формате в случае полей операндов разной длины требует соответствующего знания особенностей этих команд. Если указатель длины первого десятичного операнда больше, чем указатель длины второго операнда, то обычно не возникает никаких трудностей для успешного завершения выполнения команды. Однако если указатель длины второго операнда больше, чем указатель длины первого операнда, то выполнение программы, использующей команды десятичной арифметики, вполне может привести к десятичному переполнению и программному прерыванию. Можно было бы порекомендовать программисту использовать в этих случаях явно выраженный указатель длины.

3. Явные указатели длины

Использование явных указателей длины относится к действительному кодированию величины длины внутри операнда. Этот прием можно применять для подтверждения установленного указателя длины поля или для изменения неявной длины. Во многих типичных ситуациях этот способ задания может играть существенную роль. В памяти можно хранить большой массив данных, имеющий одну общую метку, но, используя настройку адреса и явные указатели длины, можно пересылать малые порции данных или манипулировать ими по своему усмотрению. Независимо от длины любого поля данных указатель длины, имеющий такую величину, которая допускается кодом операции конкретной команды, можно использовать для пересылки данных в это поле и последующие поля, если это необходимо. Стоит еще раз напомнить, что явный указатель длины можно использовать только в тех командах языка Ассемблера, формат которых допускает задание указателя длины.

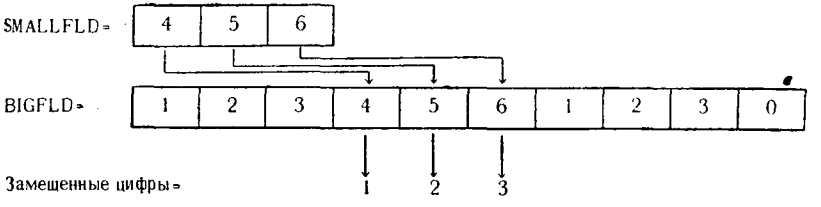
Следующие примеры иллюстрируют случаи применения указателя длины.

Пример 1. Поля данных, на которые должны воздействовать команды, и операторы соответствующих команд таковы:

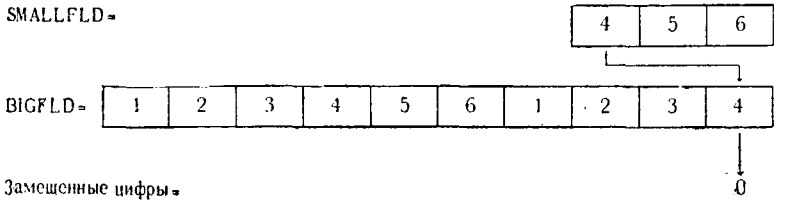



```
BIGFLD      DC      CL10'1231231230'
SMALLFLD    DC      CL3'456'
*
            MVC     BIGFLD+3(3), SMALLFLD   001
            MVC     BIGFLD+9(1), SMALLFLD   002
```

Предложение 001 интерпретируется следующим образом: «Переслать 3 байта данных (как показано явным указателем длины, равным 3) из поля SMALLFLD в четвертый, пятый и шестой байты поля BIGFLD(BIGFLD+3)». Эта пересылка выглядит так:



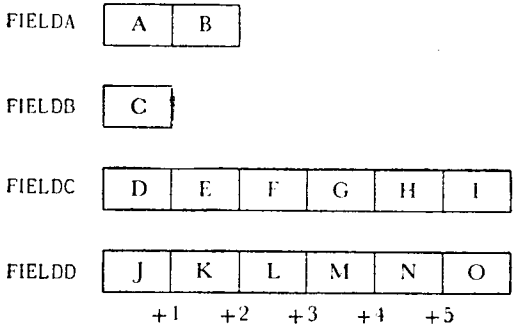
Предложение 002 утверждает следующее: «Переслать один байт данных, как указано явным указателем длины, равным 1, взятый по адресу поля SMALLFLD, в десятый байт поля BIGFLD(BIGFLD+9)». Эта процедура показана ниже:



Приведеный только что пример показал, каким образом все «посылаемое» поле в целом (или его часть) может быть переслано в любую часть принимающего поля для любой явной

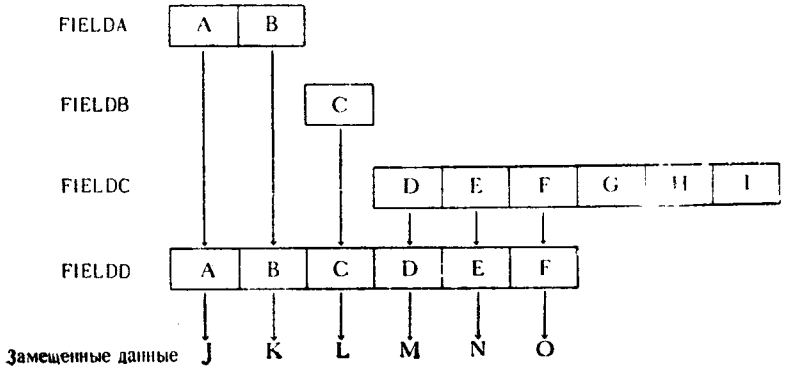
длины, если указатели длины не превышают количества байтов, допустимого этой командой.

Пример 2. Поля данных и предложение имеют следующий вид:

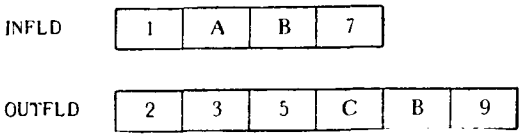



```
FIELDA   DC      CL2'AB'
FIELDB   DC      CL1'C'
FIELDC   DC      CL6'DEFGHI'
FIELDD   DC      CL6'JKLMNO'
*
         MVC     FIELDD(6),FIELDA
```

Это предложение предписывает: «Начиная с адреса, определяемого меткой FIELDA, переслать 6 байтов данных в FIELDD». Так как это предложение определяет длину поля пересылаемых данных, равную 6 байтам, то в поле FIELDD будут пересланы поля FIELDA (2 байта), FIELDB (1 байт) и первые 3 байта поля FIELDC. Выполнение этого предложения можно проиллюстрировать следующим образом:

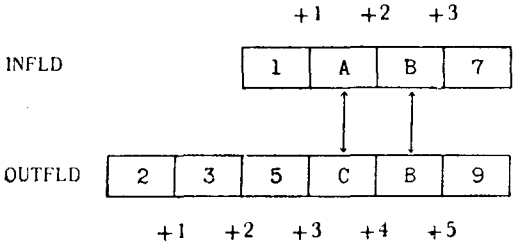


Пример 3. Этот пример относится к сравнению частей двух полей данных. Поля данных и команда сравнения имеют следующий вид:



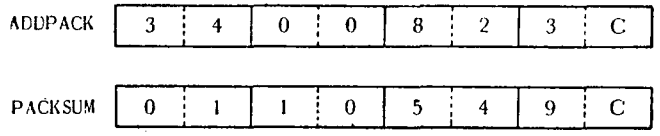

```
INFLD    DC     CL4'1AB7'
OUTFLD   DC     CL6'235CB9'
*
         CLC    OUTFLD+3(2),INFLD+1
```

Байты данных, которые сравниваются при выполнении этого предложения, представлены ниже:



Как видно из приведенного выше рисунка, сравниваются только две пары байтов, содержащих буквы.

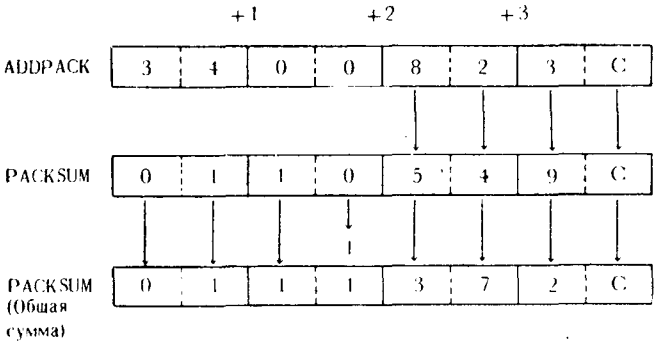
Пример 4. Этот пример показывает, каким образом часть упакованного десятичного поля можно прибавить к целому упакованному десятичному полю. Данные, содержащиеся в полях, показаны ниже:



Предложение, которое необходимо выполнить:

```
         AP     PACKSUM(4),ADDPACK+2(2)
```

Выполнение этого предложения можно показать с помощью следующего рисунка:



Приведенный пример показывает, как последние 2 байта поля ADDPACK, содержащие упакованную десятичную величину +823, складываются с содержимым всего поля PACKSUM. Принимается, что результирующая сумма занимает все 4 байта поля PACKSUM.

Кроме приведенных выше, в книге имеется много других примеров использования явных указателей длины.

В. САМООПРЕДЕЛЕННЫЕ СИМВОЛЫ И ВЕЛИЧИНЫ

Самоопределенным символом (или величиной) называется такой символ (или величина), который записывается в операнде в своем собственном представлении. Обычно под ними понимаются литералы, символы с непосредственным представлением и абсолютные величины. Способы их задания описаны ниже.

1. Литералы

Хотя в данной книге построению и композиции литералов посвящен специальный раздел, уже здесь необходимо привести общие соображения относительно их функций. Как правило, литералом является сравнительно небольшая по объему область данных, или самоопределенная величина, которая должна быть использована в качестве операнда. Литералы применяют чаще всего в тех случаях, когда значение величины или содержимое данных литерала должно остаться неизменным. В силу того что литерал не имеет метки, программист не имеет возможности обратиться к нему из другой точки своей проблемной программы. Если литерал с одинаковым значением необходимо использовать во многих местах проблемной программы, то его приходится каждый раз целиком переписывать заново. Полное описание различных типов литералов и их представлений содержится в разделе «Определение констант и литералов».

Все приведенные ниже предложения содержат литералы. Использование литерала определено в соответствии с тем

предложением, в котором он появляется.

```
AP      PAKCNTR(3),=PL1'1'
```

Это предложение прибавляет упакованную десятичную величину +1, представленную литералом, к числу в трехбайтовой области с меткой PAKCNTR. Программист мог бы создать помеченную однобайтовую десятичную константу и использовать ее метку в качестве второго операнда данного оператора вместо литерала. Рассмотренная команда может являться частью программы, выполняющей подсчет или осуществляющей управление количеством проходов процесса обработки через заданный участок программы.

```
CLC     PACKFLD(3),=X'00000C'
```

Это предложение сравнивает 3 байта поля упакованных десятичных данных, адресованного с помощью метки PACKFLD, с трехбайтовым шестнадцатеричным литералом. Литерал представляет собой трехбайтовое упакованное десятичное поле, содержащее число +0. В этом конкретном случае программист хочет только знать, содержит ли поле PACKFLD поразрядную двоичную конфигурацию, соответствующую числу +0. Программист мог бы использовать команду Compare Packed (CP) (Сравнение десятичное), если бы не то обстоятельство, что команда CP рассматривает плюс нуль и минус нуль как равные величины.

```
CLC     RECORD+5(2),=CL2'ID'
```

В этом примере сравниваются шестой и седьмой байты поля RECORD с двухбайтовым литералом, содержащим буквенные символы ID. Предполагается, что данные, находящиеся в RECORD, будут изменяться всякий раз, когда программа использует новую запись данных. Литерал применяется для проверки содержимого каждой новой секции данных на наличие символов ID, и, таким образом, использование литерала столь же целесообразно, как и использование двухбайтовой константы, содержащей ID. Программисту зачастую бывает легче проследить логику выполнения программы, если он использует этот тип литерала, нежели интерпретировать предложение, в котором второй операнд является меткой, адресующей константу.

Сравним три набора предложений, приведенных на рис. 5.5, каждый из которых включает предложение с использованием литерала и аналогичный оператор с использованием метки, которая адресует константу с точно такой же конфигурацией, что и у литерала.

В каждом примере литерал самоопределен не только в отношении содержания данных, но и в отношении формата и длины.

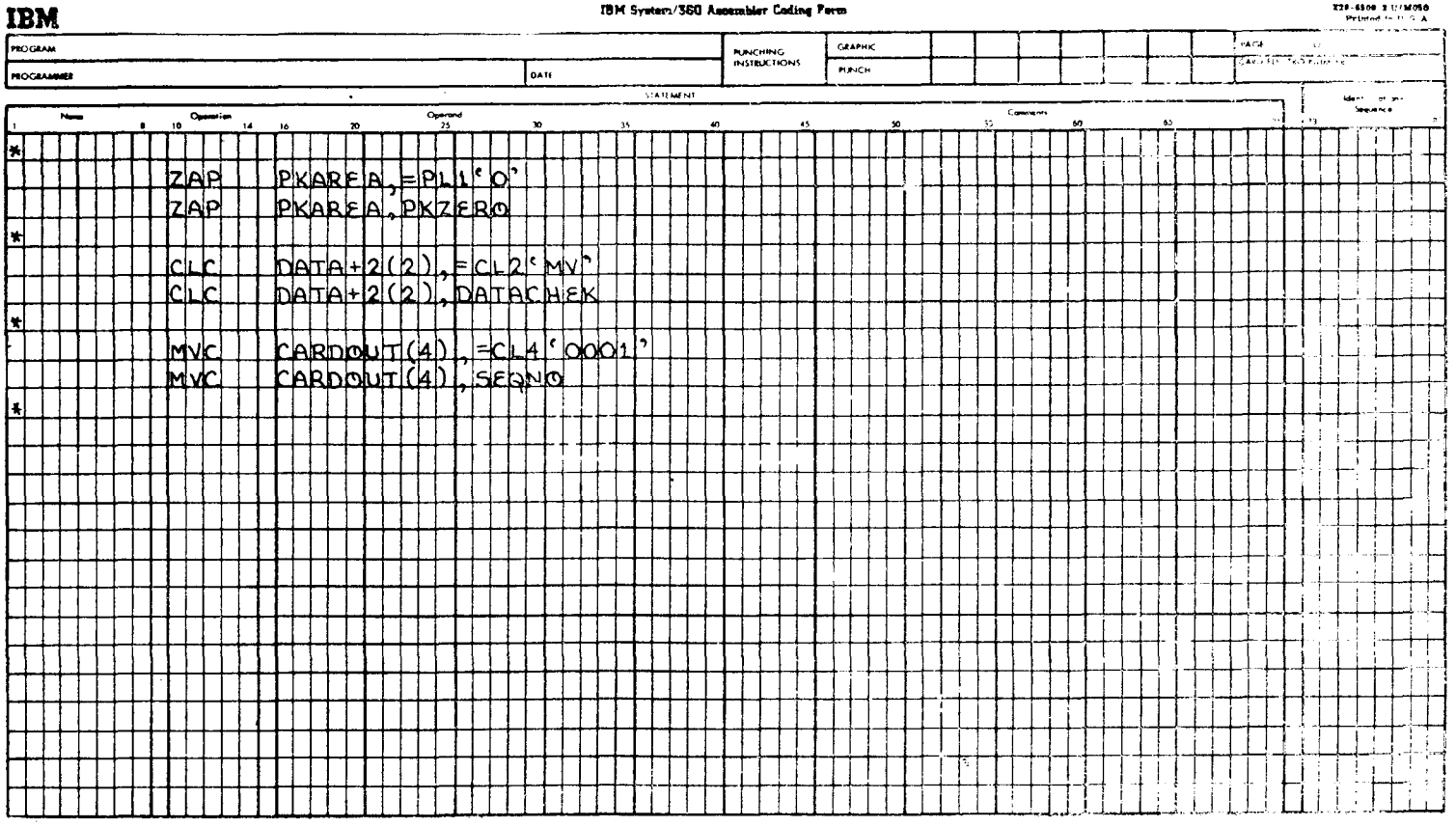

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
*
ZAP PKAREA,=PL1'0'
ZAP PKAREA,PKZERO
*
CLC DATA+2(2),=CL2'MV'
CLC DATA+2(2),DATACHEK
*
MVC CARDOUT(4),=CL4'0001'
MVC CARDOUT(4),SEQNO
*

Рис. 5.5.

Предложения, использующие в качестве второго операнда помеченные данные, не являются самоопределенными. Просматривая проблемную программу, программисту придется найти расположение описания меток второго операнда для того, чтобы ознакомиться с характеристиками и содержимым констант.

2. Непосредственные символы или величины

Как показано в описании некоторых команд языка Ассемблера, определенные форматы команд требуют использования непосредственных данных. Конфигурация записи непосредственных данных весьма близка способу записи литерала с учетом следующих исключений:

1. Впереди непосредственных данных не ставится знак равенства (=).
2. Они могут содержать только один байт данных.
3. Они не могут быть заменены константой или символом.

Функциональные команды языка Ассемблера, которые используют непосредственный символ или представление, перечислены ниже:

AND IMMEDIATE (NI)
EXCLUSIVE OR IMMEDIATE (XI)
OR IMMEDIATE (OI)
COMPARE LOGICAL IMMEDIATE (CLI)
MOVE IMMEDIATE (MVI)
TEST UNDER MASK (TM)

Конфигурацию непосредственных данных можно выражать в символьном, двоичном или шестнадцатеричном виде. Ее нельзя выражать в форме упакованного байта десятичных данных. Решение о том, какую конфигурацию лучше использовать, зависит, по всей вероятности, от типа используемого оператора. Операторы команд CLI и MVI, вероятно, стоит записывать с использованием символьной конфигурации. Остальные команды чаще всего имеет смысл записывать, используя шестнадцатеричную или поразрядную двоичную конфигурацию, так как байт непосредственных данных, используемый в этих командах, представляет собой маску, которая содержит поразрядный набор двоичных условий.

На рис. 5.6 представлены предложения, использующие байт непосредственных данных. В каждом примере оператор показан дважды, с двумя различными конфигурациями для выражения непосредственных данных. В силу того что сфера действия этих команд ограничена 1 байтом данных, они не содержат указателей

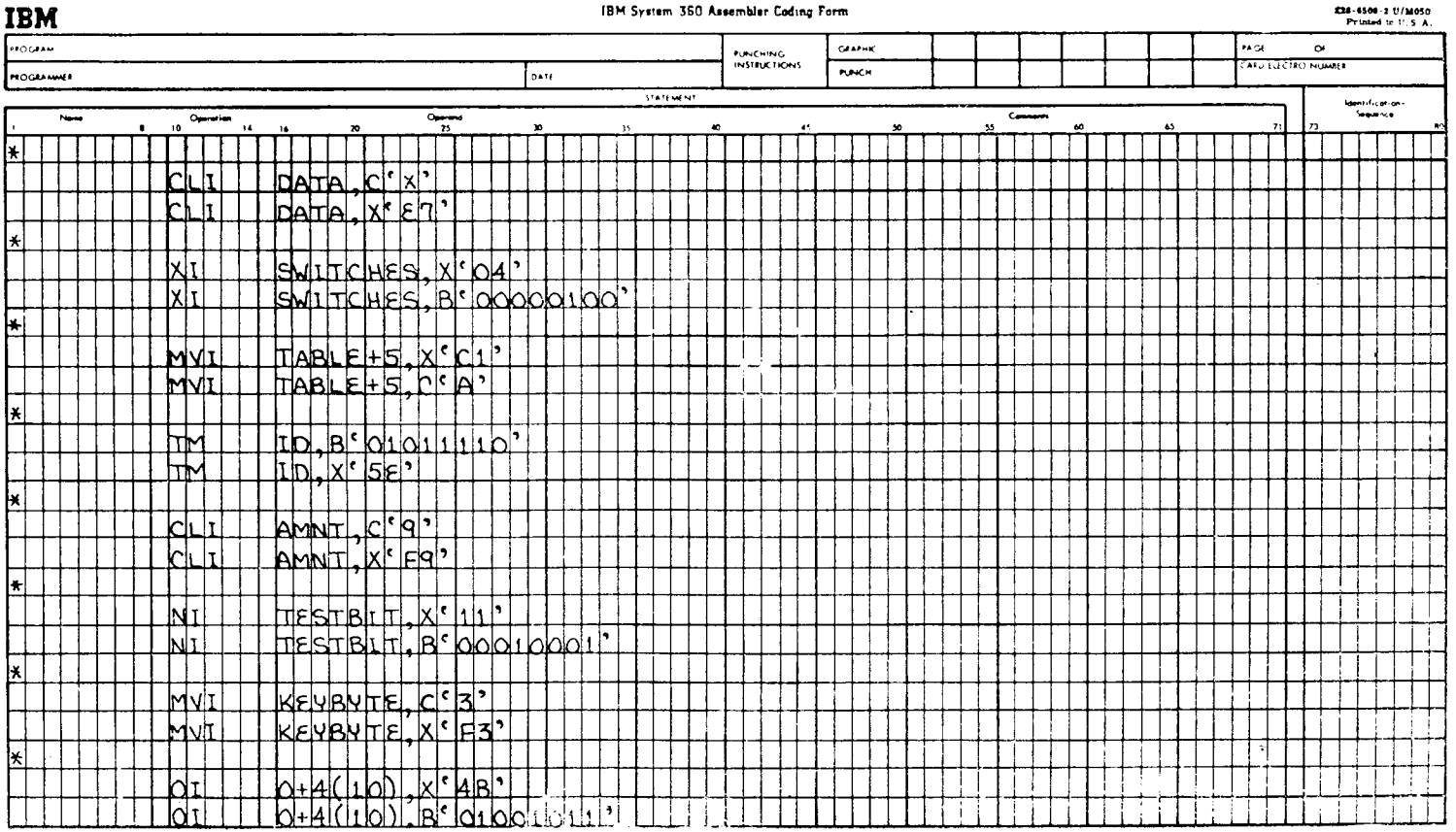

IBM
IBM System 360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operands
Comments
Identification-Sequence
*
CLI DATA,C'X'
CLI DATA,X'E7'
*
XI SWITCHES,X'04'
XI SWITCHES,B'00000100'
*
MVI TABLE+5,X'C1'
MVI TABLE+5,C'A'
*
TM ID,B'01011110'
TM ID,X'5E'
*
CLI AMNT,C'9'
CLI AMNT,X'F9'
*
NI TESTBIT,X'11'
NI TESTBIT,B'00010001'
*
MVI KEYBYTE,C'3'
MVI KEYBYTE,X'F3'
*
OI 0+4(10),X'4B'
OI 0+4(10),B'01001011'

Рис. 5.6.

длины. Заметим, что байт данных, который используется для представления непосредственного символа, должен быть самоопределенным. Непосредственный символ P'4' является неверным, так как для интерпретации компилятор был бы вынужден преобразовать его в шестнадцатеричную конфигурацию X'4C'. Однако, с другой стороны, непосредственный символ, выраженный как X'4C', является корректным.

Каждое предложение из наборов, приведенных на рис. 5.6, будет интерпретировано компилятором как совпадающее с другим из этого же набора. Поэтому программист может по своему усмотрению задавать байт непосредственных данных в форме, которую он предпочитает.

3. Абсолютные величины

Под абсолютной величиной понимается десятичный самоопределенный терм (или выражение). Он записывается в своей основной численно-цифровой форме и не содержит символов, указывающих знак величины.

Типичный пример использования абсолютной величины можно найти в связи с командой Load Address (LA) (Загрузка адреса), когда она используется для загрузки величины, управляющей циклом, в регистр. Такой тип использования показан ниже:

Это предложение вызывает загрузку абсолютной величины, равной 1024, в общий регистр 5. После завершения выполнения этой команды общий регистр 5 будет содержать величину +1024.

Абсолютная величина может также служить для задания смещения в операндах многих команд языка Ассемблера, что часто используется при определении адреса не имеющей метки точки внутри описанного поля.

```
TABLE     DC      50CL14'Я'
*
MODULE    LA      6,TABLE
          MVC     4(2,6),=CL2'ID'
```

Константа TABLE задана состоящей из 50 сегментов по 14 байтов каждый. Команда Load Address (LA) загружает адрес первого байта TABLE в общий регистр 6. Команда Move Cha-

racters (MVC) (Пересылка символов) предписывает: «По адресу, который представляет собой пятый байт поля TABLE (смещение, равное 4 плюс адрес TABLE, который содержится в общем регистре 6), переслать двухбайтовый литерал, задаваемый вторым операндом». Оператор команды MVC распадается на следующие компоненты:

Если в процессе выполнения программы десятичный адрес TABLE был равен 23410, то первый операнд в только что показанном предложении MVC будет вырабатывать адрес 23414.

Логически величина, которая используется для представления номера регистра (от 0 до 15) как часть операнда или как сам операнд, также является абсолютной величиной. Это не означает, что каждый раз, когда компилятор встречает величину, лежащую в диапазоне значений от 0 до 15, он будет предполагать наличие общего регистра; такая величина истолковывается как номер регистра, только если она занимает соответствующее место в предложении.

Г. ИНИЦИАЛИЗАЦИЯ ПРОГРАММЫ

По мере чтения этой книги программист, вероятно, захочет проверить степень усвоения изучаемых принципов. Вместо того чтобы с самого начала требовать от программиста изучения распределения регистров, областей сохранения, программных секций и т. п., приведем несколько вариантов готовых кодов, обеспечивающих необходимые действия в начале программы.

Используя команды, показанные на рис. 5.7, программист может ввести небольшие программы, написанные им на языке Ассемблера, в соответствующие места внутрь программной секции и скомпилировать свою программу.

Группа 1 предложений от метки INITL до метки SAVEREG инициализирует программу и производит присваивание значений базовым регистрам. Данная модель организована с тремя базовыми регистрами, обслуживающими до 4096 байтов каждый, так что программист будет иметь для своей программы около 12 288 байтов.

IBM

IBM System/360 Assembler Coding Form

X28-6808-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 3 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
INITL	CSECT			000021
BEGIN	SAVE	(14,12),,*		000022
	BALR	R3,0		000023
	USING	BR3,R3		000024
BR3	L	R4,BR4		000025
	L	R5,BR5		000026
	USING	BR3+4096,R4		000027
	USING	BR3+8192,R5	→ (1)	000028
	ST	R13,SAVEREG+4		000029
	LA	R13,SAVEREG		000030
	BC	15,START		000031
*				000032
*				000033
BR4	DC	A(BR3+4096)		000034
BR5	DC	A(BR3+8192)		000035
SAVEREG	DS	18F		000036
*				000037
*				000038
START	- -	- - - - - - - -	→ (2)	000039
		Insert as many program logic statements here as necessary	→ (3)	

Рис. 5.7а.

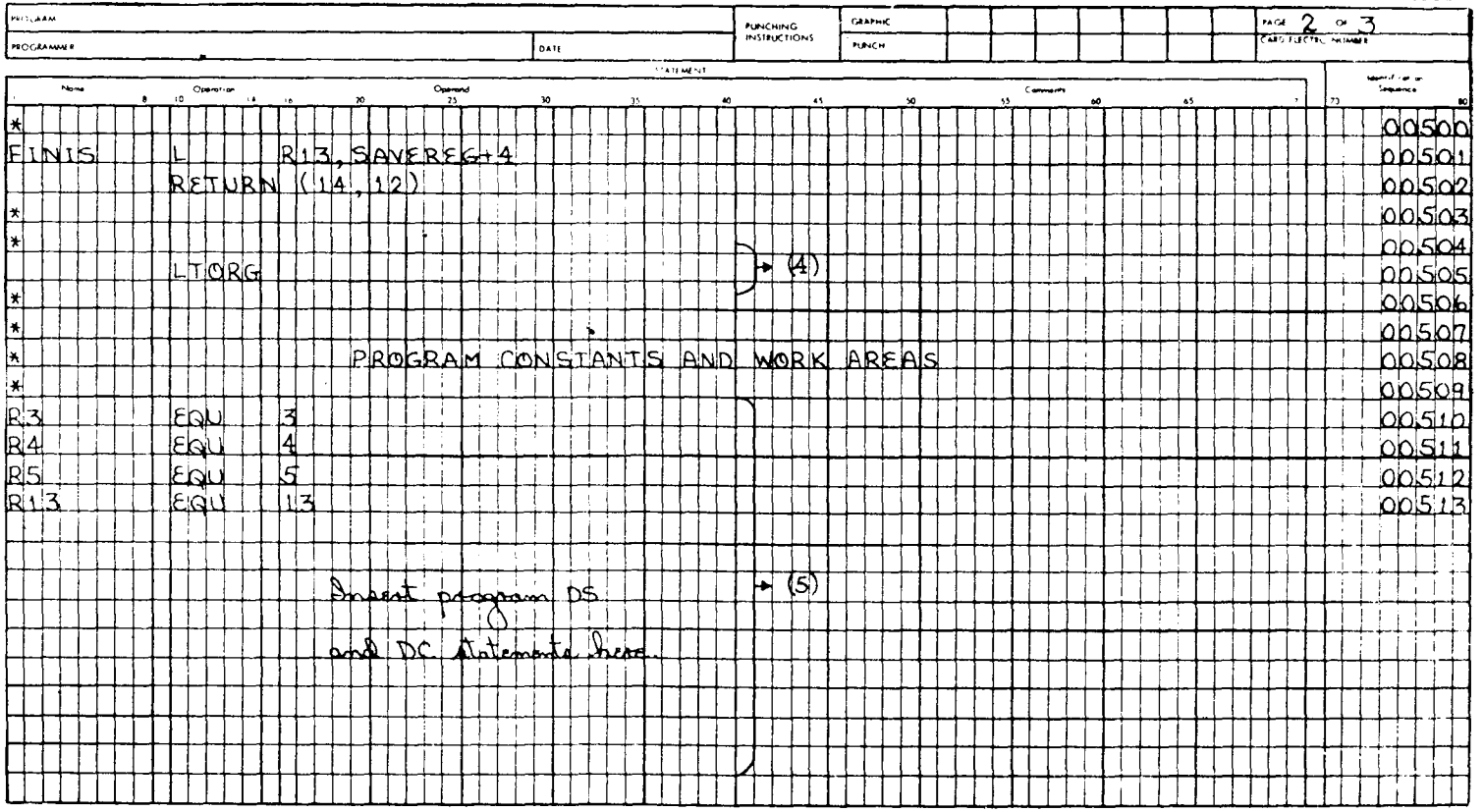
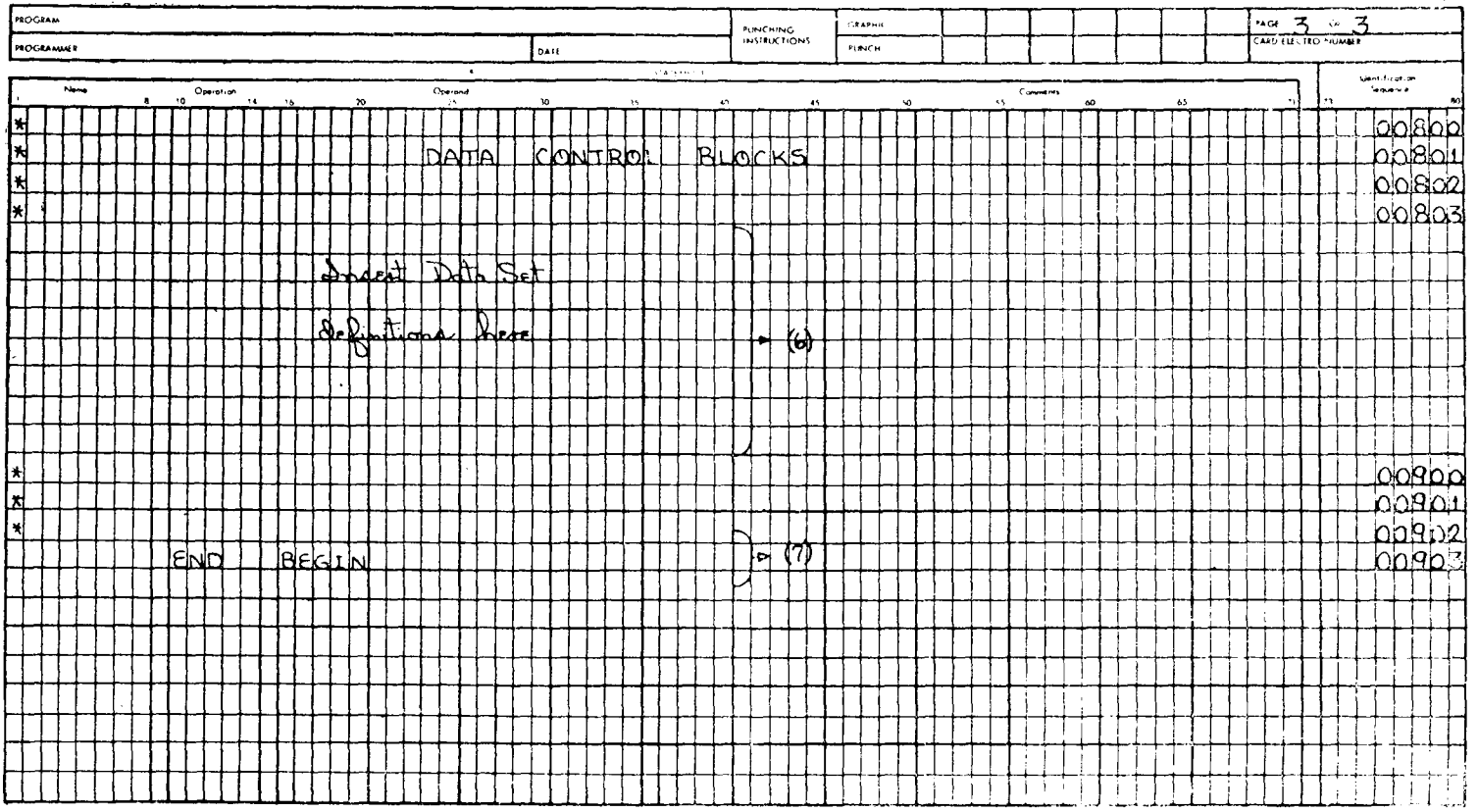

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 2 OF 3
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
*
FINIS L R13,SAVEREG+4
RETURN (14,12)
*
*
LTORG
(4)
*
*
* PROGRAM CONSTANTS AND WORK AREAS
*
R3 EQU 3
R4 EQU 4
R5 EQU 5
R13 EQU 13
Insert program DS
and DC statements here.
(5)
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513

Рис. 5.76.


IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 3 OF 3
CARD ELECTRO NUMBER
Name
Operation
Operand
Comments
Identification Sequence
DATA CONTROL BLOCKS
Insert Data Set
definitions here
(6)
END BEGIN
(7)
00800
00801
00802
00803
00900
00901
00902
00903

Рис, 5.7в.

Группа 2, помеченная START, должна содержать первое предложение отлаживаемой программы.

Группа 3 идентифицирует область, в которой программист может размещать предложения своей программы. Они могут располагаться между предложениями с метками START и FINIS.

Группа 4 представлена предложением literal org (организовать литеральный пул), обеспечивающим соответствующие программные ссылки для литералов, которые программист может включить в текст программы.

Группа 5 содержит несколько предложений «Equate» (приравнять), которые присваивают символические метки нескольким общим регистрам. Обращение к этим регистрам можно произвести, задавая только номер регистра или задавая символическую метку, как это делается в группе 1 предложений. В эту же группу включена область, в которой программисту следует разместить все предложения DS (Define Storage — Определить память) и DC (Define Constant — Определить константу), используемые внутри его программы.

Группа 6 может содержать предложения блоков управления данными для устройств ввода-вывода, которые необходимо использовать в программе. Точная форма этих предложений будет зависеть от типа используемой операционной системы, а также количества и типа устройств ввода-вывода.

Группа 7, предложение END (Конец), представляет заключительный оператор проблемной программы. Коды, записываемые программистом, могут находиться в любой из перечисленных выше групп предложений, но обязательно перед оператором END.

Как компиляция исходного модуля, так и последующая отладка объектной (оттранслированной) программы потребуют применения перфокарт «Языка управления заданиями». Они необходимы для того, чтобы дать задаче имя, разъяснить системе, какие функции необходимо выполнить, какие устройства должны быть использованы, а также сообщить некоторую другую относящуюся к делу информацию, которая потребуется операционной системе для выполнения задачи.

Предложения внутри рассматриваемого набора кодов написаны в соответствии с полной Операционной системой (OS). В случае, если программист будет пользоваться другими операционными системами, ему, возможно, понадобится отчасти изменить некоторые форматы команд.

ЧАСТЬ III

Логические применения языка Ассемблера

Глава 6

Формирование рабочих областей внутри программы

А. РЕЗЕРВИРОВАНИЕ ОБЛАСТЕЙ ПАМЯТИ

В этом разделе рассматриваются поля и рабочие области, используемые проблемной программой для хранения, накопления и обработки данных. Такие области памяти могут применяться для хранения входных или выходных записей, для построения таблиц и индексов, в качестве рабочих областей при выполнении арифметических операций, например для хранения частного и остатка, для сохранения содержимого регистров или для любых других данных, изменяющихся во время выполнения программы.

1. Определение областей

Для резервирования связной области памяти используется предложение Define Storage (Определить память) с кодом операции DS. Используя это предложение, можно зарезервировать область основной памяти для работы проблемной программы. В отличие от предложения Define Constant (DC) (Определить константу) оно не формирует данные определенной конфигурации внутри резервируемой области памяти. До тех пор пока проблемная программа действительно не поместит данные некоторого типа в область, определенную предложением DS, содержимое этой области непредсказуемо, так как данные, записанные в эту область памяти предыдущей проблемной программой, вполне возможно, все еще находятся в ней.

Типичное DS-предложение может выглядеть следующим образом:

```
RECORD    DS    CL300
```

Это предложение присваивает метку RECORD 300 последовательным байтам памяти как части проблемной программы.

Вполне возможно, программист пожелает адресоваться к меньшим объемам памяти внутри больших областей, не используя механизм настройки адресов. Распределение памяти в этом случае может быть задано следующим образом:

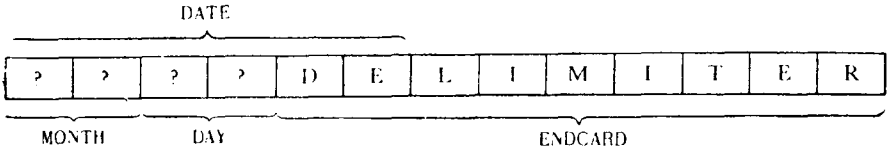
```
DATAFLD    DS    OCL100
NAME       DS    CL20
ADDRESS    DS    CL40
PHONE      DS    CL9
DETAIL     DS    CL31
```

Предложение DATAFLD присваивает эту метку 100 байтам памяти, но в действительности не резервирует эту память. Параметр нуль (0), который стоит перед CL в первом предложении, указывает, что данное предложение не резервирует память. Однако если переслать 100 байтов данных по адресу, определяемому меткой DATAFLD, эти данные будут помещены в области, зарезервированные последующими предложениями DS или DC. Смысл такого набора операторов очевиден — в программе предусматривается размещение 100-байтовой записи в область, адресуемую меткой DATAFLD. Внутри этой записи будут содержаться, например, фамилия, адрес, номер телефона и некоторые дополнительные детальные сведения. Данные, которые должны быть помещены во всю эту область, можно переслать туда обращением к метке DATAFLD. После того как эти данные помещены в память, к ним можно обращаться как к целому с помощью метки DATAFLD или по частям с помощью меток подполей. Заметим, что составная область памяти, резервируемая в действительности предложениями DS всех подполей, равна по длине 100 байтам, что в точности соответствует оператору DATAFLD. Применяя операторы DS таким образом, очень важно следить чтобы общая длина подполей была равна длине области, заданной первой меткой. Несоблюдение этого правила может привести к искажению данных, записанных в память. Например, предложения определения памяти внутри проблемной программы могут выглядеть следующим образом:

```
DATE       DS    OCL6
MONTH      DS    CL2
DAY        DS    CL2
ENDCARD    DC    CL9'DELIMITER'
```

Метка DATE здесь используется для обращения к 6 байтам данных. Общая длина подполей, определенных внутри DATE, равна только 4 байтам. В силу этого любое обращение к DATE

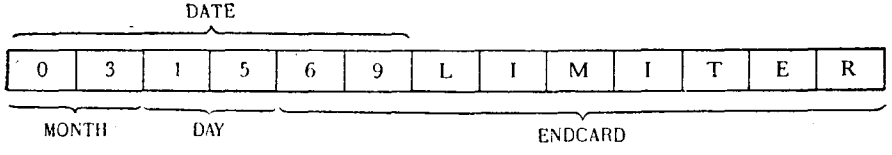
подразумевает обращение к полям MONTH, DAY и первым 2 байтам константы ENDCARD. В момент начала работы программы подполя и метки могут выглядеть следующим образом:



Теперь предположим, что в программе предусмотрена пересылка некоторых данных из другой области, которая в настоящий момент содержит «031569» (15 марта 1969 года), в область, адресуемую меткой DATE. Затем предусмотрено сравнение содержимого входной области для считывания перфокарт с константой ENDCARD, чтобы проверить, содержит ли эта область слово DELIMITER, указывающее на окончание входного потока данных на перфокартах. Команды проблемной программы таковы:

```
MVC     DATE,SOMEAREA       (001)
CLC     CARDIN,ENDCARD      (002)
BC      8,JOBDONE           (003)
```

Предложение 001 пересылает данные из SOMEAREA (031569) в область, адресованную меткой DATE. Содержимое областей DATE и ENDCARD теперь окажется следующим:



Так как длина области DATE была определена равной 6 байтам, все 6 байтов данных оказались пересланными из SOMEAREA в DATE. Последние 2 байта SOMEAREA (69) перекрыли и тем самым разрушили первые 2 байта константы ENDCARD (DE).

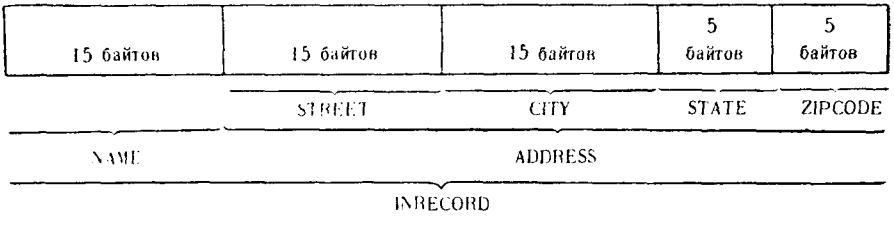
Предложения 002 и 003 осуществляют сравнение содержимого входного поля с содержимым ENDCARD (которым предположительно должно быть слово DELIMITER) и переход к программе окончания работы при выполнении условия равенства. Если входная область (CARDIN) в конце концов и будет содержать слово DELIMITER, сравнение не приведет к условию равенства из-за того, что константа ENDCARD имеет теперь вид 69LIMITER. Соответственно требующийся по логике выпол-

нения программы переход к JOBDONE не произойдет, и программа, по всей вероятности, закончится аварийно.

Параметр 0, указывающий, что память не резервируется, можно также использовать в предложении DS при определении полей внутри заданной ранее области, имеющей метку.

```
INRECORD    DS    0CL55
NAME        DS    CL15
ADDRESS     DS    0CL40
STREET      DS    CL15
CITY        DS    CL15
STATE       DS    CL5
ZIPCODE     DS    CL5
```

Этот набор предложений задает область памяти, предназначенную для хранения входных записей. Общая длина области равна 55 байтам, и в целом к ней можно обращаться по метке INRECORD. Первое подполе внутри INRECORD—поле NAME—состоит из первых 15 байтов INRECORD. Следующая за NAME метка ADDRESS относится к остальным 40 байтам INRECORD. Поля STREET, CITY, STATE и ZIPCODE являются подполями как INRECORD, так и ADDRESS; они резервируют в памяти области, к которым можно обращаться с помощью меток ADDRESS и INRECORD. Взаимосвязь всех этих меток можно представить следующим образом:



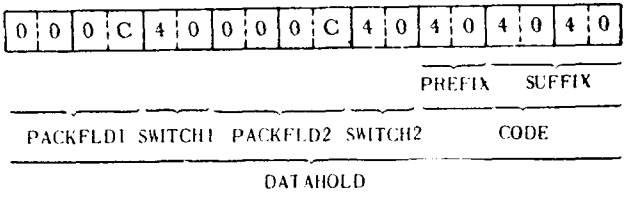
Поля STREET, CITY, STATE и ZIPCODE можно переслать или видоизменить по отдельности, адресуясь к любой из этих меток; их можно также переслать или видоизменить в совокупности, адресуясь к метке ADDRESS. Точно так же можно переслать с помощью метки INRECORD все поле целиком или с помощью других меток какую-либо его часть.

Предложения DS и DC могут чередоваться в точности так же, как это было только что показано для предложений DS. Предложение DS необходимо использовать, если одновременно с определением частей области ей нужно приписать общую

метку, не выделяя память. В качестве примера рассмотрим следующие предложения:

```
DATAHOLD    DS    0CL9
PACKFLD1    DC    PL2'0'
SWITCH1     DC    X'40'
PACKFLD2    DC    PL2'0'
SWITCH2     DC    X'40'
CODE        DS    0CL3
PREFIX      DC    X'40'
SUFFIX      DC    X'4040'
```

Логическая и шестнадцатеричная конфигурация областей, представленных этими операторами, выглядит следующим образом:



Ко всей области из 9 байтов можно обратиться с помощью метки DATAHOLD. Поля PACKFLD1 и PACKFLD2 содержат упакованную десятичную величину +0, что позволяет использовать их для операции десятичной арифметики. Каждая из однобайтовых областей SWITCH1 и SWITCH2 содержит шестнадцатеричную конфигурацию «пробел». Поле CODE — другое подполе DATAHOLD — в свою очередь внутренне доопределяется подполями PREFIX и SUFFIX, которые также содержат коды пробелов.

2. Описание перекрывающихся полей

Иногда программисту удобно определить некоторую область в нескольких форматах или различными подполями. Это можно сделать, используя предложения DS и ORG. Предложение ORG вызывает принудительное присвоение следующему распределяемому байту памяти адреса, заданного единственным операндом команды ORG. Практическим примером совместного использования предложений DS и ORG служит программа считывания данных с перфокарт. Нормальное определение входного поля

обычно достаточно, но в этом примере программа будет считывать карты, информация на которых задана в трех различных форматах. Первый символ на карте указывает, какой из трех форматов применим к данной конкретной карте. Для эффективности работы с отдельными полями данных на карте каждого типа входное поле сначала определяется в целом, а затем полностью переопределяется для каждого из трех различных форматов. Используя метки, применимые к системе учета в торговле, размещение информации на картах можно задать с помощью предложений, приведенных на рис. 6.1а и 6.1б.

Информация с карт, вводимая в память программами ввода-вывода, может быть считана в область, помеченную как CARDIN. К помещенным в CARDIN данным возможно обращение с помощью меток CARDIN, CODE или DATA, а также с помощью любой другой метки, указанной в списке предложений DS. Карта с именем CODE1 содержит данные, относящиеся к спросу. Информация для этой карты будет представлена в формате и с метками, показанными для карты CODE1; с помощью этих меток можно получить доступ к любому полю. Карта CODE2 содержит данные о заказах на покупку товара, а CODE3 — информацию о его получении. После определения типа данных и пересылки их во входную область происходит переход к одной из трех подпрограмм. Каждая из этих подпрограмм обрабатывает данные, представленные на картах соответствующего типа.

Заметим, что в каждом случае формат карт содержит различное количество байтов данных, а остальные байты данных, к которым нет обращений из программы, для каждого формата определены меткой MT (empty — пустой). Такое средство используется для согласования общего количества байтов, относящихся к следующему, более высокому уровню символических обращений. Хотя при определении этих областей формировалось собственное представление для каждого из четырех различных форматов, включая основную входную область, результат всех этих определений задает область памяти общим размером 80 байтов.

Необязательно, чтобы предложение ORG ссылалось на первый символ в поле данных, которое должно быть внутренне доопределено. Если некоторое поле необходимо внутренне доопределить, то к нему можно обращаться по первой метке, с помощью которой оно определено. Эта идея иллюстрируется рис. 6.2, где используется тот же набор доопределений, что и в последнем рассматриваемом наборе предложений DS.

В этом примере каждая из последующих доопределяемых областей может использовать метки, определенные ранее для

IBM

IBM System 360 Assembler Coding Form

X28-6508-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification Sequence
*				0001
CARDIN	DS	0CL80		0002
CODE	DS	CL1		0003
DATA	DS	CL79		0004
*				0005
	ORG	CARDIN		0006
CODE1	DS	CL1		0007
REQSTNNO	DS	CL8		0008
PARTNO1	DS	CL12		0009
REQDATE	DS	CL6		0010
REQQTY	DS	CL8		0011
MT1	DS	CL45		0012
*				0013
	ORG	CARDIN		0014
CODE2	DS	CL1		0015
PURCHNO1	DS	CL7		0016
VENDOR	DS	CL6		0017
PARTNO2	DS	CL12		0018
PODATE	DS	CL6		0019
POQTY	DS	CL8		0020
BUYER	DS	CL8		0021
MT2	DS	CL32		0022
*				0023

Рис. 6.1а.

IBM

IBM System/360 Assembler Coding Form

X28-6509-3 U/M06b
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 2 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT | Identification-Sequence

Name	Operation	Operand	Comments	Identification-Sequence
	ORG	CARDIN		0024
CODE3	DS	CL1		0025
RECVGNO	DS	CL7		0026
PURCHNO2	DS	CL7		0027
PARTNO3	DS	CL12		0028
RECQTY	DS	CL8		0029
BACKORDR	DS	CL4		0030
MT3	DS	CL41		0031
*				0032

Рис. 6.1б.

IBM

IBM System/360 Assembler Coding Form

X28-6509-2 U/2650
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				0001
CARDIN	DS	CCL80		0002
CODE	DS	CL1		0003
DATA	DS	CL79		0004
*				0005
	ORG	DATA		0006
PARTNO	DS	CL12		0007
QTY	DS	CL6		0008
REQSTNNO	DS	CL8		0009
REQDATE	DS	CL6		0010
MT1	DS	CL47		0011
*				0012
	ORG	REQSTNNO		0013
PURCHNO	DS	CL7		0014
VENDOR	DS	CL6		0015
PODATE	DS	CL6		0016
BUYER	DS	CL8		0017
MT2	DS	CL34		0018
*				0019
	ORG	VENDOR		0020
RECVGNO	DS	CL7		0021
BACKORDR	DS	CL4		0022
MT3	DS	CL43		0023
*				0024

Рис. 6.2.

других полей. Перекрытие этих полей показано ниже:

```
CARDIN   CODE     (CODE)      (CODE)      (CODE)
         DATA     PARTNO      (PARTNO)    (PARTNO)
                  QTY         (QTY)       (QTY)
                  REQSTNNO    PURCHNO     (PURCHNO)
                  REQDATE     VENDOR      RECVGNO
                  MT1         PODATE      BACKORDR
                              BUYER       MT3
                              MT2
```

Метки этих ранее определенных полей, указанные внутри скобок, повторяются в последующих предложениях, так как их интерпретация и длина поля идентичны определенным ранее. Такое определение подразумевает «пошаговое снижение», т. е. каждое доопределение выполняется на последующем уровне полей.

3. Выравнивание границ

Области памяти и поля данных, используемые непосредственно в арифметических операциях с фиксированной точкой или при выполнении регистровых команд, должны быть выравнены по границам полуслова, полного слова или двойного слова в зависимости от команды, которая должна быть выполнена. Границы определяются следующим образом:

1. Полуслово — Адрес ячейки памяти делится на 2
2. Полное слово — Адрес ячейки памяти делится на 4
3. Двойное слово — Адрес ячейки памяти делится на 8

Нельзя ожидать, что средний программист может точно знать, каким образом каждая из областей памяти или данных выравнена при компиляции его программы.

Поэтому в распоряжение программиста предоставляются операнды типа H, F и D предложений DS и DC. Эти операнды будут автоматически обеспечивать выравнивание границ по полуслову, полному слову и двойному слову соответственно. Следующие далее примеры иллюстрируют некоторые из способов, которыми эти операнды могут быть выражены в предложении DS:

```
FULLWD   DS   F
```

Это предложение резервирует 4 байта памяти, выравненные по границе полного слова и адресуемые меткой FULLWD:

```
SAVEREG  DS   18F
```

Это предложение присваивает метку SAVEREG последовательности из 18 полных слов памяти, или 72 последовательно адресуемым байтам. Каждые следующие 4 байта этой области будут иметь адрес памяти, который делится на 4. Это предложение представляет область памяти, которая резервируется в каждой проблемной программе как область сохранения содержимого регистров.

```
HAFWDS3   DS    3H
```

Данное предложение определяет область памяти, состоящую из 6 байтов — трех последовательных полуслов. Так как невозможно знать заранее, какое из этих полуслов будет выравнено также и по границе полного слова, попытка использовать два соседних полуслова в качестве поля длиной в полное слово может привести к ошибке:

```
ALIGN       DS    0F
DATABITS    DC    B'01001100'
CHECKBITS   DC    X'HCHC'
NULLBITS    DC    B'00000000'
```

Первое предложение в этой группе служит для выравнивания по границе полного слова сегментов данных, не превышающих по длине полного слова. Предложение ALIGN будет формировать ссылку для этой метки к границе полного слова, ближайшей по отношению к той точке программы, в которой встретился данный оператор, но не будет резервировать память. Однобайтовая область DATABITS, которая следует сразу же за ALIGN, начнется на границе полного слова, определенной меткой ALIGN. Поля CHECKBITS (2 байта) и NULLBITS (1 байт) займут оставшуюся часть области длиной в полное слово. Хотя весь этот набор операторов можно заменить одной константой длиной в полное слово с соответствующей двоичной конфигурацией, изложенный выше способ гораздо более целесообразен. Вдобавок к каждому из трех определенных таким способом подполей можно обращаться непосредственно с помощью их собственных меток.

```
            DS    0D
DUBPACK     DC    PL8'0'
```

Этот набор предложений формирует восьмибайтовое поле, содержащее упакованную десятичную величину +0 и выравненное по границе двойного слова. Это один из способов задания типичной упакованной десятичной области длиной в двойное слово для команд Convert To Binary (CVB) (Преобразование в двоичную) и Convert To Decimal (CVD) (Преобразование в де-

сятичную). Такую область можно определить предложением

DUBPAK DS D

Но при таком способе выделения памяти программист должен использовать команду ZAP для пересылки упакованного десятичного поля в DUBPAK до выполнения команды CVB или другим способом обеспечить в этом поле формат упакованного десятичного числа перед его применением.

Б. ОПРЕДЕЛЕНИЕ КОНСТАНТ И ЛИТЕРАЛОВ

1. Что такое константа?

В соответствии с принятой в программировании терминологией под *константой* понимают часть выражения, которая сохраняет свою конфигурацию до тех пор, пока ее не изменят преднамеренно. Язык Ассемблера подтверждает такое определение. Константа представляет собой такие данные, как величины выражения или символы, определенные в проблемной программе с помощью предложения Define Constant (DC) (Определить константу). К этим данным, после того как они определены, можно обращаться из любой подпрограммы основной программы, не изменяя их исходную конфигурацию до тех пор, пока это не потребуется в соответствии с логикой выполнения программы. Выделение памяти для константы осуществляется таким образом, что ее можно адресовать с помощью символической метки или истинного адреса.

Так как содержимое константы предполагается использовать в некоторой подпрограмме или предложении, то для обращения к ней логично использовать символическую метку. Гораздо легче адресовать константу по ее символической метке, чем определять адрес, по которому она действительно располагается. Следовательно, хотя в предложении DC метка не обязательна, в практике программирования принято «метить» все константы.

Предложение DC записывается следующим образом:

метка DC *данные*

Назначение метки уже было описано ранее. Метка может представлять собой любую разрешенную комбинацию алфавитно-цифровых символов кода EBCDIC длиной от 1 до 8 байтов для OS. Стандартные правила, относящиеся к формированию меток вообще, применимы и к константам: а) первый символ должен быть буквой; б) остальные символы могут быть либо буквами, либо цифрами, либо произвольной комбинацией букв и цифр; в) метка не может содержать пробелов; г) она не может содержать специальных символов.

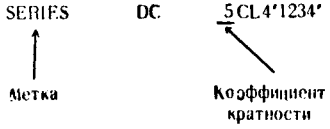
Правильно	*Неправильно*
Символические метки	*Символические метки*
P124689	86523
RTE	GO THERE
ROUTINE2	3FINTS
GOTDATA	DATAFIELD
KX7RC6	NO_GO
M345N346	244MASK

Мнемонический код операции команды «определить константу» DC указывает компилятору с языка Ассемблера на необходимость сформировать и поместить в память константу, определенную в операнде (в части «данные») этой команды, и присвоить адрес этой константы символической метке.

Часть «данные» предложения DC может включать в себя четыре компонента или подполя:

коэффициент кратности;
тип константы;
модификатор длины;
собственно данные.

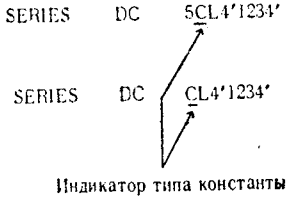
а) Коэффициент кратности определяет количество повторений константы. Если коэффициент кратности не задан, то по умолчанию он принимается равным единице. Поэтому если необходимо сформировать одиночную константу, то коэффициент кратности можно опустить. Для образования константы, которая содержит последовательность символов 12341234123412341234, предложение DC можно записать так:



В этом примере константа SERIES формируется в виде пяти групп по 4 байта, причем каждая группа будет содержать численные символы 1234.

б) Тип константы. Необходимо определить формат, в котором программист хочет представить константу. Тип константы определяется одиночным символом, который сообщает компилятору с языка Ассемблера информацию, необходимую для формирования этой константы в машинном формате. В данном разделе будут детально рассмотрены различные типы констант, такие, как C для символьного формата, B для двоичного, X для шестнадцатеричного и т. д. Если в предложении DC указан коэффициент кратности, индикатор типа константы следует

сразу же за ним; в противном случае тип константы является первым компонентом поля операнда в этом предложении.



в) **Модификатор длины** определяет действительную длину формируемой константы. Если модификатор длины не задан, длина сформированной константы определяется самой константой или подразумевается типом константы. Определенные типы констант подразумевают заранее заданную конкретную длину, например константа типа F подразумевает полное слово, константа типа H — полуслово и т. п. Следующий список предложений DC дает примеры длин, которые подразумеваются или задаются при формировании констант:

CONST1	DC	C'NOW'	неявная длина 3 байта
CONST2	DC	CL5'DATE'	заданная длина 5 байтов
CONST3	DC	F'23968'	неявная длина 4 байта
CONST4	DC	H'1'	неявная длина 2 байта

Формат и способы выравнивания данных для констант различного типа будут рассмотрены при изучении каждого типа констант. Имеются четыре типа модификаторов для явного задания длины.

1. Модификатор длины константы в байтах.
2. Модификатор длины в битах.
3. Масштабный модификатор для величин с фиксированной точкой.
4. Экспонентный модификатор для величин с фиксированной точкой.

Благодаря более частым применениям в программировании на языке Ассемблера в данном разделе будут рассмотрены и проиллюстрированы на примерах модификаторы типов 1 и 2.

Тип 1 (модификатор длины константы в байтах) используется для того, чтобы установить действительное количество байтов, которые необходимо выделить константе независимо от ее явной или неявной длины. Он записывается как Ln, где n — десятичная величина, представляющая количество байтов.

Тип 2 (модификатор длины константы в битах) устанавливает действительное количество битов, которые должны быть

выделены константе. Он записывается как L.n, где n — десятичная величина, указывающая количество двоичных позиций, которые необходимо выделить константе. Этот вид модификатора длины обычно используется только с константами типа B, F, H или D. Если значение константы не соответствует длине, указанной модификатором длины в битах, то к значению константы будет добавлено количество дополнительных нулей в старших битах, достаточное для заполнения поля указанной в модификаторе длины.

Важно знать и помнить, что выравнивание границ не выполняется, когда в константе с фиксированной точкой длиной в полуслово, полное слово или двойное слово используется модификатор длины.

г) Собственно данные. Данные, которые собственно и составляют содержимое константы, являются последним компонентом операнда в предложении DC. Для констант типа A, S, V или Y собственно данные заключаются в круглые скобки; во всех других константах они заключены в апострофы. Хотя в данном разделе будет приведено много примеров констант всех типов, приведем несколько примеров, показывающих связь между компонентом константы и способом ее записи:

```
1. CHARCONS  DC  CL6'ABCDEF'  (Символьная)
2. BINCON    DC  B'10110001'  (Двоичная)
3. FIXDCON   DC  4F'398'      (С фиксированной точкой)
4. HEXCON    DC  XL3'DF236C'  (Шестнадцатеричная)
5. PACKCON   DC  PL4'253774'  (Упакованная десятичная)
6. ADCONST   DC  A(15975)     (Адресная константа)
```

1. CHARCONS — это символьная константа в 6 байтов, содержащая символы ABCDEF. Модификаторы определяют следующее:

C — Символьный тип константы
L6 — Явная длина, равная 6 байтам
'ABCDEF' — Действительное содержимое 6 байтов

После компиляции эту константу можно интерпретировать в одном из следующих форматов:

Символьн.	A		B		C		D		E		F	
Шестн.	C	1	C	2	C	3	C	4	C	5	C	6
Двоичн.	1100	0001	1100	0010	1100	0011	1100	0100	1100	0101	1100	0110

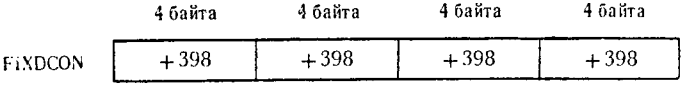
2. BINCON — однобайтовая двоичная константа. Так как указатель длины не был задан, то длина константы подразумевается равной восьми двоичным позициям, или 1 байту. Модификаторы, определяющие эту константу, выглядят так:

B — Двоичная константа
'10110001' — Двоичная конфигурация константы.
8 битов заданы без модификатора длины;
константа компилируется как 1 байт.

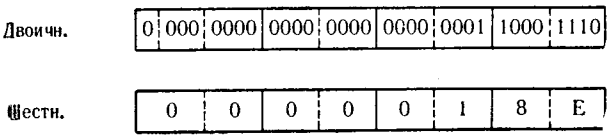
3. FIXDCON — константа с фиксированной точкой длиной в полное слово. Модификаторы интерпретируются следующим образом:

4 — Коэффициент кратности, равный 4; какая бы одиночная константа ни была задана, она окажется сформированной четыре раза подряд в непрерывном поле памяти.
F — Константа с фиксированной точкой длиной в полное слово. Так как модификатор длины не задан явно, константа будет состоять из 4 байтов, выравненных по границе полного слова.
398 — Величина с фиксированной точкой равна +398.

В силу того что для константы, состоящей из 4 байтов, задан коэффициент кратности, равный 4, суммарная область памяти, отводимая под FIXDCON, будет иметь длину 16 байтов:

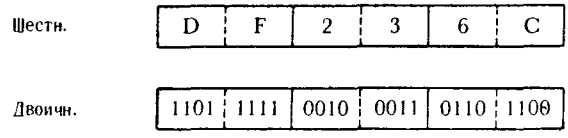


Первые 4 байта FIXDCON и последующие четырехбайтовые приращения выглядят следующим образом:



4. HEXCON — трехбайтовая шестнадцатеричная константа. Компилятор будет интерпретировать ее компоненты так:

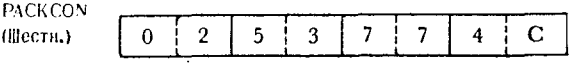
X — Шестнадцатеричная константа.
L3 — Указатель длины в байтах, равный 3.
DF236C — Шестнадцатеричные цифры, составляющие данные константы.



5. PACKCON — четырехбайтовая упакованная десятичная константа, содержащая величину +253774. Компоненты интерпретируются таким образом:

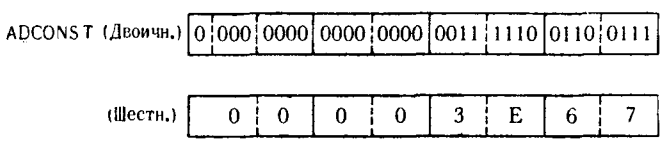
P — Упакованная десятичная константа.
L4 — Указатель длины в байтах, равный 4.
253774 — Упакованная десятичная величина, равная +253774, которая должна занимать четырехбайтовую область константы.

Так как эта величина занимает не всю длину, указанную для константы, старший полубайт заполняется шестнадцатеричным нулем:



6. ADCONST — адресная константа. Отсутствие определенного указателя длины подразумевает 4 байта (полное слово). Компоненты интерпретируются следующим образом:

A — Адресная константа длиной в 4 байта.
(15975) — Адрес, содержащийся в полном слове ADCONST, представленный как величина с фиксированной точкой.



Более детальное и тщательное изучение констант различного типа будет проведено в процессе обсуждения каждой из их разновидностей.

д) **Литеральный вариант константы.** Обычная интерпретация константы заключается в том, что конкретной конфигурации

данных или величине присваивается метка, обеспечивающая доступ к ней из любой точки проблемной программы. Однако литерал рассматривается как «разовая» константа, иными словами, как буквальная интерпретация данных или величины, выраженной в предложении. Один и тот же литерал можно использовать внутри программы любое количество раз, но всякий раз он должен быть полностью задан заново, причем к нему нельзя обращаться ни из какой другой точки программы.

Литерал записывается точно в таком же формате, как и операнд в константе, за исключением того, что перед ним должен стоять знак равенства (=). Ниже приведены другие правила, которые относятся к использованию литерала:

1. В литерале нельзя задавать больше одного операнда.
2. Нельзя указывать коэффициент кратности, равный нулю.
3. Коэффициенты кратности и модификаторы длины должны выражаться как самоопределенные десятичные величины без знака.
4. Литерал не может задавать адресную константу типа S.

Два набора команд, представленных на рис. 6.3 и 6.4, отражают различие между использованием констант и литералов. В первом наборе PACKS определена как четырехбайтовая упакованная десятичная константа со значением, равным +31649. Для того чтобы впоследствии обратиться к этой величине, необходимо только написать метку этой константы, с помощью которой будет сформирован ее адрес.

Во втором наборе весь литерал повторяется всякий раз при его использовании. Довольно очевидно, что применение обычных констант может ускорить и сделать более эффективным кодирование программы, а также уменьшает вероятность появления ошибки при повторном написании величины. В процессе трансляции литерала компилятор с языка Ассемблера не сможет сообщить, верно или неверно была записана величина, кроме как с точки зрения правильности общего формата. Однако если в наименовании константы была допущена ошибка и это искаженное имя не совпадает с каким-нибудь правильным именем в программе, компилятор укажет на наличие неопределенного символа.

Если некоторая величина или символьные данные используются в программе только один или два раза, то практичнее применять литерал вместо константы.

е) Выражение констант. Способы выражения констант и их интерпретация компилятором в значительной степени зависят от типа формируемой константы. В данном разделе будут обсуждаться альтернативы и требования, которые необходимо

IBM

IBM System/360 Assembler Coding Form

X28-6509-3 U/M090
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*		USING A CONSTANT		
*				
PACKS	DC	PL4'31649'		
*				
ROUTE1	- -	- - - - - -		
	AP	BIGFIELD,PACKS		
	- -	- - - - - -		
	- -	- - - - - -		
	SP	PACKTWO,PACKS		
	- -	- - - - - -		
	MP	PACKANSR,PACKS		
	- -	- - - - - -		

Рис. 6.3.

IBM

IBM System/360 Assembler Coding Form

X28-6809-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*		USING A LITERAL		
*				
ROUTE1				
	AP	SIGFIELD,=PL4'31649'		
	SP	PACKTWO,=PL4'31649'		
	MP	PACKANSR,=PL4'31649'		

Рис. 6.4.

принимать во внимание при использовании 12 наиболее распространенных типов констант. Они представлены в табл. 6.1.

Таблица 6.1

Код типа	Тип константы	Общее применение
C	Символьная	Для представления символов EBCDIC как алфавитно-цифровых, так и специальных
X	Шестнадцатеричная	Для выражения двоичного формата, шестнадцатеричной конфигурации, величин с фиксированной точкой или упакованных десятичных величин
B	Двоичная	Для указания поразрядного двоичного формата константы
F	Полное слово	Для определения четырехбайтовой величины с фиксированной точкой, выравненной по границе полного слова (выравнивание не производится при задании модификатора длины)
H	Полуслово	Для определения двухбайтовой величины с фиксированной точкой, выравненной по границе полуслова (если не задан модификатор длины)
D	Двойное слово	Для определения восьмибайтовой величины с плавающей точкой. Эта величина выравнивается по границе двойного слова, если не задан модификатор длины
P	Упакованная десятичная	Для выражения упакованной десятичной величины
Z	Зонная десятичная	Для выражения десятичной величины в зонном формате, в котором каждая цифра занимает 1 байт, причем крайний правый байт содержит знак поля
A	Адресная	Для определения адресной константы в формате полного слова, если не задана другая длина
Y	Адресная	Для определения адресной константы в формате полуслова, если не задана другая длина
S	Адресная	Для определения адресной константы, выраженной в форме «база — смещение» с максимальной длиной в полуслово
V	Адресная	Определяет область памяти, резервируемую для адресной константы внешнего символа. Хотя задание такой константы обычно приводит к формированию четырехбайтовой области (полное слово), можно указать меньшую длину

Каждый тип константы имеет специализированное применение, для которого он первоначально был разработан. Однако они никоим образом не ограничены рамками только данного конкретного применения. Часто программисты по своему усмотрению определяют тип константы, используемый для заданного применения, хотя необходимо отметить, что этот выбор должен

основываться на разумной логике и ясности определения. Например, если логика выполнения программы требует поразрядной двоичной конфигурации, то константу можно без труда определить как двоичную, шестнадцатеричную или даже как константу длиной в полуслово или в полное слово. Из всех этих возможностей шестнадцатеричный формат чаще всего оказывается наиболее предпочтительным: он компактен в написании и в то же время позволяет легко распознавать поразрядную конфигурацию.

По мере описания различных типов констант попытаемся рассмотреть все случаи их практического применения.

2. Константа типа C (символьная)

Символьная константа является своего рода «рабочей лошадкой» большинства коммерческих программ. Данные в сообщениях, выходные поля для вывода данных на печать, ввод данных, аргументы для поиска — в этих и во многих других применениях в равной мере используют константы этого типа. Символьная константа может иметь указатель длины, величина которого лежит в диапазоне от 1 до 256 байтов. Однако способ выражения константы может ограничить ее длину до величины, меньшей чем 256 байтов, — факт, который в большинстве книг не отмечается явно. Если программист попытается написать на бланке кодирования предложение DC, в котором сама константа физически занимает 256 позиций, компилятор пометит две последние строки константы как ошибочные. Это происходит потому, что максимальное количество строк продолжения, допустимых для обычного оператора команды, равно двум, что вместе с основной строкой обеспечивает максимум 165 байтов, из которых 1 байт отводится на определение типа константы и по 1 байту — на открывающий и закрывающий апострофы (см. рис. 6.5). Если в действительности необходимо сформировать нетривиальную константу с длиной, большей чем 165 байтов, можно сформировать две константы меньшего размера. Для того чтобы совместно использовать данные, содержащиеся в обеих константах, достаточно адресоваться только к метке первой константы и установить указатель длины, равный сумме длин обеих констант.

После определения символьной константы каждый ее символ будет скомпилирован как однобайтовый символ EBCDIC. Если модификатор длины не указан, то фактическая длина константы будет определяться количеством символов, которые составляют константу. Если модификатор длины задан и его значение отличается от количества символов, составляющих константу, то произойдет одно из двух:

IBM

IBM System/360 Assembler Coding Form

X28-6606-2 U/M065
Printed in U.S.A.

PROGRAM		PUNCHING INSTRUCTIONS	GRAPHIC								PAGE OF
PROGRAMMER	DATE		PUNCH								CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
BIGFIELD	DC	C'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890987654321ZYXWVUTSR*		
		QPONMLKJIHGFEDCBABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890987654*		
		321ZYXWVUTSRQPONMLKJIHGFEDCBABCDEFGHIJKLMNOPQRSTUVWXY'		

Рис. 6.5.

1. Если заданный указатель длины меньше количества символов в константе, то все правые байты константы, выходящие за пределы заданной длины, будут опущены.

2. Если заданный указатель длины больше количества символов в константе, то недостающие справа от нее байты будут заполнены пробелами (X'40').

Исключение из правила «один символ соответствует одному байту» происходит, когда в качестве части самой константы в нее включается коммерческое И (&) или апостроф ('). Для того чтобы включить любой из этих символов в константу, его необходимо последовательно повторить дважды — это будет выглядеть как && или (''). Однако при рассмотрении длины константы эти два символа считаются за один.

В следующих ниже примерах предложение DC дается вместе с результирующим полем константы, которое оно формирует. Две соседние вертикальные черты во всех показанных полях выделяют однобайтовые приращения.

Пример 1.

```
                DATA1    DC      C'RECORD'
DATA1           | R | E | C | O | R | D |
```

Модификатор длины не задан; следовательно, длина определяется количеством символов в выражении собственно константы.

Пример 2.

```
                DATA2    DC      CL7'DEVICES'
DATA2           | D | E | V | I | C | E | S |
```

Модификатор длины и количество байтов константы совпадают; следовательно, сформированная константа совпадает с заданной.

Пример 3.

```
                DATA3    DC      CL6'123456789'
DATA3           | 1 | 2 | 3 | 4 | 5 | 6 |
```

Хотя выражение для константы содержало девять символов, модификатор длины L6 ограничил длину скомпилированной константы. Три «лишних» символа 789 были проигнорированы.

Пример 4.

```
DATA4    DC      CL10'SERIAL#'
```

DATA4	S	E	R	I	A	L	#	␣	␣	␣

Скомпилированная константа в 10 байтов была сформирована в результате задания модификатора длины L10. Так как выражение константы не содержит достаточного количества символов, чтобы заполнить 10 байтов, правые 3 байта определяемой области заполняются кодами пробела. (Символ ␣ используется для изображения пробела.)

Пример 5.

```
DATA5    DC      C'TO&&FRO'
```

DATA5	T	O	&	F	R	O

В данном примере иллюстрируется случай использования в константе символов «коммерческое И» или апостроф. Для того чтобы компилятор сформировал одиночный байт, содержащий коммерческое И, необходимо в предложении DC задать последовательность из двух этих символов. Несмотря на то что выражение содержит семь символов, сформированная константа состоит только из 6 байтов.

Пример 6.

```
DATA6    DC      5CL2'A'
```

DATA6	A	␣	A	␣	A	␣	A	␣	A	␣

В этом примере однобайтовое выражение было использовано для формирования двухбайтовой одиночной константы, в результате чего в правый байт этой пары помещен символ пробела. Так как коэффициент кратности равен 5, общая длина поля составляет 10 байтов.

Пример 7.

```
DATA7    DC      CL250'␣'
```

Эта константа компилируется как 250-байтовая область, причем каждый байт содержит код пробела. Первый байт этой области содержит пробел из-за того, что символ пробела задан в выражении самой константы; остальные 249 байтов заполняются пробелами в силу того, что модификатор длины L250 превышает количество символов, заданное в выражении константы.

Литеральный вариант символьной константы записывается так же, как и поле операнда константы, но перед ним ставится знак равенства. Литералы в предложениях записываются так:

```
MVC   DATARECD(4),=CL4'1222'
CLC   INPUT(5),=C'MERGE'
MVC   TABLE(20),=10CL2'DF'
```

3. Константа типа X (шестнадцатеричная)

Шестнадцатеричные константы находят широкое применение при формировании используемых в программе полей данных или величин. Их удобно использовать для задания конфигураций масок, величин с фиксированной точкой различной длины и для формирования упакованных десятичных величин и полей.

Шестнадцатеричная константа может иметь общую длину от 1 до 256 байтов, соответствующую диапазону от 2 до 512 шестнадцатеричных цифр. Для формирования константы можно использовать любую из допустимых шестнадцатеричных цифр — от 0 до 9 и от A до F.

Определение длины сформированной X-константы несколько отличается от соответствующего определения для константы типа C. Если задан модификатор длины, константа формируется в соответствии с указанной длиной. Если же модификатор длины не задан, длина константы будет определяться из шестнадцатеричного выражения в соответствии с одним из приведенных ниже правил:

1. Если количество заданных в предложении DC шестнадцатеричных цифр четно, длина константы в байтах будет равна половине этого количества, так как на каждый байт приходится по две шестнадцатеричные цифры.

2. Если количество заданных в предложении DC шестнадцатеричных цифр нечетно, то длина константы в байтах будет на единицу больше целой части, полученной от деления этого количества на 2. Левый (старший) полубайт будет содержать шестнадцатеричный нуль. Например, 15 шестнадцатеричных цифр скомпилируются в восьмибайтовую константу ($15 \div 2 = 7 + 1 = 8$).

Если заданный модификатор длины не согласуется с количеством шестнадцатеричных цифр в выражении для константы, имеет место одно из двух:

1. Если модификатор длины задает бóльшую длину, чем требуется для размещения заданных шестнадцатеричных цифр,

старшие полубайты поля заполняются шестнадцатеричными нулями.

2. Если модификатор длины задает меньшую длину, чем требуется для размещения всех шестнадцатеричных цифр константы, все старшие цифры, не поместившиеся в поле, заданное модификатором длины, игнорируются.

В следующих примерах шестнадцатеричных констант сформированные поля представлены в полубайтовых приращениях. В зависимости от назначения их можно также интерпретировать в символьном или двоичном форматах.

Пример 1.

```
                HEXCON1     DC      X'39FD1E'

HEXCON1 |  3  |  9  |  F  |  D  |  1  |  E  |
```

В этом примере длина сформированной константы определяется количеством шестнадцатеричных цифр в выражении константы. Модификатор длины не задан, поэтому здесь нет ни подтверждения этой длины, ни конфликта между явным и неявным указателями длины.

Пример 2.

```
            HEXCON2     DC      XL5'069F41'

HEXCON2 |  0  |  0  |  0  |  0  |  0  |  6  |  9  |  F  |  4  |  1  |
```

Заданный модификатор определил длину сформированной константы, равной 5 байтам (L5). Так как имеющееся количество шестнадцатеричных цифр недостаточно для заполнения этой длины, в старшие незаполненные полубайты помещены шестнадцатеричные нули.

Пример 3.

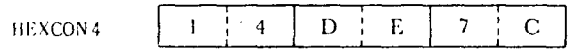
```
                HEXCON3     DC      X'FFFE1'

HEXCON3 |  0  |  F  |  F  |  F  |  E  |  1  |
```

Здесь демонстрируется правило формирования длины константы, содержащей нечетное число шестнадцатеричных цифр. Целое от деления 5 (число цифр константы) на 2 равно 2, при-

бавляем 1, что в итоге дает длину, равную 3 байтам. В дополнительный старший полубайт, который потребовался для доукомплектования поля константы до целого числа байтов, помещен шестнадцатеричный нуль.

Пример 4.

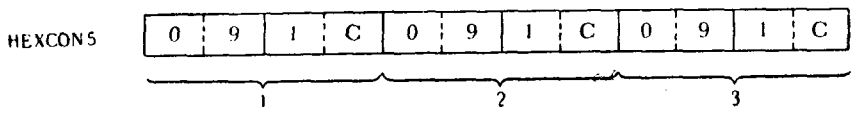
```
HEXCON4   DC    XL3'06914DE7C'
```



Модификатор длины в этом предложении DC задает общую длину сформированной константы равной 3 байтам. Так как в выражении для константы содержится девять цифр, три старшие из них компилятором отбрасываются.

Пример 5.

```
HEXCON5   DC    3XL2'91C'
```



Модификатор задает длину поля, равную 2 байтам, но в выражении для константы содержатся только три цифры, поэтому в старший полубайт двухбайтовой константы помещается шестнадцатеричный нуль. Так как коэффициент кратности равен 3, эта двухбайтовая константа формируется трижды, образуя в итоге область памяти длиной в 6 байтов.

Пример 6.

```
HEXCON6   DC    XL4'C'
```

В этом примере правильное четырехбайтовое упакованное десятичное поле, содержащее величину +0, формируется путем задания константы типа X. Шестнадцатеричная цифра C (стандартный знак плюс для упакованных десятичных величин) помещена в младший полубайт, а остальные старшие полубайты заполнены шестнадцатеричными нулями.

Шестнадцатеричный литерал можно записать точно таким же образом, как операнд шестнадцатеричной константы, за исключением того, что по правилам написания литералов перед ними

необходимо помещать знак равенства. Следующие операторы команд используют шестнадцатеричные литералы:

```
MVC     CHEXFLD(3),=X'050505'
MVC     PRTLINE(1),=X'40'
OC      SOURCE(4),=XL4'FEFEFEFE'
CLC     DATAFLD(2),=XL2'C1C2'
```

4. Константа типа B (двоичная)

Двоичная константа используется для выражения постоянных данных в виде комбинации битов. Это означает, что константа должна быть выражена только с помощью 1 или 0. Минимальная длина двоичной константы равна 1 байту, максимальная — 256 байтам. Опыт показывает, что для целей кодирования обычно удобнее использовать не двоичные, а шестнадцатеричные константы. Существование двоичных констант оправдывает только присущая им способность определять малые группы битов.

Неявная длина двоичной константы выражается в байтах. Если в выражении для константы содержится не кратное 8 число битов, то скомпилированная константа будет дополнена в своих старших битах нулями до целого числа байтов.

Если длина константы в байтах не согласуется с количеством битов в выражении для данных, имеет место одно из двух:

1. Если заданная длина константы больше требуемой для размещения битов константы, старшие неопределенные биты будут заполнены двоичными нулями.
2. Если длина меньше требуемой для размещения битов константы, для согласования с указанной длиной необходимое количество старших битов будет опущено.

Следующие примеры двоичных констант иллюстрируют связь перечисленных возможностей с соответствующими конфигурациями полей. Сформированные поля показаны только в поразрядном двоичном формате.

Пример 1.

```
BINCON1     DC      B'0110000010100101'

BINCON1     0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 1
```

Неявная длина этой константы в точности равна 2 байтам, или 16 битам,

Пример 2.

```
                BINCON2    DC      BL2'1101100101'

BINCON2         0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1
                Дополнение нулями  Заданная константа
```

Модификатор длины этой константы L2 задает длину 16 битов, в то время как в выражении константы задано только 10 битов. Неопределенные шесть старших двоичных позиций заполнены нулями.

Пример 3.

```
                BINCON3    DC      BL1'01101010110'

BINCON3         0 1 0 1 0 1 1 0
```

Указатель длины этой константы равен 1 байту (8 битам), в то время как в выражении для константы содержится 11 битов. Три старших бита (011) в процессе компиляции константы игнорируются.

Литеральный вариант двоичной константы выражается точно таким же способом, как и операнд предложения DC. Как и в случае всех литералов, перед ним должен быть помещен знак равенства. Все правила, относящиеся к двоичной константе, применимы к двоичному литералу. Литералы этого типа могут иметь следующий вид:

```
CLC     HOLDBYTE(1),=BL1'1101'
MVC     INSERT(2),=B'1111000000001111'
IC      6,=BL1'10110011'
```

5. Константа типа F (с фиксированной точкой длиной в полное слово)

Константа типа F используется для формирования четырехбайтовой величины с фиксированной точкой, которая выражена в предложении DC с помощью десятичного числа. Если модификатор длины не задан, выполняется выравнивание по границе полного слова. В случае когда модификатор длины задан, выравнивание границ не производится.

Хотя имеется много разновидностей констант длиной в полное слово (F), полуслово (H) и двойное слово (D), здесь они обсуждаться не будут. Автор не считает, что знание констант, включая масштабирование, модификаторы масштабов, экспоненты, модификаторы экспонент, неявные или заданные пере-

менные дробные части, является необходимым для изучения программирования на языке Ассемблера в рамках данного руководства. Любые дополнительные сведения подобного рода легко почерпнуть в технических руководствах, поставляемых фирмами-разработчиками вычислительных машин.

Константу типа F можно задать в виде десятичного числа со знаком или без знака. Если знак величины не задан, она считается положительной.

Как упоминалось, в константе типа F можно задать модификатор длины, максимальный диапазон изменения которой 8 байтов. Формирование константы типа F с модификатором длины в 8 байтов может показаться противоестественным, однако это позволяет программисту образовывать величину с фиксированной точкой в поле с длиной, большей 4 байтов, без предварительного резервирования области в двойное слово и загрузки в нее величины с фиксированной точкой.

В приведенных ниже примерах формируются константы длиной в полное слово, выравненные по границе полных слов, как для положительных, так и для отрицательных величин. Десятичная величина в выражении константы преобразуется в величину с фиксированной точкой, которая представлена как в двоичном, так и в шестнадцатеричном формате.

Пример 1.

```
FULLWD1   DC    F'3295'
```

S							
FULLWD1 (Двоичн.)	0000	0000	0000	0000	0000	1100	1101	1111
(Шестн.)	0	0	0	0	0	C	D	F

Пример 2.

```
FULLWD2   DC    F' - 659327'
```

S							
FULLWD2 (Двоичн.)	1111	1111	1111	0101	1110	1000	1011	0001
(Шестн.)	F	F	F	5	E	8	B	1

Пример 3.

```
FULLWD3   DC    F'12593446'
```

S							
FULLWD3 (Двоичн.)	0000	0000	1100	0000	0010	1001	0010	0110
(Шестн.)	0	0	C	0	2	9	2	6

В каждом из предшествующих примеров шестнадцатеричная конфигурация результирующей константы приведена для облег-

чения преобразования значения с фиксированной точкой в соответствующий десятичный эквивалент с помощью таблицы шестнадцатерично-десятичных преобразований.

Литералы длиной в полное слово пишутся по тем же самым правилам, что и литералы других типов; впереди ставится знак равенства, а содержимое совпадает с операндом константы.

```
A     10,=F'675021'
L     6,=F'32596'
LM    6,8,=3F'4096'
```

В последнем операторе Load Multiple (Загрузка групповая) литерал будет формировать последовательность из трех полных слов, каждое из которых содержит величину с фиксированной точкой +4096. Первое полное слово будет загружаться в общий регистр 6, второе — в общий регистр 7, третье — в общий регистр 8.

Максимальные значения, которые можно выразить в четырехбайтовой константе типа F, равны +2147483647 и —2147483648.

6. Контанта типа H (с фиксированной точкой длиной в полуслово)

Исключая из рассмотрения те же самые разновидности, что и для констант типа F, можно сказать, что константа типа H обычно используется для формирования двухбайтовых величин с фиксированной точкой, выравненных по границе полуслова. При задании модификатора длины выравнивание границ не производится. Константа типа H может содержать модификатор длины до 8 байтов.

Константу можно задать как величину со знаком или без знака; в последнем случае предполагается, что величина положительная. Максимальные значения, которые могут выражаться константой типа H, равны 32767 и —32768.

В следующих примерах формируются константы длиной в полуслово, выравненные по двухбайтовой границе и представляющие собой величины с фиксированной точкой.

Величины иллюстрируются как в двоичном, так и в шестнадцатеричном представлении; последнее облегчает преобразование константы в ее десятичный эквивалент.

Пример 1.

```
                HAFCON 1    DC      H'15987'

HAFCON 1 (Двоичн.)  0011 1110 0111 0011
         (Шестн.)     3    E    7    3
```

Пример 2.

```
HAFCON2    DC    H'-896'
```

HAFCON2	(Двоичн.)	1111	1100	1000	0000
	(Шестн.)	F	C	8	0

Пример 3.

```
HAFCON3    DC    3H'10775'
```

HAFCON3	(Двоичн.)	0010	1010	0001	0111	0010	1010	0001	0111	0010	1010	0001	0111
	(Шестн.)	2	A	1	7	2	A	1	7	2	A	1	7
		1				2				3			

Перед литералом длиной в полуслово помещается знак равенства, как показано в следующих операторах команд:

```
AH    5,=H'24'
LH    10,=H'-13006'
MH    6,=H'256'
```

7. Константа типа D (двойное слово)

Хотя вообще считается, что константы длиной в двойное слово относятся к арифметическим операциям с плавающей точкой, они очень широко используются в обычных применениях с фиксированной точкой. Так как в коммерческих программах, написанных на языке Ассемблера, величины, превышающие по своему объему полное слово, встречаются весьма редко, можно считать, что в основном константы длиной в двойное слово используются для создания восьмибайтовой области, заполненной нулями.

Например, при выполнении программы может возникнуть необходимость формирования двух отдельных величин длиной в полное слово каждая в пределах определенной константы длиной в двойное слово с последующим размещением этого двойного слова, содержащего, по предположению, две величины, в два последовательных регистра с помощью команды LM. Такое поле, содержащее первоначально нулевую величину, можно создать, используя следующее предложение:

```
DUBCON    DC    D'0'
```

Два полных слова этого поля также будут выравнены по границе полного слова. В любую часть этого поля можно поме-

стить данные или величины из общих регистров или, наоборот, из любой части поля загрузить их в общие регистры, можно производить манипуляции внутри любой части этого поля, используя настройку адреса или указывая длину (там, где это является необходимым) в операторе команды. Это может выглядеть следующим образом:

```
L       8,DUBCON+4
STM     5,6,DUBCON
CVB     9,DUBCON
MVI     DUBCON+7,X'00'
```

8. Константа типа P (упакованная десятичная)

По всей вероятности, упакованная десятичная константа является вторым наиболее широко используемым типом констант в коммерческом программировании на языке Ассемблера. Форма ее выражения вполне аналогична шестнадцатеричной константе в том отношении, что каждая цифра в ней занимает после компиляции половину байта. Знак упакованной десятичной константы можно выразить явно или опустить; в последнем случае константа считается положительной величиной. Когда компилятор формирует упакованную десятичную константу, знак помещается правее самой правой цифры собственно константы и вместе с ней заполняет самый правый байт поля. Остальные байты скомпилированной константы заполняются остальными парами цифр.

Если количество цифр в величине, выражаемой константой, не соответствует в точности ее неявно заданной длине, то в старший полубайт поля помещается шестнадцатеричный нуль.

Если модификатор задает длину, не согласующуюся с количеством десятичных цифр в выражении для константы, имеет место одно из двух:

1. Если длина больше требуемой для размещения величины, выраженной константой, избыточные старшие полубайты заполняются шестнадцатеричными нулями.
2. Если длина меньше требуемой для размещения величины, выраженной константой, старшие избыточные цифры этой величины игнорируются.

Длина упакованной десятичной константы может быть от 1 до 16 байтов. Для более наглядного представления данных внутри десятичной величины может быть помещена десятичная точка. Она полностью игнорируется компилятором, и константа компилируется таким образом, как будто в предложении DC никакой десятичной точки нет. Основная идея ее применения

заключается только в том, чтобы помочь программисту интерпретировать величину константы. За исключением знака и десятичной точки, содержимое константы должно задаваться в десятичной форме.

Следующие примеры упакованных десятичных констант содержат предложения DC и шестнадцатеричное представление (в полубайтах) полей данных, которые должны формироваться компилятором.

Пример 1.

```
                PAKCON1   DC    P'32'
PAKCON1 | 0 | 3 | 2 | C |
          ---
Заполнение нулями
```

Несмотря на то что собственно константа содержит только две цифры, необходимо сформировать константу как двухбайтовое поле, так как в правый полубайт скомпилированной константы должен быть помещен подразумеваемый знак. Когда длина была принудительно установлена равной 2 байтам, для заполнения поля не оказалось достаточного количества цифр. Соответственно в свободный старший полубайт помещен нуль.

Пример 2.

```
                PAKCON2   DC    PL6'32964'
PAKCON2 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 9 | 6 | 4 | C |
          -------------------
Заполнение нулями
```

В соответствии со значением модификатора длины предложения DC компилятор формирует шестибайтовое упакованное десятичное поле константы. Выражение константы не обеспечивает заполнения этих байтов, так что дополнительные старшие полубайты заполняются нулями.

Пример 3.

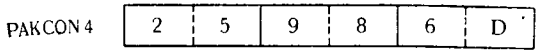
```
                PAKCON3   DC    PL2'7968'
PAKCON3 | 9 | 6 | 8 | C |
```

В этом примере программист, очевидно, не учел необходимости отведения полубайта на знак константы. Была задана явная длина в 2 байта, из-за чего остается только три полубайта для

размещения упакованных десятичных цифр. Соответственно старшая цифра собственно константы игнорируется.

Пример 4.

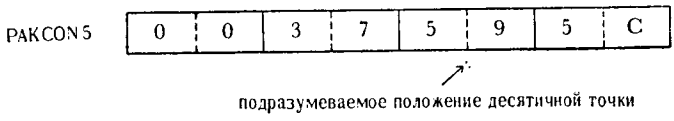
```
PAKCON4   DC    P'-25986'
```



Собственно константа была записана как отрицательная величина. Константа поэтому содержит стандартный символ знака минус для упакованных десятичных чисел.

Пример 5.

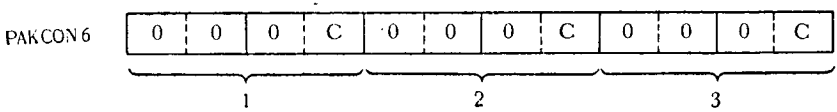
```
PAKCON5   DC    PL4'375.95'
```



Выражение константы в этом примере содержит десятичную точку. Компилятор игнорирует десятичную точку и формирует действительную константу без нее. Главной целью задания десятичной точки в предложении DC было напоминание программисту, что любое последующее использование этой константы должно подразумевать неявную десятичную точку в указанной позиции.

Пример 6.

```
PAKCON6   DC    3PL2'0'
```



Это предложение DC формирует три двухбайтовых упакованных десятичных поля, каждое из которых содержит величину +0. Каждую из этих двухбайтовых констант можно адресовать независимо: PAKCON6, PACKON6 + 2 и PAKCON6 + 4. Если нужно выполнить некоторую операцию, используя любое из этих трех полей, то в команду необходимо включить явный указатель длины. Например,

```
AP      PAKCON6+4(2), PAKDATA
```

Литеральная форма упакованной десятичной константы записывается с предшествующим ей знаком равенства. Все правила, применимые к формированию упакованной десятичной константы, справедливы и для ее литеральной формы. Примерами литералов такого типа являются:

```
AP    PACOUNTR,=PL1'1'
SP    BIGPACK,=P'32964'
CP    PAKCHECK(3),=PL3'-1596'
MP    PAKQUOT,=PL2'9.95'
DP    DIVPAK,=PL3'+17.82'
```

9. Константа типа Z (зонная десятичная)

Зонная десятичная константа используется для формирования численной константы в коде EBCDIC, в которой правый байт содержит знак десятичной величины, выражаемой константой. При соответствующем указателе длины каждая цифра собственно константы предложения DC формирует при компиляции 1 байт константы.

Если модификатор длины не задан, скомпилированная константа будет иметь указатель длины, равный той величине, которая подразумевается в соответствии с количеством цифр внутри собственно константы.

Если задан модификатор длины, который не согласуется с количеством числовых цифр, заключенных внутри собственно константы, имеет место одно из двух:

1. Если длина больше требуемой для размещения цифр, выраженных в константе, то в незаполненные старшие байты скомпилированной константы будут помещены нули.
2. Если длина меньше требуемой для размещения цифр, избыточные цифры будут игнорироваться компилятором.

Максимальная длина зонной десятичной константы равна 16 байтам. Задающее константу выражение может содержать положительный или отрицательный знак. Если знак не задан, то константа считается положительной. Внутри выражения, определяющего константу, может быть помещена десятичная точка, но только в целях информации и предполагаемого выравнивания. Десятичная точка игнорируется компилятором, и константа формируется без каких-либо ссылок к ней.

В следующих примерах поля, представляющие скомпилированные зонные десятичные константы, даны как в символьном, так и в шестнадцатеричном форматах. Заметим, что коды зонных десятичных цифр со знаком совпадают с кодами букв в EBCDIC.

Пример 1.

```
ZONECON1    DC    Z'9236'
```

ZONECON1	9		2		3		F	
	F	9	F	2	F	3	C	6

Самый правый символ F представляет положительную зонную десятичную цифру 6.

Пример 2.

```
ZONECON2    DC    ZL6'2048
```

ZONECON2	0		0		2		0		4		H	
	F	0	F	0	F	2	F	0	F	4	C	8

Самый правый символ H представляет положительную зонную десятичную цифру 8.

Пример 3.

```
ZONECON3    DC    ZL4'1651'
```

ZONECON3	1		6		5		A	
	F	1	F	6	F	5	C	1

Самый правый символ A представляет положительную зонную десятичную цифру 1.

Пример 4.

```
ZONECON4    DC    Z'-296883'
```

ZONECON4	2		9		6		8		8		L	
	F	2	F	9	F	6	F	8	F	8	D	3

Самый правый символ L представляет отрицательную зонную десятичную цифру 3.

Пример 5.

```
ZONECON5    DC    ZL5'763,13'
```

ZONECON5	7		6		3		1		C	
	F	7	F	6	F	3	F	1	C	3

Самый правый символ С представляет положительную зонную десятичную цифру 3. Десятичная точка не компилируется как часть самой константы, но появляется в листинге исходной программы как напоминание программисту о ее подразумеваемой позиции.

Зонный десятичный литерал записывается в операторе команды точно таким же образом, как операнд зонной десятичной константы — в предложении DC. Как и для всех других литералов, перед зонным десятичным литералом ставится знак равенства:

```
MVC     DATAHOLD(5),=Z'23965'
CLC     ZONEDEC(2),=ZL2'99'
MVC     OUTPUT(4),=ZL4'+377.7'
```

10. Адресные константы

Адресные константы используются для того, чтобы обеспечить программу константами, определяющими адрес памяти. Адрес, который используется для формирования операнда предложения DC, вообще говоря, может быть задан либо в абсолютной, либо в перемещаемой форме, такой, как символическая метка. Применение адресных констант упрощает распределение базовых регистров для памяти, используемой программой. Символическая метка самой константы адресует область, содержащую соответствующий адрес символа или величины, которая находится в операнде предложения DC. Выражения адресных констант заключаются в скобки, а не в апострофы. Адресные константы типа А и Y могут быть заданы с помощью литералов.

а) Адресная константа типа А. Адресная константа типа А имеет неявно выраженную длину 4 байта, что приводит к выравниванию по границе полного слова. Если задан модификатор длины, выравнивание по границе слова не производится.

Адресная константа типа А записывается следующим образом:

```
START1      DC      A(BASE1)
START2      DC      A(BASE1+4096)
START3      DC      A(BASE1+8192)
SETADDR     DC      A(TABLE)
```

б) Адресная константа типа Y. Неявно выраженная и максимальная длина адресной константы типа Y равна 2 байтам. Если модификатор длины не задан, константа будет выравнена по границе полуслова. Любая программа, которая должна выполняться под управлением Операционной системы на вычисли-

тельной машине Системы/360, не должна содержать адресных констант типа Y, заданных в перемещаемой форме. То же самое ограничение справедливо и для случая, когда объем основной памяти превышает 32 768 байтов.

Адресная константа типа Y может выглядеть так:

```
SUBRTE    DC    Y(ENTRY1)
SUBRTEA   DC    Y(ENTRY1+16394)
```

в) Адресная константа типа S. Адресная константа типа S компилируется как двухбайтовое полуслово, если не задан модификатор длины. Она имеет ограничение максимальной длины, равное 2 байтам, а наличие модификатора длины предотвращает принудительное выравнивание по границе полуслова. Когда константа скомпилирована, старшие четыре двоичные позиции в ней указывают номер базового регистра, младшие 12 двоичных позиций представляют величину смещения.

Константа может быть задана одним из двух следующих способов:

1. Набором абсолютных выражений, представляющих смещение и номер базового регистра. Выражение такого типа может иметь вид S(512(4)), где 512 показывает величину смещения, а 4 — номер базового регистра.
2. Как абсолютный или символический адрес, например S(BASE1).

г) Адресная константа типа V. Константа типа V отчасти отличается от других адресных констант в том отношении, что она компилируется с абсолютным значением содержимого, равным нулю. Действительной целью применения константы такого типа в языке Ассемблера является резервирование места в памяти для адреса внешнего символа. Этот адрес не будет фактически вырабатываться до тех пор, пока не будет загружена программа, содержащая ссылку на этот внешний символ.

Если модификатор длины не задан, адресная константа типа V компилируется как четырехбайтовое полное слово с соответствующим выравниванием границ. Выражение, задающее константу, представляет собой одиночный перемещаемый символ.

```
ADCONVA   DC    V(EXREF)
ADCONVB   DC    V(OUTMOD)
ADCONVC   DC    V(PROG2)
```

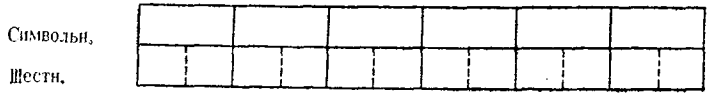
В каждом из этих примеров выражение, заключенное в скобки, при компиляции не получает идентификатора внутри той программы, в которой оно компилируется.

Упражнения

В следующих задачах константы закодированы программистом, использовавшим операторы DC. Проанализируйте отдельные операторы и заполните поля этих констант таким образом, как это делается компилятором с языка Ассемблера. Выразите содержимое каждого поля, как задано, в символьной, шестнадцатеричной или в двоичной конфигурации. Если в каком-либо случае не хватает места для того, чтобы поместить все поле по длине на одной строке, то все последующие строки представляют собой продолжение области памяти для этого поля. Обратите внимание, пожалуйста, что символьная форма пробела (X'40') представлена как ␣.

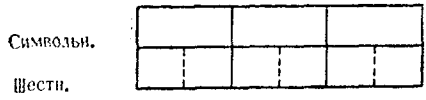
1.

```
HEXCONA   DC   XL6'C1C2C3E7E8E9'
```



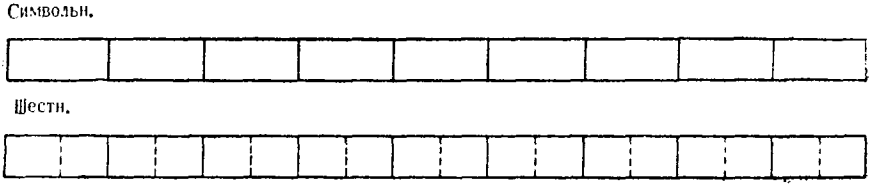
2.

```
HEXCONB   DC   XF7F8F9
```



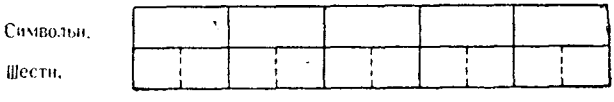
3.

```
HEXCONC   DC   3XL3'F7F8C9'
```



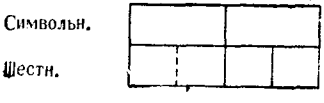
4.

```
HEXCOND   DC   XL5'5B1B1B6161'
```



5.

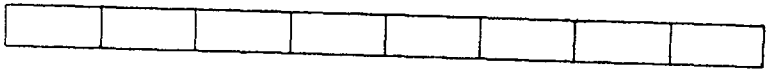
```
HEXCONE   DC   XL2'4040'
```



6.

```
HEXCONF   DC   2XL4'5BF6F1F0'
```

Символьн.



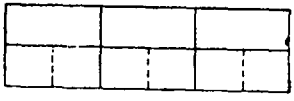
Шестн.

7

```
CHARA     DC   CL3'ABC'
```

Символьн.

Шестн.

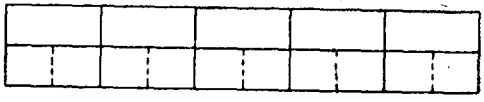


8.

```
CHARB     DC   5CL1'9'
```

Символьн.

Шестн.



9.

```
CHARC     DC   2CL4'A¢B'
```

Символьн.

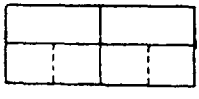
Шестн.

10.

```
CHARD     DC   CL2'$ -
```

Символьн.

Шестн.

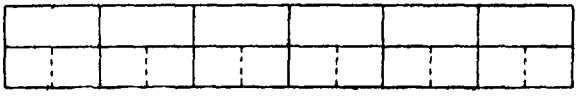


11.

```
CHARE     DC   CL6'ADHLPT'
```

Символьн.

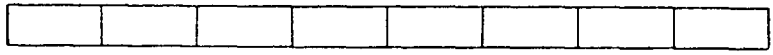
Шестн.



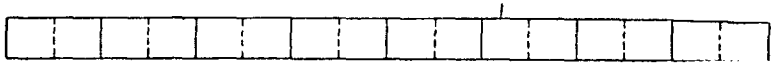
12.

```
CHARF    DC    4CL2'135'
```

Символьн.



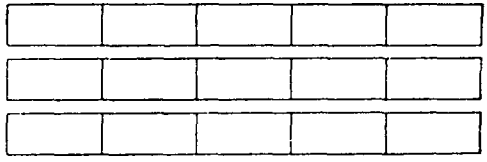
Шестн.



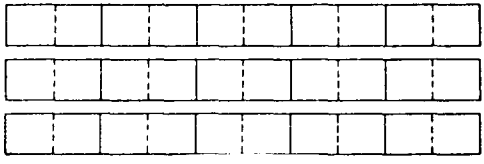
13.

```
CHARG    DC    3CL5'Ж'
```

Символьн.



Шестн.



14.

```
PKONA    DC    4PL2'+225'
```

Шестн.

15.

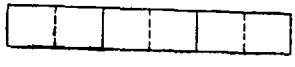
```
PKONB    DC    PL6'-51983'
```

Шестн.

16.

```
PKONC    DC    PL3'610'
```

Шестн.



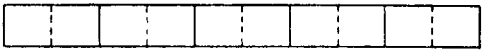
17.

```
PKOND    DC    2P'159.8762'
```

Шестн.

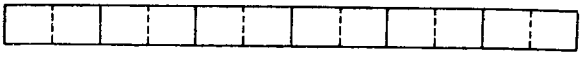
18.

```
                PKONE    DC    5PL1'-.9'
Шестн.
```



19.

```
                PKONF    DC    2PL3'+259877'
Шестн.
```

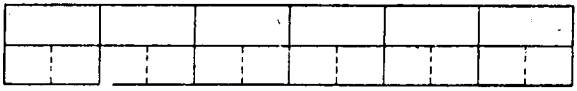


20.

```
                ZCONA    DC    ZL6'-2158'
Символьный
Шестн.
```

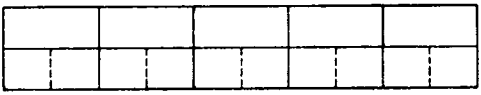
21.

```
                ZCONB    DC    2Z'+789'
Символьн.
Шестн.
```



22.

```
                ZCONC    DC    ZL5'+333'
Символьн.
Шестн.
```

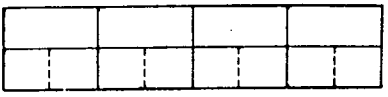


23.

```
                ZCOND    DC    3ZL2'-5'
Символьн.
Шестн.
```

24.

```
                ZCONE    DC    ZL4'4172'
Символьн.
Шестн.
```

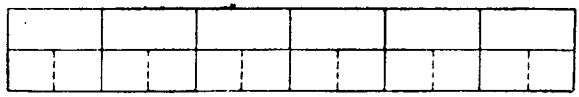


25.

```
ZCONF    DC    ZL6'+85917'
```

Символьн.

Шестн.

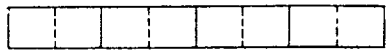


26.

```
BINCONA  DC    4BL1'01000000'
```

Шестн.

Двоичн.



27.

```
BINCONB  DC    BL3'110100011101001 0'
```

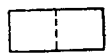
Шестн.

Двоичн.

28.

```
BINCONC  DC    2BL.2'11111111'
```

Шестн.

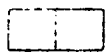


Двоичн.

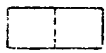
29.

```
BINCOND  DC    3BL1'11101110'
```

Шестн.



Двоичн.



30.

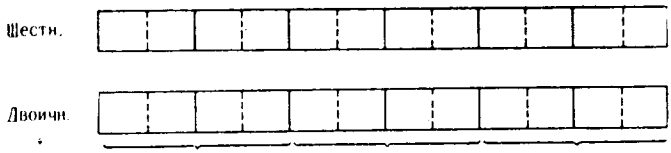
```
BINCONE  DC    BL3'1111111111111111'
```

Шестн.

Двоичн.

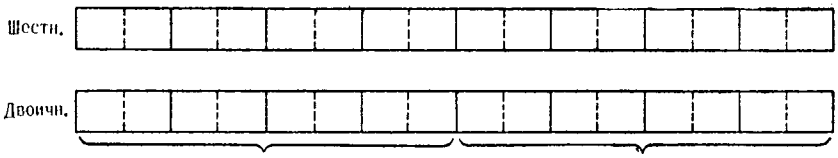
31.

```
FXCONA   DC   3H'32767'
```



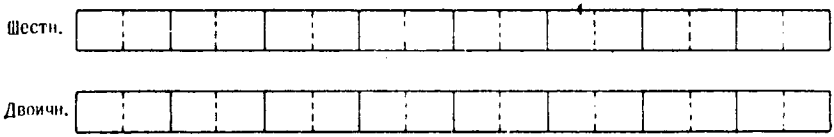
32.

```
FXCONB   DC   2F'32768'
```



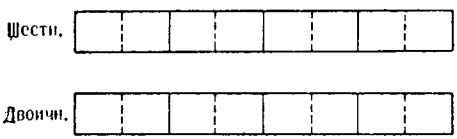
33.

```
FXCONC   DC   D'0
```



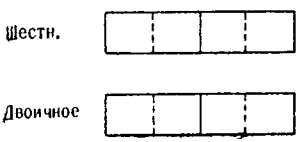
34.

```
FXCOND   DC   F'295315647'
```



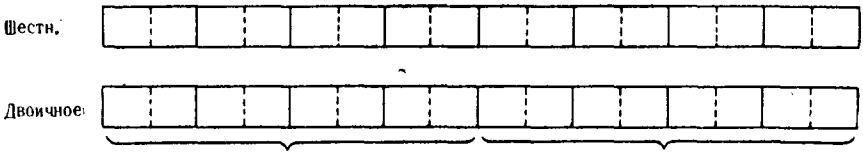
35.

```
FXCONE   DC   H'-6153'
```



36.

```
FXCONF   DC   2F'-386995'
```



Величина с фиксированной точкой

Глава 7

Сравнение данных

А. КОМАНДЫ СРАВНЕНИЯ ДАННЫХ

Прежде чем изучать общие применения команд сравнения языка Ассемблера, необходимо ознакомиться с построением и ограничениями, применимыми к каждой из этих команд. Сказанное в равной степени относится ко всем последующим главам данной книги, описывающим логические применения других команд языка Ассемблера. Каждой главе предпослано введение, содержащее краткое описание всех команд, применения которых в ней обсуждаются.

Читателю рекомендуется тщательно изучить описание каждой из команд прежде, чем приступать к ознакомлению с той частью главы, где обсуждается их применение. В дальнейшем вводные разделы этих глав можно использовать как краткое справочное руководство по командам языка Ассемблера.

Команда Compare Logical (Сравнение кодов)

Мнемоника	Код операции	Формат операндов
CL	55	$R_1,D_2(X_2,B_2)$

Производится поразрядное сравнение двоичной конфигурации регистра первого операнда с полем второго операнда. Сравнение выполняется слева направо. Если до окончания сравниваемых полей встретилось условие неравенства, дальнейшее сравнение не производится и устанавливается соответствующий признак результата. Адрес второго операнда должен быть выравнен по границе полного слова.

CC [1])	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

[1]) В колонке, обозначенной CC, даны возможные значения признака результата (Condition Code), устанавливаемого в Слове состояния программы в результате выполнения данной команды. В колонке, обозначенной BC, — соответствующие значения маски в командах условного перехода (Branch on Condition). Значения представляют двоичную конфигурацию как двоичное число без знака. — *Прим. ред.*

Команда Compare Logical Characters (Сравнение кодов)

Мнемоника	Код операции	Формат операндов
CLC	D5	$D_1(L,B_1),D_2(B_2)$

Выполняется двоичное сравнение данных в памяти, заданных первым операндом, с данными в памяти, заданными вторым операндом. Сравнение производится слева направо бит за битом. Если до окончания сравниваемых полей встречается условие неравенства, сравнение прекращается и устанавливается соответствующий признак результата. С помощью одной команды можно сравнить до 256 байтов данных.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare Logical Immediate (Сравнение непосредственное)

Мнемоника	Код операции	Формат операндов
CLI	95	$D_1(B_1),I_2$

Выполняется сравнение 1 байта данных в памяти (адресуемого первым операндом) с 1 байтом непосредственно заданных данных, определенных вторым операндом (I_2). Сравнение поразрядное, слева направо бит за битом. Если до окончания сравниваемых полей встречается условие неравенства, сравнение прекращается и устанавливается соответствующий признак результата.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare Logical Registers (Сравнение кодов)

Мнемоника	Код операции	Формат операндов
CLR	15	R_1,R_2

Выполняется сравнение содержимого регистра первого операнда с содержимым регистра второго операнда. Сравнение выполняется бит за битом слева направо. Если до окончания сравниваемых полей встречается условие неравенства, сравнение прекращается и устанавливается соответствующий признак результата.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare (Сравнение)

Мнемоника	Код операции	Формат операндов
C	59	$R_1,D_2(X_2,B_2)$

Сравнивается содержимое регистра первого операнда с содержимым второго операнда (алгебраически как 32-разрядные целые числа со знаком) и в результате этого сравнения устанавливается признак результата. В процессе выполнения команды операнды не изменяются. Адрес второго операнда должен быть выравнен по границе полного слова.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare Halfword (Сравнение полуслова)

Мнемоника	Код операции	Формат операндов
CH	49	$R_1,D_2(X_2,B_2)$

До начала сравнения производится расширение содержимого 16-разрядной величины второго операнда до 32-разрядной величины. Значение каждого из 16 добавочных старших двоичных разрядов совпадает со значением исходного знакового бита второго операнда. Затем производится алгебраическое сравнение содержимого регистра первого операнда с расширенным вторым операндом как с 32-разрядным целым числом со знаком и устанавливается соответствующий признак результата. При выполнении этой команды операнды не изменяются. Второй операнд должен быть выравнен по границе полуслова.

CC	BC	Условие
0	8	Первый операнд равен второму
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare Registers (Сравнение)

Мнемоника	Код операции	Формат операндов
CR	19	R_1,R_2

Алгебраически сравнивается содержимое регистра первого операнда с содержимым регистра второго операнда, каждое из которых рассматривается как 32-разрядные числа со знаком, и устанавливается признак результата. Содержимое операндов не изменяется.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

Команда Compare Packed Decimals (Сравнение десятичное)

Мнемоника	Код операции	Формат операндов
CP	F9	$D_1(L_1,B_1),D_2(L_2,B_2)$

Сравнивается содержимое первого операнда в упакованном десятичном формате с содержимым второго операнда в упакованном десятичном формате. Сравнение производится справа налево по всем цифрам обоих операндов. Если указатели длины двух операндов не равны между собой, то более короткое поле расширяется нулями со стороны старших разрядов до длины более протяженного. Двоичная конфигурация самых правых полубайтов каждого поля проверяется только с целью определения вида знака — плюс или минус. Все допустимые коды знака плюс рассматриваются как совпадающие; аналогично рассматриваются допустимые коды знака минус. Положительный нуль (+0) принимается равным отрицательному нулю (—0). Конфигурация содержимого операндов не изменяется в процессе выполнения сравнения. Максимальная подразумеваемая или явная длина для любого из сравниваемых полей равна 16 байтам.

CC	BC	Условие
0	8	Первый и второй операнды равны
1	4	Первый операнд меньше второго
2	2	Первый операнд больше второго

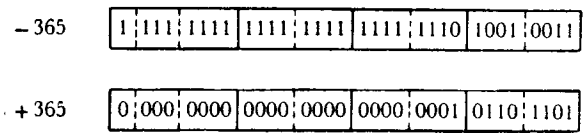
Б. ПРИМЕНЕНИЕ КОМАНД СРАВНЕНИЯ

Было бы нелогично предполагать, что можно написать программу, в которой нет ни одного оператора сравнения. Хотя не все сравнения определяются программистом (операционная система сама во многих случаях выполняет функции принятия решения, основанные на сравнениях), в типичной программе, вероятно, найдется много мест, связанных с принятием решений, результат которых основан на сравнении.

Так как в результате выполнения большинства команд языка Ассемблера в Слове состояния программы устанавливается признак результата, каждую из них можно несколько вольно трактовать как своего рода команду сравнения. Даже если не рассматривать установку признака результата в качестве отличительной черты команд сравнения, все же имеется несколько команд, которые не содержат слова «сравнение» в их наименованиях, но по существу являются командами сравнения. К ним относятся, например, команды Branch on Count (Переход по счетчику). Branch on Index High (Переход по индексу больше) и Test under Mask (Проверить по маске). Однако, исходя из целей данного раздела, в нем будут рассмотрены только команды, которые содержат в своем наименовании слово «сравнение». Для удобства последующего рассмотрения они разделены на три группы: сравнение кодов, сравнение чисел с фиксированной точкой и сравнение упакованных десятичных чисел.

1. Сравнение кодов

К этой группе относятся команды с мнемоникой CL, CLC, CLI, CLR. Действие этих команд никак не связано с символьным форматом полей сравниваемых операндов некоторых из них. Выполнение каждой из этих команд начинается с левой (старшей) двоичной позиции полей операндов и заключается в поразрядном сравнении соответствующих двоичных позиций последовательно слева направо. Каждая двоичная позиция в первом операнде сравнивается с соответствующей двоичной позицией во втором операнде. Как только при очередном сравнении выявлено неравенство, сравнение прекращается и в PSW (Program Status Word — Слово состояния программы) устанавливается признак результата, соответствующий состоянию неравенства операндов. Если условие неравенства не обнаружено, то сравнение продолжается по всей длине полей операндов бит за битом до тех пор, пока неявная или явная длина первого операнда не окажется исчерпанной. Так как при сравнении кодов результат определяется сравнением отдельных пар битов независимо от интерпретации этих битов в формате операндов, то при сравнении десятичных чисел —365 и +365, представленных в виде чисел с фиксированной точкой, —365 окажется «больше» чем +365. Поля этих двух операндов выглядят так:



Первой двоичной позицией, подлежащей сравнению, оказывается знаковый бит, причем поле отрицательной величины —365 содержит в этом разряде единицу, а поле положительной величины +365 — нуль. После обнаружения этого неравенства сравнение прекращается и признак результата устанавливается таким образом, чтобы показать, что код числа —365, представленного в виде величины с фиксированной точкой, больше, чем соответствующий код числа +365. И это соответствует действительности, так как производилось сравнение кодов операндов, а не арифметических значений с фиксированной точкой.

Сравнение кодов — CL. Эта команда используется для сравнения содержимого регистра с содержимым полного слова в памяти, например для сравнения конфигурации содержимого общего регистра с заранее установленной поразрядной конфигурацией константы с целью определения, выполнено ли некоторое условие или достигнута точка, заданная при выполнении программы.

Для того чтобы лучше уяснить различные формы представления поля данных, в следующем примере показана логическая последовательность перехода от одной формы к другой. Предположим, что некоторое полное слово в памяти, адресуемое с помощью метки CONA, содержит поразрядную конфигурацию, соответствующую кодам EBCDIC для последовательности букв ABCD. Переход от формата EBCDIC к его интерпретации в двоичном формате для содержимого поля CONA выглядит следующим образом:

Символьн.	A		B		C		D	
Шестн.	C	1	C	2	C	3	C	4
Двоичн.	1100	0001	1100	0010	1100	0011	1100	0100

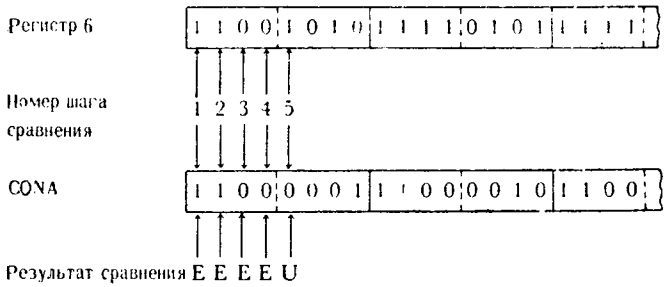
Используя это поле в качестве одного из операндов, оператор сравнения можно закодировать так:

```
CL      6, CONA
```

Пусть перед выполнением этой команды общий регистр 6 содержит следующую конфигурацию:

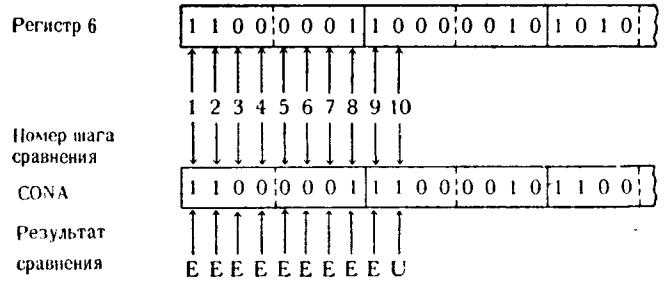
Общий регистр 6	1100	1010	1111	0101	1111	1100	1100	0100

Последовательность выполнения сравнения имеет следующий вид:



Процесс сравнения прекратится на пятом шаге в связи с обнаружением неравенства. Признак результата будет указывать, что общий регистр 6 содержит поразрядную конфигурацию, которая логически (как двоичное число без знака) больше по своему значению, чем поразрядная конфигурация CONA.

В следующем примере команды CL поле CONA содержит ту же самую величину, что и в предыдущем случае, а содержимое общего регистра 6 изменено; это приводит к следующим шагам сравнения:



В этом примере сравнение заканчивается на десятом шаге и устанавливается признак результата, показывающий, что общий регистр 6 содержит поразрядную конфигурацию, логически меньшую, чем поразрядная конфигурация поля CONA.

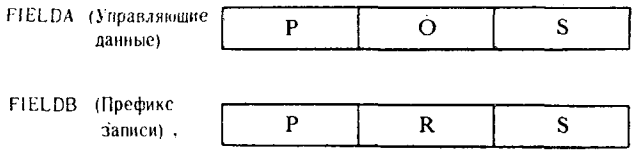
Сравнение кодов — CLC. Команда CLC обычно используется для сравнения содержимого двух полей символьных данных в памяти. Подразумевается, что оба поля содержат буквы, числа или знаки в коде EBCDIC, хотя это ограничение не обязательно. Эту команду можно применять для идентификации входных данных, используя наборы управляющих данных, для проверки полей данных с последующим принятием решения о направлении ветвления в программе, для установления критерия выбора

данных, подлежащих обработке, или для любого другого конкретного использования, где она могла бы указать на предусмотренную программистом логическую ветвь программы.

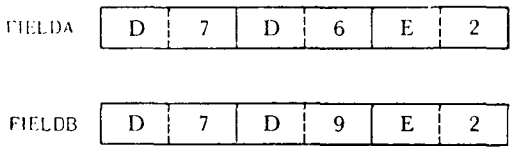
В следующем примере поля FIELDA и FIELDB содержат по 3 байта символьных данных в коде EBCDIC. Поле FIELDA содержит управляющие данные, с которыми должны совпадать префиксные поля определенных записей. Если префиксное поле записи FIELDB совпадает с полем FIELDA, то программист может захотеть использовать определенные данные из этой записи. Действие команды CLC можно показать на следующем примере:

```
CLC     FIELDA, FIELDB
```

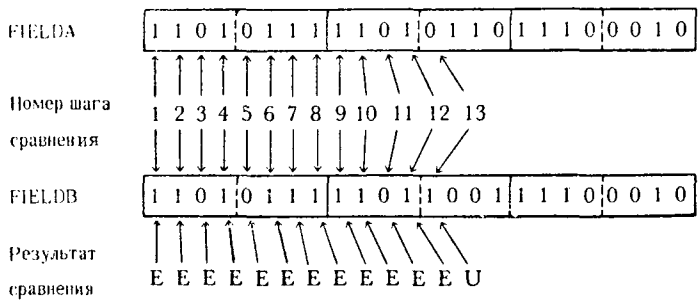
Содержимое этих полей в символьном формате выглядит так:



Шестнадцатеричная конфигурация этих полей выглядит следующим образом:



Поразрядная двоичная конфигурация полей и результирующее действие команды CLC выглядят так:



Как видно из примера, сравнение заканчивается при проверке 13-й (слева) двоичной позиции из-за обнаружения неравенства. Признак результата покажет, что FIELDA меньше, чем FIELDB, и, основываясь на этом, в соответствии с алгоритмом

программы будет решено, какое действие необходимо предпринять.

С помощью этой команды также можно без труда проверять числа, представленные в коде EBCDIC. Предположим, что программист захотел проверить трехбайтовое поле данных, содержащее числовые данные в коде EBCDIC, чтобы удостовериться в том, что это число не больше чем 099. Если число 099 представлено в виде константы в области памяти с меткой NUMBRA, а данные, подлежащие проверке, помещены в поле памяти с меткой NUMBRB, то оператор команды выглядит так:

```
CLC     NUMBRB, NUMBRA
```

Если сравниваемая величина 099 не была заготовлена ранее в виде константы, то программист вместо второго операнда может использовать литерал. В процессе компиляции программы компилятор присвоит литералу адрес в памяти. Предложение в этом случае записывается так:

```
CLC     NUMBRB, =CL3'099'
```

В обоих случаях сравнение будет выполняться аналогично, если соответствующие операнды описывают одинаковые поля.

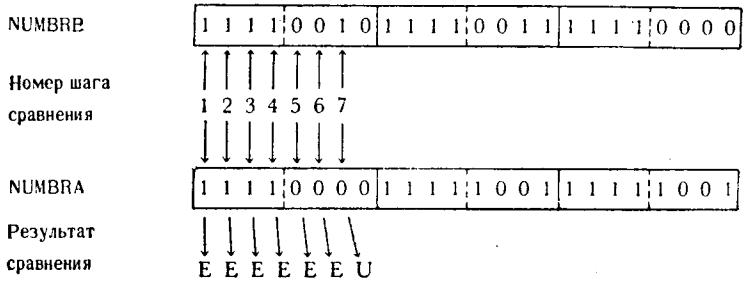
Символьный формат:

NUMBRB	2	3	0
NUMBRA	0	9	9

Шестнадцатеричная конфигурация:

NUMBRB	F	2	F	3	F	0
NUMBRA	F	0	F	9	F	9

Поразрядная двоичная конфигурация:



На седьмом (слева) бите команда обнаружит условие неравенства и прекратит сравнение. Признак результата покажет, что поле NUMBRB не равно, а именно больше, чем NUMBRA.

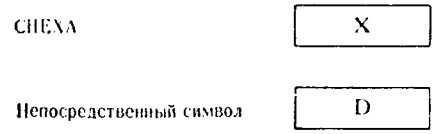
Сравнение кодов непосредственное — CLI. Так как эта команда сравнивает только поля длиной 1 байт, ее целесообразнее всего использовать для проверки условия «включено — выключено» в однобайтовом переключателе или для поиска однобайтового управляющего символа в поле данных. Первый операнд содержит метку, или символический адрес, 1 байта данных в памяти; второй операнд представляет собой однобайтовый самоопределенный (непосредственно заданный) символ или величину.

Например, программист планирует проверить однобайтовое управляющее поле в каждой записи вводимых данных на содержание в нем буквы D в формате кода EBCDIC. Это однобайтовое поле CHEXA можно сравнить с D одним из следующих способов, используя команду CLI:

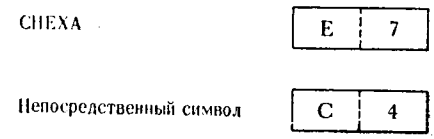
```
CLI     CHEXA,C'D'
CLI     CHEXA,X'C4'
CLI     CHEXA,B'11000100'
```

В каждой из этих трех команд сравнение будет выполнено идентичным способом. Заметим, что конфигурация второго операнда вне зависимости от того, выражен ли он в двоичной, шестнадцатеричной или символьной форме, интерпретируется как буквенный символ D во всех трех примерах. Предположив, что CHEXA содержит символ X, представление данных можно истолковывать следующим образом:

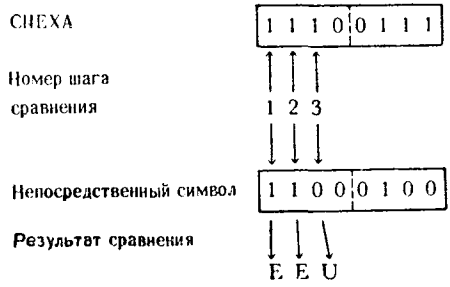
Символьный формат:



Шестнадцатеричная конфигурация:



Последовательность действий команды CLI над двоичными конфигурациями данных выглядит так:



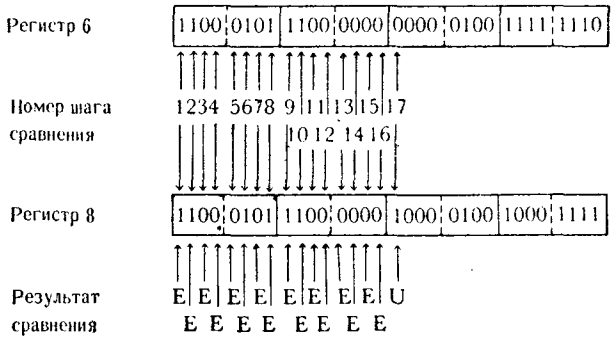
Во время сравнения третьих позиций обнаруживается неравенство и выполнение команды заканчивается. Значение признака результата дает возможность программисту определить, что сравниваемые величины не равны, причем значение содержимого первого операнда больше значения содержимого второго операнда.

Сравнение кодов — CLR. Команда CLR сравнивает двоичную конфигурацию двух общих регистров. Как и в других командах сравнения кодов, соответствующие двоичные позиции сравниваются слева направо бит за битом.

Оператор команды, сравнивающей содержимое общего регистра 6 и общего регистра 8, имеет вид

```
CLR     6,8
```

Следующая иллюстрация показывает процесс выполнения команды сравнения при заданном содержимом регистров:



Сравнение приводит к равенству до 17-й двоичной позиции. Здесь отмечается неравенство, сравнение прекращается и

установленное значение признака результата в PSW показывает, что величина в общем регистре 6 меньше величины в общем регистре 8.

2. Сравнение величин с фиксированной точкой

Для сравнения величин с фиксированной точкой используются команды с мнемоникой C, CH и CR. Каждая из этих команд сравнивает первый операнд со вторым операндом, устанавливая соответствующий признак результата. Оба операнда рассматриваются как 32-разрядные целые числа со знаком, причем 31 бит отводится на само число, а старший бит — на знак. Сравнение выполняется алгебраически на основе значения каждого операнда с фиксированной точкой. По существу команда определяет значение целых чисел длиной 31 бит, учитывает знак каждого из них и производит сравнение. Значение положительного (+) целого числа определяется суммированием значений соответствующих степеней двойки для каждой двоичной позиции, которая содержит единицу. Значение отрицательного (—) целого числа, выраженного в форме дополнения до 2, может определяться суммированием степеней двойки для целочисленной части, как если бы она представляла положительное число, и вычитанием полученной величины из максимальной отрицательной величины, которая равна —2 147 483 648 для полного слова и —32 768 для полуслова.

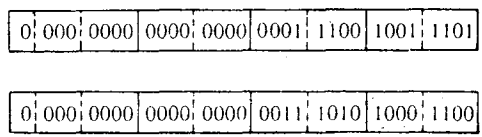
Сравнение — C. Эта команда используется для алгебраического сравнения содержимого общего регистра как числа с фиксированной точкой со значением содержимого полного слова данных из памяти, также рассматриваемого как число с фиксированной точкой.

Для того чтобы понять логику выполнения этой команды, предположим, что необходимо выполнить сравнение общего регистра 10 с полным словом FLWDA, находящимся в памяти. Предложение кодируется так:

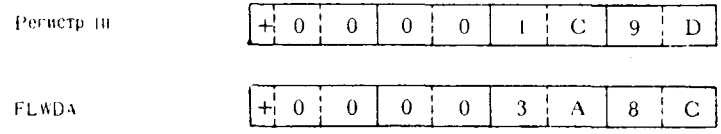
```
C     10, FLWDA
```

Данные, которые необходимо сравнить с помощью этой команды, имеют следующий вид:

Двоичная конфигурация:



Шестнадцатеричная конфигурация:



Программист может определить значение этих полей, преобразуя отдельные шестнадцатеричные величины в те значения величин с фиксированной точкой, которые они представляют. В этом ему помогут таблица степеней двойки и таблица шестнадцатерично-десятичного преобразования.

Метод выполнения преобразования иллюстрируется следующим примером:

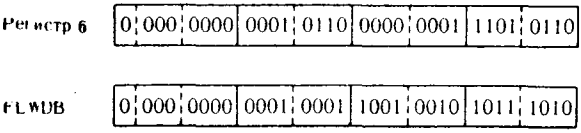
Справа налево	Регистр 10		FLWDA	
	16-ичное	10-ичное	16-ичное	10-ичное
Первая 16-ичная цифра	D	13	C	12
Вторая 16-ичная цифра	9	144	8	128
Третья 16-ичная цифра	C	3072	A	2560
Четвертая 16-ичная цифра	1	4096	3	12288
Целочисленное значение		7325		14988
Знаковый бит		+		+
Величина с фиксированной точкой		+7325		+14988

Алгебраическое сравнение показывает, что значение величины с фиксированной точкой поля FLWDA больше, чем значение величины, содержащейся в общем регистре 10. Признак результата в PSW будет показывать, что первый операнд меньше второго.

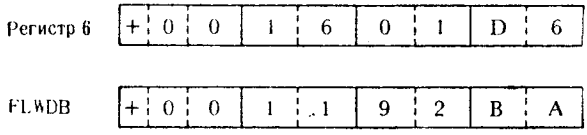
Примером, в котором в операндах содержатся другие значения величин, может служить оператор:

```
C     6, FLWDB
```

Двоичная конфигурация:



Шестнадцатеричная конфигурация:



Определение значений величин с фиксированной точкой по их шестнадцатеричному представлению.

Справа налево	Регистр 6		FLWDB	
	16-ичное	10-ичное	16-ичное	10-ичное
Первая 16-ичная цифра	6	6	A	10
Вторая 16-ичная цифра	D	208	B	176
Третья 16-ичная цифра	1	256	2	512
Четвертая 16-ичная цифра	0	0	9	36864
Пятая 16-ичная цифра	6	393216	1	65536
Шестая 16-ичная цифра	1	1048576	1	1048576
Целочисленное значение		1442262		151671
Знаковый бит		+		+
Величина с фиксированной точкой		+1442262		+1151674

В результате выполнения команды сравнения выясняется, что алгебраическое значение в общем регистре 6 больше, чем значение величины в поле FLWDB. Признак результата покажет, что величина с фиксированной точкой первого операнда оказалась больше величины второго операнда.

Чтобы полностью прочувствовать механизм работы команд с фиксированной точкой, нужно хорошо разобраться в преобразовании поразрядной двоичной конфигурации в величину с фиксированной точкой посредством интерпретации шестнадцатеричных значений. Предыдущие два примера иллюстрировали преобразование обычных положительных величин. Такое же значение имеет знание принципов образования и интерпретации дополнений до 2 величин с фиксированной точкой. Хотя эта тема подробно рассматривается в других разделах книги, повторим некоторые основные правила.

Для целей кодирования программы не требуется знание машинного формата величин. Однако, если программист захочет проверить правильность полученного результата, ему придется вычислять значения величин по их действительному представлению в машине.

Если проверяется значение отрицательной величины с фиксированной точкой, представленной в виде дополнения до 2, это значение может быть определено следующим образом:

Двоичная конфигурация:

Регистр 3	1	111	1111	1110	1100	0111	1001	1101	0010

Шестнадцатеричная конфигурация:

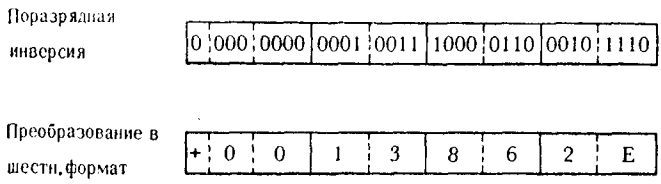
Регистр 3	–	7	F	E	C	7	9	D	2

Определение значения отрицательной величины с фиксированной точкой по ее шестнадцатеричному представлению:

Справа налево	Регистр 3		
	16-ичная	10-ичная	
Первая 16-ичная цифра	2	2	
Вторая 16-ичная цифра	D	208	
Третья 16-ичная цифра	9	2304	
Четвертая 16-ичная цифра	7	28672	
Пятая 16-ичная цифра	C	786432	
Шестая 16-ичная цифра	E	14680064	
Седьмая 16-ичная цифра	F	251658240	
Восьмая 16-ичная цифра	7	1879048192	(за исключением
Целочисленное значение		2146204114	знакового бита)
Максимальное отрицательное значение *полного слова* с фиксированной точкой			2147483648
Целочисленное значение в общем регистре 3			2146204114
Разность			1279534
Знаковый бит			—
Отрицательное значение величины с фиксированной точкой в регистре 3			— 1279534

Целочисленную часть регистра 3 можно проверить, преобразуя ее обратно в положительную величину с помощью перехода к дополнительному коду. Это можно выполнить таким образом:

Общий регистр 3	1	111	1111	1110	1100	0111	1001	1101	0010
Вычесть единицу									–1
Результат	1	111	1111	1110	1100	0111	1001	1101	0001



Справа налево	Регистр 3	
	16-ичное	10-ичное
Первая 16-ичная цифра	E	14
Вторая 16-ичная цифра	2	32
Третья 16-ичная цифра	6	1536
Четвертая 16-ичная цифра	8	32768
Пятая 16-ичная цифра	3	196608
Шестая 16-ичная цифра	1	1048576
Целочисленное значение		1279534

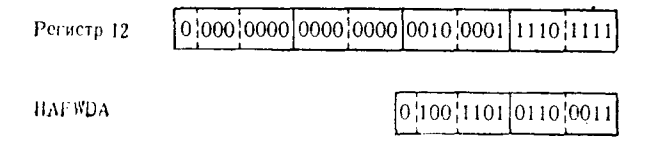
В процессе преобразования отрицательной конфигурации содержимого общего регистра 3 в положительную величину была проверена точность интерпретации исходной отрицательной конфигурации.

Сравнение полуслова — CH. Команда CH алгебраически сравнивает содержимое общего регистра с содержимым полуслова в памяти. При выполнении команды сначала производится обращение к памяти и выбирается полуслово данных. Затем это 16-разрядное полуслово расширяется до 32-разрядного полного слова. Формируемым дополнительным старшим разрядам присваивается то же самое значение, что и знаковому биту в полуслове. После этого выполняется алгебраическое сравнение двух операндов.

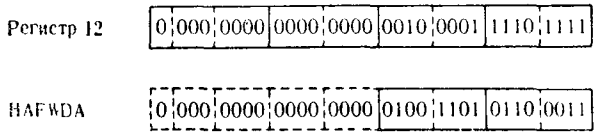
Действие этой команды иллюстрируется следующим примером:

```
CH      12, HAFWDA
```

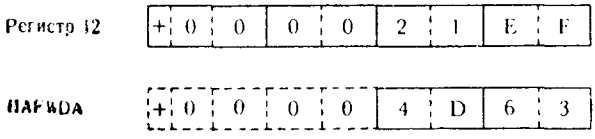
Исходная поразрядная конфигурация полей:



Поразрядная конфигурация после расширения полуслова HAFWDA:



Шестнадцатеричная интерпретация этих полей:

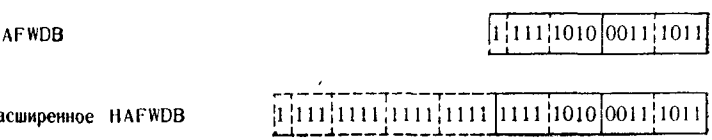


Определение значения величины с фиксированной точкой по ее шестнадцатеричному представлению:

Справа налево	Регистр 12		Расширенное HAFWDA	
	16-ичное	10-ичное	16-ичное	10-ичное
Первая 16-ичная цифра	F	15	3	3
Вторая 16-ичная цифра	E	224	6	96
Третья 16-ичная цифра	1	256	D	3328
Четвертая 16-ичная цифра	2	8192	4	16384
Целочисленное значение		8687		19811
Знаковый бит		+		+
Величина с фиксированной точкой		+8687		+19811

В результате алгебраического сравнения обнаруживается, что величина в поле HAFWDA больше величины в регистре 12. Признак результата в PSW отразит этот факт.

Полуслово в памяти, содержащее отрицательную величину, расширяется посредством добавления 16 дополнительных старших двоичных разрядов, каждый из которых имеет значение знакового бита исходного полуслова.



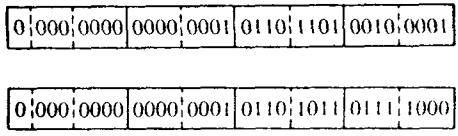
Значение поля HAFWDB не изменилось, но само поле имеет теперь размер, облегчающий сравнение с содержимым регистра. Исходные данные, по-прежнему хранящиеся в ячейке памяти HAFWDB, не изменяются командой CH.

Сравнение — CR. Команда CR используется для алгебраического сравнения величин с фиксированной точкой, содержащихся в двух общих регистрах. Содержимое регистра первого операнда сравнивается с содержимым регистра второго операнда. В этой команде, как и во всех других командах сравнения величин с фиксированной точкой, оба операнда рассматриваются в качестве 32-разрядных целых чисел со знаком, состоящих из знакового бита и 31-разрядной целочисленной части.

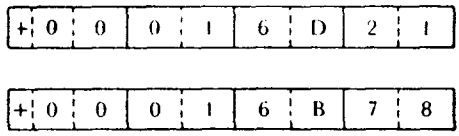
Программист может оценить содержимое двух регистров операндов команды CR следующим образом:

```
CR    6,9
```

Двоичная конфигурация регистров:



Шестнадцатеричная конфигурация регистров:

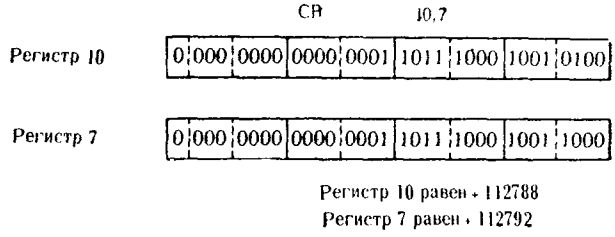


Определение значения величины с фиксированной точкой по ее шестнадцатеричному представлению:

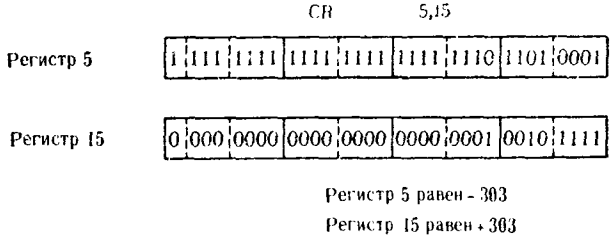
Справа налево	Регистр 6		Регистр 9	
	16-ичное	10-ичное	16-ичное	10-ичное
Первая 16-ичная цифра	1	1	8	8
Вторая 16-ичная цифра	2	32	7	112
Третья 16-ичная цифра	D	3328	B	2816
Четвертая 16-ичная цифра	6	24576	6	24576
Пятая 16-ичная цифра	1	65536	1	65536
Целочисленное значение		93473		93048
Знаковый бит		+		+
Величина с фиксированной точкой		+93473		+93048

После окончания операции сравнения признак результата в PSW показывает, что величина первого операнда в общем регистре 6 больше величины второго операнда в общем регистре 9.

Ниже дано несколько других примеров интерпретации значений операндов команды CR. В этих примерах промежуточные иллюстрации определения значений полей опущены.



Признак результата покажет, что первый операнд в общем регистре 10 меньше второго операнда в общем регистре 7.



После выполнения этой команды признак результата покажет, что величина с фиксированной точкой первого операнда в регистре 5 меньше соответствующей величины второго операнда в регистре 15.

3. Сравнение упакованных десятичных чисел

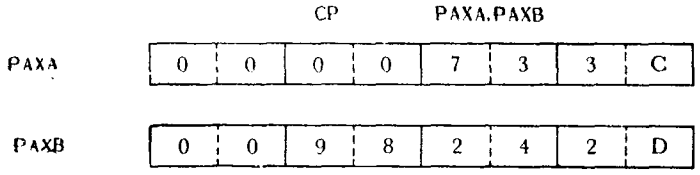
Единственной командой этой группы является команда Сравнение десятичное — CP. Сравниваются поля данных в памяти в соответствии с правилами операций над упакованными десятичными числами. Данные сравниваются справа налево по одному полубайту. В силу того что четыре двоичные позиции используются для представления как десятичных, так и шестнадцатеричных цифр, в дальнейшем не делается различия между упакованными десятичными цифрами и соответствующими шестнадцатеричными символами.

Сравнение десятичное — CP. Действие этой команды в основном понятно из предшествующего изложения. Однако следует заметить, что в отличие от других команд сравнения эта команда предусматривает указание длин обоих операндов. Максимальная длина каждого операнда — 16 байтов. Если поля операндов имеют неодинаковую длину, то более короткое поле будет расширено добавлением дополнительных байтов, заполненных шестнадцатеричными нулями, до совпадения длин обоих полей. Дополнительные байты присоединяются со стороны старших цифр более короткого поля.

Как и для других команд десятичной арифметики, знак плюс может быть представлен шестнадцатеричными символами A(1010), C(1100), E(1110) или F(1111); знак минус — символами B(1011) или D(1101). Поля, подлежащие сравнению, должны содержать в младшем полубайте в качестве символа знака одну из указанных шестнадцатеричных цифр. Любая шестнадцатеричная цифра со значением, меньшим 10 (A), рассматривается как упакованная «числовая» цифра и не может служить знаком для упакованного десятичного поля. Любые допустимые представления знака плюс — A, C, E или F — при сравнении считаются равными.

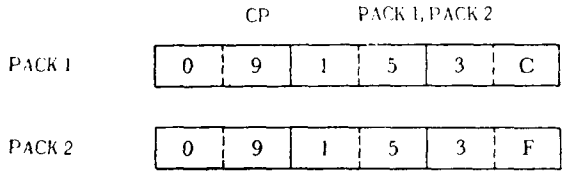
Аналогичному правилу подчиняются символы, представляющие знак минус, — B и D. В некотором противоречии с общими правилами упакованная десятичная величина +0 считается равной упакованной десятичной величине —0.

В примерах этой команды в отличие от команд сравнения величин с фиксированной точкой поразрядная двоичная конфигурация полей не используется. Для наиболее ясной иллюстрации значений упакованных десятичных полей каждый полубайт представлен шестнадцатеричной цифрой. В некоторых примерах операнды команд будут содержать явные указатели длины. Это делается для того, чтобы подчеркнуть различие в длинах между операндами и способ выравнивания длины операндов.

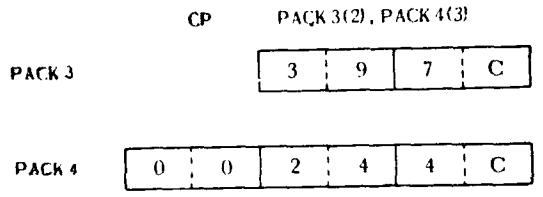


Как видно PAXA содержит упакованную десятичную величину +733; содержимым PAXB является упакованная десятичная величина —98 242. В результате действия команды CP признак результата отобразит тот факт, что значение первого опе-

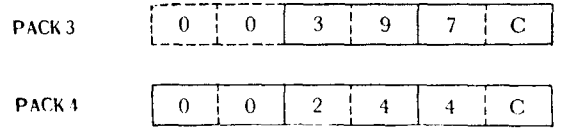
ранда больше значения второго операнда.



Признак результата примет значение, показывающее, что оба операнда содержат равные величины — как цифра C, так и цифра F представляют допустимые коды знака плюс в упакованных десятичных полях.



Прежде чем выполнять собственно сравнение, более короткое поле извлекается из памяти и расширяется до размера более длинного поля.



В результате сравнения признак результата покажет, что упакованное десятичное значение первого операнда PACK3 больше соответствующего значения второго операнда PACK4 несмотря на то, что указатель длины поля PACK3 меньше. Содержимое исходного поля памяти PACK3 при выполнении команды CP не изменяется, хотя длина поля была увеличена в целях сравнения.

Упражнения

1. Команды сравнения кодов начинают свое действие с _____ двоичной позиции и ведут сравнение в направлении _____ двоичной позиции. Сравнение заканчивается, как только обнаруживается несовпадение между двумя соответствующими _____ позициями.

2. Второй операнд предложения CLI должен представлять собой _____ символ или величину.

3. Команды C, CH и CR рассматриваются как команды сравнения чисел с _____.

4. Какие шестнадцатеричные символы допустимы для обозначения знака минус в отрицательных упакованных десятичных величинах?

5. Чему равна максимально допустимая длина упакованного десятичного поля, являющегося вторым операндом в предложении CP? _____

6. Упакованная десятичная знаковая цифра X'C' при сравнении окажется (равна) (не равна) (больше) (меньше) упакованной десятичной знаковой цифре X'A'.

7. При выполнении команды CL величина с фиксированной точкой —39 510 окажется (больше) (меньше) (равна) величины с фиксированной точкой +39 510.

8. Максимальная длина любого из полей, сравниваемых по команде CLC, равна _____.

9. Следующие два поля сравниваются с помощью команды сравнения кодов. Обведите кружочком те двоичные позиции в каждом из полей, где прекратится сравнение.

0000	1010	1100	0100	1101	1011	1111	1000

0000	1010	1100	0100	1101	1010	1111	1000

10. Если поле с меткой BYTE содержит букву S в коде EBCDIC, следующая команда установит признак результата в состояние, соответствующее тому, что BYTE (больше) (меньше) (равен) второго операнда:

```
CLI     BYTE, B'11100010'
```

11. Общий регистр 6 содержит величину с фиксированной точкой +3 175 815. Полное слово FWDTWO содержит шестнадцатеричную конфигурацию X'00307588'. После выполнения следующей команды признак результата покажет, что величина в регистре 6 (больше) (меньше) (равна) величины с фиксированной точкой, находящейся по адресу FWDTWO:

```
C       6,FWDTWO
```

12. Величину «нуль» с фиксированной точкой можно представить в положительной или отрицательной форме. Так ли это? (Да) (Нет).

Тщательно проанализируйте каждую из следующих ниже команд. Отметьте те команды, которые содержат неправильные типы операндов, неправильные указатели длины или содержат операнды в неверно заданных форматах. В большинстве примеров регистровые операнды закодированы как R3, R4, R5, R6, R7, R8, R9 и т. д.; номера регистров приравнены этим символическим меткам, чтобы помочь читателю при определении тех операндов, которые являются регистрами.

13.	_____	CP	DATAPAK,=X'03952C'
14.	_____	CLR	R12,R15
15.	_____	CLC	INDATA,0(5)
16.	_____	CLI	VALUE,X'40'
17.	_____	CL	DATAFLD,INPUT
18.	_____	CR	R9,R8
19.	_____	CH	HAFWD,R7
20.	_____	C	DATA,0(8)
21.	_____	CP	PACKSUM(5),DATAPAK(3)
22.	_____	C	R8,=F'32957'
23.	_____	CLC	0+5(3,9),R7
24.	_____	CLI	7(5),B'10110001'
25.	_____	CP	R13,DUBLPACK
26.	_____	CL	R3,R2
27.	_____	C	R12,FULLWD6
28.	_____	CLR	R10,R11
29.	_____	CLI	R6,F'6'
30.	_____	CLC	DATA(95),BIGSET
31.	_____	C	R13,R12
32.	_____	CR	R6,DATA(9)
33.	_____	CP	0+10(6,8),5+7(10,7)
34.	_____	CLI	SWITCH,=C'X'
35.	_____	CH	R9,HAFWD7
36.	_____	CL	R6,FULLWD5RG
37.	_____	CP	SETPACK+20(3),INSET+52(3)
38.	_____	CLI	SETUP,C'1'
39.	_____	CP	PACKIT(25),INPACK(25)
40.	_____	CLR	SETVALUE,R6
41.	_____	CLC	0(4,3),5(4,6)
42.	_____	C	R10(7),FULLWD8
43.	_____	CL	R6,4(8)

44.	_____	CH	R3,FULLWD
45.	_____	CLR	0(4,9),R3
46.	_____	CLI	TWOBYTE(2,6),C'AB'
47.	_____	CLC	INDATA+9(6),CONSTA
48.	_____	CR	R3(6),R11
49.	_____	C	R9,8(3)
50.	_____	CP	PACKINITL+15(3),=PL3'519'
51.	_____	CLI	0+95(11),X'C1'
52.	_____	CLR	R9,R13
53.	_____	CH	R12,HAFWD4

Глава 8

Переходы

А. КОМАНДЫ ПЕРЕХОДОВ

Команда Branch on Condition (Условный переход)

Мнемоника	Код операции	Формат операндов
BC	47	$M_1,D_2(X_2,B_2)$

Команда BC сопоставляет двоичную конфигурацию маски в первом операнде с признаком результата в PSW. Если обнаружено соответствие, в программе происходит переход к команде, расположенной по адресу, указанному вторым операндом.

Значение признака результата не изменяется.

Команда Branch on Condition to Register (Условный переход)

Мнемоника	Код операции	Формат операндов
BCR	07	M_1,R_2

Команда BCR вызывает переход к команде, расположенной по адресу, находящемуся в регистре второго операнда, если значение маски в первом операнде соответствует значению признака результата.

Признак результата не изменяется.

Команда Branch on Count (Переход по счетчику)

Мнемоника	Код операции	Формат операндов
BCT	46	$R_1,D_2(X_2,B_2)$

Команда BCT вызывает переход по адресу, определяемому вторым операндом, всякий раз, когда значение величины в регистре первого операнда отлично от нуля. При каждом выполнении этой команды происходит алгебраическое уменьшение содержимого регистра первого операнда на 1. Если после этого оно становится равным нулю, переход не производится и выполняется следующая по порядку команда.

Признак результата не изменяется.

Команда Branch on Count to Register (Переход по счетчику)

Мнемоника	Код операции	Формат операндов
BCTR	06	R_1,R_2

Команда BCTR вызывает переход к команде, адрес которой находится в регистре второго операнда, если значение величины в регистре первого операнда отлично от нуля. При каждом выполнении этой команды значение величины в регистре первого операнда алгебраически уменьшается на 1. Если после этого оно станет равным нулю, переход не производится и выполняется следующая по порядку команда.

Признак результата не изменяется.

Команда Branch on Index High (Переход по индексу больше

Мнемоника	Код операции	Формат операндов
BXH	86	$R_1,R_3,D_2(B_2)$

Эта команда использует три регистра, один из которых имеет четный номер, а два других — нечетный [1]). Регистр первого операнда представляет собой первый из нечетно пронумерованных общих регистров, регистр третьего операнда (R_3) имеет четный номер, а следующий регистр с нечетным номером просто подразумевается. Под воздействием этой команды происходит алгебраическое сложение содержимого регистра третьего операнда (обычно отрицательного приращения, например —1) с содержимым регистра первого операнда, а затем алгебраическое сравнение полученной таким образом величины с содержимым следующего нечетно пронумерованного регистра, номер которого больше номера регистра [2]) третьего операнда. Если в результате сравнения выполняется одно из условий: «равно» или «меньше», то выбирается следующая по порядку команда; если же выполняется условие «больше», то происходит переход к команде, адрес которой определен вторым операндом.

Признак результата не изменяется

Команда Branch on Index Low or Equal
(Переход по индексу меньше или равно)

Мнемоника	Код операции	Формат операндов
BXLE	87	$R_1,R_3,D_2(B_2)$

Эта команда работает в некотором смысле подобно команде BXH и использует три регистра, номера которых выбираются аналогично тому, как это делается для команды BXH. Команда будет производить алгебраическое сложение содержимого регистра третьего операнда (обычно положительного приращения) с содержимым регистра первого операнда и сравнение получен-

[1]) Автор описывает типичное применение. Однако обязательным требованием является только нечетность номера регистра, содержащего заданную величину индекса, с которой сравнивается сумма первого и третьего операндов. — *Прим. ред.*

[2]) Или равен ему. — *Прим. ред.*

ной величины с содержимым следующего нечетно пронумерованного регистра. Если в результате сравнения выполняется одно из условий: «меньше» или «равно», то происходит переход к команде по адресу, заданному вторым операндом. Если же выполняется условие «больше», то будет выбираться следующая по порядку команда.

Признак результата не изменяется.

Команда Branch and Link (Переход с возвратом)

Мнемоника	Код операции	Формат операндов
BAL	45	$R_1,D_2(X_2,B_2)$

При выполнении команды BAL сначала в регистре первого операнда запоминаются младшие 32 бита обновленного PSW, в которых содержится адрес следующей по порядку команды программы. Затем происходит переход к команде по адресу, определенному вторым операндом.

Признак результата не изменяется.

Команда Branch and Link Registers (Переход с возвратом)

Мнемоника	Код операции	Формат операндов
BALR	05	R_1,R_2

При выполнении команды BALR в регистре первого операнда запоминаются младшие 32 бита обновленного PSW, в которых содержится адрес следующей по порядку команды, а затем производится переход по адресу, содержащемуся в регистре второго операнда. Если второй операнд определяет общий регистр с номером 0, то данные о PSW запоминаются, но переход не производится.

Признак результата не изменяется.

Б. ИСПОЛЬЗОВАНИЕ ПЕРЕХОДОВ В ПРОГРАММАХ

Термин «переход» можно определить как переход при выполнении программы от одной команды к другой, необязательно расположенной непосредственно за первой командой. Переход обычно имеет место в результате принятия решения, которое основывается на действии предыдущей команды. Вообще говоря, команды переходов можно разделить на две группы — условные переходы и переходы с возвратом.

Условные переходы служат для выбора одного из нескольких возможных путей, предусмотренных алгоритмом программы, в зависимости от признака результата или значения величины в

регистре. Программные переходы с возвратом обычно применяются для перехода к другой программе, подпрограмме или модулю, чтобы выполнить команды, содержащиеся в них, и затем возвратиться к оператору, следующему по порядку непосредственно за оператором, вызвавшим переход.

1. Команды условного перехода

Прежде чем обсуждать команды условного перехода, целесообразно ознакомиться с фактором, определяющим, будет ли команда условного перехода изменять обычную последовательность выполнения программы. Речь идет о признаке результата, определяемом 34 и 35 разрядами PSW. Эти две позиции позволяют задать четыре различные конфигурации — 00, 01, 10 и 11. В результате выполнения большинства команд языка Ассемблера устанавливается определенное значение признака результата. Команды условного перехода BC и BCR используют первый операнд в качестве маски, выделяющей одно или несколько заданных состояний признака результата в качестве условия выполнения перехода.

Конфигурация признака результата и соответствующие им значения масок показаны ниже.

Состояния битов 34—35 PSW	Эквивалентные операнды для команд BC и BCR
00 (0)	8
01 (1)	4
10 (2)	2
11 (3)	1

Значение маски в команде условного перехода может включать несколько величин, т. е. соответствовать одновременно нескольким значениям признака результата. Эта возможность детально обсуждается при рассмотрении в настоящем разделе команд условного перехода.

Применение условных переходов позволяет программисту использовать множество логических путей и заранее предопределять выбор дальнейшего пути на основании результатов обработки данных его программой. Например, если определенное числовое поле содержит нуль, то программист может захотеть перейти к некоторой стандартной программе, чтобы пропустить последующие арифметические операции; если поле содержит положительную величину, то программист может захотеть выбрать другой путь в алгоритме программы; если поле имеет отрицательное значение, программист может захо-

теть перед дальнейшим выполнением программы преобразовать число, находящееся в этом поле, в положительную величину.

Некоторые команды условного перехода, такие, как команды BCT и BXH, не имеют в предложении поля маски. Они сравнивают значение содержимого операнда с заданным численным значением и затем решают, нужно ли в результате этого сравнения производить переход.

Условный переход — BC. Команда BC осуществляет переход к команде, расположенной по адресу, который подразумевается вторым операндом, если признак результата в PSW соответствует маске в первом операнде команды. Маска может принимать одиночное или составное значение из четырех основных численных величин — 8, 4, 2 и 1. В зависимости от команды, установившей признак результата, каждая из этих четырех основных возможностей имеет разный смысл. Например, для команд сравнения кодов интерпретация определенных состояний битов признака результата и соответствующих значений маски для каждого состояния такова:

8 Операнды равны между собой
4 Первый операнд меньше второго
2 Первый операнд больше второго
1 Значение «1» для команд сравнения не используется

Команды, выполняющие арифметические операции, устанавливают значения признака результата, для которых эквивалентные значения маски обычно имеют следующий смысл:

8 Результат равен нулю
4 Результат отрицателен (—1 или меньше)
2 Результат положителен (+1 или больше)
1 Арифметическое переполнение

В случае когда необходимо предпринять переход к определенному адресу при осуществлении одного из нескольких возможных условий, маска команды условного перехода может составляться в виде произвольной комбинации величин 8-4-2-1.

На примере интерпретации признака результата для команд сравнения кодов можно видеть, что имеется целый ряд допустимых конфигураций кодов маски. Поскольку маска 8 означает «переход по условию равенства», легко видеть, что если необходимо проверить, оказался ли первый операнд в предшествующей команде сравнения меньше или равен второму, то для перехода по этому условию следует задать значение маски 12 в команде BC — 8 для условия «равен» и 4 для условия «меньше». Комбинации величин маски и их общепринятая интерпретация сведены в табл. 8.1.

Таблица 8.1

Значение маски	Интерпретация маски	
14	Переход по условию равно, больше или меньше	(8 + 4 + 2)
13	равно, меньше или переполнение	(8 + 4 + 1)
12	равно или меньше	(8 + 4)
11	равно, больше или переполнение	(8 + 2 + 1)
10	равно или больше	(8 + 2)
9	равно или переполнение	(8 + 1)
8	равно	(8)
7	не равно (больше, меньше или переполнение)	(4 + 2 + 1)
6	больше или меньше	(4 + 2)
5	меньше или переполнение	(4 + 1)
4	меньше	(4)
3	больше или переполнение	(2 + 1)
2	больше	(2)
1	переполнение	(1)

Как установлено ранее, точная интерпретация этих смешанных величин зависит от команд, которые устанавливают признак результата. Кроме перечисленных в таблице значений, в качестве первого операнда команды BC можно использовать значения маски 0 или 15. Когда в качестве первого операнда применяется 0, переход не происходит, так как для нулевого кода маски не существует соответствующего значения признака результата. Подобная запись рассматривается как предложение No Operation — NOP (Нет операции). Программист, однако, может впоследствии в любой момент времени в процессе обработки переслать в маску допустимую величину, и тогда действующим станет это новое условие перехода. Для осуществления *безусловного перехода*, т. е. перехода, который будет иметь место при любом состоянии битов признака результата, в качестве значения маски в команде BC можно задать 15. Это гарантирует переход, потому что любой признак результата будет соответствовать одной из величин, составляющих значение маски — 8+4+2+1.

Следующие примеры содержат последовательности команд CLC и BC:

Пример 1.

```
CLC     INPUT, HOLDATA
BC      7,NOGOOD
```

Предложение BC в этом примере означает следующее: «Если в результате предшествующего сравнения выработалось условие неравенства, т. е. первый операнд оказался больше или меньше второго операнда, выбрать следующую команду по адресу, соответствующему метке NOGOOD».

Пример 2.

```
CLC     OUTPUT,HOLD1
BC      8,PROCESS
```

Предложение BC имеет следующий смысл: «Если два операнда в предшествующем операторе сравнения оказались равными друг другу, произвести переход к программе с меткой PROCESS».

Пример 3.

```
CLC     FLD1,FLD2
BC      8,MOVEIT
BC      4,SUBTR
BC      2,ADDIT
```

Это пример выбора одного из ряда возможных путей в программе, основанного на результатах одной операции сравнения.

Первое предложение BC утверждает: «Если операнды равны, перейти к программе с меткой MOVEIT».

Во втором предложении BC говорится: «Если FLD1 имеет меньшее логическое значение, чем FLD2, перейти к программе с меткой SUBTR».

Третье предложение BC означает следующее: «Если FLD1 логически больше, чем FLD2, перейти к выполнению программы, помеченной ADDIT». Так как команда CLC в любом случае установит признак результата в состояние, соответствующее одному из значений маски, заданных в последующих командах BC (8, 4 или 2), то один из переходов этого набора команд будет иметь место.

Пример 4.

```
BC 15, GOTHERE
```

Это предложение можно записать безотносительно к тому, какая команда непосредственно ему предшествовала. Оно фактически утверждает следующее: «Независимо от конфигурации признака результата перейти к команде или программе, имеющей метку GOTHERE». Такой переход называется *безусловным переходом.*

Условный переход — BCR. Эта команда почти идентична кокоманде BC, за исключением того, что переход, если он будет

иметь место, произойдет по адресу, содержащемуся в регистре, который определяется вторым операндом. Команда будет сопоставлять первый операнд (маску) с признаком результата, и, если соответствующие условия выполняются, в итоге будет иметь место переход. Все общие правила, применимые к команде BC, применимы также и здесь.

Для иллюстрации ниже приведено несколько примеров, в которых за различными командами десятичной арифметики следуют команды BCR.

Пример 1.

```
CP      PAKFLD1,PAKFLD2
BCR     10,2
BCR     4,3
```

Первая команда сравнивает два поля упакованных десятичных данных. Первое предложение BCR утверждает следующее: «Если оператор CP выявил, что PAKFLD1 равно (8) или больше, чем (2) PAKFD2, перейти по адресу, находящемуся в общем регистре 2». Второе предложение BCR утверждает: «Если PAKFLD1 меньше, чем PAKFLD2, перейти по адресу, содержащемуся в общем регистре 3».

Пример 2.

```
LA      9,BADADDR
LA      10,ZEROSUM
LA      11,NEGSUM
AP      SUMPK,INPAK
BCR     8,10
BCR     4,11
BCR     1,9
```

Первые три команды загружают адреса трех отдельных программ в общие регистры 9, 10 и 11. Команда Add Packed — AP (Сложение) складывает два поля с упакованными десятичными величинами, причем результирующая сумма остается в поле SUMPK. Первое предложение BCR говорит: «Если общая сумма, хранящаяся в SUMPK, равна нулю, выбрать следующую команду по адресу, находящемуся в общем регистре 10». Этим адресом является адрес программы, названной ZEROSUM, который загружен в общий регистр 10 второй командой LA (Загрузка адреса). Второе предложение BCR указывает, что, «если общая сумма, хранящаяся в SUMPK, отрицательна (меньше чем —0), следующая команда выбирается по адресу, находя-

щемуся в общем регистре 11». Это адрес программы NEGSUM, который был загружен третьей командой LA. Третье предложение BCR утверждает: «Если во время сложения этих полей произойдет переполнение, следующая команда будет выбрана по адресу, находящемуся в общем регистре 9». Это адрес программы BADADDR, загруженный в общий регистр 9 первой командой LA. В этом примере также предусмотрено, что, если результат десятичного сложения окажется положительной величиной, при выполнении всех команд BCR перехода не будет и программа перейдет к выполнению следующей за ними команды. Это ясно из того, что переходы в примере обнаруживают условия равенства нулю, отрицательности результата и переполнения.

Пример 3.

```
BCR     15,8
```

Это предложение вызывает безусловный переход, и следующая команда выбирается по адресу, находящемуся в общем регистре 8. Первый операнд требует перехода независимо от значения признака результата. Перед выполнением команды в общий регистр 8 должен быть загружен адрес программы, к которой должен производиться переход.

Это один из способов возврата из программы, переход к которой был произведен по команде BAL, определившей общий регистр 8 в качестве регистра возврата.

Переход по счетчику — BCT. Команда вызывает переход к команде по адресу, определенному вторым операндом, если величина, находящаяся в регистре первого операнда, не равна нулю. Команда BCT часто используется для просмотра таблиц, организации циклов и управления счетом. Каждый раз, когда при выполнении программы встречается эта команда, происходит следующее:

1. Величина в регистре первого операнда уменьшается на величину с фиксированной точкой, равную 1.
2. Проверяется, равно ли нулю содержимое общего регистра.
3. Если содержимое регистра отлично от нуля, следующая команда выбирается по адресу, определенному вторым операндом предложения.
4. Если содержимое регистра равно нулю, переход не происходит и выбирается следующая по порядку адресов команда.

Программист может перезагружать регистр операнда любой подходящей величиной для повторного использования в соответствии с алгоритмом программы.

Пример использования этой команды приведен ниже:

```
        SR      5,5                 001
        LA      5,100               002
LOOP    AP      PAKSUM,=PL1'3'      003
        BCT     5,LOOP              004
        ZAP     BIGSUM,PAKSUM       005
```

Осуществляемое этой программой перемножение величин можно было бы выполнить одной командой. Однако этот пример дает простую иллюстрацию использования команды BCT.

Предложение 001 вычитает содержимое общего регистра 5 само из себя, тем самым очищая регистр.

Предложение 002 загружает абсолютную величину 100 в общий регистр 5.

Предложение 003 складывает однобайтовую упакованную десятичную величину +3, представленную в форме литерала, с содержимым области, выполняющей функции накопителя.

Предложение 004 уменьшает содержимое регистра 5 на единицу при каждом выполнении этой команды и затем осуществляет переход к предложению 003.

Это продолжается до тех пор, пока содержимое общего регистра 5 не уменьшится до нуля. На этот раз перехода к LOOP не будет и программа перейдет к предложению 005.

Так как первоначально в общий регистр 5 была загружена величина +100, программа в процессе своей работы пройдет через предложение LOOP, включая и самый первый проход, 100 раз; при каждом выполнении предложения LOOP к накопителю PAKSUM будет прибавляться упакованная величина +3. Когда команда BCT выполнится в сотый раз, содержимое общего регистра 5 станет равно нулю и переход не произойдет. Поле PAKSUM содержит теперь упакованную десятичную величину +300 (100 циклов, в каждом из которых в накопитель добавлялась величина +3).

Переход по счетчику — BCTR. Команда BCTR вызывает переход к выполнению команды, расположенной по адресу, определенному вторым операндом, если величина, находящаяся в регистре первого операнда, не равна нулю. Всякий раз, когда при выполнении программы встречается эта команда, происходит следующее:

1. Величина в общем регистре первого операнда уменьшается на 1.
2. Содержимое этого общего регистра сравнивается с нулем.
3. Если содержимое отлично от нуля, следующая выполняемая команда выбирается по адресу, содержащемуся в регистре второго операнда.

4. Если содержимое регистра первого операнда становится равным нулю, переход не происходит и программа выполняет следующую по порядку адресов команду.

Основные возможности команды BCTR иллюстрируются примером организации цикла, аналогичным приведенному выше для команды BCT. В этом примере, однако, используется метод построения «цикла в цикле» (рис. 8.1).

Здесь первые четыре предложения с 001 до 004 «чистят» общие регистры, которые должны использоваться в предложениях BCTR. Внешним циклом (LOOPONE) управляют общие регистры 6 и 8, внутренним (LOOPTWO) — регистры 7 и 9.

Предложение 005 загружает абсолютную величину 50 в общий регистр 6 — регистр управления циклом LOOPONE.

Предложение 006 загружает адрес программы LOOPONE в общий регистр 8; к этому адресу перехода впоследствии обращается предложение 013. Эти два предложения создали базу для управления внешним циклом LOOPONE.

Предложение 007 загружает абсолютную величину 100 в общий регистр 7 — регистр управления циклом для LOOPTWO.

Предложение 008 загружает адрес LOOPTWO в общий регистр 9; к этому адресу перехода впоследствии обращается предложение 010.

Предложение 009 прибавляет упакованную десятичную величину +2, представленную в форме литерала, к области, служащей накопителем для цикла LOOPTWO, метка которой SMALPAK.

Предложение 010 — команда BCTR для LOOPTWO. При каждом входе в программу LOOPTWO она выполняется 100 раз, после чего происходит обращение к предложению 011.

Предложение 011 прибавляет упакованное десятичное содержимое накопителя для LOOPTWO (SMALPAK) к накопителю для цикла LOOPONE. При каждом выполнении этого предложения к накопителю BIGPAK добавляется величина +200, так как в итоге 100 циклов прохождения через LOOPTWO содержимое области SMALPAK будет равно +200 (100 циклов, в каждом из которых в накопитель добавлялась величина +2).

Предложение 012 использует команду ZAP, чтобы очистить SMALPAK перед очередным входом в цикл.

Предложение 013 — команда BCTR для внешнего цикла LOOPONE. Так как предложение 005 загрузило в общий регистр 6 абсолютную величину +50, эта команда выполняется 50 раз, после чего происходит обращение к предложению 014.

Предложение 014 осуществляет безусловный переход к подпрограмме, которая служит для завершения выполнения основной программы.

IBM

IBM System/360 Assembler Coding Form

GX28-6508-7 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | CARD ELECTR. NUMBER | PAGE OF

PROGRAMMER | DATE

STATEMENT

Name	Operation	Operand	Comments	Identification Sequence
*				
LOOPRTE	SR	6,6		001
	SR	7,7		002
	SR	8,8		003
	SR	9,9		004
	LA	6,50		005
	LA	8,LOOPONE		006
LOOPONE	LA	7,100		007
	LA	9,LOOPTWO		008
LOOPTWO	AP	SMALPAK,=PL1'2'		009
	BCTR	7,9		010
	AP	BIGPAK,SMALPAK		011
	ZAP	SMALPAK,=PL1'0'		012
	BCTR	6,8		013
	BC	15,FINIS		014
*				

Рис. 8.1.

После завершения программы упакованная десятичная величина, накопленная в BIGPAK, равна +10000. Интерпретируя параметры циклов, это значение можно определить следующим образом:

LOOPONE умножить на (LOOPTWO умножить на 2) равно BIGPAK,

или

50 умножить на (100 умножить на 2) равно 10000.

На этом примере удобно показать, как пропуск по ошибке одного оператора может повлиять на решение. Если бы программист пропустил оператор, который очищает накопитель внутреннего цикла после каждой сотни циклов (предложение 12), то результирующее значение BIGPAK при завершении всей программы оказалось бы равным 255000. Исключите предложение 012 и проверьте этот результат.

Команда BCTR может также использоваться для уменьшения значения общего регистра без перехода, для чего достаточно в качестве регистра второго операнда определить регистр 0. Например, выполнение оператора BCTR 6,0 уменьшает содержимое общего регистра 6 на единицу, но не приводит к переходу независимо от текущего значения регистра 6.

Переход по индексу больше — BXH. Команда BXH использует три регистра: регистр-накопитель, регистр приращений, регистр сравниваемой величины. Первые два из них представлены в операндах команды наряду с еще одним операндом — символическим или действительным адресом перехода.

Каждый раз, когда выполняется предложение BXH, происходит следующее:

1. Содержимое второго общего регистра (как положительное или отрицательное приращение) складывается с содержимым первого общего регистра; результат помещается в первый регистр.
2. Новое значение содержимого первого общего регистра сравнивается со значением содержимого третьего общего регистра.
3. Если в результате сравнения величина в первом регистре оказывается больше величины в третьем регистре, то происходит переход в соответствии с адресом, заданным в предложении.
4. Если в результате сравнения величина в первом регистре оказывается меньше величины в третьем регистре или равна ей, перехода не будет и произойдет обращение к следующей по порядку адресов команде программы.

При практическом применении этой команды для удобства и единообразия принято использовать три последовательных общих регистра — с нечетным, четным и снова нечетным номерами соответственно.

В следующем примере (рис. 8.2) предполагается, что программист имеет в своем распоряжении таблицу, содержащую 50 элементов, каждый длиной 4 байта. Таким образом вся таблица имеет длину 200 байтов. Если первым символом в элементе является символ A, то программист намеревается ввести символ D во вторую позицию этого элемента таблицы; в противном случае он не вносит изменений в таблицу.

Предложение 001 загружает адрес таблицы TABLE, увеличенный на 49 элементов таблицы (или 196 байтов), в общий регистр 12. Это делается потому, что поиск в таблице программист собирается начать с самой старшей позиции в памяти и, постепенно уменьшая адрес, прийти к начальной точке таблицы. Регистр 12 должен использоваться как базовый, который будет непосредственно указывать на элементы TABLE.

Предложение 002 загружает в общий регистр 3 величину +196; в общий регистр 4 величину —4 и в общий регистр 5 величину —4.

Предложение 003 — команда CLI сравнивает первый байт элемента таблицы, текущей адрес которого находится в регистре 12, с символом A.

Предложение 004 проверяет признак результата, установленный командой CLI в предложении 003. Если обнаружено условие неравенства, происходит переход к предложению 006. Если признак результата показывает, что при сравнении было обнаружено равенство операндов, происходит обращение к предложению 005, т. е. к следующему по порядку предложению.

Предложение 005 представляет команду MVI (Пересылка непосредственная), пересылающую символ D во второй байт элемента таблицы, текущий адрес которого находится в регистре 12.

Предложение 006 — команда AR (Сложение) складывает содержимое общего регистра 5 (величина —4) с текущим адресом, находящимся в общем регистре 12. Это действие уменьшает значение содержимого общего регистра 12 на 4 байта, так что в следующий раз при выполнении оператора RELOOK регистр 12 будет содержать адрес следующего младшего элемента таблицы.

Предложение 007 — команда BXH. Каждый раз при ее выполнении значение в общем регистре 3 изменяется на величину содержимого общего регистра 4, а результат сравнивается с содержимым общего регистра 5. Если регистр 3 содержит величину, большую, чем величина в регистре 5, то происходит

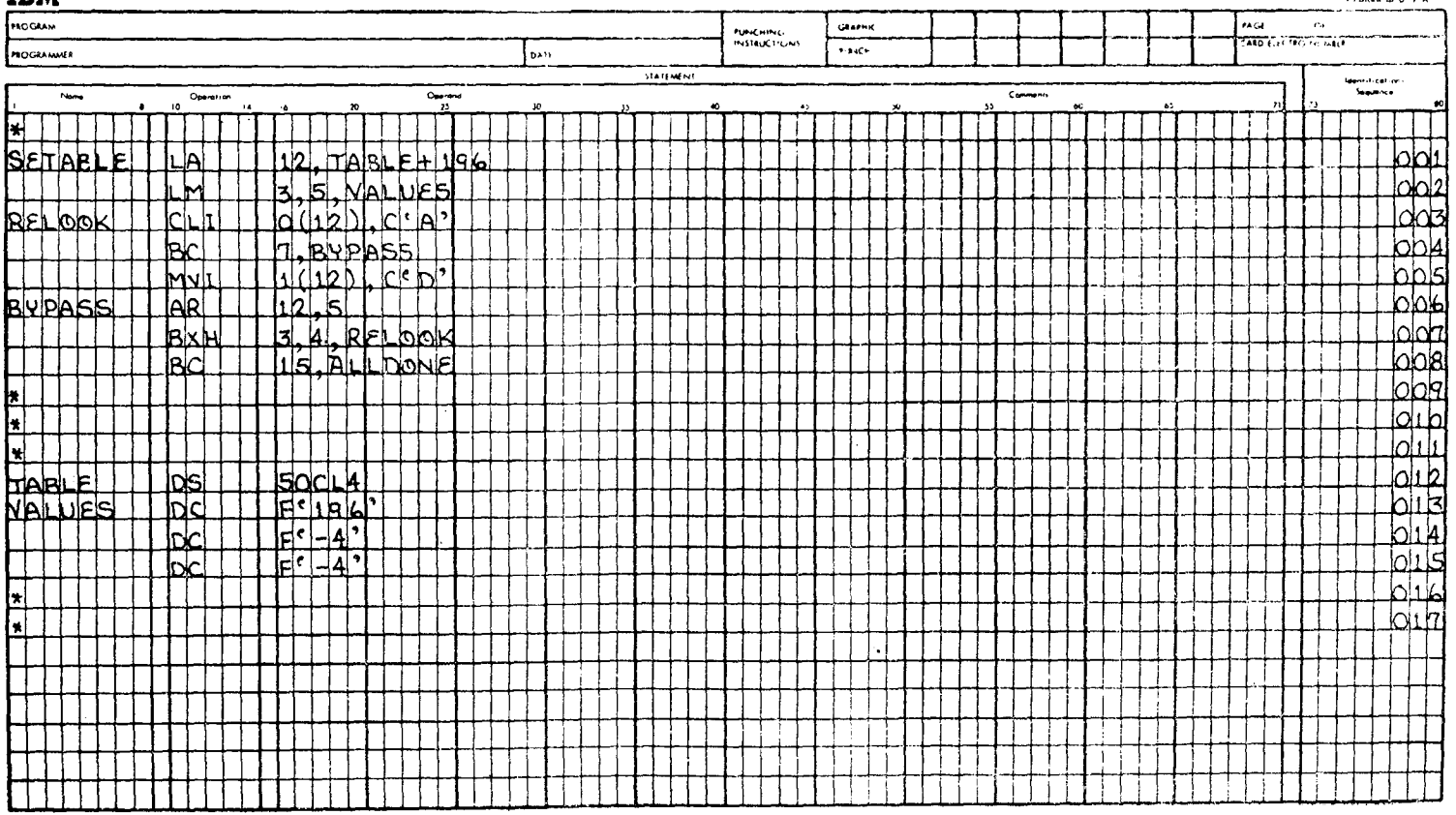

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
*
SETABLE LA 12,TABLE+196 001
LM 3,5,VALUES 002
RELOOK CLI 0(12),C'A' 003
BC 7,BYPASS 004
MVI 1(12),C'D' 005
BYPASS AR 12,5 006
BXH 3,4,RELOOK 007
BC 15,ALLDONE 008
* 009
* 010
* 011
TABLE DS 50CL4 012
VALUES DC F'196' 013
DC F'-4' 014
DC F'-4' 015
* 016
* 017

Рис. 8.2.

возврат к оператору RELOOK. Когда величина, находящаяся в регистре 3, примет значение —4, будет зафиксировано условие равенства содержимому регистра 5 (в котором находится —4), и перехода не будет. Затем выполняется следующее по порядку предложение.

Предложение 008 осуществляет безусловный переход к предполагаемой программе, которая, судя по всему, должна завершить выполнение этой части программы.

Подставим в этот пример действительные десятичные адреса ячеек памяти, для чего предположим, что первый байт TABLE имеет адрес 15000. В этом случае TABLE займет байты с 15000-го по 15199-й, всего 200 байтов.

Предложение 001 загружает адрес 15196 в общий регистр 12. Это адрес первого байта последнего четырехбайтового элемента таблицы.

Предложение 003 проверяет байт с адресом 15196 на содержание в нем символа A. Если зафиксировано условие равенства, предложение 005 пересылает символ D по адресу 15197 — во второй байт только что просмотренного четырехбайтового элемента таблицы.

Предложение 006 прибавляет —4 (вычитает величину 4 из 15196) к общему регистру 12, в результате чего в регистре 12 окажется теперь величина 15192.

Предложение 007 — команда BXH, прибавляет —4 к общему регистру 3 (по существу вычитает 4 из 196), сравнивает новое значение регистра 3, равное 192, со значением регистра 5 (величиной —4), после чего происходит переход к RELOOK — предложению 003.

Предложение 003 будет теперь проверять байт по адресу 15192 на наличие в нем символа A. Этот цикл будет повторяться до тех пор, пока величина в общем регистре 3 при выполнении команды BXH не станет равна величине в регистре 5 или меньше ее.

Переход по индексу меньше или равно — BXLE. Оператор команды BXLE использует операнды точно такого же типа, что и команда BXH (три регистра и символический или действительный адрес перехода). Функции, которые выполняют регистры, также аналогичны — первый регистр выступает в роли накопителя, второй регистр содержит величину приращения, а третий содержит значение, с которым должно сравниваться содержимое первого регистра. Здесь тоже предполагается, что три общих регистра имеют последовательные номера, причем первый из них нечетный.

Выполнение этой команды осуществляется следующим образом:

1. Величина во втором регистре (регистр приращений) суммируется с величиной в первом регистре, результат помещается в первый регистр.

2. Затем новое значение в первом регистре сравнивается с величиной, находящейся в третьем регистре.

3. Если значение в первом регистре при сравнении оказывается меньше величины значения в третьем регистре или равно ему, происходит переход к команде, расположенной по адресу, указанному вторым операндом.

4. Если значение величины в первом регистре при сравнении оказывается больше значения величины в третьем регистре, то переход не производится и выполняется следующая по порядку команда.

Рассмотрим использование этой команды на следующем примере. Предположим, что программист имеет 75-байтовую таблицу, по одному байту на каждый элемент, которую он хочет переместить в другую область, расположив в обратном порядке. Распределение памяти, величины, используемые общими регистрами, и операторы команд приведены на рис. 8.3.

Предложение 001 загружает три регистра по команде LM (Загрузка групповая). Команда LM загружает в общий регистр 5 значение нуль, представленный в формате полного слова с фиксированной точкой, в общий регистр 6 величину +1 для приращений, а в общий регистр 7 величину +74, которая будет использоваться как эталон для сравнения.

Предложение 002 загружает адрес TABLE1, увеличенный на 74 байта, в общий регистр 8. Общий регистр 8 теперь содержит адрес самого старшего байта таблицы TABLE1.

Предложение 003 загружает адрес TABLE2 в общий регистр 9. Это адрес самого младшего байта таблицы TABLE2. Регистры 8 и 9 теперь содержат противоположные по значению границы для TABLE1 и TABLE2, чтобы данные, находящиеся в TABLE1, могли быть расположены в TABLE2 в обратном порядке.

Предложение 004 пересылает 1 байт данных из TABLE1, адресованный общим регистром 8, в TABLE2 по адресу, определенному общим регистром 9.

Предложение 005 прибавляет значение содержимого общего регистра 6 (равное +1) к содержимому общего регистра 9. Регистр 9 содержит теперь адрес текущего элемента таблицы TABLE2.

Предложение 006 вычитает значение общего регистра 6 (равное +1) из общего регистра 8. Общий регистр 8 содержит теперь адрес текущего элемента таблицы TABLE1.

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M360
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE | OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
MIXTABLE	LM	5,7,REGVALS		001
	LA	8,TABLE1+74		002
	LA	9,TABLE2		003
REPEAT	MVC	0(1,9),0(8)		004
	AR	9,6		005
	SR	8,6		006
	BXLE	5,6,REPEAT		007
	BC	15,MOVEDIT		008
*				009
*				010
*				011
TABLE1	DS	CL75		012
TABLE2	DS	CL75		013
REGVALS	DC	F'0'		014
	DC	F'1'		015
	DC	F'74'		016
*				017
*				018

Рис. 8.3.

Предложение 007 — команда BXLE. При каждом выполнении этой команды значение величины в общем регистре 5 увеличивается на величину содержимого регистра 6 и результат сравнивается со значением содержимого общего регистра 7. Если содержимое общего регистра 5 при сравнении оказывается меньше содержимого общего регистра 7 или равно ему, производится переход к REPEAT. Однако, когда значение регистра 5 увеличивается таким образом, что при сравнении окажется больше содержимого регистра 7, перехода не произойдет и следующим будет выполняться предложение 008.

Предложение 008 представляет собой безусловный переход к другой части основной ветви алгоритма программы. Этот переход осуществляется после завершения всех циклов подпрограммы, включающей предложения с 001 по 007.

Для конкретности предположим, что заданы следующие действительные десятичные адреса:

TABLE1	Ячейки памяти от 32000 до 32074
TABLE2	Ячейки памяти от 32075 до 32149

Функции операторов предыдущей программы в этом случае состоят в следующем:

Предложение 001 функционирует, как описано ранее.

Предложение 002 загружает в общий регистр 8 адрес 32074. Эта величина представляет собой адрес TABLE1 (32000), увеличенный на 74, или адрес последнего однобайтового элемента TABLE1.

Предложение 003 загружает в общий регистр 9 адрес 32075 первого элемента таблицы TABLE2.

Предложение 004 пересылает 1 байт данных из ячейки с адресом 32074 по адресу 32075. Это соответствует пересылке значения из последнего элемента TABLE1 в первый элемент TABLE2.

Предложение 005 прибавляет значение величины в общем регистре 6 (величину +1) к текущему содержимому общего регистра 9 (+32075), в результате чего общий регистр 9 теперь содержит +32076.

Предложение 006 вычитает содержимое общего регистра 6 (+1) из содержимого общего регистра 8 (+32074), в результате чего общий регистр 8 теперь содержит +32073. После выполнения предложения 007 общий регистр 5 содержит теперь значение +1, общий регистр 6 содержит +1, а общий регистр 7— величину +74. В итоге будет совершаться переход к предложению 004 REPEAT.

Предложение 004 пересылает 1 байт данных (предпоследний элемент таблицы TABLE1) из ячейки с адресом 32073 по адресу 32076 второго элемента TABLE2.

Предложение 005 увеличивает адрес, находящийся в общем регистре 9, до +32077.

Предложение 006 уменьшает адрес, находящийся в общем регистре 8, до +32072.

Этот процесс будет продолжаться циклически от оператора REPEAT до оператора команды BXLE до тех пор, пока общий регистр 5 не будет содержать величину +75, большую величины +74, содержащейся в общем регистре 7. После чего следующим будет выполняться предложение 008.

2. Команды перехода с возвратом

Команды, которые относятся к этой группе, могут рассматриваться как «принудительные» переходы, которые по усмотрению программиста дают возможность проблемной программе вернуться к следующему по порядку оператору. Их можно использовать для организации связи между программами, подпрограммами, модулями или любыми их комбинациями. Наиболее частое использование команд перехода с возвратом заключается в обращении к общей подпрограмме из многих точек основной программы. Это дает возможность программисту писать подпрограмму только однажды и затем при необходимости вызывать ее, вместо того чтобы повторять те же самые операторы во многих местах своей программы.

Переход с возвратом — BAL. По этой команде в общем регистре первого операнда запоминается адрес следующей команды программы, который содержится в обновленном PSW, затем в PSW заносится адрес, определяемый вторым операндом, после чего происходит переход в соответствии с этим адресом. После того как получившая управление программа закончит работу, программист может вернуться к команде, следующей непосредственно за командой BAL, с помощью команды безусловного перехода по адресу, находящемуся в общем регистре первого операнда команды BAL. Такой метод возврата основывается на предположении, что команды получившей управление программы не изменяют содержимого регистра возврата.

Предположим, что программисту требуется написать простую математическую подпрограмму с применением десятичной арифметики, к которой многократно обращается основная программа. Поля данных и коды команд программы приведены на рис. 8.4.

Подразумеваемая последовательность выполнения этих операторов выглядит так: 001, 002, 085, 086, 003, —, 010, 011, 085, 086, 012, —, 017, 018, 085, 086, 019, 020.

Предложение 001 сначала заполняет поле PAKANSR нулями, затем помещает в него значение PACK1.

IBM

IBM System/360 Assembler Coding Form

X33-6503-3 U/M050
Printed in U.S.A.

PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE OF

STATEMENT

Name	Operation	Operand	Comments	Sequence
ROUTE1	ZAP	PAKANSR,PACK1		001
	BAL	10,DOMULT		002
	AP	CUMULAT,PAKANSR		003
	ZAP	PAKANSR,PACK2		010
	BAL	10,DOMULT		011
	AP	CUMULAT,PAKANSR		012
	ZAP	PAKANSR,PACK3		017
	BAL	10,DOMULT		018
	AP	CUMULAT,PAKANSR		019
	BC	15,DUNCA+C		020
*				
DOMULT	MP	PAKANSR,PAKMULT		085
	BCR	15,10		086
*				087
*				088
PACK1	DC	PL3'0'		089
PACK2	DC	PL3'0'		090
PACK3	DC	PL3'0'		091
CUMULAT	DC	PL6'0'		092
PAKANSR	DC	PL4'0'		093
PAKMULT	DC	PL1'5'		094
*				095

Рис. 8.4

Предложение 002 вызывает переход с возвратом к предложению 085.

Предложение 085 умножает содержимое PAKANSR, равное в этот момент содержимому PACK1, на PAKMULT (величина +5 в упакованном десятичном формате).

Предложение 086 вызывает переход обратно к предложению 003. Это происходит потому, что команда BAL загрузила адрес предложения 003 в общий регистр 10.

Предложение 003 прибавит новое значение PAKANSR к величине, содержащейся в накопителе CUMULAT.

Предложения с 004 до 009 не показаны, но подразумевается, что они служат для получения записей данных, дополнительной обработки и т. д.

Предложение 010 заполняет поле PAKANSR нулями, затем помещает в него значение PACK2.

Предложение 011 вызывает переход с возвратом к предложению 085, загружая в то же самое время адрес предложения 012 в общий регистр 10.

Предложение 085 умножает содержимое PAKANSR, равное в этот момент содержимому PACK2, на величину +5, находящуюся в поле PAKMULT.

Предложение 086 вызывает переход к оператору 012, адрес которого находится в этот момент в общем регистре 10.

Предложение 012 прибавляет новое значение PAKANSR к накопителю CUMULAT.

Подразумевается, что предложения с 013 по 016 обрабатывают дополнительные данные и т. п.

Предложение 017 заполняет поле PAKANSR нулями, а затем помещает в него значение PACK3.

Предложение 018 — команда BAL — загружает адрес предложения 019 в общий регистр 10 и затем вызывает переход к предложению 085.

Предложение 085 умножает содержимое PAKANSR, равное в этот момент содержимому PACK3, на величину +5, находящуюся в поле PAKMULT.

Предложение 086 вызывает переход обратно к предложению 019, адрес которого загружен в общий регистр 10.

Предложение 019 прибавляет новое значение PAKANSR к накопителю CUMULAT.

Предложение 020 вызывает переход к другой части основной программы.

Хотя этот набор команд вовсе необязательно является самым лучшим средством достижения поставленной цели, он достаточно хорошо иллюстрирует применение команд перехода с

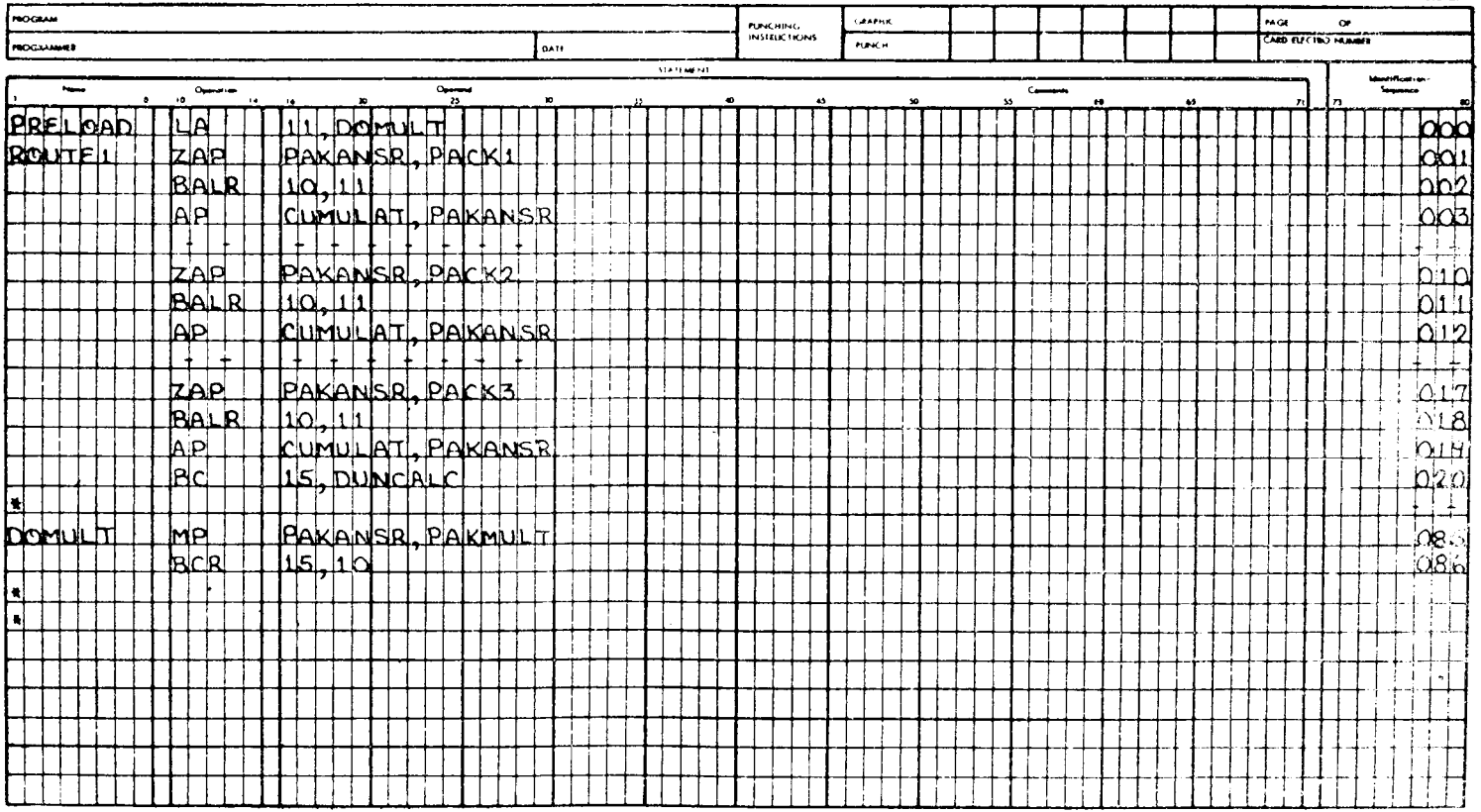

IBM
IBM System 360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
PRELOAD LA 11,DOMULT 000
ROUTE1 ZAP PAKANSR,PACK1 001
BALR 10,11 002
AP CUMULAT,PAKANSR 003
ZAP PAKANSR,PACK2 010
BALR 10,11 011
AP CUMULAT,PAKANSR 012
ZAP PAKANSR,PACK3 017
BALR 10,11 018
AP CUMULAT,PAKANSR 01H
BC 15,DUNCALC 020
DOMULT MP PAKANSR,PAKMULT 085
BCR 15,10 086

Рис. 8.5.

возвратом и средства возвращения к последовательности команд, выдавшей команду BAL.

Переход с возвратом — BALR. Команда BALR в основном выполняет те же самые функции, что и команда BAL, за исключением того, что переход осуществляется по адресу, который содержится в регистре, определенном вторым операндом команды. Точнее, при выполнении команды адрес следующей по порядку адресов команды запоминается в регистре первого операнда и затем в PSW помещается адрес перехода, находящийся в регистре второго операнда. На рис. 8.5 представлена программа, подобная рассмотренной в предыдущем примере (рис. 8.4), иллюстрирующая применение команды.

Основное различие между этим примером и аналогичным примером для команды BAL заключается в операторе PRELOAD, который загружает адрес DOMULT в общий регистр 11. Впоследствии команды BALR используют адрес в общем регистре 11 как адрес перехода к DOMULT.

Упражнения

1. В общем команды переходов можно разбить на две группы: условные переходы и _____ переходы.

2. Единственными командами перехода, использующими в качестве первого операнда маску, являются команды _____ и _____.

3. Команды BC и BCR, содержащие в качестве маски величину _____, рассматриваются как команды принудительного или _____перехода.

4. Четырьмя отдельными значениями маски, которые по отдельности или совместно используются для проверки признака результата в PSW, являются _____, _____, _____ и _____.

5. Каждый раз, когда выполняется команда BCT, величина, содержащаяся в _____операнде, будет автоматически уменьшаться на величину_____.

6. Для каждого из следующих наборов команд укажите отношение сравнения между двумя операндами в первом операторе, при котором выполнение второго оператора приведет к переходу. (Это отношение может выглядеть так: «первый операнд больше второго или равен ему» и т. п.)

```
а)      COMPA   CL      8,FLWDA
                BC      12,ROUTE1
б)      COMPC   CH      3,HALFWDA
                BC      6,ROUTE2
```

```
в)      COMPE   CP      PACKFLDA,=PL3'8952'
                BC      10,ROUTE3
г)      COMPG   CLC     DATAFLD+3(7),INCARD+20
                BC      2,ROUTE4
д)      COMPI   CLI     FIELD6+3,X'40'
                BC      8,BYPASS
е)      COMPK   CLR     10,4
                BC      14,NEWROUTE
ж)      COMPM   C       3,FULLWDG
                BC      2,ROUTE5
з)      COMPN   CR      3,13
                BC      4,ROUTE6
```

7. Максимальная величина, которая может быть представлена первым операндом команды BCT, равна _____.

8. В предложении BCTR адрес перехода находится в _____.

9. Каждая из команд BXH и BXLE всего использует _____ регистра.

10. Переход по команде BCT не произойдет, если при ее выполнении величина в регистре первого операнда уменьшится до значения _____.

11. Команда BAL загружает в первый регистр _____ следующей команды.

12. Предложение BALR содержит два операнда, причем оба они указывают _____.

13. Команду «Нет операции» можно сформировать, закодировав значение маски _____ в качестве первого операнда команд BC или BCR.

14. В предположении, что общий регистр 7 перед выполнением приведенной ниже команды содержит величину с фиксированной точкой +75, определите, какая величина с фиксированной точкой будет содержаться в регистре 7 после е выполнения.

```
LOOPUP  BCTR 7,9
```

15. Предположим следующее:

Регистр 5 содержит величину с фиксированной точкой +3275.
Регистр 6 содержит величину с фиксированной точкой —25.
Регистр 7 содержит величину с фиксированной точкой +50.

После выполнения приведенной ниже команды регистр 5 будет содержать величину с фиксированной точкой _____, ре-

гистр 6 — величину с фиксированной точкой _____, регистр 7 — величину с фиксированной точкой _____.

```
BXH     5,6,RESEEK
```

16. Если перед выполнением предложения BRNCH общий регистр 9 содержал величину с фиксированной точкой +32500, каково будет его содержимое сразу же после выполнения предложений

```
BRNCH   BAL     9,EXITRTE
BACK    CLI     SWITCH,C'A'
```

17. Если общий регистр 10 содержит величину +1 перед выполнением команды BCT предложения ENDLOOP, после команды BCT будет выполняться предложение _____.

```
ENDLOOP   BCT     10,RELOOP
CONTIN    CLC     DATA,INPUT
```

18. В каждом из следующих наборов команд будет иметь место переход. Основываясь на заданных значениях сравниваемых полей, укажите метку предложения, к которому будет совершен переход.

а) Регистр 8 содержит величину с фиксированной точкой —3598; регистр 12 содержит величину с фиксированной точкой +15782.

```
COMPARA   CR    8,12
          BC    4,SUBRT1   _____
          BC    8,SUBRT2   _____
          BC    3,SUBRT3   _____
```

б) Регистр 9 содержит величину с фиксированной точкой —783; FLWD2 содержит величину с фиксированной точкой +6.

```
COMPARB   CL    9,FLWD2
          BC    4,SUBRT4   _____
          BC    2,SUBRT5   _____
          BC    9,SUBRT6   _____
```

в) PAKFLDA содержит упакованную десятичную величину —0.

```
COMPARF   CP    PAKFLDA,=P'0'
          BC    8,SUBRT7   _____
          BC    7,SUBRT8   _____
```

г) FIELD1 содержит символы кода EBCDIC 'S1FA'; FIELD2 содержит символы кода EBCDIC 'S12A'.

```
COMPARG   CLC   FIELD1,FIELD2
          BC    10,SUBRT9     _____
          BC    4,SUBRT10     _____
          BC    15,SUBRT11    _____
```

д) SETBYTE содержит символ кода EBCDIC'X'.

```
COMPARH   CLI   SETBYTE,X'4O'
          BC    6,SUBRT12A    _____
          BC    8,SUBRT12B    _____
```

е) Регистр 8 содержит величину с фиксированной точкой +9810516; регистр 10 содержит величину с фиксированной точкой +5.

```
COMPARJ   CLR   8,10
          BC    15,SUBRT14    _____
          BC    1,SUBRT15     _____
          BC    4,SUBRT16     _____
```

ж) Регистр 9 содержит величину с фиксированной точкой +3516; HAFWORD6 содержит величину с фиксированной точкой −3516.

```
COMPARK   CH    9,HAFWORD6
          BC    2,SUBRT17     _____
          BC    8,SUBRT18     _____
          BC    4,SUBRT19     _____
```

Глава 9

Пересылка, запись в память и загрузка данных

А. КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

Команда Move Immediate — MVI (Пересылка)

Мнемоника	Код операции	Формат операндов
MVI	92	$D_1(B_1),I_2$

Эта команда пересылает байт непосредственных данных, заданных вторым операндом, по адресу, указанному первым операндом. Независимо от неявной длины первого операнда эта команда пересылает только 1 байт данных.

Признак результата не изменяется.

Команда Move Characters — MVC (Пересылка символов)

Мнемоника	Код операции	Формат операндов
MVC	D2	$D_1(L,B_1),D_2(B_2)$

Команда MVC пересылает данные из области памяти, адрес которой указан вторым операндом, в область памяти, определенную первым операндом. Данные пересылаются байт за байтом слева направо, причем общее количество пересылаемых байтов определяется явной или неявной длиной первого операнда. Одна команда может пересылать до 256 байтов данных.

Признак результата не изменяется.

Команда Move Numeric — MVN (Пересылка цифр)

Мнемоника	Код операции	Формат данных
MVN	D1	$D_1(L,B_1),D_2(B_2)$

Команда MVN пересылает только правые 4 бита каждого байта (правый полубайт), которые в зонном десятичном формате содержат цифровую часть данных. Правый полубайт каждого байта данных из области памяти, указанной вторым операндом, пересылается в соответствующее место каждого байта области памяти, заданной первым операндом. Правильность значений пересылаемых полубайтов не проверяется; поэтому с помощью этой команды можно пересылать как цифры в диапазоне от 0 до 9, так и шестнадцатеричные цифры A, B, C, D, E и F. Данные пересылаются по порядку слева направо по полу-

байту, причем общее количество пересылаемых полубайтов определяется явной или неявной длиной первого операнда. Левые 4 бита в каждом байте поля первого операнда не изменяются. Получаемые в результате выполнения этой команды данные не проверяются на соответствие кодам алфавитно-цифровых символов, тем самым допускается формирование непечатаемых символов. С помощью одной команды можно пересылать до 256 полубайтов.

Признак результата не изменяется.

Команда Move Zone — MVZ (Пересылка зон)

Мнемоника	Код операции	Формат данных
MVZ	D3	$D_1(L,B_1),D_2(B_2)$

Команда MVZ пересылает только левые 4 бита каждого байта (левый полубайт), которые в зонном десятичном формате содержат зонную часть данных. Это, однако, не мешает пересылать любые цифровые значения, которые могут содержаться в пересылаемых полубайтах. Команда пересылает левые 4 бита каждого байта поля, указанного вторым операндом, в соответствующие 4 бита каждого байта области памяти, определенной первым операндом. Данные пересылаются по одному полубайту слева направо, причем общее количество пересылаемых полубайтов определяется явной или неявной длиной первого операнда. Правые 4 бита (правый полубайт) каждого байта результирующего поля не изменяются. С помощью одной команды можно переслать до 256 полубайтов.

Признак результата не изменяется.

Команда Store Character — STC (Запись в память символа)

Мнемоника	Код операции	Формат операндов
STC	42	$R_1,D_2(X_2,B_2)$

Команда STC помещает младшие 8 битов (1 байт) общего регистра, указанного первым операндом, по адресу, указанному вторым операндом.

Признак результата не изменяется.

Команда Store Halfword — STH (Запись в память полуслова)

Мнемоника	Код операции	Формат операндов
STH	40	$R_1,D_2(X_2,B_2)$

Команда STH помещает младшие 16 битов (2 байта) общего регистра, указанного первым операндом, в область памяти длиной в полуслово, указанную вторым операндом. Второй операнд должен быть расположен на границе полуслова.

Признак результата не изменяется.

Команда Store — ST (Запись в память)

Мнемоника	Код операции	Формат операндов
ST	50	$R_1,D_2(X_2,B_2)$

Команда ST помещает содержимое общего регистра, указанного первым операндом, в полное слово памяти, указанное вторым операндом. Второй операнд должен быть расположен на границе слова.

Признак результата не изменяется.

Команда Store Multiple — STM (Запись в память групповая)

Мнемоника	Код операции	Формат операндов
STM	90	$R_1,R_3,D_2(B_2)$

Команда STM записывает содержимое нескольких общих регистров (двух, трех, четырех или больше) в связную область памяти, состоящую из соответствующего числа слов. Первый операнд указывает начальный регистр из числа подлежащих запоминанию, а второй операнд (R_3) указывает последний из этих регистров [1]). Область памяти, адресуемая операндом $D_2(B_2)$, должна начинаться с границы слова и состоять из достаточного для запоминания указанных регистров количества слов — по одному слову на регистр. Общие регистры записываются в память в порядке возрастания номеров, начиная с регистра, указанного первым операндом; после запоминания регистра 15, если регистр, указанный в R_3, еще не встретился, произойдет переход к запоминанию регистра 0. Например, если в операнде R_1 указан регистр 13, а в операнде R_3 указан регистр 2, то команда запишет в память по порядку следующие регистры: 13, 14, 15, 0, 1 и 2.

Признак результата не изменяется.

Команда Insert Character — IC (Прочитать символ)

Мнемоника	Код операции	Формат операндов
IC	43	$R_1,D_2(X_2,B_2)$

Команда IC помещает 1 байт, адресуемый вторым операндом, в младшие 8 битов регистра, указанного первым операндом. Старшие 24 бита этого регистра остаются без изменения.

---

[1]) Автор имеет в виду второй по порядку операнд в записи формата операндов, являющийся, строго говоря, третьим (R_3) операндом команды STM. — *Прим. ред.*

Признак результата не изменяется.

Команда Load Halfword — LH (Загрузка полуслова)

Мнемоника	Код операции	Формат операндов
LH	48	$R_1,D_2(X_2,B_2)$

Команда LH сначала выбирает из памяти полуслово по адресу, указанному во втором операнде, и расширяет его до полного слова 16 дополнительными старшими битами, совпадающими по значению со знаковым битом первоначального полуслова. Затем полученные 32 бита загружаются в общий регистр, указанный первым операндом. Содержимое полуслова в памяти не изменяется. Второй операнд должен быть расположен на границе полуслова.

Признак результата не изменяется.

Команда Load — L (Загрузка)

Мнемоника	Код операции	Формат операндов
L	58	$R_1,D_2(X_2,B_2)$

Команда L загружает полное слово, адресуемое вторым операндом, в общий регистр, указанный первым операндом. После выполнения этой команды содержимое первого операнда идентично содержимому второго операнда. Второй операнд должен быть расположен на границе слова.

Признак результата не изменяется.

Команда Load Register — LR (Загрузка)

Мнемоника	Код операции	Формат операндов
LR	18	R_1,R_2

Команда LR помещает содержимое регистра второго операнда в регистр первого операнда. После завершения команды содержимое обоих регистров одинаково.

Признак результата не изменяется.

Команда Load Multiple — LM (Загрузка групповая)

Мнемоника	Код операции	Формат операндов
LM	98	$R_1,R_3,D_2(B_2)$

Команда LM загружает полные слова из памяти, начиная с адреса, указанного последним операндом, в ряд регистров, начинающийся с регистра, указанного первым операндом, и кончающийся регистром, указанным операндом R_3. Последовательность регистров, подлежащих загрузке, может начинаться с

большего номера и кончаться меньшим номером. Например, если в качестве R_1 указан регистр 13, а в качестве R_3 — регистр 4, то будут загружены регистры 13, 14, 15, 0, 1, 2, 3 и 4.

Признак результата не изменяется.

Команда Load Address — LA (Загрузка адреса)

Мнемоника	Код операции	Формат операндов
LA	41	$R_1,D_2(X_2,B_2)$

Команда LA формирует 24-разрядный адрес, определенный вторым операндом, и помещает этот адрес в младшие 24 бита общего регистра первого операнда. Старшие 8 битов регистра первого операнда заполняются нулями. Второй операнд может быть абсолютным десятичным значением или символическим адресом с индексом и смещением.

Признак результата не изменяется.

Команда Load and Test Register — LTR (Загрузка и проверка)

Мнемоника	Код операции	Формат операндов
LTR	12	R_1,R_2

Команда LTR помещает содержимое регистра второго операнда в регистр первого операнда, после чего содержимое обоих регистров становится одинаковым. Значение признака результата определяется значением и знаком данных, загружаемых в регистр первого операнда:

CC	BC	Условие
0	8	Новое значение в регистре первого операнда равно нулю
1	4	Новое значение в регистре первого операнда меньше нуля
2	2	Новое значение в регистре первого операнда больше нуля

Команда Load Complement Register — LCR (Загрузка дополнения)

Мнемоника	Код операции	Формат операндов
LCR	13	R_1,R_2

Команда LCR помещает дополнение до 2 содержимого регистра второго операнда в регистр первого операнда. В обоих операндах можно указать один и тот же регистр, и тогда содержимое этого регистра будет заменено на его дополнение. Значение признака результата определяется значением дополнения, получаемого в регистре первого операнда:

ОС	ВС	Условие
0	8	Новое значение в регистре первого операнда равно нулю
1	4	Новое значение в регистре первого операнда меньше нуля
2	2	Новое значение в регистре первого операнда больше нуля
3	1	Условие переполнения

Команда Load Positive Register — LPR
(Загрузка положительная)

Мнемоника	Код операции	Формат операндов
LPR	10	R_1,R_2

Если регистр второго операнда содержит отрицательную величину, в регистр первого операнда загружается положительное дополнение этой величины. Если регистр второго операнда уже содержит положительную величину, содержимое этого регистра помещается в регистр первого операнда без изменения. При возникновении условия переполнения устанавливается соответствующее значение признака результата.

СС	ВС	Условие
0	8	Регистр первого операнда содержит нуль
2	2	Регистр первого операнда содержит положительную величину
3	1	В результате выполнения команды возникло переполнение

Команда Load Negative Register — LNR
(Загрузка отрицательная)

Мнемоника	Код операции	Формат операндов
LNR	11	R_1,R_2

Если регистр второго операнда содержит положительную величину, в регистр первого операнда загружается отрицательное дополнение этой величины. Если регистр второго операнда уже содержит отрицательную величину, то это содержимое без изменения загружается в регистр первого операнда. Если регистр второго операнда содержит нулевое значение, оно будет помещено в регистр первого операнда с положительным знаком.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно нулю
1	4	Содержимое регистра первого операнда меньше нуля

Команда Shift Left Logical — SLL (Сдвиг влево кода)

Мнемоника	Код операции	Формат операндов
SLL	89	$R_1,D_2(B_2)$

Все 32 бита общего регистра первого операнда сдвигаются влево на число позиций, определяемое вторым операндом. Для определения числа позиций сдвига используются только 6 младших битов адреса, формируемого вторым операндом. Сдвигу подвергаются все биты регистра, включая бит знака, и любые биты, выходящие в результате сдвига за пределы регистра, теряются, не вызывая условия переполнения. В соответствующее число младших битов помещаются нули.

Признак результата не изменяется.

Команда Shift Left Double Logical — SLDL (Сдвиг влево двойной кода)

Мнемоника	Код операции	Формат операндов
SLDL	8D	$R_1,D_2(B_2)$

Все 64 бита, содержащиеся в паре общих регистров, номер первого из которых четный, сдвигаются влево на число позиций, определяемое вторым операндом. Для определения числа позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Первый операнд должен содержать адрес первого регистра пары. Сдвигу подвергаются все биты, и любые биты, выходящие в результате сдвига на пределы четного регистра, теряются; условие переполнения при этом не возникает. В соответствующее число младших битов помещаются нули.

Признак результата не изменяется.

Команда Shift Right Logical — SRL (Сдвиг вправо кода)

Мнемоника	Код операции	Формат операндов
SRL	88	$R_1,D_2(B_2)$

Все 32 бита общего регистра первого операнда сдвигаются вправо на число позиций, определяемое вторым операндом. Для определения количества позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом.

Сдвигу подвергаются все биты регистра, и биты, сдвинутые из младших позиций, теряются; это обстоятельство не отражается в признаке результата. Освободившиеся старшие биты заполняются нулями.

Признак результата не изменяется.

Команда Shift Right Double Logical — SRDL
(Сдвиг вправо двойной кода)

Мнемоника	Код операции	Формат операндов
SRDL	8C	$R_1,D_2(B_2)$

Все 64 бита пары общих регистров, номер первого из которых четный, сдвигаются вправо на число позиций, определяемое вторым операндом. Первый операнд должен содержать адрес первого регистра пары. Для определения количества позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Сдвигу подвергаются все биты, и любые биты, выходящие в результате сдвига за пределы нечетного регистра, теряются; это обстоятельство не отражается в признаке результата. Освободившиеся старшие биты заполняются нулями.

Признак результата не изменяется.

Команда Shift Left Algebraically — SLA
(Сдвиг влево арифметический)

Мнемоника	Код операции	Формат операндов
SLA	8D	$R_1,D_2(B_2)$

Эта команда сдвигает влево 31 бит общего регистра первого операнда на число позиций, определяемое вторым операндом. Для определения числа позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Знаковый бит регистра первого операнда не сдвигается. Младшие биты, освободившиеся при сдвиге, заполняются нулями. Сдвиг значащего бита из старшей позиции целочисленной части отражается в признаке результата условием переполнения.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно нулю
1	4	Содержимое регистра первого операнда меньше нуля

2	2	Регистр первого операнда содержит положительную величину
3	1	При выполнении сдвига возникло условие переполнения

Команда Shift Left Double Algebraically — SLDA
(Сдвиг влево двойной арифметический)

Мнемоника	Код операции	Формат операндов
SLDA	8F	$R_1,D_2(B_2)$

63 бита пары общих регистров, номер первого из которых четный, сдвигаются влево на число позиций, определяемое вторым операндом. Первый операнд должен содержать адрес первого регистра пары. Для определения числа позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Бит знака нечетного общего регистра сдвигается наравне с другими битами, однако бит знака четного регистра не сдвигается. Младшие биты, освободившиеся при сдвиге, заполняются нулями. Сдвиг значащего бита из старшей позиции целочисленной части четного регистра отражается в признаке результата условием переполнения.

CC	BC	Условие
0	8	Регистры первого операнда содержат нулевое значение
1	4	Содержимое регистров первого операнда меньше нуля
2	2	Содержимое регистров первого операнда больше нуля
3	1	При выполнении сдвига произошло переполнение

Команда Shift Right Algebraically — SRA
(Сдвиг вправо арифметический)

Мнемоника	Код операции	Формат операндов
SRA	8A	$R_1,D_2(B_2)$

Эта команда сдвигает вправо 31 бит общего регистра первого операнда на число позиций, определяемое вторым операндом. Для определения числа позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Биты, сдвигаемые из младших позиций, теряются; признак результата не отражает этого обстоятельства. Знаковый бит первого операнда остается без изменения, а старшие биты, освободившиеся при сдвиге, устанавливаются равными знаковому биту.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно нулю
1	4	Содержимое регистра первого операнда меньше нуля
2	2	Содержимое регистра первого операнда больше нуля

Команда Shift Right Double Algebraically — SRDA
(Сдвиг вправо двойной арифметический)

Мнемоника	Код операции	Формат операндов
SRDA	8E	$R_1,D_2(B_2)$

Команда SRDA сдвигает вправо 63 бита пары общих регистров, номер первого из которых четный, на число позиций, определяемое вторым операндом. Первый операнд должен содержать адрес первого регистра пары. Для определения числа позиций сдвига используются только младшие 6 битов адреса, формируемого вторым операндом. Биты, сдвигаемые из младших позиций нечетного регистра, теряются. Знаковый бит четного регистра не изменяется, а знаковый бит нечетного регистра сдвигается наравне с остальными битами. Старшие биты пары регистров устанавливаются равными знаковому биту четного регистра, и число таких битов равно числу позиций сдвига. При сдвиге значащих битов из младших позиций условие переполнения не возникает.

CC	BC	Условие
0	8	Содержимое регистров первого операнда равно нулю
1	4	Содержимое регистров первого операнда меньше нуля
2	2	Содержимое регистров первого операнда больше нуля

Б. ПРИМЕНЕНИЕ КОМАНД ПЕРЕСЫЛКИ, ЗАПИСИ В ПАМЯТЬ, ЗАГРУЗКИ И СДВИГА

При написании проблемной программы возникает необходимость пересылать данные из одной области памяти в другую. Эти перемещения осуществляются почти исключительно командами пересылки, загрузки, записи в память и сдвига. Команды

пересылки выполняют перемещение данных из области памяти с одним адресом в область памяти с другим адресом; команды загрузки перемещают данные из областей памяти в регистры или из регистров в регистры; команды записи в память перемещают данные из регистров в области памяти; команды сдвига оперируют данными внутри регистров. Ниже на примерах показана работа отдельных команд из этих четырех групп. Некоторые команды рассмотрены здесь вскользь, поскольку они подробно описаны в других главах книги.

В связи с большим числом рассматриваемых в этой главе команд, приведенные в конце главы упражнения также разбиты на группы по типам команд. Читателю рекомендуется выполнять упражнения для каждой группы команд сразу после ознакомления с соответствующим разделом этой главы.

1. Пересылка данных в проблемной программе

Хотя любую передачу данных из одного места машины в другое с полным основанием можно назвать пересылкой, термин *пересылка* означает перемещение данных из одной области основной памяти в другую область основной памяти. Такие действия выполняют следующие команды Ассемблера:

1. Пересылка — MVI
2. Пересылка — MVC
3. Пересылка цифр — MVN
4. Пересылка зон — MVZ
5. Пересылка со сдвигом — MVO

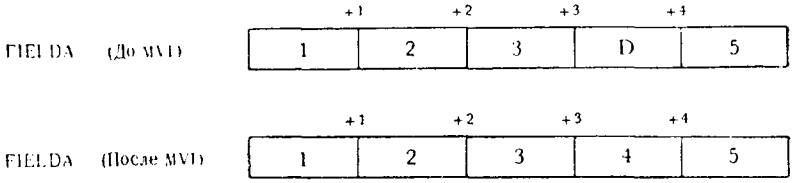
Команда MVO достаточно полно описана в следующей главе и поэтому здесь не рассматривается.

Из последующего изложения станет очевидно, что с помощью одной команды можно пересылать от 1 до 256 байтов данных. Особое внимание будет уделено пересылке полей данных, занимающих более 256 байтов, а также приему программирования, позволяющему заполнять некоторое поле содержимым 1 байта.

Пересылка — MVI. С помощью команды MVI удобно пересылать в некоторое поле 1 байт непосредственных данных. Эта операция выполняется значительно быстрее, чем пересылка 1 байта командой MVC. Разница в скорости выполнения возникает из-за характера второго операнда. В команде MVC второй операнд генерирует адрес символа, подлежащего пересылке. В команде MVI пересылаемый символ является составной частью генерируемой машинной команды. Работа команды MVI иллю-

стрируется следующим примером:

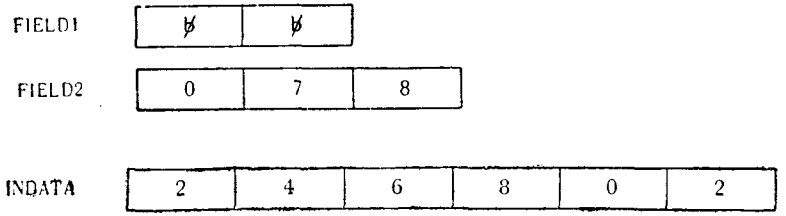
```
FIELDA DC  CL5'123D5'
       MVI FIELDA+3,C'4'
```



Предложение MVI пересылает цифру 4 (в коде EBCDIC) в четвертый байт поля, адресуемого меткой FIELDA. Поскольку символ 4 является частью самой машинной команды, для него не требуется дополнительной памяти.

Пересылка символов — MVC. Эта команда пересылает до 256 байтов данных из одной области памяти в другую. Количество пересылаемых байтов определяется явной или неявной длиной области, адресуемой первым операндом.

```
FIELD1   DC   CL2'ЬЬ'
FIELD2   DC   CL3'078'
INDATA   DC   CL6'246802'
```

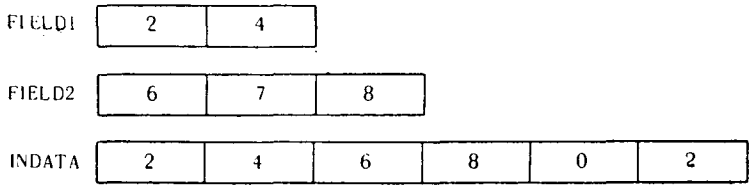



```
MUVIT    MVC    FIELD1,INDATA          001
         MVC    FIELD2(1),INDATA+2     002
```

Предложение 001 содержит указание: «Переслать данные из области памяти с начальным адресом, генерируемым меткой INDATA, в область памяти с меткой FIELD1; число пересылаемых байтов взять равным указателю длины поля FIELD1». Первые 2 байта из INDATA, содержащие символы 24, пересылаются в первые 2 байта поля FIELD1.

Предложение 002 задает пересылку данных, начиная с адреса INDATA+2, по адресу FIELD2. В соответствии с указанием явной длины в первом операнде пересылается 1 байт. Третий байт поля INDATA, содержащий символ 6, пересылается в первый байт поля FIELD2.

После выполнения этих двух команд содержимое полей будет такое:



Команда MVC довольно проста. Ее можно применять для различных преобразований данных, при условии что оба операнда находятся в основной памяти.

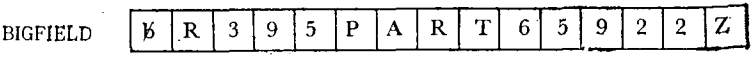
Одно из применений этой команды состоит в заполнении всего поля одним символом (содержимым 1 байта). Так как данные пересылаются слева направо с младших адресов памяти к старшим, содержимое первого байта любого поля может быть послано во все остальные байты этого поля. Благодаря этому можно, например, заполнить поле пробелами, не запасаясь для этого константой с числом пробелов, равным длине очищаемого поля. Для выполнения такой операции требуется две команды — одна для того, чтобы заслать в первый байт очищаемого поля код пробела, и вторая для того, чтобы заполнить этим кодом остальные байты поля.

Такое применение команды MVC называют *скользящей пересылкой.*

Ниже приведены определение поля в памяти, содержимое этого поля перед выполнением команд и сами команды:

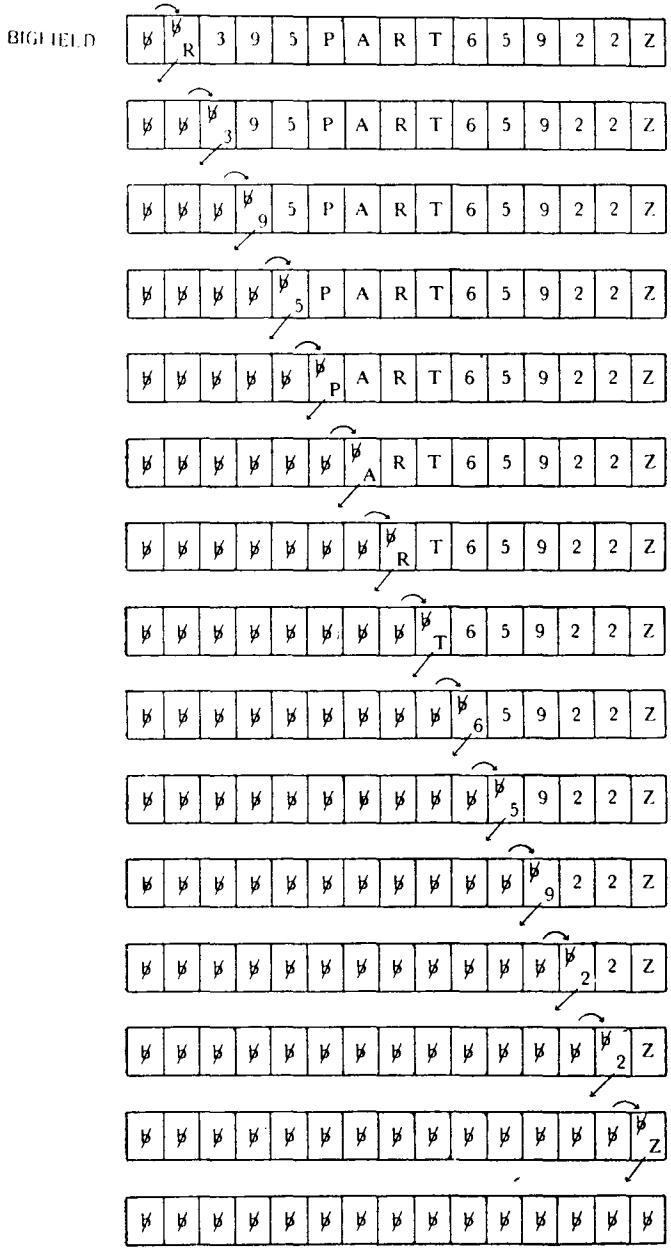

```
MVI     BIGFIELD,C'␣'                   001
MVC     BIGFIELD+1(14),BIGFIELD         002
```

Предложение 001 — команда MVI — помещает код пробела (X'40') в первый байт поля BIGFIELD. Теперь поле BIGFIELD будет иметь следующий вид:



Предложение 002 вызывает распространение первого байта BIGFIELD на все поле. Оно указывает, что каждый байт этого

поля, начиная с первого, должен быть послан в ближайший справа байт. Побайтовая пересылка данных выглядит следующим образом:



Как указано в предложении 002, длина первого операнда составляет 14 байтов, начиная с адреса BIGFIELD+1, в результате чего оставшиеся 14 байтов этого поля заполняются пробелами. Таким же образом можно заполнить пробелами или другим символом любое поле длиной до 256 байтов.

В некоторых случаях программисту может понадобиться пересылка данных с длиной, большей 256 байтов. Пересылка таких данных осуществляется цепочкой команд MVC. Первое предложение MVC кодируется с указанием максимальной длины. Первый и второй операнды следующего предложения адресуют байты, непосредственно следующие за последними байтами полей, к которым обращались первый и второй операнды предыдущего предложения. Такую цепочку команд лучше всего показать на примере:

```
DATAFLD   DS    CL750
INPUT     DS    CL890
*
          MVC   DATAFLD(256),INPUT                001
          MVC   DATAFLD+256(256),INPUT+256        002
          MVC   DATAFLD+512(238),INPUT+512        003
```

Будем считать, что поля памяти, определенные предложениями DS, содержат нужные нам данные. Последующие предложения MVC предназначены для пересылки 750 байтов данных из поля INPUT в поле, определенное меткой DATAFLD.

Предложение 001 помещает первые 256 байтов (слева направо) поля INPUT в первые 256 байтов поля DATAFLD.

Предложение 002 пересылает следующие 256 байтов из INPUT в DATAFLD. Это будут байты поля INPUT с 257-го по 512-й, и попадут они в соответствующие позиции поля DATAFLD.

Предложение 003 пересылает из INPUT в DATAFLD 238 байтов, начиная с 513-го и кончая 750-м. Теперь все 750 байтов данных пересланы из INPUT в DATAFLD: 256 байтов — предложением 001, 256 следующих байтов — предложением 002 и остальные 238 байтов — предложением 003.

Программист может составить цепочку команд MVC какой угодно длины; можно пересылать тысячи байтов данных, если только в этом есть необходимость.

Если нужно заполнить одинаковым содержимым все байты очень длинного поля данных, можно составить цепочку из команд MVC, каждая из которых осуществляет скользящую пересылку. На рис. 9.1 представлен пример такой цепочки.

IBM

IBM System/360 Assembler Coding Form

X28-6500-3 U/M360
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
MOVELOT	MVI	HOLDAREA,C' '		
	MVC	HOLDAREA+1(255),HOLDAREA		
	MVC	HOLDAREA+256(256),HOLDAREA+255		
	MVC	HOLDAREA+512(256),HOLDAREA+511		
	MVC	HOLDAREA+768(42),HOLDAREA+767		
	BC	15,BACK		
*				
*				
HOLDAREA	DS	CL810		
*				

Рис. 9.1.

В результате выполнения этих команд все 810 байтов поля HOLDAREA будут заполнены кодами пробела (X'40').

С помощью команд управления циклом и команд MVC можно пересылать в области памяти программы целые группы, или сегменты, данных. Этот прием иногда называют *скользящей таблицей* или *гибким индексированием.* Суть дела заключается в добавлении сегментов данных в большой блок, состоящий из упорядоченной последовательности сегментов одинаковой длины, или удалении сегментов из него. Поскольку подробный пример с необходимыми объяснениями действия команд занял бы много места, ниже приведены общие правила, которые помогут программисту самостоятельно освоить этот метод. Следует помнить, что описываемый прием относится только к пересылке данных в пределах основной памяти и предназначен для того, чтобы последовательно вводить (или удалять) некоторый сегмент произвольных данных в соответствующее место блока, состоящего из сегментов одинаковой длины.

1. Увеличение или сокращение общего количества данных, содержащихся внутри блока, должно происходить только на верхнем конце блока; расширение производится с младших адресов к старшим; сокращение производится со старших адресов в сторону младших. Из этого следует, что начальный адрес блока не должен изменяться, в то время как адрес последнего сегмента, принадлежащего блоку, может меняться, если это необходимо.

2. Вслед за последним сегментом данных должен следовать ограничитель, указывающий их конец, если даже не вся выделенная для блока память занята сегментами.

3. Последняя позиция выделенной для блока области должна содержать сегмент-ограничитель блока. Если программа написана верно, ограничитель предотвратит попытку переслать сегмент в область памяти, не отведенную под блок.

Для того чтобы ввести сегмент с данными в соответствующую позицию внутри блока, надо выполнить следующие действия:

1. Отыскать адрес позиции в блоке, куда необходимо поместить сегмент. Это можно сделать, последовательно сравнивая номер нового сегмента с номерами существующих сегментов. Новый сегмент надо поместить на место первого встретившегося при поиске сегмента, номер которого больше номера нового сегмента. Фактическое занесение данных по найденному адресу будет выполнено позднее, а пока следует запомнить этот адрес в области сохранения или в регистре.

2. Просмотреть остаток блока и найти ограничитель сегментов.

3. Построить циклическую подпрограмму, которая перешлет каждый сегмент, включая ограничитель сегментов, на длину одного сегмента в направлении конца блока. Каждый проход по циклу уменьшает адреса обоих операндов в команде MVC на длину сегмента.

4. Циклическая подпрограмма должна заканчивать цикл, как только будет переслан сегмент, занимающий область, куда следует поместить новый сегмент. При первой пересылке сегмента данных, а именно ограничителя сегментов, следует проверить, не затрагивает ли эта пересылка ограничителя блока.

5. Новый сегмент следует затем поместить на свое место в блоке. При пересылке ранее существовавшая здесь копия сегмента будет уничтожена, однако эта информация теперь хранится на месте следующего по порядку сегмента.

Для того чтобы удалить из блока сегмент данных и уплотнить оставшиеся сегменты, необходимо выполнить следующие действия:

1. Определить адрес удаляемого сегмента.

2. Загрузить этот адрес в два общих регистра X и Y. Прибавить к содержимому второго регистра длину сегмента.

3. Построить циклическую подпрограмму с командой MVC, где указана явная длина, равная длине сегмента. Первый общий регистр (X) должен задавать адрес первого операнда в этой команде, а второй общий регистр (Y) — адрес второго операнда. Команда MVC будет пересылать каждый сегмент в направлении начала блока на длину одного сегмента. Первое выполнение этой команды вызовет пересылку ближайшего сегмента с более высоким порядковым номером на место, занятое удаляемым сегментом.

4. Каждый проход по циклу должен увеличивать значения обоих регистров на величину, равную длине сегмента.

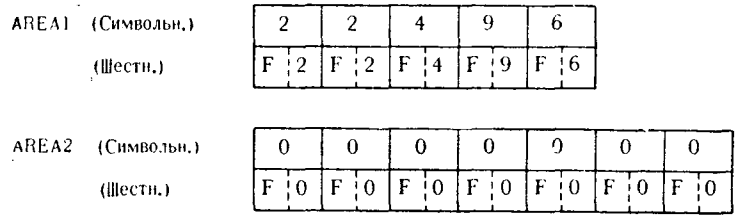
5. После пересылки каждого сегмента программа должна проверить, не является ли этот сегмент ограничителем сегментов. Если это так, то уплотнение сегментов в блоке окончено.

Пользуясь этими общими правилами, программист сможет написать программу, реализующую «скользящую таблицу». У начинающего программиста такое задание может вызвать некоторые затруднения, однако, успешно выполнив его, он получит хорошую практику в обработке данных.

Пересылка цифр — MVN. Команда MVN является одной из двух команд языка Ассемблера, которые пересылают не все поле данных, а лишь отдельные его части. Хотя данные пересылаются из одной области памяти в другую в строгом порядке, эта команда перемещает только правую половину (4 бита) каж-

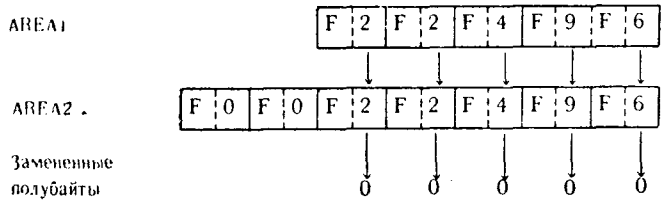
дого байта данных. Алфавитно-цифровые символы кода EBCDIC содержат в правой тетраде (правом полубайте) шестнадцатеричную цифру от 0 до 9; поэтому эта команда и носит название «Пересылка цифр». Как уже говорилось, правая тетрада каждого байта посылающего поля пересылается в правую тетраду соответствующего байта принимающего поля. Левые тетрады каждого байта обоих полей не изменяются. Одна команда MVN может работать с полями данных длиной до 256 байтов.

```
AREA1    DC    CL5'22496'
AREA2    DC    CL7'0000000'
```



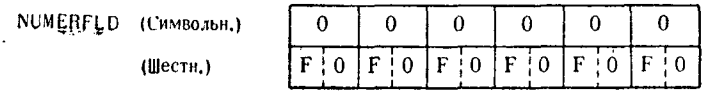

```
MVN      AREA2+2(5),AREA1
```

Это предложение указывает, что правая половина каждого из 5 первых байтов для AREA1 должна быть переслана в правую половину каждого байта поля AREA2+2 (начиная с третьего байта этого поля). Действие команды выглядит так:



Эту команду можно использовать также для того, чтобы преобразовать алфавитно-цифровые символы кода EBCDIC в соответствующие правым тетрадам цифровые значения. Ниже иллюстрируется это применение:

```
ALPHAFLD    DC    CL6'AKTDNW'
NUMERFLD    DC    CL6'000000'
```



ALPHAFLD (Символьн.)

A	K	T	D	N	W

(Шестн.)

C	1	D	2	E	3	C	4	D	5	E	6

В этом примере команда MVN пересылает только правую шестнадцатеричную цифру каждого байта из поля ALPHAFLD в соответствующую правую тетраду каждого байта поля NUMERFLD и тем самым преобразует буквенные символы EBCDIC в цифровые символы кода EBCDIC.

Команду MVN применяют также для того, чтобы привести к полностью числовому виду информацию некоторого поля, имевшего до этого лишь частично цифровое содержимое. Такая ситуация может возникнуть, например, в случае, когда цифровое поле составлено так, что старшие (незначащие) нули цифровых символов кода EBCDIC представлены пробелами.

Данные такого вида могли получиться в результате предварительного редактирования их другой программой. Предположим, что проблемная программа должна распознавать числа в диапазоне от 0000001 до 9999999, представленные в EBCDIC. Однако вследствие предварительного редактирования данные считываются в проблемную программу в другом формате с диапазоном от ЬЬЬЬЬЬ1 до 9999999. Пусть для нашей программы цифровые величины должны быть выражены в полностью цифровой форме. Тогда поля данных могут иметь следующий вид:

```
DATAFLD  DS   CL7
NUMCONV  DS   CL7'0000000'
*
         MVN NUMCONV,DATAFLD
```

DATAFLD (Символьн.)

Ь	Ь	Ь	5	9	7	4

(Шестн.)

4	0	4	0	4	0	F	5	F	9	F	7	F	4

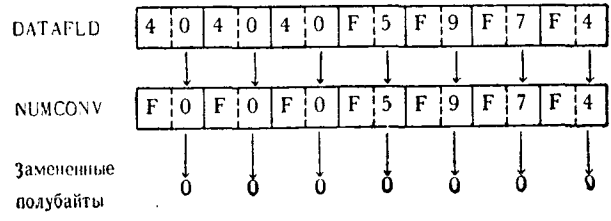
NUMCONV (Символьн.)

0	0	0	0	0	0	0

(Шестн.)

F	0	F	0	F	0	F	0	F	0	F	0	F	0

Выполнение команд выглядит так:

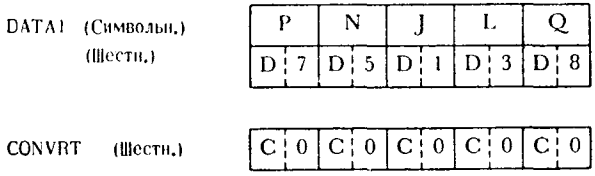


Приведенное предложение заменило конфигурацию ЬЬЬ5974 на полностью цифровую 0005974. Такое же преобразование можно выполнить, пересылая только левые половины байтов. Этот прием описан ниже, при рассмотрении команды пересылки зон.

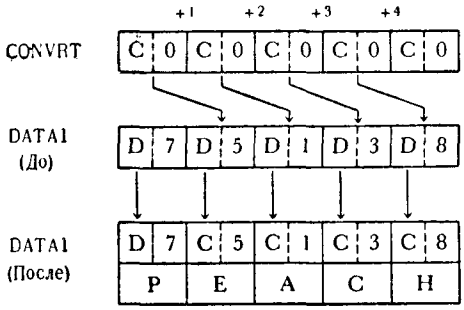
Пересылка зон — MVZ. Команда MVZ, подобно команде пересылки цифр, пересылает данные лишь частично, затрагивая только часть байта. Эта команда отличается от команды MVN тем, что пересылает только левые тетрады (4 бита) каждого байта передающего поля в принимающее поле. Левая половина алфавитно-цифрового символа кода EBCDIC называется зонной частью байта; поэтому эта команда и получила такое наименование. Одна команда MVZ может работать с полями длиной до 256 байтов.

Пересылка частей байтов данных происходит так, как показано ниже:

```
DATA1    DC     CL5'PNJLQ'
CONVRT   DC     X'COCOCOCOCO'
*
         MVZ    DATA1+1(4),CONVRT
```



Выполнение команды приведет к следующему изменению содержимого полей:



Как видно из примера, команда MVZ изменила содержимое 5 байтов данных с PNJLQ на PEACH. Это произошло только за счет изменения зонной части в 4 последних байтах.

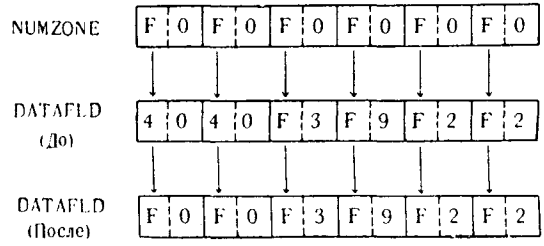
Как упоминалось в связи с обсуждением команды MVN, команду пересылки зон можно использовать для того, чтобы преобразовать поле, содержащее коды пробелов и цифр, в полностью цифровое поле. Так как код пробела равен 40 (X'40'), его можно легко превратить в цифру 0 (X'F0') пересылкой шестнадцатеричной цифры F в зонную часть каждого байта. Ниже приведен пример такого преобразования:

```
DATAFLD    DC    CL6'bb3922'
NUMZONE    DC    CL6'000000
```

DATAFLD	(Символьн.)	b	b	3	9	2	2
	(Шестн.)	40	40	F3	F9	F2	F2
NUMZONE	(Символьн.)	0	0	0	0	0	0
	(Шестн.)	F0	F0	F0	F0	F0	F0

```
MVZ       DATAFLD, NUMZONE
```

Выполнение команды вызывает следующие действия:

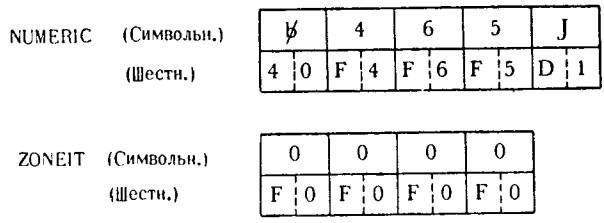


Поле DATAFLD, которое первоначально содержало набор цифровых символов и пробелов слева от них, теперь имеет конфигурацию поля, целиком заполненного цифровыми символами. Такая операция может оказаться очень полезной программисту, который хочет быть уверенным в том, что перед выполнением команды «Упаковать» в младшем (самом правом) байте исходного поля не содержится код пробела. Если эта команда будет работать с полем, содержащим пробел в младшем байте, то результирующее поле упакованных десятичных чисел будет непригодно для операций десятичной арифметики (см. гл. 10).

В предыдущих примерах предполагалось, что проверяемые цифровые данные не имеют знака, т. е. в зонной части младшего байта нет специального кода знаков «плюс» или «минус». Например, шестнадцатеричная конфигурация C1 (символ A) в упакованном десятичном формате рассматривается как

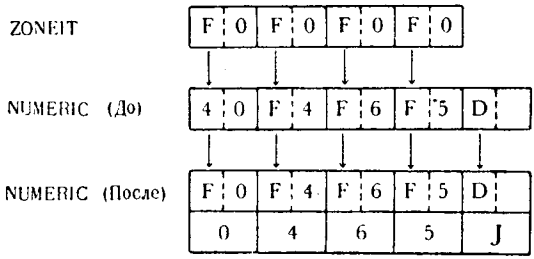
стандартное значение +1; шестнадцатеричная конфигурация D1 (символ J) представляет значение —1; шестнадцатеричная конфигурация F1 (цифровой символ 1) рассматривается как значение +1. Если все цифровые данные, обрабатываемые командой MVZ, должны рассматриваться как положительные числа, пересылка шестнадцатеричной цифры F в левую половину младшего байта поля не вызовет нежелательных последствий. Если же младший байт поля цифровых символов содержит код знака минус, то команда MVZ «сотрет» отрицательную зону и заменит ее положительной. Этого можно избежать, не затрагивая младшего байта поля при пересылке. Например:

```
NUMERIC   DC   CL'ϕ465J'
ZONEIT    DC   CL'0000'
```




```
MVZ       NUMERIC(4),ZONEIT
```

Это предложение указывает, что зонную часть каждого байта поля ZONEIT следует переслать в соответствующие позиции только первых 4 байтов поля NUMERIC. Это будет выглядеть так:



Первоначальное содержимое поля NUMERIC представляло собой набор частично цифровых данных; в младшем байте находился символ J в зонном десятичном формате, соответствующий значению —1. Если бы команда MVZ переслала цифровые зоны во все байты поля NUMERIC, младший байт приобрел бы шестнадцатеричную конфигурацию F, соответствующую символу числа 1, который рассматривается как положительное число

без знака. Это привело бы к изменению знака первоначального отрицательного числа, представленного в зонном десятичном формате.

2. Команды записи в память

Команды записи в память предназначены для пересылки в области основной памяти всего содержимого регистров или его части. К этой группе относятся четыре команды; они могут записывать в память 1, 2, 4 байта или группу вплоть до 64 байтов. Кроме операции записи в память 1 байта, остальные команды записи требуют, чтобы области памяти были соответственным образом выравнены по определенной границе. Двухбайтовые области памяти должны находиться на границе полуслова, четырехбайтовые — на границе полного слова; группы четырехбайтовых областей должны примыкать друг к другу и располагаться на границе полного слова.

Рассматриваемые команды особенно полезны для сохранения адресов и чисел, находящихся в регистре. Это часто бывает необходимым, особенно в случае, когда один и тот же общий регистр используется проблемной программой в разных целях. На время необходимых вычислений можно записать содержимое некоторых регистров в память с тем, чтобы после их окончания вновь загрузить регистры нужными значениями с помощью команд загрузки.

Существуют четыре команды записи в память: ST, STC, STH и STM.

Выбор программиста в каждом конкретном случае целиком зависит от размера полей занимаемых данных.

Запись в память символа — STC. Эта команда оперирует только с 1 байтом из 4 байтов, содержащихся в общем регистре. Ее название связано с тем, что символ в коде EBCDIC представляется комбинацией из 8 битов — 1 байтом. При выполнении команды STC младший байт данных из общего регистра, указанного первым операндом, в неизменном виде помещается в однобайтовую область памяти, указанную вторым операндом.

Хотя эта команда имеет очень много применений, одно из главных состоит в изменении существующего указателя длины в некоторой команде. Например, предположим, что проблемная программа содержит следующее предложение:

```
MOVEDATE    MVC     FLD1(6),FLD2
```

Это предложение пересылает 6 байтов из области FLD2 в область FLD1. Первые 2 байта машинной команды, порожденной этим предложением, имеют шестнадцатеричную конфигурацию

D205, где D2 — код операции команды MVC, 05 — указатель длины машинного кода команды; он всегда на единицу меньше указателя длины в предложении на языке Ассемблера. Если в каком-нибудь месте проблемной программы программисту потребуется изменить указатель длины предложения с меткой MOVEDATA, с тем чтобы эта команда пересылала 11 байтов данных, он может сделать это с помощью следующих команд:

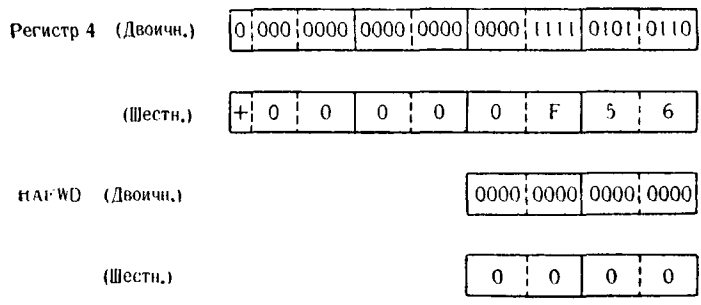
```
LA      8,10
STC     8,MOVEDATA+1
```

Первое предложение загружает число с фиксированной точкой +10 (что соответствует указателю длины для пересылки 11 байтов) в общий регистр 8. Теперь младший байт общего регистра 8 содержит шестнадцатеричную конфигурацию 0A. Вторая команда записывает младший байт общего регистра 8 в байт памяти, адресуемый выражением MOVEDATA+1, или, другими словами, во второй байт команды, адресуемой MOVEDATA. Теперь первые 2 байта этой команды в машинном коде примут вид D20A, что при выполнении команды приведет к пересылке 11 байтов для полей, указанных в операндах. Таким же способом по желанию программиста команда STC может использоваться для модификации других команд.

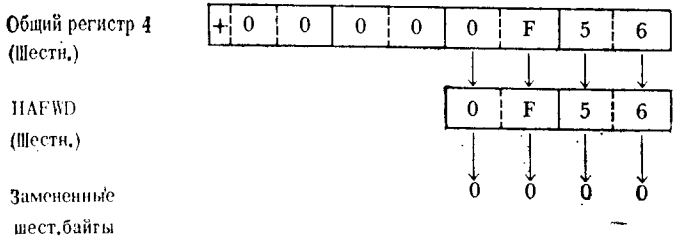
Запись в память полуслова — STH. Эта команда выбирает содержимое 2 правых байтов общего регистра, указанного первым операндом, и помещает его в неизмененном виде в область памяти длиной в полуслово по адресу, указанному вторым операндом. Команду STH можно использовать для записи чисел в память.

Предположим, что общий регистр 4 содержит число +3926:

```
HAFWD   DC      H'0'
*
        STH     4,HAFWD
```



Выполнение команды приведет к следующим изменениям:



Содержимое двух младших байтов общего регистра 4 заменило собой данные, первоначально находившиеся в области HAFWD. Обычно команда STH используется для того, чтобы извлечь из общего регистра число, умещающееся в полуслове, или отправить на хранение величины относительных приращений адресов, являющихся указателями относительных позиций, занимаемых сегментами таблиц.

Запись в память — ST. Команта ST помещает содержимое общего регистра, не изменяя его, в полное слово памяти. В проблемной программе один и тот же общий регистр чаще всего используется для многих целей. Это требует временного запоминания содержимого регистра с тем, чтобы после использования регистра для других целей вновь восстановить его прежнее содержимое. Такое применение команды ST бывает полезно в программах, реализующих арифметические операции, а также для запоминания приращений и истинных значений адресов. Команда ST — одно из средств для запоминания результатов арифметической подпрограммы, манипулирующей с числами с фиксированной точкой. Например,

```
AR      8,9
ST      8,FULLWD
```

Первое предложение складывает два числа с фиксированной точкой, содержащиеся в регистрах 8 и 9. Второе предложение помещает сумму, находящуюся в общем регистре 8, в полное слово памяти, сохраняя это число для дальнейшего использования.

Предложения, приведенные на рис. 9.2, иллюстрируют применение одного регистра для решения различных задач.

В целом программа выполняет два довольно простых процесса поиска в таблице. Предложения 001—005 производят первый поиск в таблице, используя общий регистр 9 в качестве указателя адреса текущего сегмента первой таблицы TABLE1. Как только в таблице будет найден искомый элемент (успешное

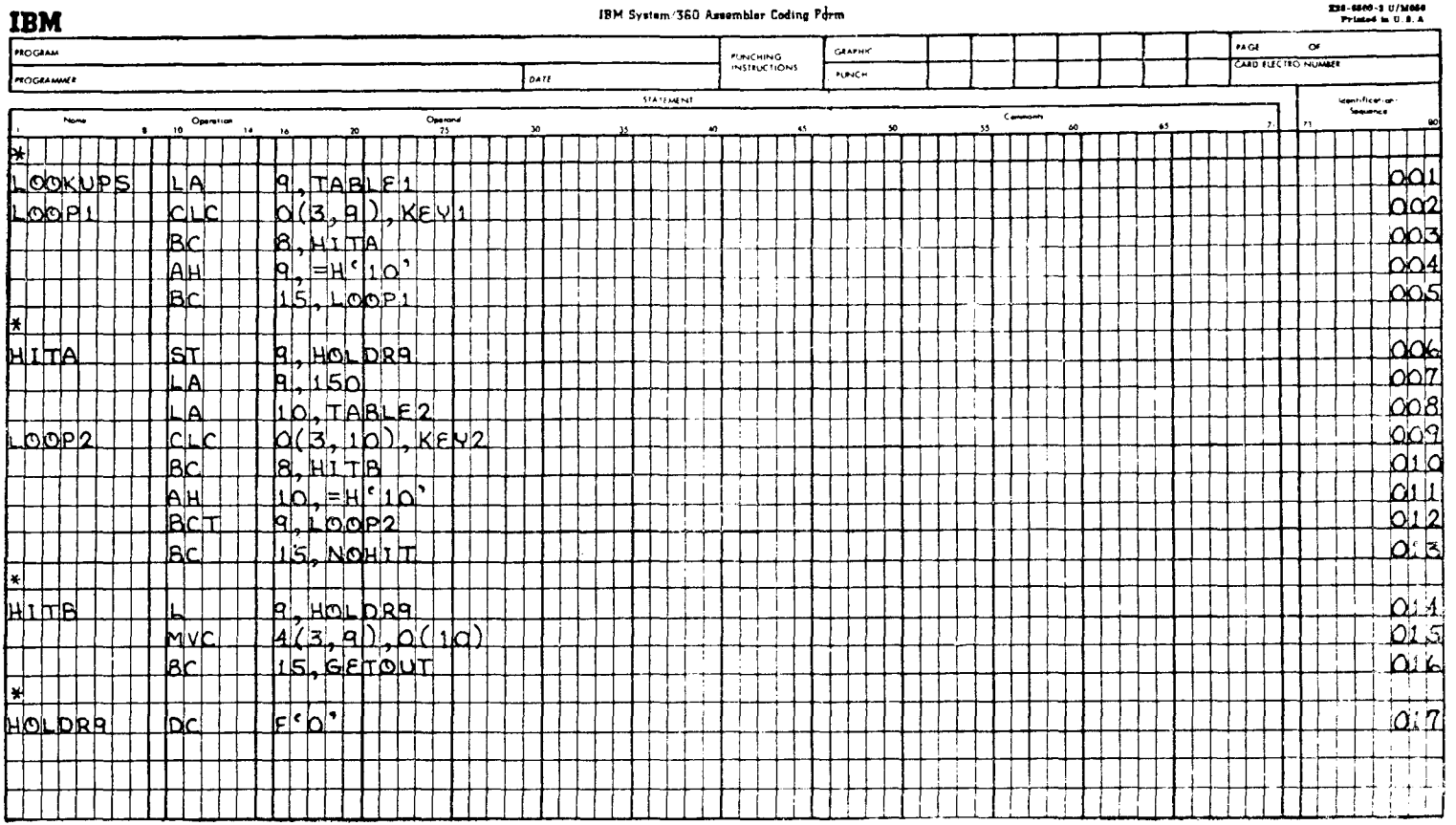

IBM
IBM System/360 Assembler Coding Form
X28-6500-3 U/M050
Printed in U.S.A.
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
*
LOOKUPS LA 9,TABLE1 001
LOOP1 CLC 0(3,9),KEY1 002
BC 8,HITA 003
AH 9,=H'10' 004
BC 15,LOOP1 005
*
HITA ST 9,HOLDR9 006
LA 9,150 007
LA 10,TABLE2 008
LOOP2 CLC 0(3,10),KEY2 009
BC 8,HITB 010
AH 10,=H'10' 011
BCT 9,LOOP2 012
BC 15,NOHIT 013
*
HITB L 9,HOLDR9 014
MVC 4(3,9),0(10) 015
BC 15,GETOUT 016
*
HOLDR9 DC F'0' 017

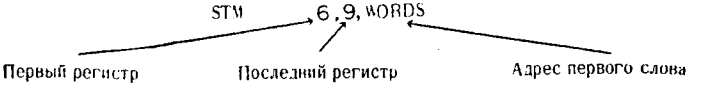
Рис. 9.2.

сравнение в предложении 002), происходит переход к подпрограмме HITA поиска во второй таблице. Предложения 006—013 также используют регистр 9, однако на этот раз в качестве счетчика циклов в команде BCT. При входе в подпрограмму HITA адрес, содержащийся в регистре 9, записывается в область памяти HOLDR9 для последующего использования после завершения подпрограммы HITA. Затем в регистр 9 загружается число 150 — количество циклов просмотра и производится поиск в другой таблице.

При обнаружении искомого элемента в TABLE2 производится переход к подпрограмме HITB, в которой предполагается использовать найденный в таблице TABLE1 адрес, записанный в HOLDR9. Поэтому содержимое HOLDR9 загружается в общий регистр 9 (предложение 014). Затем предложение 015, используя адрес, содержащийся в общем регистре 10 и представляющий сегмент из второй таблицы, пересылает данные в сегмент первой таблицы, указанный адресом в общем регистре 9. Значение счетчика циклов, содержащееся в регистре 9 при выходе из подпрограммы HITA, запоминать не надо, поскольку это число устанавливается вновь каждый раз при входе в эту подпрограмму.

Любая проблемная программа может использовать команду ST в разнообразных вариантах применения. Функции этой команды очень просты, однако с ее помощью создаются весьма сложные программы.

Запись в память групповая — STM. Команда STM применяется для того, чтобы переслать содержимое двух или более общих регистров в соответствующее количество следующих друг за другом полных слов памяти. Первые два операнда указывают номера первого и последнего регистров, подлежащих записи в память; последний операнд указывает адрес первого полного слова области памяти, куда следует записать содержимое регистров. Эту команду можно рассматривать таким образом:



Хотя общие регистры перенумерованы от 0 до 15, в команде STM последний регистр не обязательно должен иметь номер, больший номера первого регистра. В следующих примерах мы увидим различные применения этой команды, причем во всех случаях для запоминания используется одна и та же область, состоящая из полных слов:

```
SAVEWDS    DS    16F
```

Пример 1.

```
STM     8,2,SAVEWDS
```

Это предложение записывает содержимое общих регистров 8, 9, 10, 11, 12, 13, 14, 15, 0, 1 и 2 в первые 11 полных слов области SAVEWDS в том порядке, в каком они перечислены.

Пример 2.

```
STM     5,7,SAVEWDS
```

Это предложение записывает содержимое общих регистров 5, 6 и 7 в первые три полных слова области SAVEWDS.

Пример 3.

```
STM     13,0,SAVEWDS+8
```

Это предложение записывает содержимое общих регистров 13, 14, 15 и 0 в третье, четвертое, пятое и шестое слова области SAVEWDS.

Пример 4.

```
STM     9,8,SAVEWDS
```

Это предложение записывает в область SAVEWDS содержимое всех 16 регистров. Первым записывается содержимое регистра 9, вторым — регистра 10 и т. д.

3. Команды загрузки

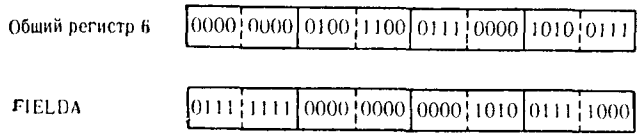
Команды загрузки предназначены для занесения (загрузки) в регистры данных и чисел. Эти команды могут загрузить в один регистр от 1 до 4 байтов.

Фактически команды загрузки выполняют функции, противоположные функциям команд записи в память, — данные, находящиеся в областях памяти, числа, адреса или содержимое других регистров можно загрузить с помощью этих команд в один или несколько общих регистров.

Прочитать символ — IC. Хотя название этой команды не содержит слова «загрузка», ее в полной мере можно отнести к командам этой группы. С ее помощью содержимое 1 байта памяти помещается в младший байт общего регистра. Введение этого байта не изменяет содержимого остальных 3 байтов регистра.

Команда IC может быть использована для изменения адреса, содержащегося в регистре, для сохранения или восстановления определенного значения в части регистра и для подобных

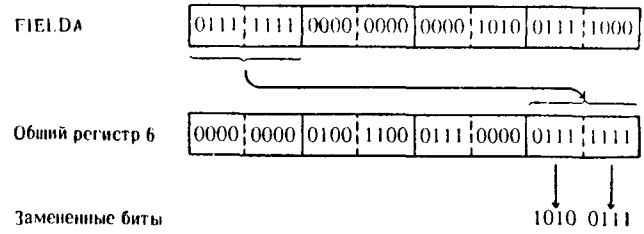
применений. Рассмотрим содержимое общего регистра 6 и поля FIELDA:



При выполнении предложения

```
IC      6,FIELDA
```

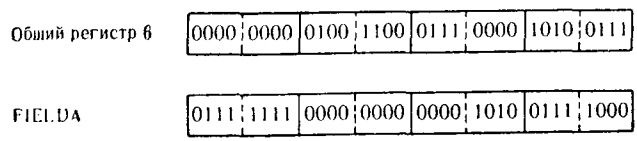
произойдет следующее:



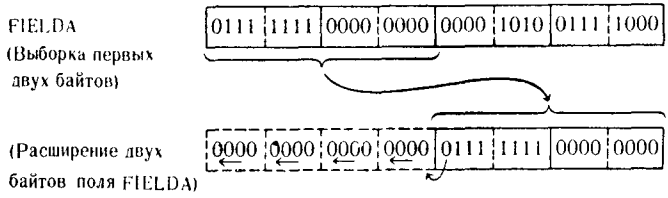
Как мы видим, выполнение команды вызвало загрузку в младший байт регистра 6 содержимого первого байта области FIELDA.

Загрузка полуслова — LH. Команда LH применяется для загрузки в общий регистр содержимого 2 байтов из области памяти, размещенной на границе полуслова. Содержимое полуслова сначала извлекается из области памяти, расширяется до конфигурации полного слова и затем помещается в общий регистр первого операнда. Как и команда IC, эта команда часто применяется для перезаписи значения константы в регистре. Это значение может представлять адрес, счетчик циклов или другую величину по усмотрению программиста. Чтобы лучше оттенить сходство и отличие команд IC и LH, в следующем примере использованы те же величины, что и в предыдущем.

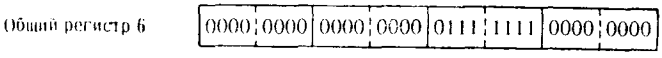
```
FIELDA  DS      F
*
        LH      6,FIELDA
```



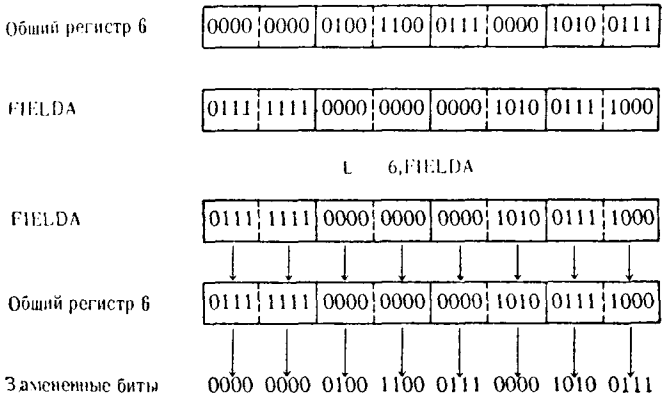
Значение полуслова, содержащееся в первых 2 байтах области FIELDA, сначала извлекается, а затем расширяется до размера полного слова с помощью распространения значения знакового бита на 16 старших битов.



Затем в общий регистр помещается новое содержимое:

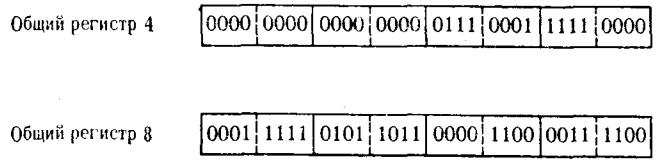


Загрузка — L. Команда L противоположна команде ST. 4 байта данных, адресуемых вторым операндом, загружаются в общий регистр первого операнда. С помощью этой команды чаще всего устанавливают начальное значение в регистре или восстанавливают в нем адрес или число, которые были ранее записаны в память для последующего использования. Рассмотрим эту команду с применением уже знакомых нам величин.

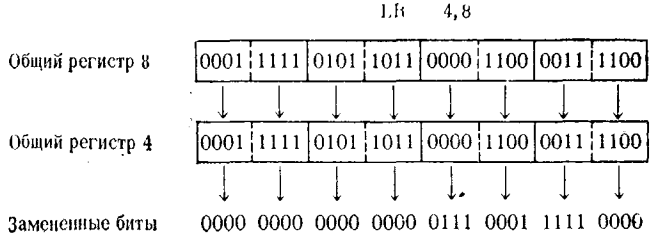


Загрузка — LR. Эта команда помещает содержимое общего регистра второго операнда в общий регистр первого операнда. Пересылаемые данные не изменяются. Содержимое общего регистра, откуда считываются данные, сохраняет свою прежнюю конфигурацию. По существу эту команду, как и подобные ей команды пересылки, загрузки и записи в память, можно рассматривать как средство дублирования данных. Данные пересылаются в другие области, а в прежних сохраняется первоначальное содержимое.

Предположим, что содержимое общих регистров 4 и 8 таково:



Выполняемая команда и производимые ею действия показаны ниже:



Загрузка групповая — LM. Команда LM используется для загрузки в два или более общих регистра значений, находящихся в соответствующем количестве соседних полных слов памяти. Число загружаемых слов определяется числом регистров, указанных первыми двумя операндами команды. Регистры указываются следующим образом: первый операнд указывает первый регистр, подлежащий загрузке, а второй операнд указывает последний из ряда регистров, подлежащих загрузке. Например, можно написать такую команду:

```
LM      7,10,DATAFWDS
```

Это предложение указывает, что в регистры 7, 8, 9 и 10 должны быть загружены значения, содержащиеся в четырех полных словах памяти, расположенных друг за другом, начиная с адреса DATAFWDS.

Еще один пример:

```
FULLWD1     DS      F
FULLWD2     DS      F
FULLWD3     DS      F
FULLWD4     DS      F
FULLWD5     DS      F
FULLWD6     DS      F
*
ROUTINEA    LM      8,13,FULLWD1
```

После выполнения этой команды общий регистр 8 будет содержать величину из поля FULLWD1, общий регистр 13 — из FULLWD6.

Итак, если данные расположены в соседних словах памяти, их можно загрузить в общие регистры с помощью одной команды LM.

Загрузка адреса — LA. Команда LA применяется для того, чтобы загрузить в общий регистр адрес заданной области памяти, а также для загрузки в регистр числа. Загружаемый адрес, указанный вторым операндом, может быть представлен символическим именем или абсолютным значением, записанным в десятичной форме. Следует отметить, что значение второго операнда, будь то генерируемый адрес или заданное число, загружается только в 3 младших байта регистра, а четвертый байт устанавливается в нуль.

Применение команды показано в следующих примерах.

Пример 1.

```
LA      11,SUBRTE1
```

Если во время выполнения программы адрес, соответствующий области, указываемой именем SUBRTE1, имеет значение 83596 (или 1486C в шестнадцатеричном коде), то это число и будет загружено в общий регистр 11. После этого содержимое регистра 11 примет вид

		Адрес SUBRTE1							
Регистр 11	(Двоичн.)	0000	0000	0000	0001	0100	0110	1000	1100
	(Шестн.)	0	0	0	1	4	6	8	C

Пример 2.

```
LA      8,300
```

В этом примере в общий регистр 8 загружается число с фиксированной точкой +300. Таким способом можно устанавливать в регистре значение счетчика циклов или какой-либо другой величины.

Пример 3.

```
LA      4,2(0,4)
```

В этом примере в общий регистр 4 загружается число, равное сумме числа 2 и текущего содержимого общего регистра 4. Если, например, регистр 4 содержал число +3386 до выполне-

ния команды, то теперь он будет содержать число +3388. Это можно выразить такой формулой:

Новое содержимое регистра 4=Смещение+Содержимое индексного регистра+Прежнее содержимое регистра 4,

2+0+3386=3388.

Таким образом, команду LA можно использовать для последовательного наращивания значения общего регистра с определенным шагом.

Загрузка и проверка — LTR. При выполнении этой команды содержимое общего регистра второго операнда загружается в регистр первого операнда и одновременно производится проверка, является ли это содержимое нулем, положительным или отрицательным числом. Единственным отличием этой команды от команды LR является установка соответствующего значения признака результата в PSW. За командой LTR в программе обычно следует предложение, которое проверит значение признака результата и укажет, какие действия следует предпринять по результатам такой проверки.

В качестве регистров первого и второго операндов можно указывать как разные регистры, так и один и тот же регистр, с тем чтобы проверить просто его содержимое.

Вот пример такой проверки:

```
LTR     6,6
BC      8,ZERORTE
BC      4,NEGRTE
```

Сначала происходит загрузка общего регистра 6 своим собственным содержимым, и признак результата устанавливается в соответствии с числом, содержащимся в этом регистре. Если значение величины в регистре равно нулю, происходит переход к подпрограмме ZERORTE. Если значение отрицательное, при выполнении третьего предложения происходит переход к подпрограмме NEGRTE. Если ни одно из этих условий не выполняется, т. е. в регистре находится положительное число, управление перейдет к следующей команде.

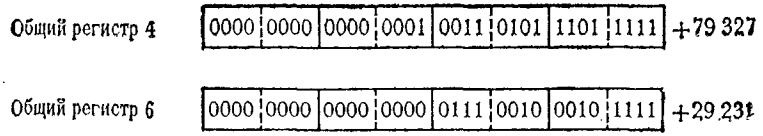
Загрузка дополнения — LCR. Эта команда выбирает число из регистра второго операнда, преобразует его в дополнительный код и помещает в таком виде в общий регистр первого операнда. Если содержимое регистра второго операнда является отрицательным числом, оно будет преобразовано в прямой

двоичный код; если оно было положительным числом, то оно преобразуется в отрицательное число, записанное в дополнительном коде.

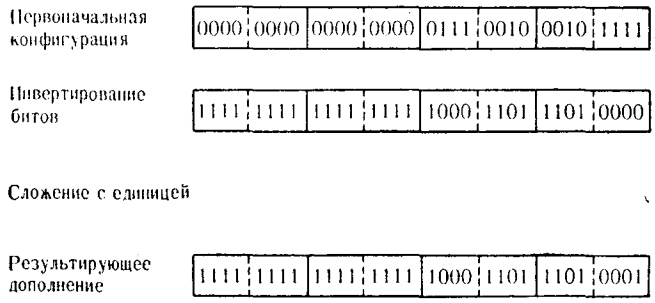
Пример 1.

```
LCR     4,6
```

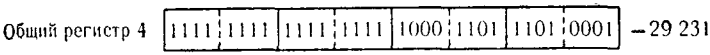
Общий регистр 4 содержит число с фиксированной точкой +79327, и общий регистр 6 содержит число с фиксированной точкой +29231. Первоначальная конфигурация этих регистров имеет вид



При выполнении команды из общего регистра 6 извлекается число, содержащееся в нем, и преобразуется в дополнительный код. Это преобразование производится так:



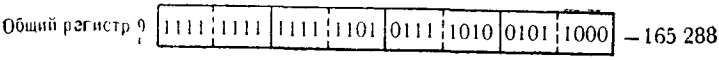
Это дополнение затем помещается в общий регистр 4, после чего регистр 4 будет иметь следующее содержимое:



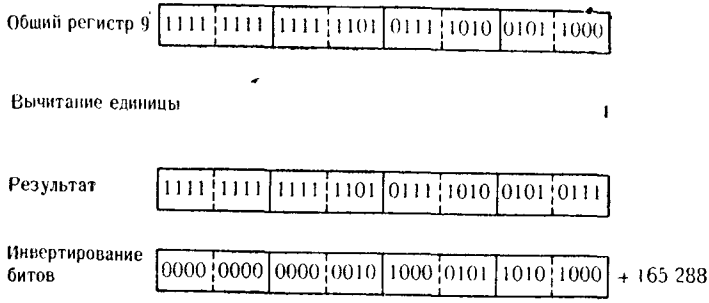
Пример 2.

```
LCR     9,9
```

В общем регистре 9 содержится число с фиксированной точкой —165288. Двоичное содержимое этого регистра следующее:



Преобразование содержащегося в регистре 9 отрицательного числа в свое дополнение, которое затем загружается в регистр 9, может быть представлено следующим образом:



Дополнение, представляющее собой величину +165288, помещается в общий регистр 9 вместо первоначального содержимого — числа —165288, находившегося здесь до выполнения команды LCR.

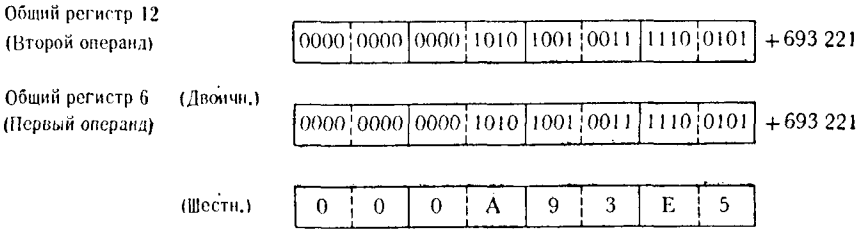
Загрузка положительная — LPR. Команда LPR помещает абсолютное значение числа с фиксированной точкой, содержащегося в общем регистре второго операнда, в общий регистр первого операнда. Если исходное число отрицательное, то перед загрузкой оно преобразуется в положительное путем перехода к дополнительному коду. Если содержимое регистра второго операнда до выполнения команды LPR уже было положительным, то оно без изменений помещается в принимающий регистр.

Если указать один и тот же регистр в обоих операндах команды LPR, то после выполнения команды в этом регистре будет абсолютное значение находившейся в нем ранее величины. Ниже приведены примеры выполнения этой команды.

Пример 1. В этом примере в регистре второго операнда содержится положительная величина, поэтому при загрузке в регистр первого операнда она не изменяется. Должна быть выполнена команда

```
LPR     6,12
```

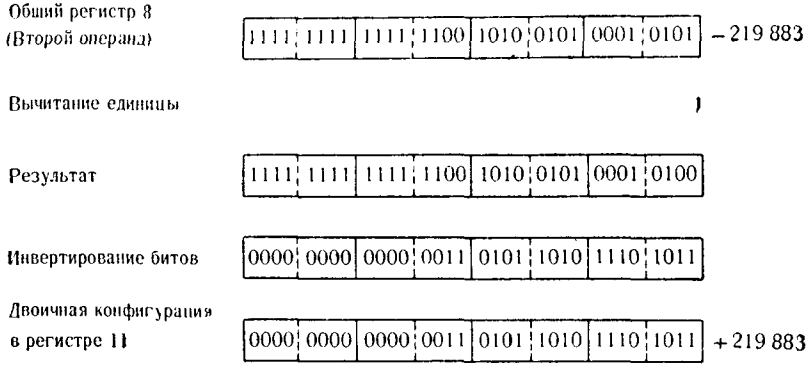
Если в общем регистре 12 содержится число +693221, то при загрузке в общий регистр 6 оно не изменится.



Пример 2.

```
LPR     11,8
```

Этот пример покажет преобразование отрицательного числа с фиксированной точкой в положительное при выполнении команды LPR. Пусть ко времени выполнения команды LPR в общем регистре 8 содержалось отрицательное число с фиксированной точкой —219883. Команда будет выполняться следующим образом:



Пример 3.

В этом примере один и тот же общий регистр используется в обоих операндах команды LPR. Это является гарантией того, что число с фиксированной точкой, содержащееся в этом общем регистре, обязательно станет положительным независимо от его представления до выполнения команды LPR.

```
LPR     4,4
```

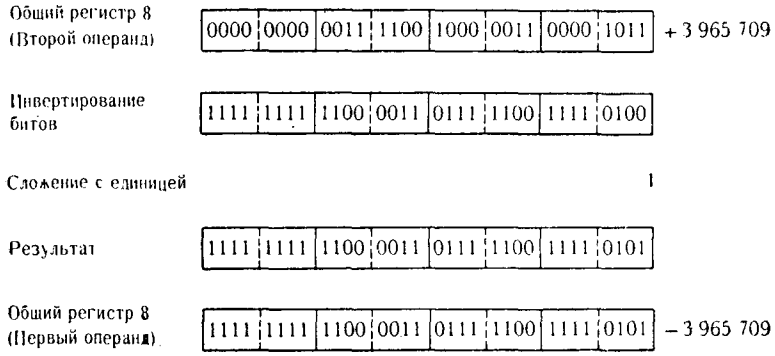
Если в регистре 4 первоначально содержалось число +36984, то и теперь в нем будет содержаться то же число. Однако если в общем регистре 4 содержалось число —2675037, то теперь в нем будет находиться двоичное представление числа с фиксированной точкой +2675037.

Загрузка отрицательная — LNR. Эта команда выполняет функцию, в точности противоположную команде LPR. Команда LNR сначала выбирает число с фиксированной точкой из общего регистра второго операнда и анализирует его знак. Если содержимое этого регистра отрицательное, то оно без изменения помещается в общий регистр первого операнда. Однако если содержимое второго операнда положительное, то команда LNR перед загрузкой в регистр первого операнда преобразует его в дополнительный код. Эта операция демонстрируется на следующих примерах.

Пример 1.

```
LNR     8,8
```

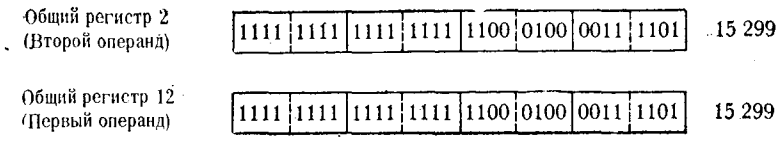
Если в общем регистре 8 до выполнения этой команды содержалось число с фиксированной точкой +3965709, то последовательность действий команды LNR будет такой:



Пример 2.

```
LNR     12,2
```

В этом примере предполагается, что в общем регистре 2 находится число с фиксированной точкой —15299. Так как это число отрицательное, то оно загружается без изменений в общий регистр 12.



4. Операции сдвига

Термин *сдвиг* используется для описания перемещения битов информации в пределах одного или более последовательно расположенных общих регистров. Перемещение осуществляется на число двоичных позиций, определяемое вторым операндом команды сдвига. Несмотря на то что при помощи команды сдвига можно умножать или делить числа с фиксированной точкой, на практике этой возможностью обычно не пользуются, в связи с тем, что множитель или делитель может быть только степенью двойки: 2, 4, 8, 16, 32, 64, 128 и т. д. Чаще всего команды сдвига используются для изменения, перемещения или выделения определенной группы битов, содержащейся в пределах одного или более общих регистров.

Эти случаи применения обсуждаются ниже, при описании отдельных команд сдвига. В примерах команд сдвига будет использовано как двоичное, так и шестнадцатеричное представления содержимого регистров.

Команды сдвига кодов. Команды сдвига кодов воздействуют на все биты в пределах указанного регистра или регистров, включая знаковой разряд. При выполнении этих команд считается, что во всех позициях содержатся логические значения 0 или 1 и все эти позиции равнозначны. Поэтому при выполнении сдвига кодов знаковый разряд перемещается в соответствии с перемещением содержимого других разрядов.

Сдвиг влево кода — SLL. Команда SLL выполняет сдвиг в пределах одного общего регистра максимально на 32 двоичные позиции. Содержимое регистра первого операнда сдвигается влево на число разрядов, определяемое вторым операндом. Значение знакового разряда игнорируется, и он сдвигается вместе с остальными разрядами. По мере того как содержимое указанного общего регистра сдвигается влево, освободившиеся младшие разряды заполняются нулями. Биты, сдвинутые за пределы регистра, теряются и замещаются битами, передвинутыми из младших разрядов. Потеря старшего значащего разряда не влияет на значение признака результата в PSW.

Пример 1.

В этом примере показана возможность изменения содержимого общего регистра, которое в программе рассматривается как представленное в шестнадцатеричной форме. Будем считать, что в общем регистре 7 содержится двоичная конфигурация, являющаяся эквивалентом шестнадцатеричного FEDCBA21. Эта

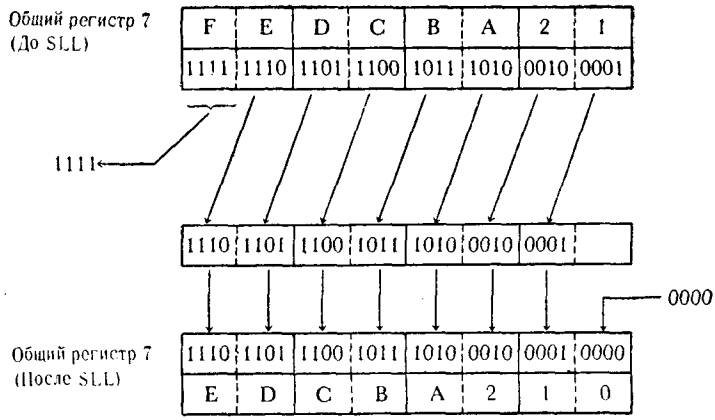
конфигурация выглядит следующим образом:

Общий регистр 7 (Двоичн.)	1111	1110	1101	1100	1011	1010	0010	0001
(Шестн.)	F	E	D	C	B	A	2	1

Должна быть выполнена следующая команда:

```
SLL     7,4
```

Это предложение осуществляет сдвиг текущего содержимого общего регистра 7 влево на четыре позиции.



Четыре старших разряда, содержащих биты 1111 (шестнадцатеричное F), в результате выполнения команды потеряны, в то время как четыре младших разряда заполнились нулями и представляют шестнадцатеричный нуль (биты 0000). Младшие 28 битов сдвинуты влево на четыре позиции, так что теперь они занимают 28 старших позиций.

Пример 2.

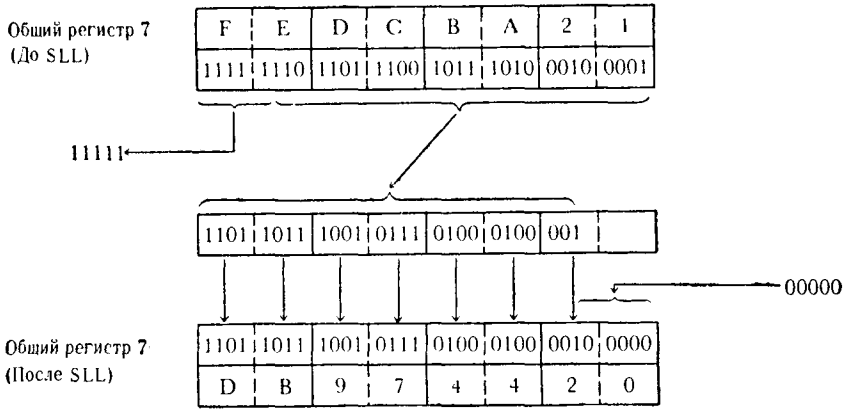
В предыдущем примере сдвиг был использован для перемещения значений всех битов влево на четыре позиции, т. е. на длину, соответствующую перемещению на один шестнадцатеричный символ.

Таким образом, каждый шестнадцатеричный разряд сдвинут на одну позицию в направлении старшего разряда в пределах общего регистра, старший шестнадцатеричный разряд теряется, а младший шестнадцатеричный разряд регистра заполняется нулевым значением. В следующем примере показан результат

сдвига того же первоначального значения на число позиций, которое не обеспечивает сохранение идентичности шестнадцатеричных разрядов. Первоначальное содержимое общего регистра 7 опять равно X'FEDCBA21'. Должна быть выполнена команда

```
SLL     7,5
```

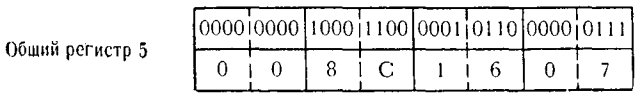
Это предложение указывает, что каждый бит общего регистра 7 должен перемещаться влево на пять позиций. Биты старших пяти позиций выдвинутся за пределы общего регистра и будут потеряны; пять младших позиций, освобожденных при сдвиге, заполнятся нулями.



Результирующее представление не имеет никакого сходства с начальным шестнадцатеричным представлением содержимого общего регистра 7. Несмотря на это, у программиста может быть достаточно оснований для использования сдвига этого типа в своей программе.

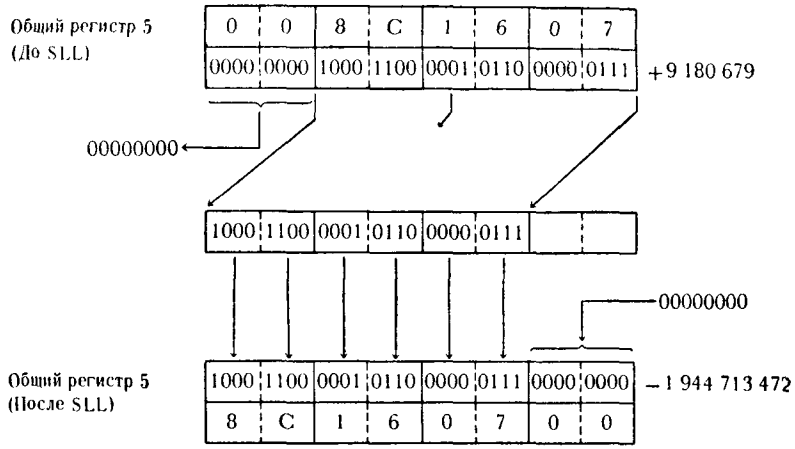
Пример 3.

Хотя команда SLL чаще всего используется для сдвига символов или шестнадцатеричных данных в пределах регистра, с ее помощью можно также выполнять сдвиг чисел с фиксированной точкой. При таком применении следует быть осторожным в связи с тем, что сдвиг кодов не выделяет особо старший знаковый разряд. В этом примере рассматривается проблема, которая может возникнуть при применении сдвига кодов для чисел с фиксированной точкой. В общем регистре 5 находится число с фиксированной точкой +9180679, представленное в виде



Программист, желая умножить это число на +256, записал следующую команду:

```
SLL     5,8
```



Результат умножения числа +9180679 на +256 должен быть равен +2350253824. Хотя за пределы регистра не был сдвинут ни один значащий разряд, некорректное использование команды сдвига кодов привело к появлению единицы в старшем разряде. Это является признаком того, что в общем регистре содержится отрицательное число с фиксированной точкой, а младшие 31 разряд содержат целую часть этого числа. Вместо правильного результата +2350253824 команда SLL сформировала отрицательное число с фиксированной точкой, равное —1944713472, или в шестнадцатеричном представлении X'8C160700'. В связи с тем что вместо арифметического сдвига был использован сдвиг кодов, программист не получит сообщения об ошибке. Поэтому следует иметь в виду, что сдвиги кодов не следует применять в арифметических операциях с фиксированной точкой, если результаты такого применения не подвергаются тщательной проверке.

Сдвиг влево двойной кода — SLDL. Команда SLDL выполняется так же, как и команда SLL, за исключением того, что при этом используются два общих регистра. Эти регистры должны представлять собой пару соседних общих регистров, номер первого из которых четный, например 4 и 5, 8 и 9, 12 и 13, 2 и 3 и т. д. Аналогичное условие должно выполняться и для любой другой команды двухрегистрового сдвига — регистр первого операнда должен иметь четный номер. Двойной сдвиг регистов может быть использован для перемещения максимум 64 битов.

Используя команду SLDL, можно выделить данные, содержащиеся в старшем байте (или байтах) регистра, и затем послать их в область памяти командой записи в память.

Пример 1.

В этом примере данные, содержащиеся в старшем байте общего регистра 7, должны быть выделены для последующего их использования в программе. Допустим, что общий регистр 6 уже очищен и содержит значение +0 и что содержимое регистра 7 в шестнадцатеричной форме имеет вид C30000D3. Процесс выборки старшего байта может быть описан следующими командами:

```
SLDL    6,8
STC     6,HOLD
```

Первая команда производит сдвиг битов.

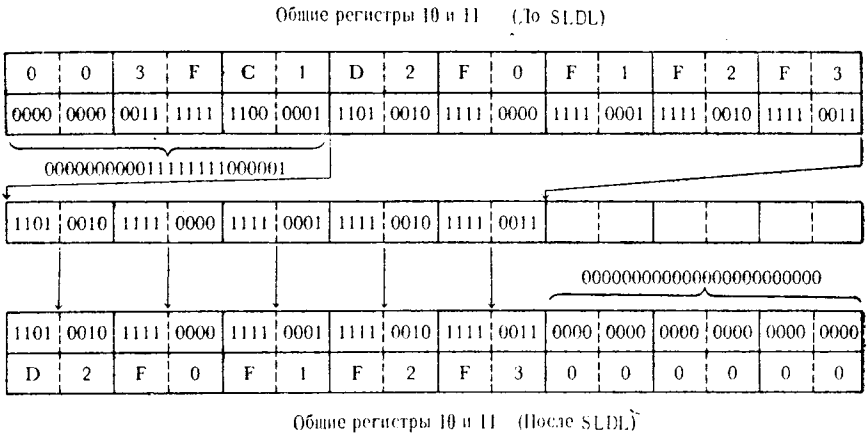
Теперь старший байт общего регистра 7 сдвинут в младший байт общего регистра 6. Вторая команда поместит этот байт в однобайтовую область памяти с меткой HOLD, после чего он может быть использован программой.

Пример 2.

Практически во всех случаях при помощи команд сдвига можно выделить любой байт или группу байтов, находящихся в общем регистре. В этом примере показано, как с помощью команды SLDL выделяются два средних байта общего регистра, которые затем выбираются командой STH:

```
SLDL    10,24
STH     10,HAFWRD
```

Пусть в общем регистре 10 первоначально содержится шестнадцатеричное X'003FC1D2', а в общем регистре 11— X'F0F1F2F3'. Команда SLDL выполняет следующее преобразование:



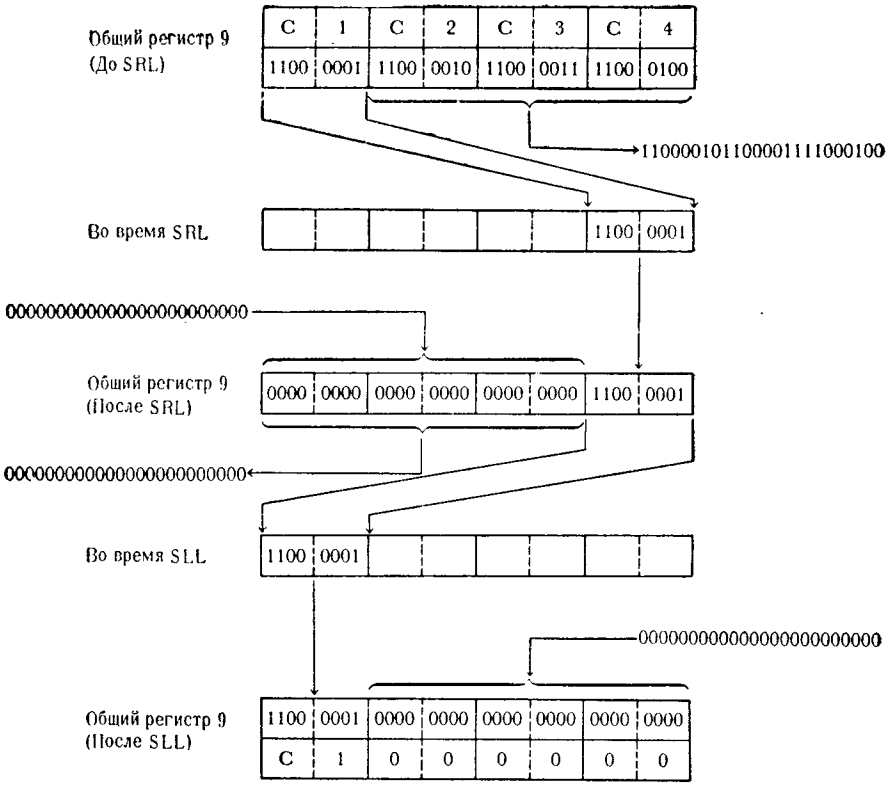
Два средних байта общего регистра 11, содержащие шестнадцатеричные символы 'F1F2', сдвинуты теперь в 2 младших байта общего регистра 10. Команда записи в память полуслова поместит эти 2 байта в полуслово с меткой HAFWRD.

Сдвиг вправо кода — SRL. Команда SRL выполняет в пределах общего регистра сдвиг битов максимально на 32 позиции. Двоичная конфигурация общего регистра первого операнда перемещается вправо на число позиций, определяемое вторым операндом. Как и в случае любого сдвига кодов, старший знаковый разряд не выделяется, а сдвигается так же, как и остальные биты данных. По мере того как содержимое указанного регистра перемещается вправо, освободившиеся старшие позиции заполняются нулями. Биты, сдвинутые за пределы младшего разряда общего регистра, теряются. Значение признака результата в PSW не отражает потерю (сдвиг за пределы регистра) значащего бита. Команду SRL можно использовать для деления, причем делителем может быть только положительное число, равное степени 2 (2, 4, 8, 16, ...). Эта команда может быть использована совместно с командой SLL для выноса нулевого младшего байта или байтов за пределы общего регистра. Некоторые применения рассматриваются в следующих примерах.

Пример 1. В этом примере показано, как совместное применение команд SRL и SLL может быть использовано для очистки

трех крайних правых байтов общего регистра. Таким же способом можно заполнить нулями любое количество двоичных разрядов или байтов вплоть до целого регистра. В этом примере общий регистр 9 содержит шестнадцатеричную конфигурацию X'C1C2C3C4', представляющую символы ABCD в коде EBCDIC. В результате выполнения команд, приведенных ниже, шестнадцатеричное представление содержимого этого регистра примет вид X'C1000000'.

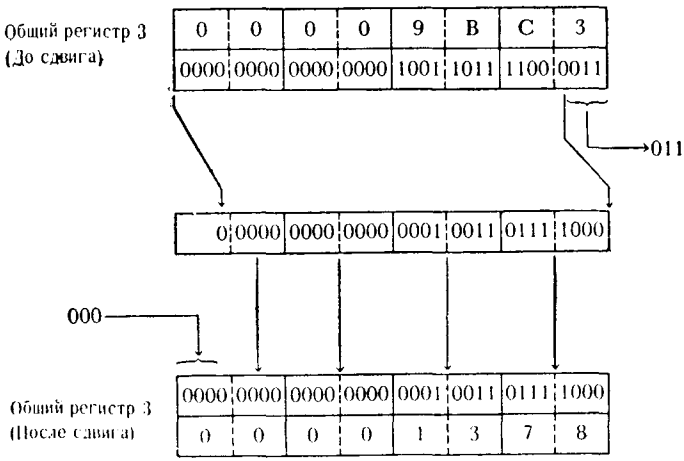
```
SRL     9,24
SLL     9,24
```



Как показано на этом примере, старший байт общего регистра 9 (X'C1') сдвигается вправо до тех пор, пока он не займет место младшего байта того же регистра. Затем он перемещается влево до тех пор, пока не займет первоначальное положение, причем при втором сдвиге происходит заполнение нулями младших 24 позиций, содержимое которых было сдвинуто за пределы регистра командой SRL.

Пример 2. В этом примере показано деление на число 8; оно возможно только потому, что 8 является степенью числа 2. Применение сдвига кодов для целей деления ограничивается положительным делимым, что связано с одинаковой обработкой этой командой всех разрядов, включая знаковый. Кроме того, при сдвиге вправо одного регистра частное от деления сохраняется, а остаток сдвигается за пределы регистра и теряется. Регистр 3 содержит число с фиксированной точкой +39875. Так как это число должно быть разделено на +8, то величина сдвига равна трем двоичным позициям.

SRL 3,3



После сдвига регистр 3 содержит шестнадцатеричное число X'1378', равное 4984, — частное от деления числа +39875 на +8. Заметьте, что 3 бита, сдвинутые из младших разрядов за пределы общего регистра 3, представляют число +3 — остаток, полученный при выполнении деления.

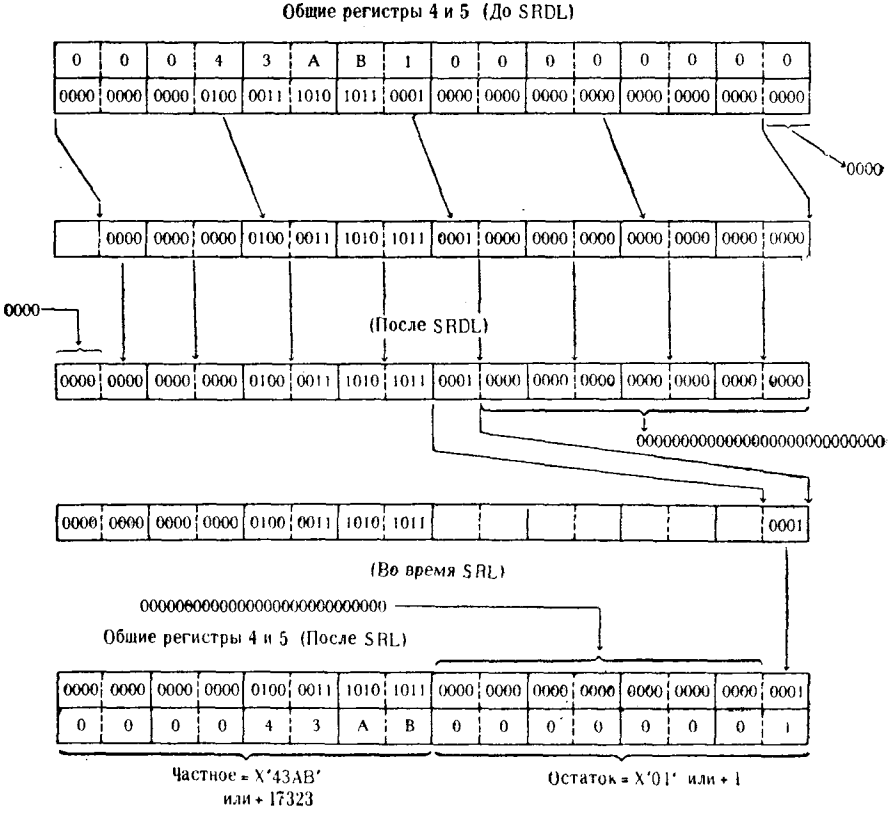
Сдвиг вправо двойной кода — SRDL. Отличие этой команды от команды SRL состоит в том, что SRDL сдвигает содержимое двух последовательных общих регистров. Первый из этих регистров должен иметь четный номер. Этот номер задается в первом операнде. Второй операнд определяет число позиций, на которое должен осуществиться сдвиг. Максимальное число двоичных позиций сдвига равно 64. Как и в случае команды SRL, эта команда может использоваться с теми же ограничениями для выполнения операции деления. Значение делимого должно быть положительным числом, а делителем может быть только одно из значений степени 2. Однако эта команда помещает остаток в регистр с нечетным номером, а частное в регистр с четным

номером при условии, что делимое находилось в регистре с четным номером. Комбинированием команды SRDL с другими командами сдвига кодов можно выполнять самые разнообразные манипуляции над содержимым регистров. По желанию могут быть сдвинуты, инвертированы или изменены байты данных, десятичные числа, шестнадцатеричные цифры или значения отдельных битов.

Пример 1. В этом примере демонстрируется программный прием выравнивания остатка в случае, если программист использует с соответствующими ограничениями команду SRDL для операции деления. В общем регистре 4 содержится положительное число с фиксированной точкой +277169, которое нужно разделить на +16. После деления методом сдвига остаток должен быть помещен в младшие разряды общего регистра 5. Для выполнения этой операции необходимы следующие команды:

```
SRDL    4,4
SRL     5,28
```

Эти команды выполняют следующие действия:



Первая команда осуществляет сдвиг вправо всех битов на четыре позиции. 4 бита, сдвинутые за пределы общего регистра 5, являются остатком. Для того чтобы представить остаток в виде числа с фиксированной точкой, эти 4 бита должны быть сдвинуты до правой границы регистра 5. Четыре позиции, занятые остатком, вычитаются из общего числа разрядов (32) общего регистра, и полученная разность (28) является числом позиций, на которое должен быть сдвинут остаток. Вторая команда выполняет сдвиг вправо кода на 28 позиций, и остаток принимает значение с фиксированной точкой +1.

Пример 2. Этот пример приводится с единственной целью — показать возможности совместного использования различных команд сдвига. Он не служит в качестве иллюстрации какого-либо конкретного применения, но может сослужить хорошую службу программисту при написании программ.

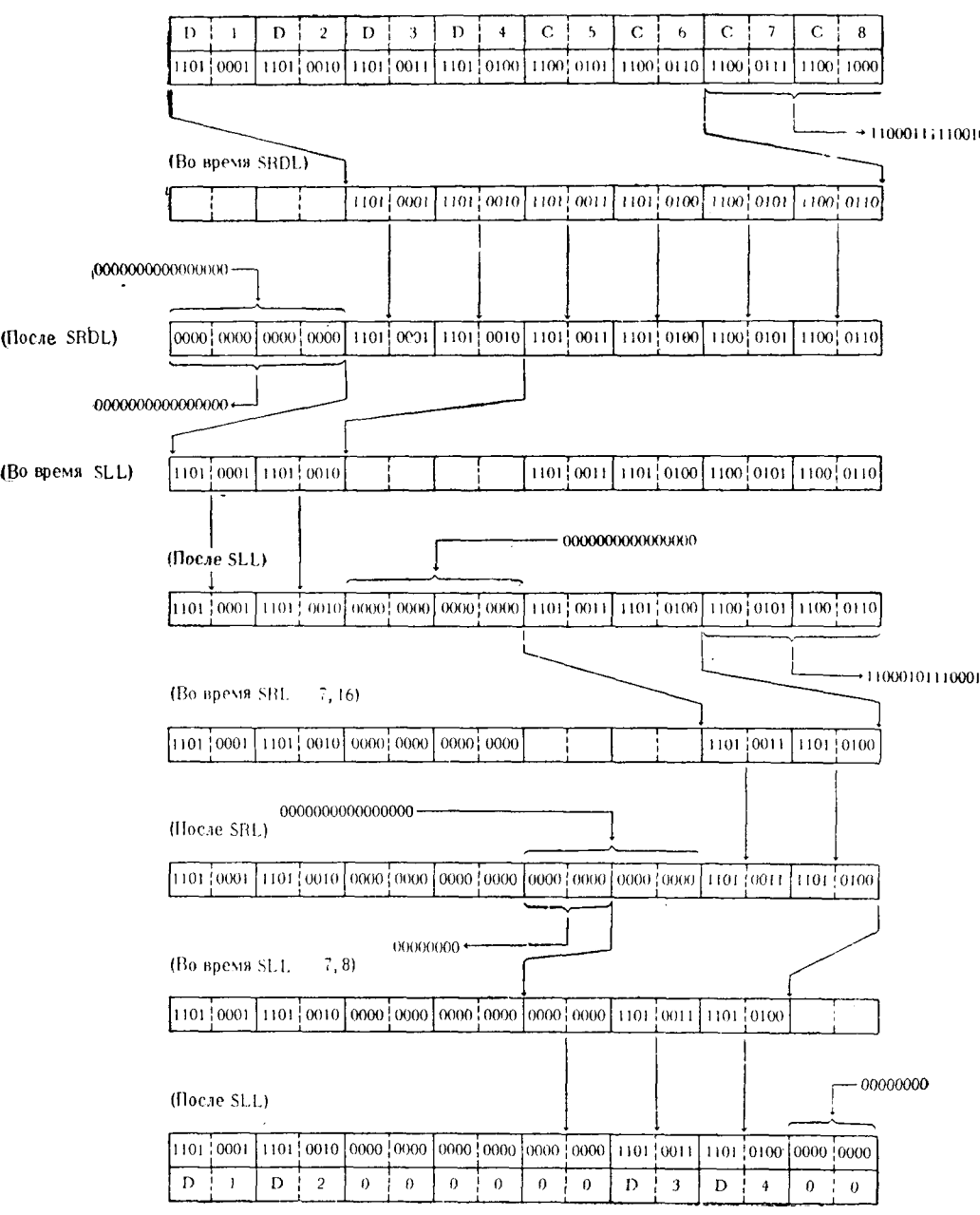
В этом примере общие регистры 6 и 7 содержат конфигурации X'D1D2D3D4' и X'C5C6C7C8' соответственно. Задача заключается в том, чтобы переслать содержимое 2 младших байтов (X'D3D4') регистра 6 в 2 средних байта регистра 7, первоначально содержащие конфигурацию X'C6C7'. Первый и последний байты регистра 7 должны быть очищены (должны приобрести значения X'00'), а 2 младших байта регистра 6 должны содержать X'0000'. Хотя та же задача может быть выполнена с помощью различных команд записи в память, загрузки и пересылки, это можно сделать, пользуясь только командами сдвига:

```
SRDL    6,16
SLL     6,16
SRL     7,16
SLL     7,8
```

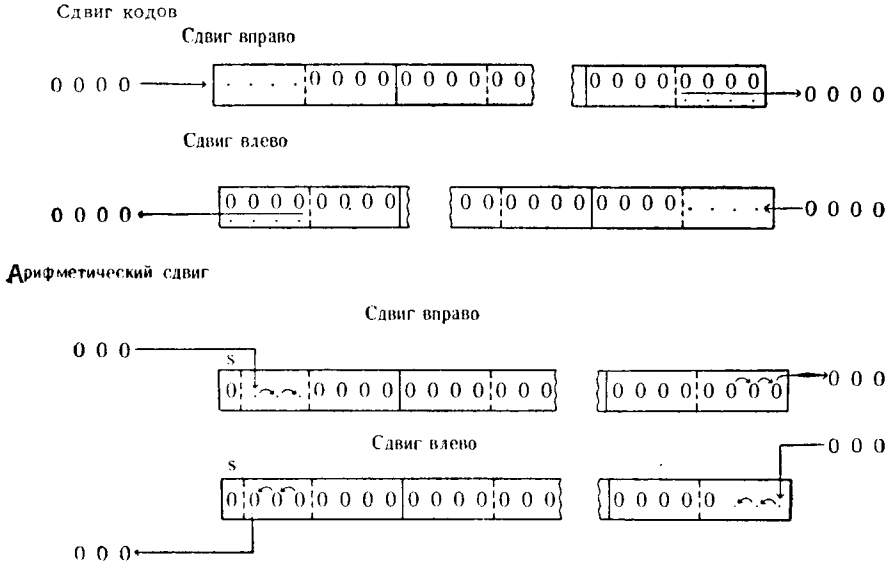
Содержимое регистров 6 и 7 (до выполнения команды SRDL) показано на стр. 264.

Приведенная последовательность команд состоит только из команд сдвига кодов.

Операции арифметического сдвига. В то время как при сдвиге кодов все разряды рассматриваются как равноправные, участвующие в операции сдвига, команды арифметического сдвига всегда рассматривают самый левый (старший) разряд как несдвигаемый знаковый разряд. Заполнение освободившихся старших или младших разрядов различно для команд сдвига кодов и арифметических сдвигов, что показано на следующем

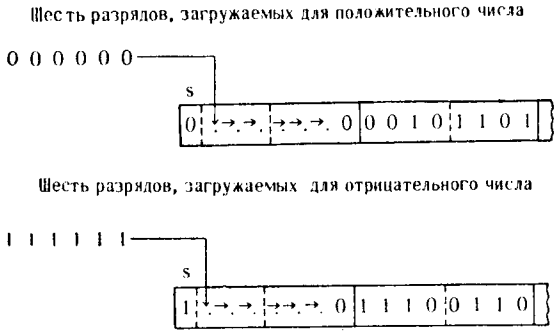

Общие регистры 6 и 7 (До SRDL)
D 1 D 2 D 3 D 4 C 5 C 6 C 7 C 8
1101 0001 1101 0010 1101 0011 1101 0100 1100 0101 1100 0110 1100 0111 1100 1000
(Во время SRDL)
0000000000000000
(После SRDL)
0000 0000 0000 0000 1101 0001 1101 0010 1101 0011 1101 0100 1100 0101 1100 0110
0000000000000000
(Во время SLL)
1101 0001 1101 0010 1101 0011 1101 0100 1100 0101 1100 0110
(После SLL)
0000000000000000
1101 0001 1101 0010 0000 0000 0000 0000 1101 0011 1101 0100 1100 0101 1100 0110
(Во время SRL 7, 16)
1101 0001 1101 0010 0000 0000 0000 0000 1101 0011 1101 0100
(После SRL)
0000000000000000
1101 0001 1101 0010 0000 0000 0000 0000 0000 0000 0000 0000 1101 0011 1101 0100
(Во время SLL 7, 8)
00000000
1101 0001 1101 0010 0000 0000 0000 0000 0000 0000 1101 0011 1101 0100
(После SLL)
00000000
1101 0001 1101 0010 0000 0000 0000 0000 0000 0000 1101 0011 1101 0100 0000 0000
D 1 D 2 0 0 0 0 0 0 D 3 D 4 0 0

примере:



Команды арифметического сдвига сохраняют содержимое регистра в формате с фиксированной точкой независимо от знака числа. Это достигается благодаря различию выполняемых функций в командах арифметического сдвига влево и арифметического сдвига вправо.

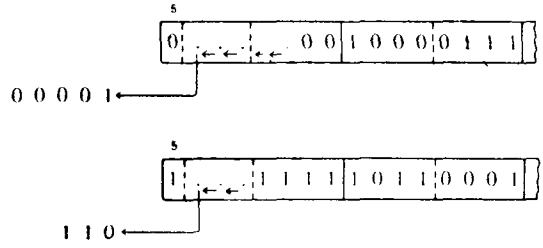
При выполнении арифметического сдвига вправо во все освобождаемые старшие разряды помещается значение знакового разряда:



Как здесь показано, значения битов, загружаемых в освободившиеся старшие разряды, совпадают со значением в знаковом разряде.

Целостность числа с фиксированной точкой при выполнении команды арифметического сдвига влево обеспечивается

«отбрасыванием» значений битов из позиции, предшествующей знаковому разряду. При этом выполняется еще одна функция, специфическая для арифметического сдвига влево. Если за пределы старшего разряда целочисленной части сдвигается бит, значение которого отлично от значения знакового разряда, то формируется сигнал переполнения, вызывающий программное прерывание. Бит, отличный по значению от знакового разряда, считается битом значащей цифры; для положительного числа им является единица, для отрицательного числа — нуль. Условия, при которых сдвиг значащего разряда может вызвать прерывание, иллюстрируются следующим примером:



Как только значащая цифра сдвигается за пределы старшего разряда целочисленной части числа с фиксированной точкой, фиксируется переполнение. Основываясь на предшествующей информации, можно заключить, что операции арифметического сдвига в основном предназначены для применений, использующих арифметические величины и действия над числами с фиксированной точкой.

Сдвиг влево арифметический — SLA. Команда SLA сдвигает 31 «числовой» разряд общего регистра первого операнда на число позиций, определяемое вторым операндом. Для всех команд арифметического сдвига под числовыми разрядами подразумеваются все разряды, кроме самого старшего. В обычном 32-разрядном регистре самый левый разряд рассматривается как знаковый, в то время как оставшиеся 31 разряд образуют целое число. В команде арифметического сдвига двойного регистра, состоящего из 64 разрядов, крайний левый разряд суммарного поля обоих регистров считается знаковым, а оставшиеся 63 разряда подлежат сдвигу.

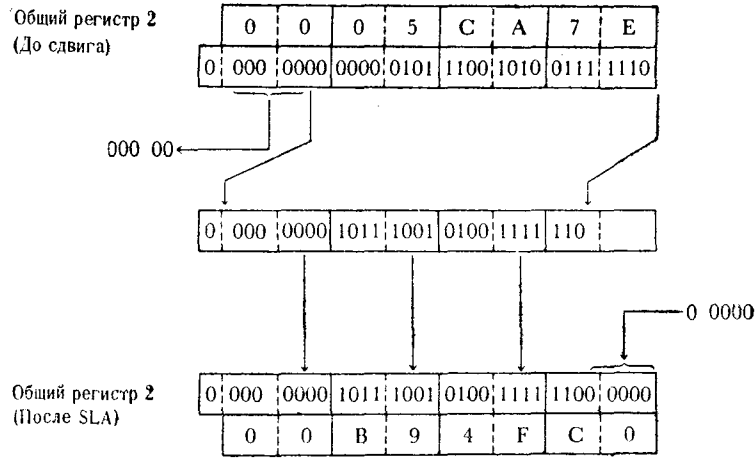
При выполнении команды арифметического сдвига влево биты будут сдвинуты влево на заданное число позиций. Биты, сдвигаемые из старшего разряда числовой части регистра, отбрасываются. Знаковый разряд не может быть сдвинут или изменен. Младшие разряды, освобождаемые от сдвигаемых влево битов, заполняются нулями. В случае сдвига из старшего раз-

ряда числовой части регистра значащей цифры регистрируется переполнение с фиксированной точкой. Арифметический сдвиг влево может быть использован для умножения при условии, что множитель может принимать только одно из значений степеней двойки (2, 4, 8, 16, 32, 128, 256 и т. д.) и что множимое является положительным числом. Если при выполнении с помощью сдвига операции умножения результат слишком велик для размещения в пределах 31-разрядного поля, то за пределы поля будет вынесен значащий бит, что соответствует условию переполнения с фиксированной точкой. На следующих примерах иллюстрируется выполнение команды SLA.

Пример 1. Пусть в общем регистре 2 содержится число с фиксированной точкой +379518. Программист намеревается с помощью команды SLA умножить это число на +32. Для выполнения этой задачи нужно закодировать команду арифметического сдвига влево на пять разрядов.

```
SLA     2,5
```

Выполнение этой команды выглядит следующим образом:



В результате сдвига содержимого общего регистра 2 получаем: 379518, умноженное на 32, равно 12144576.

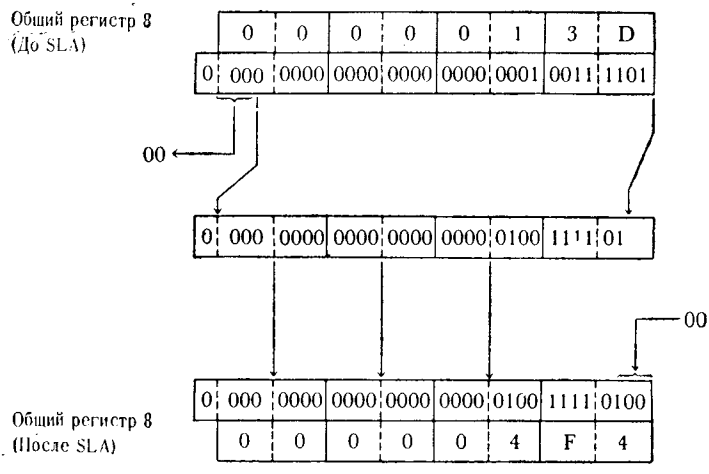
Число разрядов, на которое необходимо было осуществить сдвиг, определялось тем, что число 32 является пятым членом ряда степеней двойки: 2, 4, 8, 16, 32.

Пример 2. В этом примере программист хочет определить длину одной из нескольких таблиц переменной длины, содержа-

щихся в его программе. Пусть имеется 12 таблиц, каждая из которых состоит из четырехбайтовых сегментов. Хотя для размещения каждой таблицы зарезервирована максимальная длина 2048 байтов (512 сегментов), в программе имеется в виду использовать только ту часть таблицы, в которую были загружены данные в процессе выполнения этой программы. Для каждой таблицы известно количество занятых сегментов, но программист желает определить объем данных каждой таблицы в байтах, применив для этого команду сдвига. В связи с тем что длина каждого сегмента 4 байта, решение может быть получено выполнением следующей команды:

```
SLA     8,2
```

В этой команде предполагается, что в регистре 8 находится число используемых сегментов данных в какой-либо из названных таблиц, которое умножается на число +4 (длину сегмента в байтах) для того, чтобы найти полную длину данных в байтах. Если в этом примере число сегментов таблицы, находящееся в регистре 8, равно +317, то команда SLA выполняется следующим образом:



Команда SLA в данном случае выполнила функцию умножения — теперь в регистре 8 находится число с фиксированной точкой +1268, равное общей длине в байтах сегментов указанной таблицы.

Сдвиг влево двойной арифметический — SLDA. Команда SLDA — еще одна команда сдвига двойного регистра — рассма-

тривает совокупность двух регистров как состоящую из одного старшего знакового разряда и 63 битов поля, отводимого для целого числа. Как и в случае других сдвигов двойного регистра, выбранные регистры должны образовывать последовательную пару, причем номер первого из них должен быть четным, и команда сдвига должна адресовать регистр с четным номером в качестве первого операнда. Количество позиций сдвига 63-битного поля определяется вторым операндом команды. Биты, сдвигаемые из старшего разряда, непосредственно расположенного перед знаковым разрядом регистра с четным номером, теряются, подвергаясь предварительной проверке. Если сдвигается значащий бит, то генерируется сигнал переполнения с фиксированной точкой. Биты, сдвигаемые за пределы регистра с нечетным номером, включая его знаковый разряд, передвигаются в регистр с четным номером без какого-либо контроля. Если число с фиксированной точкой, находившееся в регистре с нечетным номером, имело отрицательное значение, то единица старшего знакового разряда этого регистра будет сдвинута в регистр с четным номером, где она будет представлять число из ряда степеней двойки, определяемое ее позицией после завершения сдвига. Младшие разряды, освобожденные в результате сдвига, заполняются нулями независимо от того, какое число (положительное или отрицательное) находилось в любом из регистров.

Пусть в регистрах 2 и 3 находятся числа +321516 и —6810 соответственно, тогда содержимое этих регистров выглядит следующим образом:

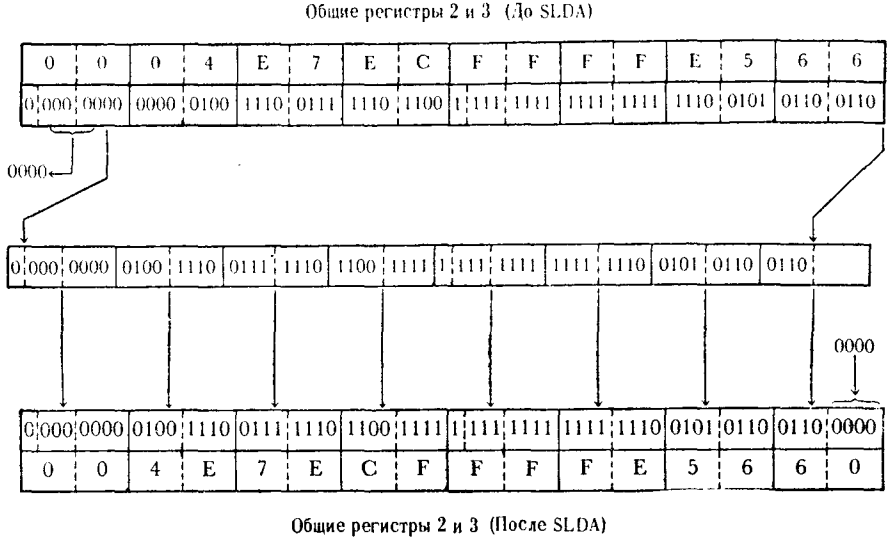
Общие регистры 2 и 3

0	0	0	4	E	7	E	C	F	F	F	F	E	5	6	6
0000	0000	0000	0100	1110	0111	1110	1100	1111	1111	1111	1111	1110	0101	0110	0110

Если каждый из этих регистров рассматривается как содержащий число с фиксированной точкой, то старший разряд считается знаковым разрядом. Однако если эти регистры используются совместно в команде SLDA, то только крайний левый разряд объединения регистров (в данном случае старший разряд регистра 2) будет считаться знаковым разрядом. Тогда команда

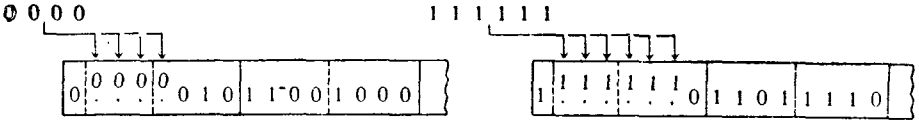
```
SLDA    2,4
```

произведет сдвиг всех 63 битов поля следующим образом:



Результатом сдвига является перемещение четырех старших разрядов, содержащих единицы, из регистра 3 в четыре младших разряда регистра 2. В регистре 2 теперь находится шестнадцатеричная конфигурация X'004E7ECF', соответствующая числу с фиксированной точкой +5144271, а в регистре 3 находится X'FFFE5660' или число с фиксированной точкой —108975.

Сдвиг вправо арифметический — SRA. Команда SRA сдвигает 31 разряд числа, содержащегося в регистре. Сдвиг происходит в направлении младших разрядов этого регистра. Разряды сдвигаются вправо на число позиций, определяемое вторым операндом команды. Как и в случае других команд арифметического сдвига, перемещение битов двоичного целого не затрагивает старший знаковый разряд. Биты, сдвигаемые из младшего разряда регистра, теряются без проверки на значимость. В освобождаемые старшие разряды помещаются значения знакового разряда.

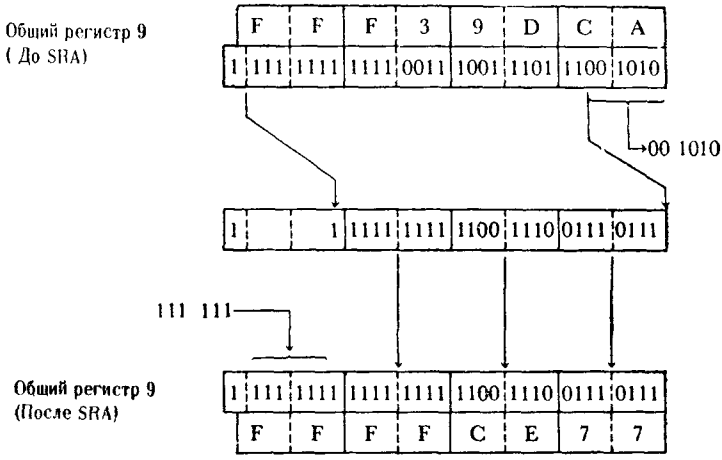


Для более подробной иллюстрации сказанного приведем два примера: первый представляет работу команды SRA для отрицательного, а второй — для положительного числа.

Пример 1. В регистре 9 находится число с фиксированной точкой —811574, которое должно быть сдвинуто на 6 разрядов

вправо арифметическим сдвигом. Выполняемая команда и результат выглядят следующим образом:

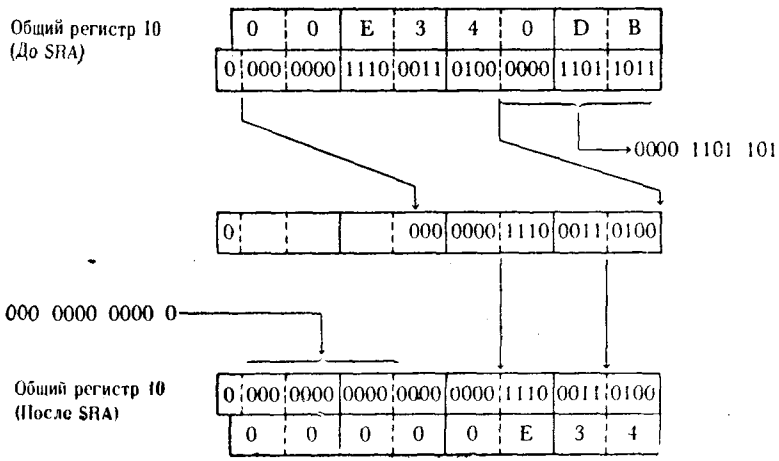
```
SRA     9,6
```



Как видно из примера, сдвиг привел к сдвигу из младших разрядов регистра значащих битов без какого-либо сообщения об этом. В освободившиеся старшие разряды загружаются единицы, т. е. значения, совпадающие со знаковым разрядом исходного отрицательного числа, находившегося в этом регистре.

Пример 2. В этом примере в регистре 10 содержится число с фиксированой точкой +14893275. Должен быть осуществлен сдвиг на 12 позиций. Запись и порядок выполнения команды будут следующие:

```
SRA     10,12
```



В этом примере по существу происходит деление заданного числа, содержащегося в регистре 10, на число 4096. Результат, содержащийся в регистре 10 (+3636), представляет частное от этого деления. Остаток от деления был потерян при сдвиге младших разрядов за пределы регистра. Значение делителя было определено числом сдвигаемых позиций — перемещение на 12 позиций соответствует двенадцатому члену ряда степеней двойки: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096.

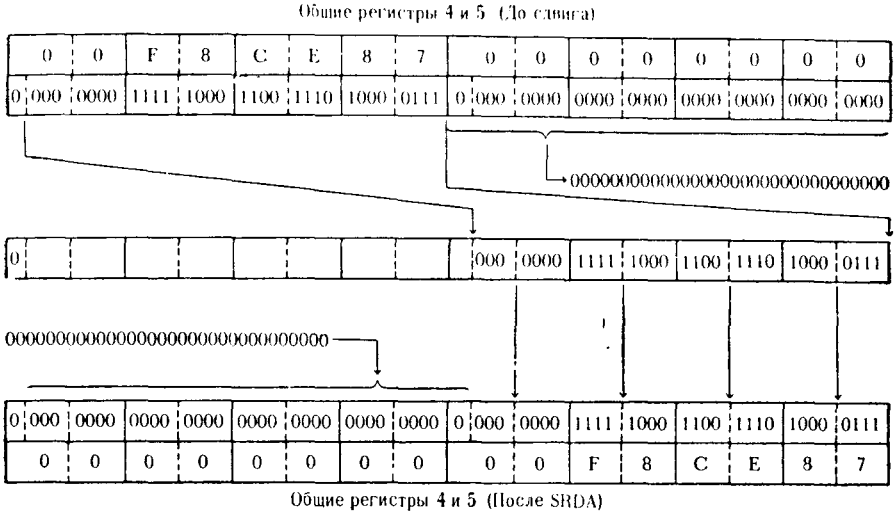
Сдвиг вправо двойной арифметический — SRDA. Команда SRDA следует правилам команды SRA, за исключением того, что она выполняет сдвиг пары общих регистров. Вследствие этого максимальная величина сдвига, выполняемого этой командой, составляет 63 разряда, что равно длине, отводимой для целочисленной части. Как и во всех командах двойного сдвига, считается, что старший разряд регистра с нечетным номером не выполняет функции знакового разряда. Он считается частью числа и сдвигается вместе с другими битами целого числа. Самый левый бит пары регистров считается знаковым битом всего числа и как таковой не может быть сдвинут или изменен этой командой. Все биты, сдвинутые за пределы младшего разряда регистра с нечетным номером, независимо от их значений считаются потерянными. Освободившиеся старшие разряды заполняются битами того же значения, что и значение знакового разряда: нулями для положительных чисел и единицами для отрицательных чисел. Положительное число с фиксированной точкой, находящееся в регистре с четным номером, может быть целиком сдвинуто в соседний регистр с нечетным номером, при этом произойдет установка регистра с четным номером в +0. Пусть в регистре 4 находится число с фиксированной точкой +16295799, тогда команда имеет вид

```
SRDA    4,32
```

и процесс ее выполнения можно представить диаграммой на стр. 273.

Содержимое, первоначально находившееся в регистре 4, переместилось в регистр 5, и этой же командой для последующего использования очищен регистр 4. Для достижения того же результата с использованием команд загрузки потребовались бы две команды.

Командой SRDA можно выполнить простейшую операцию деления положительного числа, где делитель — одно из чисел ряда степеней двойки. Делимое сдвигается вправо, и после выполнения сдвига в правом регистре (с нечетным номером) располагается остаток. Затем в регистре с нечетным номером выполняется сдвиг вправо кода с такой длиной смещения, которая переместит младший бит остатка в младший разряд регистра. Программа выполнения подобного деления, где число +93268

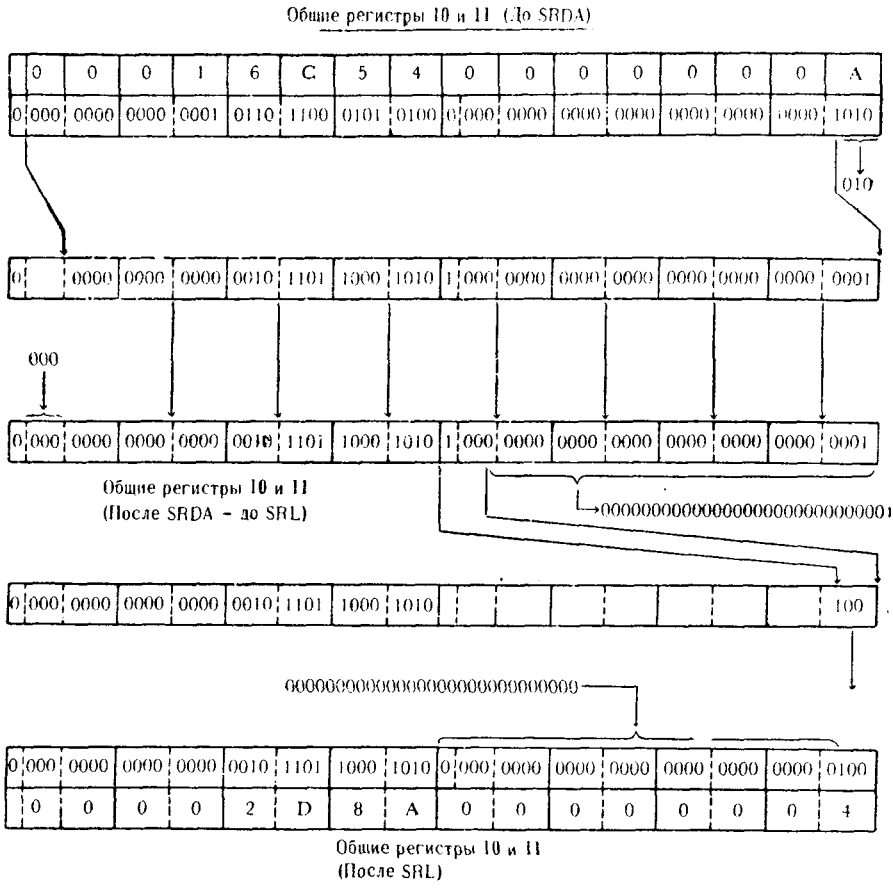


должно быть поделено на + 8, может быть записана в виде

```
SRDA    10,3
SRL     11,29
```

Хотя и не имеет никакого значения, каково содержимое регистра 11 в момент выполнения этих команд, все же примем, что в нем содержится число +10. Тогда операции, производимые этими командами, будут выглядеть так, как показано на стр. 274. Результатом деления, осуществленным командой SRDA, числа 93268 на 8 является число 11658, а также остаток, представляющий собой 3 бита, сдвинутые в старшие разряды регистра 11. Команда SRL перемещает эти 3 бита в младшие разряды регистра 11, формируя тем самым правильное число с фиксированной точкой, представляющее остаток от деления, равный +4.

Объяснения работы команд сдвига, а также приводимые примеры и комментарии в основном относятся к изменению символьных данных, данных в шестнадцатеричном представлении и действиям над числами с фиксированной точкой. Это делалось только с целью объяснения, как выполняются эти команды, без каких-либо предполагаемых ограничений их использования. Различные разряды в пределах одного или двух регистров могут быть установлены в нуль, в единицу или переведены в любое заданное состояние посредством совместного использования различных команд сдвига. Битовый «переключатель», находящийся в пределах регистра, может быть легко переведен в состояние «выключено» или установлен в состояние «включено» при по-



мощи команд сдвига. Все эти применения невозможно описать исчерпывающе; чаще всего такое использование сдвига организуется для решения специфических задач, в которых в противном случае потребовалось бы выполнение целой группы команд для достижения той же цели.

В зависимости от условий, определяющих положение чисел в регистрах, и от возможного размещения результирующих чисел существует вероятность того, что некоторые из арифметических операций, выполненных в примерах этой главы, могли бы быть выполнены более эффективно при помощи специально для этого предназначенных команд.

Упражнения

Команды пересылки.

1. Формат команд пересылки бывает двух типов: формат SS и формат _____.

2. Команда MVN пересылает только правые _____ бита каждого байта из «посылающего» поля в «принимающее».

3. Команда _____ пересылает 1 байт непосредственных данных, которые определяются вторым операндом команды, в 1 байт памяти, адресуемый с помощью первого операнда.

4. Команда _____ пересылает содержимое левой тетрады каждого байта области памяти, определяемой вторым операндом, в соответствующую левую часть каждого байта области памяти, определяемой первым операндом; максимально допустимое количество пересылаемых полубайтов равно 256.

5. Следующая команда, пересылающая данные, определяется как «_____» пересылка. После выполнения этой команды FIELD7 будет содержать байты данных, каждый из которых идентичен _____ байту того же поля.

```
SETLIKE MVC FIELD7+1(17),FIELD7
```

6. Если в BYTEA содержится код пробела (X'40'), а в BYTEB содержится X'F6', то следующая команда, мнемонический код которой _____, изменит содержимое BYTEA с пробела на символ 0 в коде EBCDIC.

```
ALTERFLD _____ BYTEA, BYTEB
```

7. Команда _____ перешлет правую тетраду каждого байта области памяти, адресованной вторым операндом, в соответствующую правую часть каждого байта области памяти, адрес которой указывается первым операндом и длина которой не превосходит 256 байтов.

8. Когда в команде MVC длина явно не указана, то она неявно определяется длиной поля, адрес которого указывается _____ операндом.

9. Команда _____ пересылает байты области памяти, адресованной вторым операндом, в область памяти, указанную первым операндом, причем неявно или ясно заданная первым операндом длина не превышает 256 байтов.

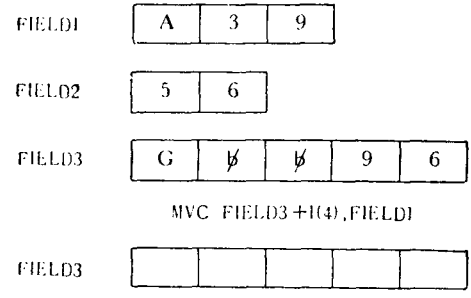
10. Если выполняется команда MVN, причем содержимое пересылаемого байта X'D7', а первоначальное содержимое «принимающего» байта X'F0, то после выполнения команды в принимающем байте будет находиться символ _____ кода EBCDIC.

11. Поля данных, превышающие по длине 256 байтов, могут пересылаться из одной области памяти в другую с помощью _____ команд MVC.

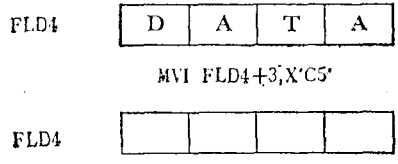
12. В следующих задачах одна или несколько областей памяти приводятся в той последовательности, в которой они рас-

полагаются в памяти. Проанализируйте содержимое полей и команду, а затем заполните символами приведенное на рисунке незаполненное принимающее поле. Содержимое этого поля запишите в исходном формате. Если перед пересылкой был шестнадцатеричный формат, то и после пересылки запишите результат в шестнадцатеричном формате и т. д.

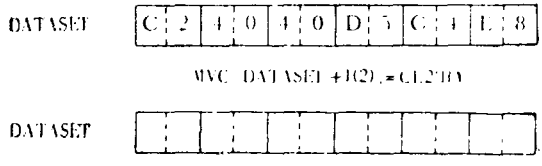
12а.



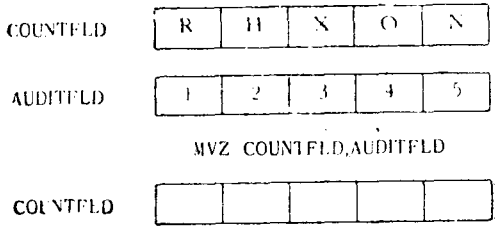
12б.



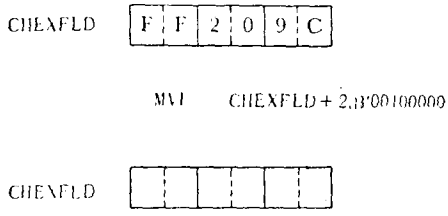
12в.



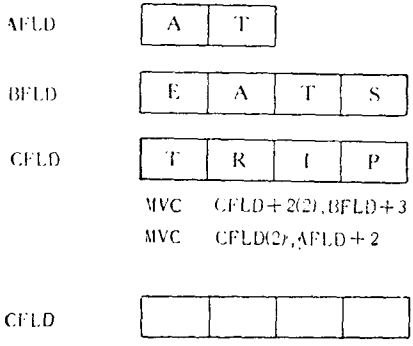
12г.



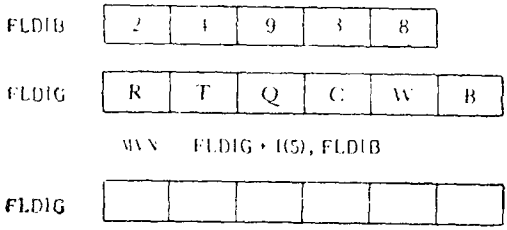
12д.



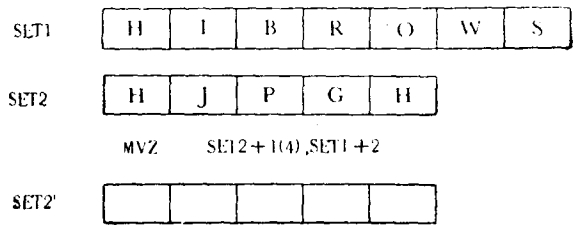
12е.



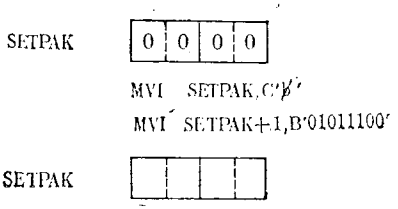
12ж.



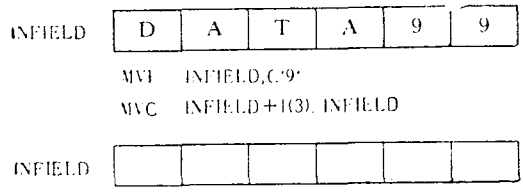
12з.



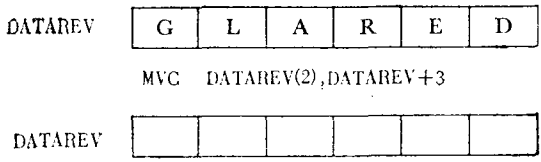
12и.



12к.



12л.



Команды записи в память

13. В следующей команде младший байт регистра первого операнда должен содержать действительный символ кода EBCDIC. (Да) (Нет).

```
STC     6,HOLDBYTE
```

14. Команда _____ занесет 2 младших байта из регистра, указанного в первом операнде, в первые 2 байта области памяти, расположенной на границе полуслова.

15. Команда _____ поместит содержимое ряда последовательно расположенных регистров в равное количество последовательно расположенных слов области памяти.

16. Команда _____ поместит содержимое регистра первого операнда в первые 4 байта области памяти, расположенной на границе слова и указанной вторым операндом.

17. При выполнении команды записи в память ST только 3 младших байта регистра, указываемого первым операндом, запоминаются в слове, указанном вторым операндом. (Да) (Нет).

18. Команда _____ поместит содержимое младшего байта регистра, указанного в первом операнде, в байт памяти, адрес которого указан во втором операнде.

19. Если в команде STH второй операнд определяет полное слово памяти, то младшие _____ байта регистра, указанного в первом операнде, запомнятся в _____ 2 байтах этого слова.

20. С помощью приведенной ниже команды производится запоминание содержимого _____ общих регистров в равном числе слов:

```
STM     12,4,FULLWDS
```

21. В следующей последовательности команд команда, обозначенная MODFY, изменит команду, обозначенную MVCHAR, таким образом, что при выполнении последней она перешлет всего _____ байтов. (В регистре 12 содержится X'00000009'.)

```
MODFY     STC     12,MVCHAR+1
MVCHAR    MVC     FIELDREC(5),FIELDSND
```

22. Если в регистре 10 содержится число с фиксированной запятой —319771, то после выполнения следующей команды в слове, обозначенном POSTVFWD, будет содержаться число с фиксированной точкой _____

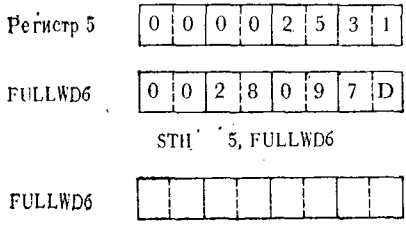
```
STOREPOS    ST    10,POSTVFWD
```

23. Следующая команда поместит содержимое указанных регистров в _____ последовательных байтах памяти, начиная с адреса, который указан меткой SETWORDS, если этот адрес соответствует границе слова памяти.

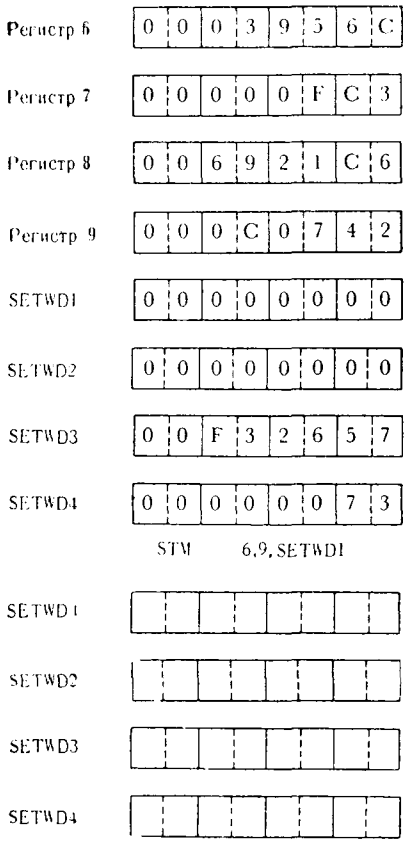
```
DATASET     STM     15,0,SETWORDS
```

24. В следующих задачах определены один или более регистров и областей памяти. Исследуйте содержимое указанных операндов и затем заполните символами поле результата, приведенное на рисунке. Записывайте содержимое этого поля в том же формате, в котором оно было первоначально; если оно было в шестнадцатеричном формате перед выполнением команды записи в память, то запишите содержимое тоже в шестнадцатеричном формате.

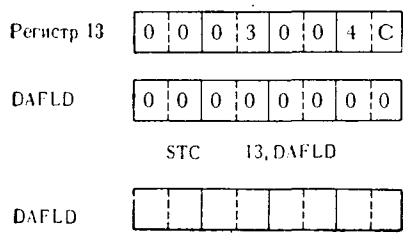
24а.



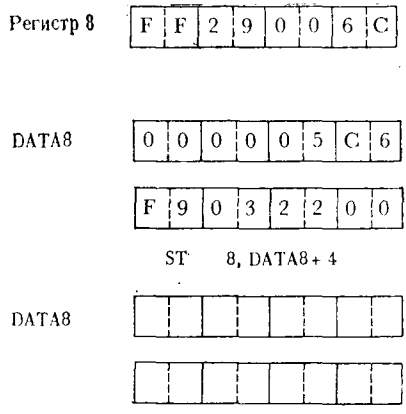
24б.


Регистр 6
0 0 0 3 9 5 6 C
Регистр 7
0 0 0 0 0 F C 3
Регистр 8
0 0 6 9 2 1 C 6
Регистр 9
0 0 0 C 0 7 4 2
SETWD1
0 0 0 0 0 0 0 0
SETWD2
0 0 0 0 0 0 0 0
SETWD3
0 0 F 3 2 6 5 7
SETWD4
0 0 0 0 0 0 7 3
STM 6,9,SETWD1
SETWD1
SETWD2
SETWD3
SETWD4

24в.


Регистр 13
0 0 0 3 0 0 4 C
DAFLD
0 0 0 0 0 0 0 0
STC 13,DAFLD
DAFLD

24г.



Команды загрузки

25. Команда _____ загружает 2 байта данных, лежащих на границе полуслова, из памяти в регистр в виде полного слова. 2 байта данных расширяются до полного слова при помощи распространения знакового разряда на все 16 старших разрядов.

26. Команда _____ загружает сформированный вторым операндом адрес, будь это действительный адрес или задаваемый меткой, в 3 младших байта регистра первого операнда. Старший байт регистра устанавливается в нуль.

27. Команда _____ загружает регистр первого операнда абсолютным значением числа, содержащегося в регистре второго операнда.

28. Команды загрузки могут размещать в регистрах от _____ до _____ байтов данных.

29. Одной из основных целей использования команды LTR является проверка содержимого регистра на _____ или _____.

30. Команда _____ загружает регистр первого операнда содержимым 4 байтов данных, расположенных на границе слова, адрес которого указан вторым операндом.

31. Команда _____ загружает 1 байт, адрес которого указан во втором операнде, в младший байт регистра первого операнда.

32. Команда _____ загружает в регистр первого операнда отрицательное число, по абсолютной величине равное числу, содержащемуся в регистре второго операнда.

33. Максимальное положительное число с фиксированной запятой, которое может быть загружено в регистр командой LH, равно _____.

34. Следующая команда загружает в указанные регистры в сумме _____ байтов памяти:

```
RELOAD    LM    9,0,REGAREAS
```

35. Команда _____ загружает в регистр первого операнда содержимое регистра второго операнда.

36. Команда _____ загружает число из регистра, указанного во втором операнде, в регистр, указанный в первом операнде, а затем устанавливает признак результата для индикации положительного, отрицательного или нулевого значения этого числа.

37. После выполнения следующей команды в старшем байте регистра 12 будет находиться шестнадцатеричное число X'_____'.

```
LOADADDR    LA    12,FIELDB+255
```

38. Первоначально в регистре 8 содержалось число X'712400FF'. После выполнения следующей команды в регистре 8 будет находиться X' _____ '.

```
LPR    8,8
```

39. Команда _____ загружает дополнение числа с фиксированой точкой, содержащегося в регистре второго операнда, в регистр первого операнда.

40. Команда _____ загружает два или более слов памяти, адресуемых последним операндом, в ряд последовательных регистров, определяемых двумя первыми операндами [1]).

41. Если полуслово памяти, содержащее буквенные символы AB в коде EBCDIC, загружается в регистр 8 при помощи команды LH, то шестнадцатеричное содержимое регистра 8 после этого станет

Регистр 8 

---

[1]) Как и во всех других случаях описания команд, содержащих три операнда, используемая автором нумерация отличается от принятой в документации на Систему/360 и соответствует порядку записи в графе «формат операндов». — *Прим. ред.*

42. После выполнения следующей команды в регистре 3 находится шестнадцатеричное X'03040506'. Следовательно, в регистре 7 должно находиться шестнадцатеричное X' _____ '.

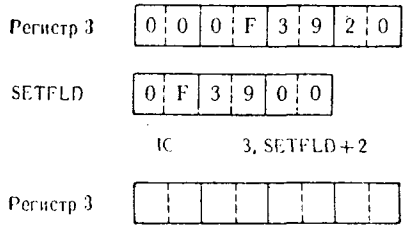
```
FLIPREG    LCR    3,7
```

43. Если после выполнения следующей команды в регистре 9 находится шестнадцатеричное X'003D0E46', то до выполнения этой команды в регистре 10 должно было находиться шестнадцатеричное X' _____ '.

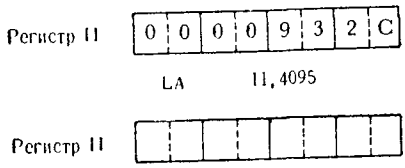
```
LTR    9,10
```

44. В следующих задачах задано содержимое одного или более регистров и областей памяти. Исследуйте содержимое приведенных операндов и команд, а затем заполните приведенные рисунки пустых регистров. Запишите результат в той же форме, в которой было определено исходное содержание регистра, т. е. если его содержимое перед выполнением команды загрузки было в шестнадцатеричном виде, то и после загрузки выразите его содержимое в шестнадцатеричном виде и т. д.

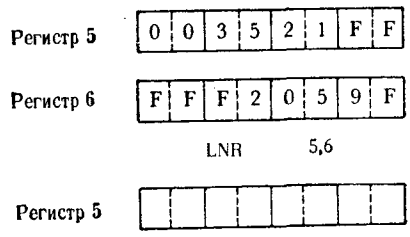
44а.



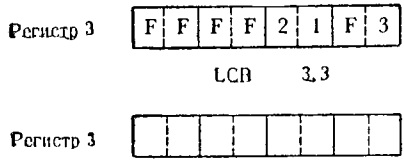
44б.



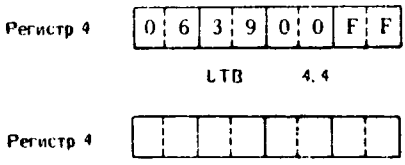
44в.



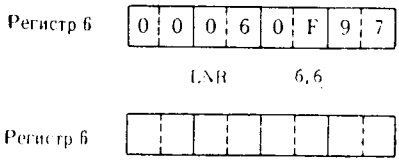
44г.



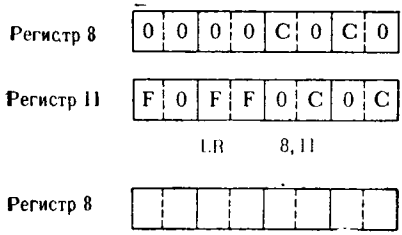
44д.



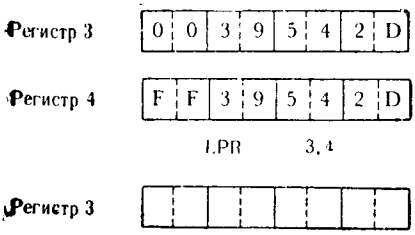
44е.



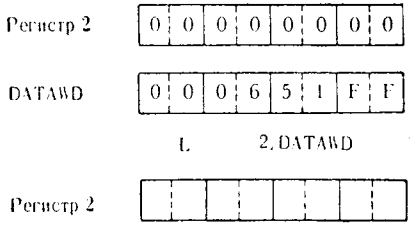
44ж.



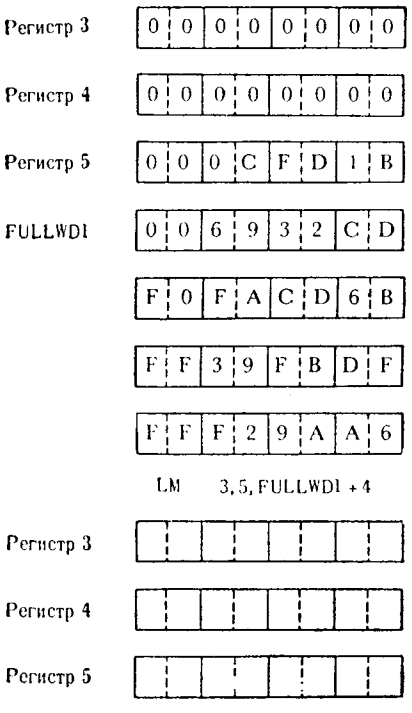
44з.



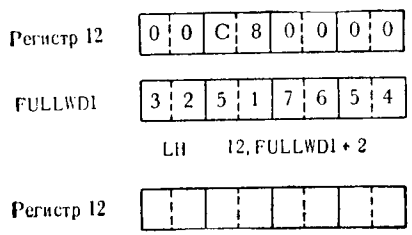
44и.


Регистр 2
0 0 0 0 0 0 0 0
DATAWD
0 0 0 6 5 1 F F
L 2, DATAWD
Регистр 2

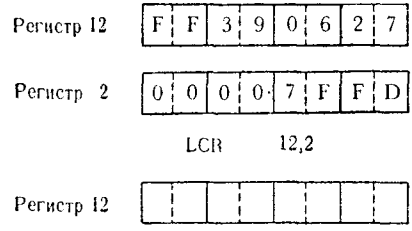
44к.


Регистр 3
0 0 0 0 0 0 0 0
Регистр 4
0 0 0 0 0 0 0 0
Регистр 5
0 0 0 C F D 1 B
FULLWD1
0 0 6 9 3 2 C D
F 0 F A C D 6 B
F F 3 9 F B D F
F F F 2 9 A A 6
LM 3, 5, FULLWD1 + 4
Регистр 3
Регистр 4
Регистр 5

44л.


Регистр 12
0 0 C 8 0 0 0 0
FULLWD1
3 2 5 1 7 6 5 4
LH 12, FULLWD1 + 2
Регистр 12

44м.



Команды сдвига

45. _____ операнд всех команд сдвига задает число двоичных позиций, определяющее длину сдвига.

46. Сдвиг влево на три двоичные позиции равносилен умножению положительного числа на _____.

47. Команда _____ сдвигает влево все биты одного регистра, включая и знаковый разряд.

48. При выполнении команды арифметического сдвига вправо свободные старшие разряды заполняются значениями, совпадающими со значением _____.

49. Команда _____ сдвигает вправо все биты целочисленной части в пределах одного регистра максимум на 31 позицию.

50. При выполнении двойного арифметического сдвига сдвигаются все биты, за исключением знакового разряда _____ регистра.

51. Команда _____ сдвигает влево все двоичные разряды, принадлежащие двум последовательным регистрам, максимум на 64 позиции.

52. В командах двойного сдвига первый операнд команды должен указывать _____ регистр пары.

53. Команда _____ сдвигает все двоичные разряды одного регистра вправо, заполняя освободившиеся старшие разряды нулями.

54. При выполнении команды арифметического сдвига влево вырабатывается сигнал переполнения всякий раз, когда за пределы старшего разряда поля числа сдвигается бит, значение которого _____ от значения знакового бита.

55. Команда _____ сдвигает влево 63 двоичных разряда целочисленной части, принадлежащих двум последовательным регистрам, максимум на 63 позиции.

56. В командах двойного сдвига знаковый разряд регистра с _____ номером расматривается как обычный разряд поля числа.

57. Команда _____ сдвигает вправо 31 двоичный разряд целочисленной части, принадлежащий одному регистру.

58. Во всех командах сдвига влево в освобождаемые позиции заносятся _____.

59. Команда _____ сдвигает вправо 64 двоичных разряда двух последовательных регистров максимум на 64 позиции, заполняя освобождаемые старшие разряды _____.

60. Сдвиг положительного числа вправо на пять позиций равносилен делению этого числа на _____.

61. Команда _____ сдвигает вправо 63 двоичных разряда целой части числа пары последовательных регистров максимум на _____позиций.

62. В результате двойного сдвига вправо кода на 32 двоичные позиции в регистре с четным номером содержится число _____

63. В следующих задачах задано содержимое одного или более регистров. Исследуйте содержимое указанных операндов, изучите команды и затем запишите результаты как в двоичной, так и в шестнадцатеричной форме.

63а.

Регистр 5

1000	0000	1110	1111	1000	0111	0100	0011

SRA 5,7

Регистр 5 (Двоичн.)

(Шестн.)

63б.

Регистры 2 и 3

0000	0000	0011	0110	0011	1111	1110	0000	0000	1111	1111	1110	0101	1001	1000	0100

SLDL 2,18

Регистры 2 и 3 (Двоичная и шестнадцатеричная формы)

(Двоичн.)

(Шестн.)

63в.

Регистр 10

0000	1000	1110	0010	1011	1010	0111	1011

SLL 10, 32

Регистр 10 (Двоичн.)

(Шестн.)

63г.

Регистр 9

1111	0100	1110	0101	1101	1011	1110	0111

SRL 9, 12

Регистр 9 (Двоичн.)

(Шестн.)

63д.

Регистры 6 и 7

1111	1000	0111	1011	1010	0000	1110	0101	0000	0000	0100	1101	1001	1111	0101	1011

SLDA 6, 4

Регистры 6 и 7 (Двоичная и шестнадцатеричная формы)

63е.

Регистры 10 и 11

1000	0000	0000	0000	0000	0000	1000	0001	0000	1111	1111	0101	1011	1101	0011	0000

SRDA 10, 31

Регистры 10 и 11 (Двоичная и шестнадцатеричная формы)

63ж.

Регистр 2	0000	0000	1000	1100	0001	1101	0100	0010

SLA 2, 7

Регистр 2 (Двоичн.)								
(Шестн.)								

63з.

Регистры 8 и 9

0010	1000	0000	1100	1111	1101	0001	1000	0111	1111	0000	0101	1111	1101	0001	1000

SLL 9, 16

SRDL 8, 16

Регистры 8 и 9 (Двоичная и шестнадцатеричная формы)

Глава 10

Арифметические операции над упакованными десятичными данными

А. КОМАНДЫ ДЕСЯТИЧНОЙ АРИФМЕТИКИ

Команда Pack — PACK (Упаковать)

Мнемоника	Код операции	Формат операндов
PACK	F2	$D_1(L_1,B_1),D_2(L_2,B_2)$

Команда PACK преобразует данные, указанные вторым операндом (предполагается, что это десятичные числа в зонном формате), в упакованный формат, располагая их по адресу, указанному первым операндом. Преобразованные данные помещаются полубайтами в область памяти, указанную первым операндом; обработка данных идет справа налево. Первая загружаемая половина байта представляет собой код знака, находившийся в зонной части самого правого (младшего) байта преобразуемого поля (указанного вторым операндом). Он помещается в младшие четыре разряда поля, указанного первым операндом. В следующий полубайт загружается десятичная цифра из того же байта поля, указанного вторым операндом, из которого был взят код знака. Теперь эти два полубайта из поля зонного формата, поменявшись местами, разместились в поле упакованных данных: значение зоны оказалось в правой, а десятичная цифра — в левой половине младшего байта. Начиная с этого места, команда будет теперь обрабатывать только правые половины (десятичные цифры) каждого байта поля зонного формата. Однако в поле упакованного формата они будут располагаться последовательно в каждом полубайте. Если команда обработала всю область данных зонного формата, а поле упакованного формата еще не заполнено целиком, то в оставшиеся старшие полубайты заносятся шестнадцатеричные нули. Если поле упакованных данных заполнено до окончания обработки данных поля зонного формата, то оставшиеся данные не используются. В связи с тем что не все данные пересылаются из одного операнда в другой (из зонных частей байтов пересылается только код знака), очевидно, что длина первого операнда может быть меньше длины второго операнда.

Требуемая для первого операнда длина равна длине второго операнда, увеличенной на 1, деленной на 2 и округленной до ближайшего большего целого.

Эта формула применяется следующим образом:

	Пример 1	Пример 2
Длина поля второго операнда	5	10
Прибавляем 1 байт для зоны знака	+1	+1
Сумма	6	11
Сумма, деленная на 2	3	$5^1/_2$
Округление до ближайшего большего целого		6
Требуемая длина для первого операнда	3	6

Максимальная длина, неявная или явная, для любого из операндов равна 16 байтам.

Признак результата не изменяется.

Команда Unpack — UNPK (Распаковать)

Мнемоника	Код операции	Формат операндов
UNPK	F3	$D_1(L_1,B_1),D_2(L_2,B_2)$

Команда UNPK преобразует данные, указанные вторым операндом (десятичные данные в упакованном формате), из упакованного формата в зонный формат и помещает их по адресу, указанному в первом операнде. Данные из поля упакованного формата перемещаются в поле зонного формата справа налево. Знак поля упакованного формата (младшие четыре разряда) помещается в левую половину младшего байта поля зонного формата. Самая правая десятичная цифра, находящаяся в одном байте со знаком поля упакованного формата, помещается в самые правые четыре разряда поля зонного формата, таким образом заполняя младший байт зонного формата десятичной цифрой со знаком. Затем из поля упакованного формата выбирается следующий полубайт, дополняется до полного байта десятичного зонного формата стандартным символом зоны в коде EBCDIC (шестнадцатеричным 'F') и помещается в поле первого операнда, которое таким образом загружается байт за байтом справа налево. Если длина области, указанной в первом операнде, больше, чем требуется для размещения данных второго операнда, то незаполненные старшие байты области первого операнда заполняются десятичными нулями в зонном формате. Если длина второго операнда больше длины области первого операнда, то неиспользуемые старшие разряды области второго операнда теряются. В случае когда в младших четырех разрядах поля упакованного формата содержится знак, следующая формула служит для вычисления длины

поля в байтах, предназначенного для хранения распакованных данных: длина первого операнда равна длине, указанной во втором операнде, умноженной на 2 и уменьшенной на 1.

	Пример 1	Пример 2
Длина области второго операнда	3	5
Умноженная на 2	×2	×2
Результат	6	10
Уменьшение на 1 из-за присутствия знака в упакованном формате	−1	−1
Требуемая длина для области первого операнда	5	9

Если поле второго операнда не содержит кода знака в младших четырех разрядах, то длина определяется следующим образом. Длина первого операнда равна длине поля упакованного формата, умноженной на 2.

Максимальная неявная или явная длина любого из операндов равна 16 байтам.

При выполнении этой команды признак результата не изменяется.

Команда Convert to Binary — CVB
(Преобразование в двоичную) [1])

Мнемоника	Код операции	Формат операндов
CVB	4F	R_1, $D_2(X_2,B_2)$

Команда CVB преобразует упакованное десятичное число второго операнда в двоичное целое число со знаком, помещая его в регистр, указанный в первом операнде. Второй операнд должен быть расположен на границе двойного слова и представляет собой поле из 8 байтов, содержащее десятичные данные в упакованном формате. Знак упакованного десятичного числа (младший полубайт) определяет знаковый разряд двоичного целого числа. Максимальное положительное десятичное число в упакованном формате, которое может быть преобразовано в двоичный формат, равно 2147483647. Любое большее число при выполнении преобразования явится причиной прерывания при делении с фиксированной точкой, что приведет к прекращению выполнения программы.

---

[1]) В документах по Системе/360 команды CVB и CVD рассматриваются при описании команд арифметики с фиксированной точкой. — *Прим. ред.*

Признак результата не изменяется.

Команда Convert to Decimal — CVD
(Преобразование в десятичную)

Мнемоника	Код операции	Формат операндов
CVD	4E	$R_1,D_2(X_2,B_2)$

Команда CVD преобразует 32-разрядное двоичное число со знаком, находящееся в регистре первого операнда, в десятичное число упакованного формата, располагая это число в области, указанной вторым операндом. Второй операнд представляет собой поле длиной 8 байтов, начинающееся с границы двойного слова. В соответствии с форматом упакованных десятичных данных в младшие четыре разряда (самый правый полубайт) поля помещается код знака преобразуемого числа.

Признак результата не изменяется.

Команда Zero and Add Packed Decimals — ZAP
(Сложение с очисткой)

Мнемоника	Код операций	Формат операндов
ZAP	F8	$D_1(L_1,B_1),D_2(L_2,B_2)$

Эта команда помещает десятичные данные упакованного формата из поля второго операнда в область памяти, указанную в первом операнде, устанавливая в нуль оставшиеся незаполненными старшие полубайты поля первого операнда. Эта команда отличается от других команд десятичной арифметики тем, что только второй операнд проверяется на правильность кодов цифр и знака. Если длина поля первого операнда больше длины поля второго операнда, то оставшиеся незаполненными старшие полубайты поля первого операнда заполняются десятичными нулями. Если длина поля первого операнда меньше длины поля второго операнда, то происходит десятичное переполнение. Максимальная неявная или явная длина для любого из операндов равна 16 байтам.

CC	BC	Условие
0	8	Значение содержимого первого операнда равно нулю
1	4	В первом операнде содержится число меньше нуля (отрицательное)
2	2	В первом операнде содержится число больше нуля (положительное)
3	1	При выполнении команды произошло переполнение

Команда Add Packed Decimals — AP (Сложение десятичное)

Мнемоника	Код операции	Формат операндов
AP	FA	$D_1(L_1,B_1),D_2(L_2,B_2)$

Эта команда осуществляет алгебраическое сложение упакованного десятичного содержимого второго операнда с упакованным десятичным содержимым первого операнда. Сумма помещается в поле первого операнда, причем определяемый по правилам алгебры знак помещается в младший полубайт. Оба операнда должны иметь правильные коды знака — шестнадцатеричные цифры от A до F (от 10 до 15) [1]). При выполнении этой команды могут возникнуть переполнения двух видов:

1. Если поля обоих операндов одинаковой длины, то в зависимости от чисел, содержащихся в операндах, может возникнуть перенос цифры из старшего разряда.

2. Если длина второго операнда больше длины первого операнда, то значащие цифры старших разрядов, позиции которых выходят за пределы длины первого операнда, теряются.

Если значение числа в поле упакованного формата меньше половины максимально возможного значения числа в этом поле, то число может быть сложено с самим собой путем указания одного и того же поля в обоих операндах.

Максимальная неявная или явная длина для любого из операндов равна 16 байтам.

CC	BC	Условие
0	8	В поле первого операнда содержится нуль
1	4	В поле первого операнда содержится число меньше нуля (отрицательное)
2	2	В поле первого операнда содержится число больше нуля (положительное)
3	0	Переполнение

Команда Subtract Packed Decimals — SP (Вычитание десятичное)

Мнемоника	Код операции	Формат операндов
SP	FB	$D_1(L_1,B_1),D_2(L_2,B_2)$

Команда SP осуществляет алгебраическое вычитание упакованного десятичного содержимого второго операнда из упакованного десятичного содержимого первого операнда. Полученная разность двух операндов со знаком, определяемым по пра-

[1]) При выполнении команды десятичной арифметики проверяется также правильность кодов цифр. — *Прим. ред.*

вилам алгебры, замещает содержимое первого операнда. Оба операнда должны содержать в младшем полубайте (самых правых четырех двоичных разрядах) правильный код знака [1]). Максимальная неявная или явная длина любого из операндов равна 16 байтам.

CC	BC	Условие
0	8	В поле первого операнда содержится нуль
1	4	В поле первого операнда содержится число меньше нуля (отрицательное)
2	2	В поле первого операнда содержится число больше нуля (положительное)
3	1	При выполнении этой команды произошло переполнение

Команда Multiply Packed Decimals — MP
(Умножение десятичное)

Мнемоника	Код операции	Формат операндов
MP	FC	$D_1(L_1,B_1),D_2(L_2,B_2)$

Содержимое второго операнда в упакованном формате (множитель) умножается на содержимое первого операнда в упакованном формате (множимое). В младших полубайтах полей упакованных данных обоих операндов должны находиться правильные коды знаков.

Результат умножения занимает все поле первого операнда. Знак результата определяется алгебраически с учетом знаков множимого и множителя. Длина множителя не должна превышать 8 байтов, а также не должна быть равной или превышать длину, указанную первым операндом. Формула для определения длины первого операнда: длина области результата равна длине множителя плюс длина множимого и плюс 1 байт.

	Пример 1	Пример 2
Длина множимого	9	6
Длина множителя	4	5
Добавочный байт	1	1
Длина поля результата	14	12

При использовании этой формулы возникновение переполнения исключается: При формировании команды MP следует помнить, что множитель и множимое должны находиться в основной памяти и обычной должна быть такая последовательность действий:

[1]) См. примечание к описанию команды AP. — *Прим. ред.*

1. Определить длину, требуемую для результата, и отвести для него память.
2. Загрузить множимое в область результата командой ZAP.
3. Записать команду MP.

Максимально возможная длина области результата равна 16 байтам.

Признак результата не изменяется.

Команда Divide Packed Decimals — DP
(Деление десятичное)

Мнемоника	Код операции	Формат операндов
DP	FD	$D_1(L_1,B_1),D_2(L_2,B_2)$

Содержимое первого операнда в упакованном формате, представляющее собой делимое, делится на содержимое второго операнда в упакованном формате (делитель). В младшем полубайте каждого из операндов должен находиться правильный код знака упакованного числа. Частное от деления и остаток замещают делимое в поле первого операнда. Частное помещается в старших байтах поля первого операнда; его длина определяется вычитанием длины второго операнда из длины поля первого операнда. Остаток от деления помещается в младшие байты поля первого операнда и занимает то же число байтов, что и поле второго операнда (делитель). Знак частного определяется алгебраически с учетом знаков делителя и делимого; знак остатка совпадает со знаком делимого. Длина делителя не должна превышать 8 байтов, а также не должна быть равной или большей длины, указанной в первом операнде.

Признак результата не изменяется.

Команда Move with Offset — MVO (Пересылка со сдвигом)

Мнемоника	Код операции	Формат операндов
MVO	F1	$D_1(L_1,B_1),D_2(L_2,B_2)$

Команда MVO перемещает данные, выбирая каждый байт из области памяти, указанной вторым операндом, и располагая его в области памяти, указанной первым операндом, но при этом смещая его влево на полбайта по отношению к соответствующей позиции. Данные для пересылки выбираются в полях последовательно справа налево. Как в первом, так и во втором операндах этой команды присутствует указатель длины, однако смещение происходит только в поле первого операнда. Если длина второго операнда меньше длины первого операнда, то старшие полубайты первого операнда заполняются шестнадцатеричными нулями. Если длина поля первого операнда меньше

длины второго операнда, то после того, как поле первого операнда будет заполнено, оставшиеся данные, подлежащие пересылке, не пересылаются.

Максимальная длина пересылаемых этой командой данных равна 16 байтам.

Признак результата не изменяется.

Б. ОПЕРАЦИИ ДЕСЯТИЧНОЙ АРИФМЕТИКИ

Язык Ассемблера Системы/360 предоставляет программисту три основных типа арифметических команд: с фиксированной точкой, десятичные и с плавающей точкой. По ранее упоминавшимся причинам команды арифметики с плавающей точкой в этой книге не рассматриваются. Поэтому рассмотрение арифметических операций ограничено командами с фиксированной точкой и командами десятичной арифметики.

Когда в этом тексте говорится о числе с фиксированной точкой (или о двоичном представлении десятичного числа), то имеется в виду целое число, занимающее полуслово, слово или общий регистр, в которых самый левый (старший) разряд указывает знак числа, а все остальные разряды задают само число. В каждой арифметической команде с фиксированной точкой по крайней мере один из операндов находится в регистре.

Числа в упакованном десятичном формате представляются полем из 1 или более байтов, в котором каждый отдельный полубайт содержит десятичную величину, выраженную в форме шестнадцатеричной цифры. Самый правый полубайт упакованного поля, используемого в арифметических операциях, должен содержать правильный код знака упакованного десятичного числа, а именно шестнадцатеричные значения от A до F.

1. Процедуры преобразования форматов упакованных десятичных чисел и чисел с фиксированной точкой

Данные, используемые в арифметических операциях какого-либо типа, часто должны быть преобразованы в формат, соответствующий типу используемых арифметических команд. В связи с этим, перед тем как перейти к изучению использования арифметических команд, целесообразно детально познакомиться со средствами преобразования данных.

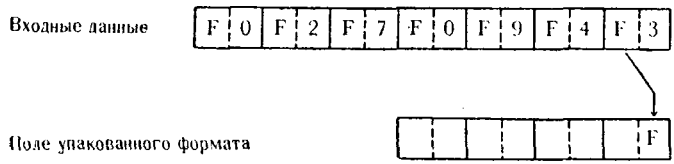
Данные, получаемые проблемной программой в коде EBCDIC, могут быть преобразованы сначала в упакованную десятичную форму, а затем в формат с фиксированной точкой. При выполнении перечисленных действий в обратном порядке данные в формате с фиксированной точкой могут быть преобразованы в форму, соответствующую коду EBCDIC.

Преобразования выполняются в следующей последовательности:

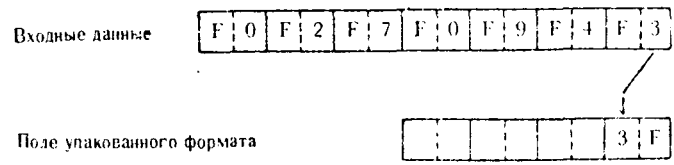
1. Входные данные поступают в коде EBCDIC.
2. Они преобразуются командой PACK в упакованный десятичный формат.
3. Десятичное упакованное число с помощью команды CVB преобразуется в формат с фиксированной точкой.

1. Данные представлены в двоичном формате с фиксированной точкой.
2. Командой CVD они преобразуются в упакованные десятичные данные.
3. Командой UNPK упакованные десятичные данные преобразуются в цифры в коде EBCDIC.

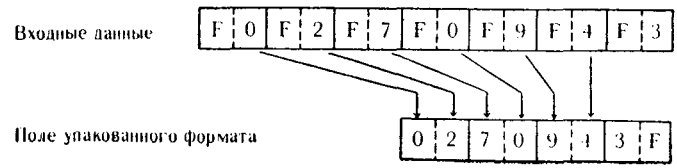
Упаковать — PACK. Команда PACK используется для преобразования числовых символов кода EBCDIC в упакованную десятичную форму. При преобразовании зонная часть младшего байта входных данных помещается в младший полубайт поля упакованного формата.



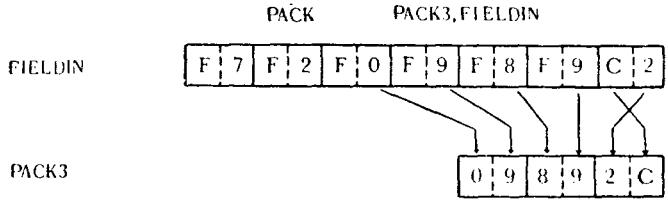
Цифровая часть младшего байта поля входных данных помещается затем в другую половину младшего байта поля упакованного формата.



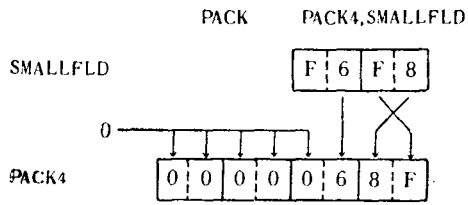
Далее цифровые части последующих старших байтов помещаются в последующие старшие полубайты поля упакованного формата.



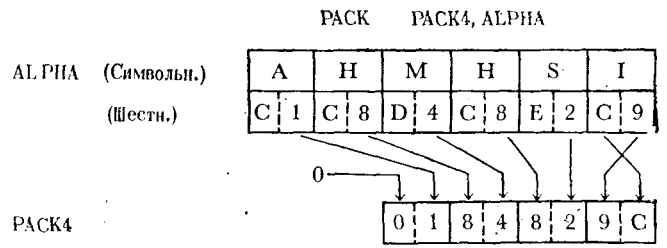
Если в поле зонного формата содержится больше цифровых символов кода EBCDIC, чем может разместиться в поле упакованного формата, то избыточные старшие цифровые разряды теряются. Ниже следует иллюстрация сказанного:



Если же в поле зонного формата не содержится достаточного количества цифровых символов кода EBCDIC для заполнения старших позиций поля упакованного формата, то эти позиции заполняются нулями.



Буквенные символы кода EBCDIC могут быть преобразованы в упакованный десятичный формат, но после этого восстановить их снова в формат кода EBCDIC невозможно. К этому вопросу мы вернемся при обсуждении команды UNPK. Тем не менее и одностороннее преобразование буквенных символов кода EBCDIC в упакованное десятичное число может иметь смысл. Преобразование этого типа показано ниже.

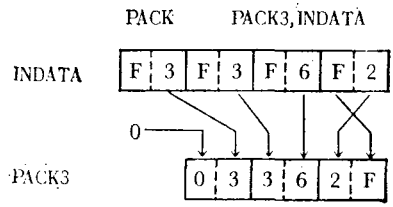


Здесь буквенные символы кода EBCDIC преобразовались в упакованное десятичное число +184829.

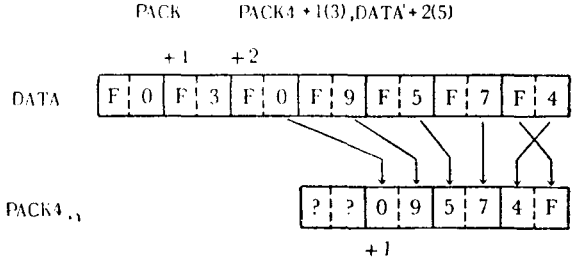
Рассмотрим еще два примера упаковки десятичных данных.

Пример 1. Поле десятичных данных в зонном формате, содержащее эквивалент четырех цифровых символов кода EBCDIC, упаковано в трехбайтное поле. В связи с тем что количества

цифр было недостаточно для заполнения всех полубайтов поля упакованного формата, в старший разряд был помещен нуль

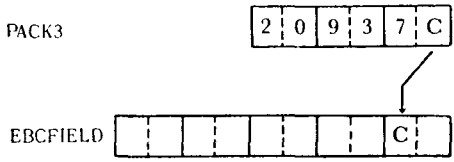


Пример 2.

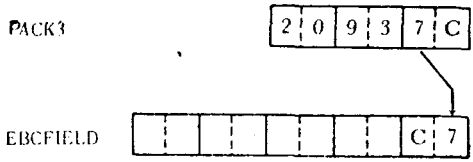


В этом примере младшие 5 байтов семибайтового поля, содержащего цифровые символы в коде EBCDIC, упаковываются в младшие 3 байта четырехбайтового поля. Содержимое старшего байта поля упакованного формата при выполнении команды PACK не изменяется.

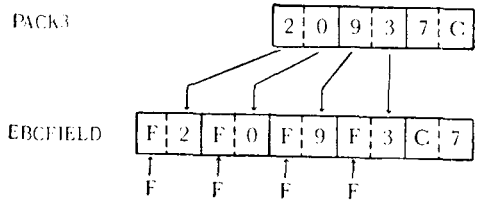
Распаковать — UNPK. Команда UNPK осуществляет процедуру, обратную выполняемой командой PACK, преобразуя упакованное десятичное число в поле цифровых символов кода EBCDIC. Сначала младший полубайт упакованного десятичного поля помещается в зонную часть младшего байта принимающего поля.



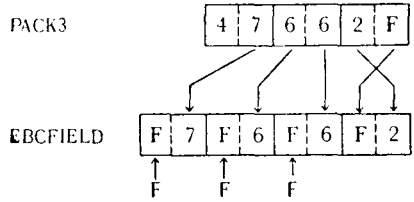
Затем младшая десятичная цифра упакованного поля помещается в цифровую часть младшего байта принимающего поля зонного формата.



Далее каждая следующая старшая упакованная десятичная цифра помещается в цифровую часть каждого следующего старшего байта поля зонного формата, в зонную часть байтов помещается стандартный символ зоны в коде EBCDIC.

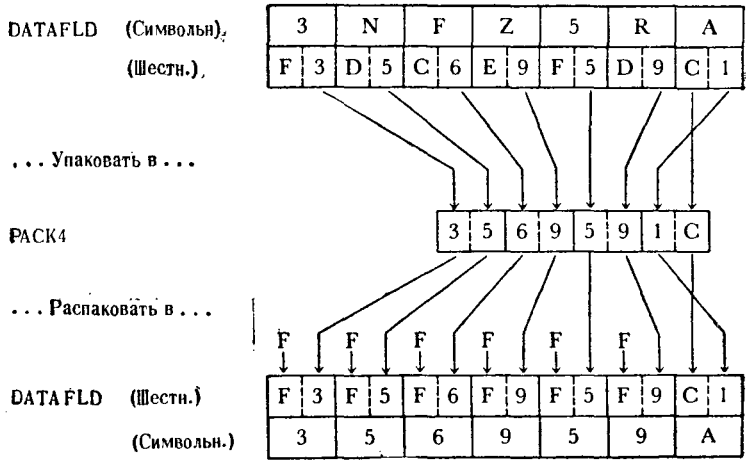


Если в поле упакованного десятичного формата содержится больше цифр, чем может поместиться в поле зонного формата, то избыточные старшие цифры теряются.



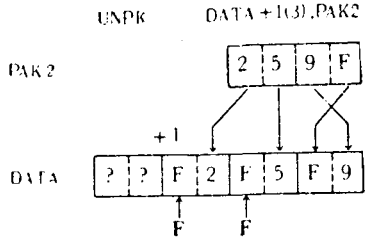
Независимо от того, буквенные или цифровые данные преобразовывались в упакованный десятичный формат, в результате последующей распаковки формируется поле цифровых символов кода EBCDIC. Это иллюстрируется на примере последовательного выполнения команд PACK и UNPK.

Причина, по которой младший байт не изменяется, заключается в том, что он представляет собой десятичную цифру со знаком в зонном формате.



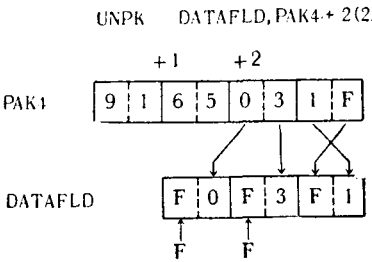
Как и во многих других командах, для модификации отдельных частей полей, используемых командой UNPK, может применяться указатель длины и настройка адреса.

Пример 1.



В этом примере три цифровых разряда двухбайтного упакованного поля были распакованы в 3 младших байта поля DATA. Старший байт поля DATA остался неизменным, т. е. таким, каким он был перед выполнением команды UNPK.

Пример 2.



Как здесь показано, 2 младших байта четырехбайтового упакованного десятичного поля, содержащие три цифровых разряда, распаковываются в трехбайтное поле цифровых символов.

Преобразовать в двоичную — CVB. Команда CVB преобразует упакованное десятичное число, содержащееся в восьмибайтовом поле, выравненном по границе двойного слова, в двоичное число с фиксированной точкой и помещает его в общий регистр. Первый операнд команды указывает регистр, в который должно быть помещено преобразованное число, а второй операнд — адрес восьмибайтового поля, содержащего упакованное десятичное число. Это вторая команда, требуемая для преобразования числа из символьной формы кода EBCDIC в число с фиксированной точкой; первым этапом была упаковка поля символов EBCDIC. Для того чтобы задать восьмибайтовое поле,

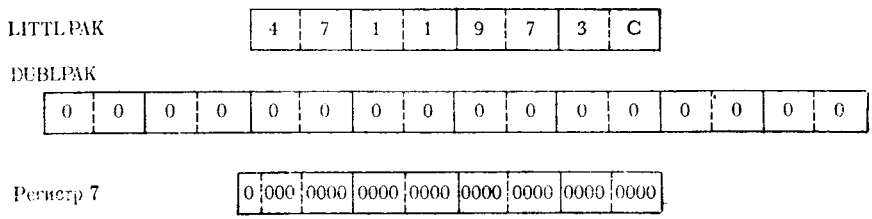
выравненное по границе двойного слова, в которое может быть помещено упакованное число, программист может воспользоваться любым из следующих типов предложений:

```
BIGDBWDA    DS    D       001
BIGDBWDA    DC    D'0'    002
```

Первое предложение резервирует в области памяти, отведенной программе, поле длиной 8 байтов, выравненное по границе двойного слова, но не изменяет содержимого этой области. Второе предложение формирует восьмибайтовое двойное слово, должным образом выравненное, содержащее число +0, или, иными словами, шестнадцатеричные нули во всех разрядах.

В любом случае преобразуемое в двоичную форму упакованное десятичное число следует пересылать в двойное слово командой ZAP. Это гарантирует то, что любое предыдущее содержимое данной области будет стерто и засылаемое число сохранит правильную конфигурацию упакованного десятичного формата.

Пусть необходимо преобразовать упакованное десятичное поле из 4 байтов LITTLPAK в двоичный формат. Сначала оно будет переслано в поле DUBLPAK, а затем преобразовано в двоичный формат с фиксированной точкой и помещено в общий регистр 7. Ниже приводится содержимое этих областей перед выполнением команд.



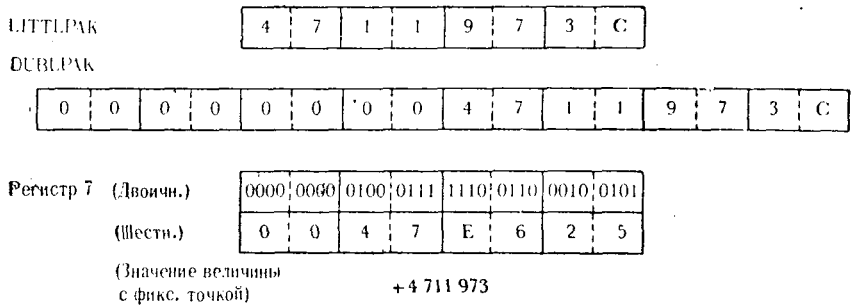
Должны быть выполнены следующие команды:

```
ZAP     DUBLPAK,LITTLPAK    001
CVB     7,DUBLPAK           002
```

Предложение 001 заносит содержимое LITTLPAK в 4 младших байта DUBLPAK, устанавливая шестнадцатеричные разряды 4 старших байтов DUBLPAK в нуль.

Предложение 002 преобразует упакованное десятичное число + 4711973, которое теперь содержится в DUBLPAK, в его эквивалент с фиксированной точкой и помещает его в общий регистр 7. После выполнения команд содержимое этих трех

областей выглядит так:

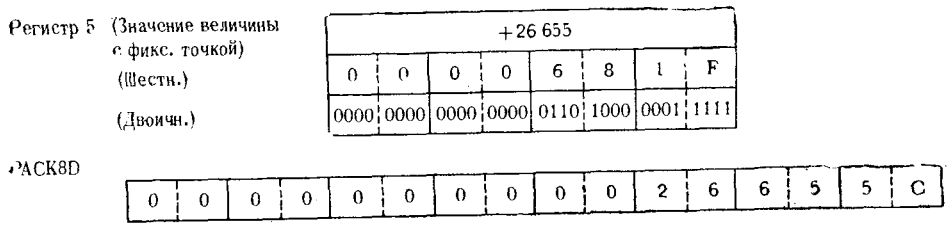


Преобразовать в десятичную — CVD. Целью команды CVD является преобразование двоичного числа с фиксированной точкой в равное ему десятичное число в упакованном формате. Число с фиксированной точкой должно находиться в общем регистре, а поле упакованного формата должно состоять из 8 байтов и размещаться, начиная с границы двойного слова. Число с фиксированной точкой, содержащееся в общем регистре первого операнда, преобразуется в упакованную десятичную форму и располагается в поле, адрес которого указывается вторым операндом.

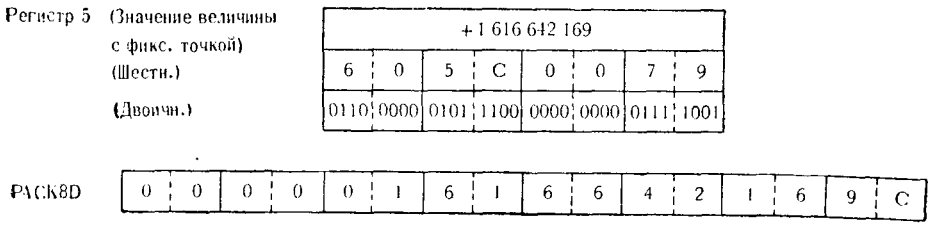
Предположим, что число с фиксированной точкой находится в общем регистре 5, поле второго операнда имеет метку PACK8D и рассмотрим примеры выполнения команды

CVD 5,PACK8D

Пример 1.



Пример 2.



Пример 3.

Регистр 5 (Значение величины с фикс. точкой)

	+3							
(Шестн.)	0	0	0	0	0	0	0	3
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0011

PACK8D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	C

В. ВЫПОЛНЕНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ НАД УПАКОВАННЫМИ ДЕСЯТИЧНЫМИ ЧИСЛАМИ

Программисту очень часто приходится решать, использовать ли операции арифметики с фиксированной точкой или операции десятичной арифметики. Естественно, что у каждого из этих двух типов арифметических операций есть свои преимущества. Один из ключевых факторов, влияющих на решение этого вопроса, — способ использования результатов вычислений. Сравнение выполнения отдельных операций арифметики с фиксированной точкой и десятичной арифметики (например, сравнение команд A и AP) показывает, что время выполнения команд с фиксированной точкой значительно меньше. Однако, если данные, которые должны быть обработаны, вводятся в программу в символьном формате или в упакованном десятичном формате и в конце концов будут выведены на печать или записаны в память в символьном или в упакованном десятичном формате, то лучшим решением может оказаться выполнение всех арифметических операций с помощью средств десятичной арифметики. Выполнение команд преобразования упакованного десятичного числа в двоичное число с фиксированной точкой (CVB) и обратного преобразования в упакованное десятичное число (CVD) занимает значительное время. Некоторое время, сэкономленное путем использования арифметики с фиксированной точкой, может оказаться меньше времени, затраченного на преобразование в двоичную форму и последующее преобразование в десятичную форму командами CVB и CVD. Однако если данные вводятся в программу в двоичном формате с фиксированной точкой, будут выведены или сохранены в двоичном формате с фиксированной точкой или будут повторно использованы в формате с фиксированой точкой, то программисту лучше использовать арифметические операции с фиксированной точкой.

Помимо этого, необходимо оценить рабочую область, требуемую каждым из этих типов команд. Для выполнения арифметических операций с фиксированной точкой требуется по крайней мере один общий регистр; операции десятичной

арифметики используют области памяти, которые предварительно должны быть выделены программистом. При работе с командами с фиксированной точкой может оказаться необходимым повторное использование нескольких регистров для различных целей. Если программист располагает достаточным количеством свободной памяти, он может воспользоваться средствами десятичной арифметики, выделив для каждой группы команд отдельную рабочую область.

Опыт показывает, что если результаты арифметических операций должны быть выведены на печать, дисплей или записаны в память в символьном или упакованном десятичном формате, то обычно предпочтительнее применение команд десятичной арифметики. Однако имеется достаточно аргументов в пользу применения каждого типа арифметических операций.

Прежде чем перейти к применениям операций десятичной арифметики, целесообразно рассмотреть структуру и правила организации арифметических полей в упакованном десятичном формате.

Структура упакованного десятичного числа непосредственно связана с понятием шестнадцатеричного символа. Каждый разряд упакованного десятичного числа представляется шестнадцатеричным символом — одна цифра соответствует четырем двоичным разрядам. Шестнадцатеричные символы или цифры от 0 до 9 представляют правильные цифровые разряды упакованных десятичных чисел, а шестнадцатеричные символы от A до F представляют знаки или зонные части правильных упакованных десятичных чисел. Для того чтобы упакованное десятичное число могло быть использовано в арифметической операции, в самом правом полубайте поля числа должен находиться правильный код знака (шестнадцатеричное число от A до F). Все остальные шестнадцатеричные цифры этого поля должны принимать шестнадцатеричные значения от 0 до 9. Если делается попытка обработать неверно упакованное десятичное поле, то произойдет программное прерывание и выполнение задачи закончится аварийно. Нужно отметить, что в отличие от арифметических команд с фиксированной точкой команды десятичной арифметики не требуют выравнивания в памяти и не используют общих регистров. Поэтому программист в операциях десятичной арифметики может использовать рабочие области длиной от 1 до 16 байтов.

Другой отличительной чертой команд десятичной арифметики является наличие указателей длины в обоих операндах. Это необходимо подчеркнуть особо, поскольку упомянутые указатели длины действительно существуют в обоих операндах либо в неявной, либо в явной форме. При использовании команд десятичной арифметики нужно учитывать влияние

различия указателей длины между операндами на выполнение команды.

В главе, посвященной операциям с фиксированной точкой, иллюстрации всех регистров и полей данных приведены в двух форматах: двоичном и шестнадцатеричном. Это сделано для того, чтобы можно было определить веса отдельных двоичных позиций. В отличие от этого в данной главе упакованные десятичные поля и остальной графический материал будут представлены только в шестнадцатеричном формате, так как это единственный формат, соответствующий упакованным десятичным данным.

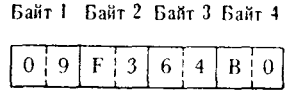
Сложение с очисткой — ZAP. Эта команда может эффективно выполнить подготовку произвольной области к использованию в качестве рабочей области упакованного десятичного формата, очищая эту область от ранее хранившихся в ней данных и устанавливая ее в нуль, или переслать упакованные десятичные данные из некоторого поля упакованного формата в поле большей длины без предварительной его очистки. Максимальная длина каждого из операндов — 16 байтов.

По окончании работы команды ZAP содержимое всех полубайтов области памяти первого операнда устанавливается в шестнадцатеричный нуль, если эти полубайты не были заполнены данными, пересылаемыми из области, указанной вторым операндом. Если длина первого операнда больше длины второго операнда, то незаполненные старшие полубайты первого операнда будут заполнены шестнадцатеричными нулями. Если длина первого операнда недостаточна для размещения всех значащих (не нулевых) шестнадцатеричных разрядов второго операнда, происходит программное прерывание, сопровождаемое установкой признака результата, указывающего на десятичное переполнение. Ниже приведены примеры использования этой команды.

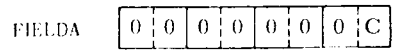
Пример 1.
Выполняемая команда:

```
ZAP     FIELDA,=PL1'0'
```

Перед выполнением команды содержимое FIELDA выглядело следующим образом:



После выполнения команды ZAP поле примет вид

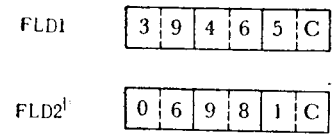


Надо заметить, что, хотя в качестве второго операнда задан однобайтовый литерал, все полубайты первого операнда были заполнены шестнадцатеричными нулями.

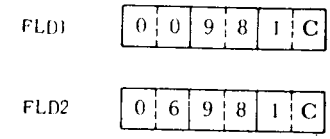
Пример 2.
Выполняемая команда:

```
ZAP     FLD1,FLD2+1(2)
```

Перед выполнением команды ZAP содержимое полей выглядело так:



После выполнения команды эти поля имеют вид

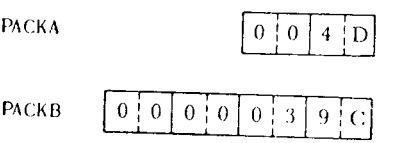


В этом примере команда указывает, что пересылаемые из FLD2 данные должны выбираться, начиная со второго байта. Поэтому только 2 байта данных были пересланы из FLD2 в FLD1, а в незаполненные полубайты FLD1 были помещены шестнадцатеричные нули.

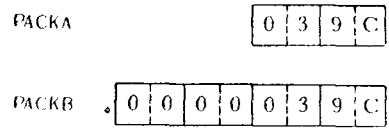
Пример 3.
Выполняемая команда:

```
ZAP     PACKA,PACKB
```

Этой командой должны быть обработаны упакованные десятичные поля:



После выполнения команды содержимым этих полей будет



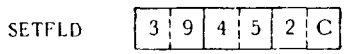
Хотя длина поля первого операнда была меньше длины второго операнда, программного прерывания (десятичного переполнения) не произошло, так как при пересылке не было потеряно ни одного значащего разряда.

Пример 4.

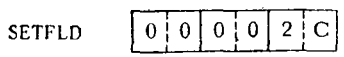
Выполняемая команда:

```
ZAP     SETFLD,SETFLD+2(1)
```

Перед выполнением команды ZAP содержимым поля SETFLD было



После выполнения команды ZAP поле SETFLD имеет вид



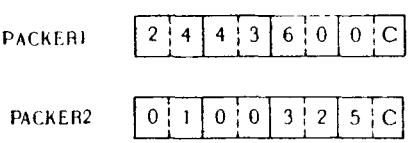
В этом примере команда ZAP очищает старшие байты упакованного десятичного поля, что достигается смещением адреса начала поля во втором операнде и пересылкой в то же самое поле. Нужно заметить, что явный указатель длины в 1 байт был использован для того, чтобы ограничить неявную длину второго операнда.

Пример 5.

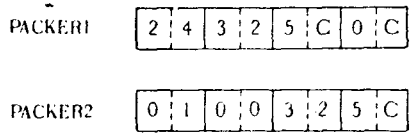
В этом примере предполагается показать использование явных указателей длины, примененных в обоих операндах команд десятичной арифметики. Используемая в этом примере команда:

```
ZAP     PACKER1+1(2),PACKER2+2(2)
```

Содержимое полей перед выполнением команды имеет вид



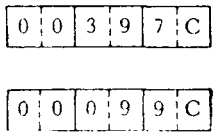
Содержимое полей после выполнения команды примет вид



Заметьте, что изменились только второй и третий байты поля PACKER1. Как указано в первом операнде, обработка поля PACKER1 начинается 1 байтом правее старшего разряда этого поля и затрагивает только 2 байта памяти. После выполнения этой команды попытка использовать четырехбайтовое содержимое PACKER1 в операциях десятичной арифметики приведет к программному прерыванию. Это объясняется тем, что в шестом полубайте слева, не являющимся младшим полубайтом поля, находится шестнадцатеричное число С, недопустимое в цифровой позиции упакованного десятичного числа. Это поле могло бы быть использовано в арифметических операциях, если бы указатель длины был равен 3, т. е. PACKER1(3), и поэтому обрабатывались бы только 3 левых байта поля. Мог бы быть использован и самый правый байт этого поля, если в команде определить его как PACKER1+3(1).

Сложение десятичное — AP. Команда, суммирующая два поля упакованных десятичных данных, имеет вид, который довольно легко интерпретируется программистом. Другими словами, соотношение 4C + 6C = 10C легче для понимания по сравнению с арифметикой с фиксированной точкой, где F+F = = 1E. При помощи этой команды упакованное десятичное число второго операнда складывается с упакованным десятичным числом первого операнда, а их сумма помещается в область первого операнда. Если при сложении сумма превышает длину области первого операнда, то происходит десятичное переполнение, вызывая программное прерывание. Это может произойти, когда поля обоих операндов имеют равную длину или когда длина поля второго операнда больше длины поля первого операнда и при этом в поле второго операнда есть значащие цифры, позиции которых выходят за пределы длины первого операнда. Нулевая сумма, являющаяся результатом правильно завершенного сложения, всегда имеет в младшем полубайте положительный знак. Правильными кодами положительного знака могут быть шестнадцатеричные цифры A, C, E или F. Правильными кодами отрицательного знака может быть одна из шестнадцатеричных цифр B или D.

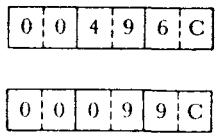
Пример 1. Перед выполнением команды AP поля операндов имеют вид



Выполняемая команда:

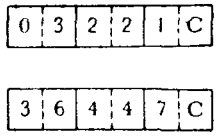
```
AP      PKFLD1,PKFLD2
```

После выполнения команды AP содержимое этих полей примет вид



Заметьте, что выполнение этой команды не изменяет содержимое поля второго операнда. Поскольку это справедливо во всех случаях применения операций десятичной арифметики, в последующих примерах этой главы приведен результат изменения только первого операнда.

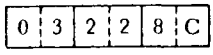
Пример 2. В этом примере используются следующие поля:



Выполняемая команда:

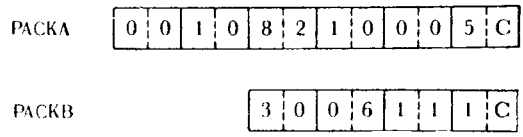
```
AP      PKA,PKB+2(1)
```

После выполнения команды содержимое PKA примет вид



В этом примере второй операнд содержит адрес, который на 2 больше начального адреса поля PKB, и указывает, что только 1 байт должен быть сложен с первым операндом. Поэтому эта операция сложения эквивалентна соотношению 03221C + 7C = = 03228C. Символ C в каждом из этих чисел представляет знак этих чисел.

Пример 3. Содержимое полей, которые будут использованы в этом примере, имеет вид

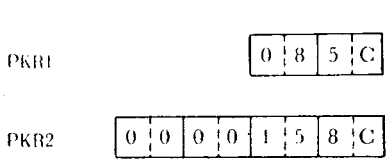


Выполняемая команда:

```
AP      PACKA,PACKB
```

После выполнения команды содержимым PACKA стало

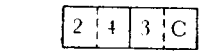
Пример 4. Поля, которые будут использованы в этом примере, перед выполнением команды AP имеют вид



Выполняемая команда:

```
AP      PKR1,PKR2
```

В результате выполнения этой команды в PKR1 содержится теперь упакованное десятичное число



В этом примере не происходит десятичного переполнения и последующего программного прерывания. Хотя длина второго операнда больше длины первого операнда, но избыточные байты поля PKR2 не содержат значащих шестнадцатеричных цифр, а сумма двух операндов не превышает по длине поля первого операнда.

Пример 5. В этом примере выполняется команда

```
AP      FIELDR,FIELDS
```

Шестнадцатеричное содержимое полей, обрабатываемых этой командой, следующее:

FIELDR	0	0	3	7	5	7	2	C
FIELDS	0	0	0	4	9	1	6	D

После выполнения команды поле FIELDR будет содержать

FIELDR	0	0	3	2	6	5	6	C

В данном случае в FIELDS находится отрицательное число —4916, что указывается шестнадцатеричной цифрой D в младшем полубайте этого поля. Сумма, полученная при сложении этого числа и положительного числа +37572, находившегося в FIELDR, отражает влияние кодов знака. Если бы отрицательное число было по абсолютной величине больше положительного, с которым оно складывалось, то полученной сумме был бы присвоен знак отрицательного числа.

Вычитание десятичное — SP. Команда SP вычитает из упакованного десятичного содержимого поля первого операнда упакованое десятичное содержимое поля второго операнда. Разность помещается в поле первого операнда. В обоих полях должны находиться упакованные десятичные числа с правильными кодами знаков. Прерывание при десятичном переполнении происходит по тем же причинам, которые были описаны для команды AP.

Пример 1. Перед началом выполнения команды SP содержимым полей, используемых командой, было

PACKERA	0	2	2	5	8	C
PACKERB	0	0	9	9	4	C

Выполняемая команда:

```
SP      PACKERA,PACKERB
```

В поле первого операнда после выполнения команды находятся следующие данные:

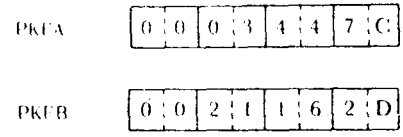
PACKERA	0	1	2	6	4	C

В этом примере показан простейший случай вычитания. И действительно, он подтверждает тот факт, что 2258С минус 994С равно 1264С.

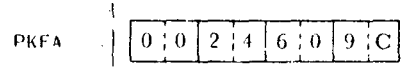
Пример 2. Должна быть выполнена команда

```
SP      PKFA,PKFB
```

Содержимое полей, используемых при выполнении команды, имеет следующий вид:

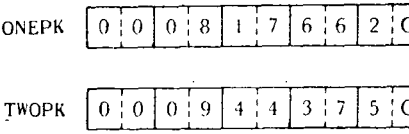


В результате выполнения команды содержимое PKFA изменится:



В этом примере в результате вычитания отрицательного числа из положительного получается положительное число, равное сумме этих двух чисел, рассматриваемых как числа без знаков.

Пример 3. Содержимое полей, которые используются в этом примере, следующее:



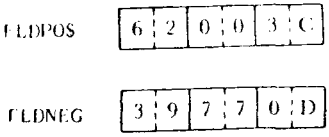
Команда, обрабатывающая эти поля, записывается так:

```
SP      ONEPK,TWOPK
```

Выполнение команды отразится на содержимом поля ONEPK следующим образом:

В этом примере в результате вычитания упакованного десятичного числа второго операнда из меньшего числа первого операнда получена отрицательная разность.

Пример 4. Этот пример приводится для иллюстрации одного из условий, являющихся причиной программного прерывания вследствие десятичного переполнения. Ниже приводится содержимое полей, обрабатываемых командой SP, перед ее выполнением:



Выполняется команда

```
SP      FLDPOS,FLDNEG
```

После выполнения команды SP область, занимаемая FLDPOS, будет иметь вид

Результатом выполнения этой команды является число, превышающее длину поля первого операнда. Это условие, называемое переносом старшего разряда, вызывает программное прерывание.

Умножение десятичное — MP. Упакованное десятичное содержимое первого операнда (множимое) умножается на упакованное десятичное содержимое второго операнда (множитель) и результат умножения помещается в область первого операнда. Есть несколько правил, которым необходимо следовать при организации рабочих областей, используемых этой командой:

1. Длина множителя не должна превышать 8 байтов.
2. Должна быть определена область для результата умножения, в которую перед выполнением команды MP должно засылаться значение множимого.
3. Длина области результата умножения должна быть равна суммарной длине множителя и множимого.
4. Длина области результата не должна превышать 16 байтов.
5. Длина множителя должна быть по крайней мере на 1 байт меньше длины области результата.
6. В области результата после записи в нее множимого должно находиться по крайней мере столько же старших нулевых разрядов, сколько могло бы уместиться в поле множителя.
7. Оба операнда должны быть правильными упакованными десятичными числами со знаками.

Если полностью следовать этим правилам, то программное прерывание не может являться следствием выполнения этой команды. Если не следовать этим правилам, то программное прерывание, вероятнее всего, произойдет из-за неправильных данных или неправильной спецификации.

Ввиду того что область результата, возможно, будет использоваться довольно часто по крайней мере некоторой группой команд, лучшим средством загрузки множимого в область результата является команда ZAP. Она не только нужным образом располагает множимое в области результата (выравнивая по правой границе поля), но также очищает старшие разряды, в которых могли находиться значащие цифры, и помещает в каждый из них шестнадцатеричный нуль. После завершения команды MP в самом правом младшем полубайте поля результата будет находиться знак результата.

Возможности использования подразумеваемой десятичной точки, а также изменения ее положения в операциях десятичной арифметики обсуждаются в следующем разделе этой главы.

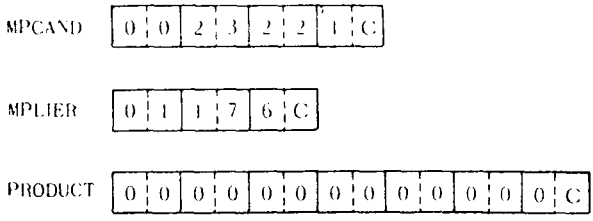
Поле результата может быть задано меньшей длины, чем полная суммарная длина множителя и множимого, но только если множимое содержит достаточное количество старших нулевых разрядов, чтобы оно уместилось в поле результата. Сказанное можно применить, только если программист знает, что по крайней мере несколько старших разрядов в поле множимого равны нулю. Если при засылке множимого командой ZAP в область результата в ней оказывается больше нулевых разрядов, расположенных слева от самого левого значащего разряда множимого, чем могло бы разместиться в поле множителя, длина области результата может быть уменьшена за счет длины лишних нулевых разрядов множимого. В общем случае на практике поступать так не рекомендуется, так как этот метод требует точного знания того, что старшие разряды, содержащиеся в поле множимого, никогда не примут значения, отличного от нулевого.

В следующих примерах показано применение правил использования команды MP.

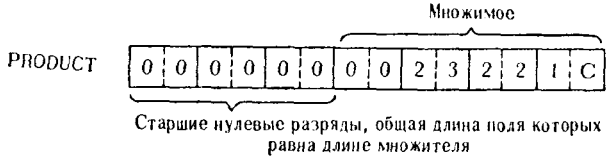
Пример 1. В этом примере заданы следующие константы и команды:

```
MPCAND    DC    PL4'23221'
MPLIER    DC    PL3'1176'
PRODUCT   DC    PL7'0'
*
          ZAP   PRODUCT,MPCAND    001
          MP    PRODUCT,MPLIER    002
```

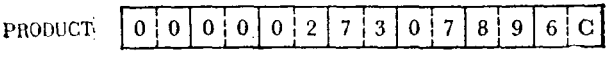
Перед выполнением этой программы указанные упакованные десятичные области содержат:



В предложении 001 для размещения множимого в младших байтах поля PRODUCT используется команда ZAP. После выполнения этой команды содержимым PRODUCT будет



Затем будет выполнено предложение 002, реализующее операцию MP, после чего содержимым поля PRODUCT станет



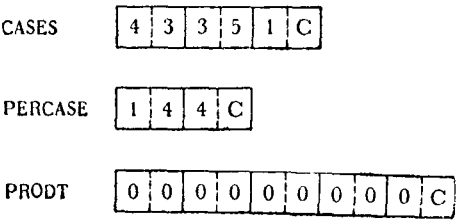
В результате эта программа выполнила вычисление

$$1176 \times 23221 = 27307896.$$

Пример 2. Для этого примера определены следующие области:

```
CASES     DC    PL3'43351'
PERCASE   DC    PL2'144'
PRODT     DC    PL5'0'
```

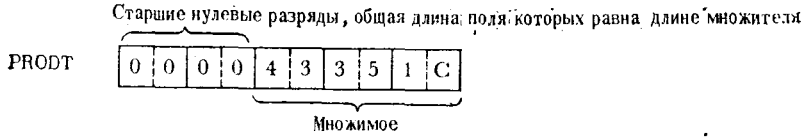
Содержимое этих областей, уже определенных ранее, следующее:



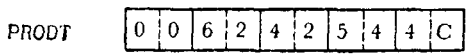
Должны быть выполнены следующие команды:

```
ZAP     PRODT,CASES     001
MP      PRODT,PERCASE   002
```

Предложение 001 занесет множимое в область результата PRODT, предварительно очистив ее, после чего ее содержимое примет вид



Предложение 002 вызовет умножение множителя PERCASE на число, содержащееся в поле PRODT. Теперь в PRODT находится результат умножения

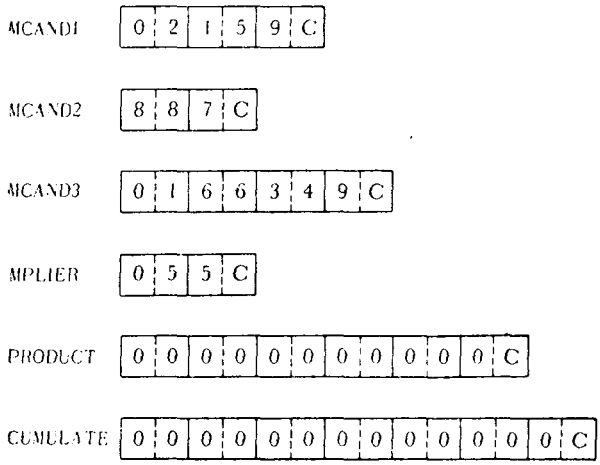


или

$$144 \times 43\,351 = 6\,242\,544$$

Заметьте, что, хотя во всех разрядах как множителя, так и множимого находились значащие цифры, в конечном результате содержатся два нулевых старших разряда. Один из этих нулевых разрядов присутствует потому, что в конечном результате содержится только один знаковый разряд, хотя знаковый разряд присутствует в младших полубайтах каждого из операндов, а длина поля результата умножения была выбрана равной суммарной длине множителя и множимого. Присутствие другого нулевого разряда объясняется тем, что при умножении не возникло переноса в старший разряд. Это может быть пояснено с помощью следующего очевидного утверждения: произведение двух одноразрядных чисел может быть, а может и не быть двухразрядным числом. Например, $2 \times 2 = 4$ и $4 \times 4 = 16$ — в каждом из этих случаев как множитель, так и множимое является только одноразрядным числом, тогда как число разрядов, занимаемых результатом, отличается на единицу. Поэтому можно заключить, что, хотя поля как множителя, так и множимого содержат максимально возможное количество значащих разрядов, все же в поле результата умножения всегда будет находиться по крайней мере один нулевой старший разряд, а если старшие разряды множителя и множимого таковы, что условие переноса не возникает, то в поле результата будут находиться два нулевых старших разряда.

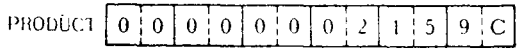
Пример 3. Ниже приводятся используемые в этом примере рабочие области, упакованные десятичные константы и их шестнадцатеричное содержимое:



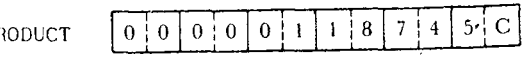
Описание этих областей и команды, которые должны быть выполнены, приведены на рис. 10.1.

Ниже приводится описание выполнения каждой из команд. В каждом случае показано текущее содержимое обработанного к данному моменту поля.

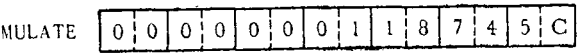
Предложение 001 загружает четырехбайтовое множимое MCAND1 в поле PRODUCT.



Предложение 002 умножает содержимое PRODUCT на упакованное десятичное содержимое поля MPLIER, результатом чего является



Предложение 003, команда ZAP , подготовит область CUMULATE, занеся в нее число, находившееся в это время в RRODUCT.



Предложение 004 помещает значение содержимого MCAND2 в младшие байты PRODUCT. Заметим, что ранее содержав-

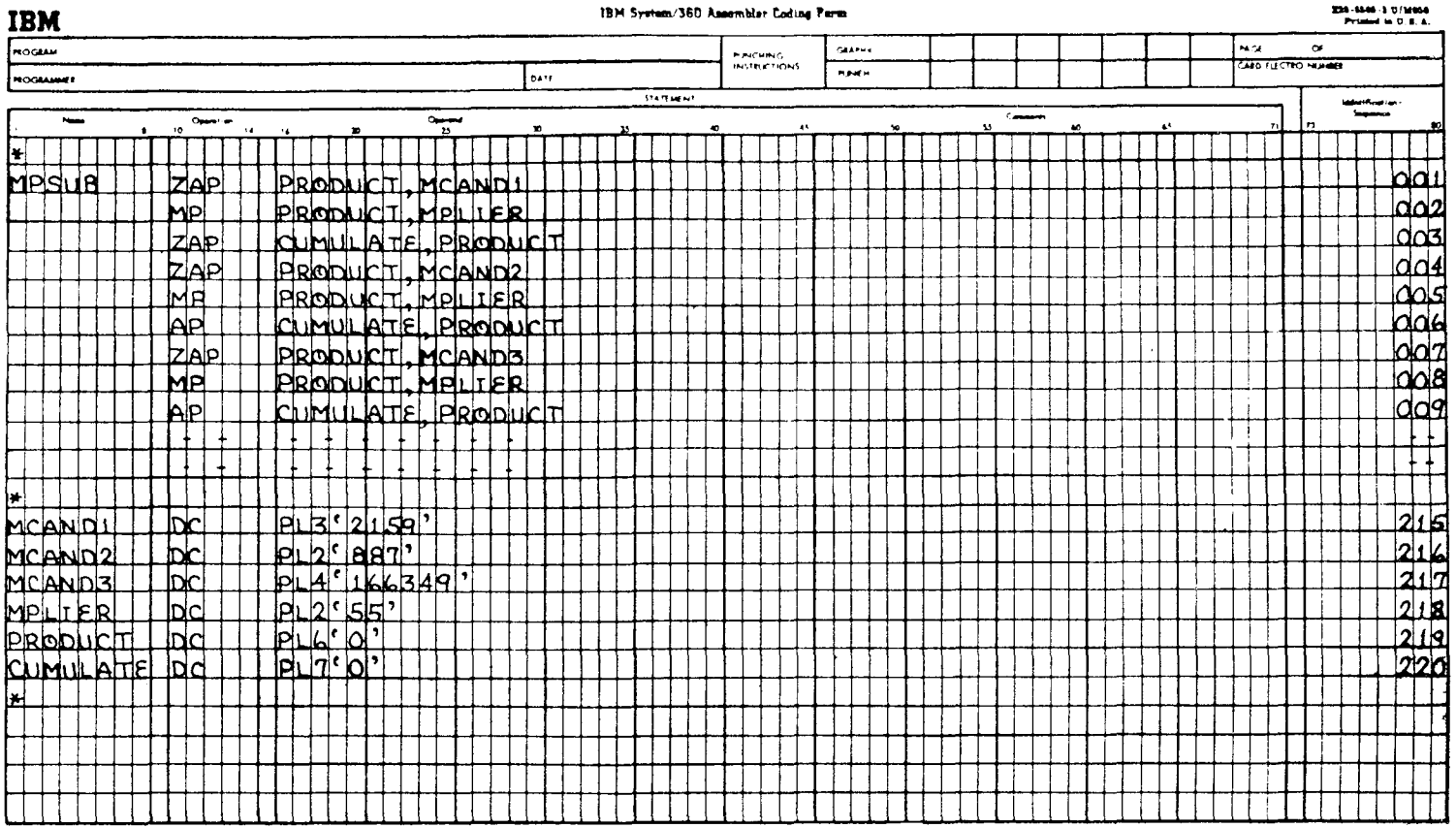

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
*
MPSUB ZAP PRODUCT,MCAND1 001
MP PRODUCT,MPLIER 002
ZAP CUMULATE,PRODUCT 003
ZAP PRODUCT,MCAND2 004
MP PRODUCT,MPLIER 005
AP CUMULATE,PRODUCT 006
ZAP PRODUCT,MCAND3 007
MP PRODUCT,MPLIER 008
AP CUMULATE,PRODUCT 009
*
MCAND1 DC PL3'2159' 215
MCAND2 DC PL2'887' 216
MCAND3 DC PL4'166349' 217
MPLIER DC PL2'55' 218
PRODUCT DC PL6'0' 219
CUMULATE DC PL7'0' 220

Рис. 10.1.

шееся в PRODUCT значение теперь полностью уничтожено.

PRODUCT

0	0	0	0	0	0	0	0	8	8	7	C

Предложение 005 умножает текущее значение величины, находящейся в поле PRODUCT (равное числу в поле MCAND2), на число, содержащееся в поле MPLIER. Теперь в PRODUCT содержится следующая величина:

PRODUCT

0	0	0	0	0	0	4	8	7	8	5	C

Предложение 006 складывает новое значение PRODUCT (+48785) с текущим значением CUMULATE (+118745), помещая полученную сумму + 167530 в CUMULATE.

CUMULATE (До AP)

0	0	0	0	0	0	0	1	1	8	7	4	5	C

PRODUCT

0	0	0	0	0	0	4	8	7	8	5	C

CUMULATE (После AP)

0	0	0	0	0	0	0	1	6	7	5	3	0	C

Предложение 007 складывает число из MCAND3 с содержимым PRODUCT, предварительно очистив последний. Эта операция является пересылкой нового множимого в младшие байты области результата.

PRODUCT

0	0	0	0	0	1	6	6	3	4	9	C

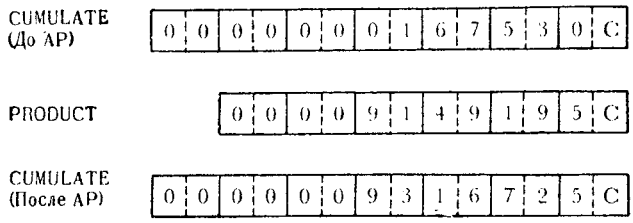
Предложение 008 для умножения текущего значения поля PRODUCT (+166349) на число из MPLIER использует команду MP.

PRODUCT

0	0	0	0	9	1	4	9	1	9	5	C

Предложение 009 складывает новое значение PRODUCT (+9149195) с текущим значением CUMULATE (+167530),

помещая полученный результат +9316725 снова в поле CUMULATE.



Выполненная в этом примере группа вычислений подтверждает тот факт, что

$$(55 \times 2159) + (55 \times 887) + (55 \times 166349) = 9316725$$

Хотя в каждой команде MP последнего примера длина множителя одна и та же, длина множимого различна. Поскольку длина области PRODUCT установлена по максимальной длине возможных комбинаций множителя и множимого, то алгоритм программы всегда может использовать одну и ту же область результата.

Заметим также, что длина поля CUMULATE на 1 байт больше длины поля PRODUCT. Если результаты арифметической операции, содержащиеся в поле упакованного формата, должны суммироваться в другом поле упакованного формата, то поле накопителя должно быть по крайней мере на 1 байт больше поля, из которого берутся данные для суммирования, для того чтобы предотвратить возможность возникновения переполнения.

Деление десятичное — DP. Упакованное десятичное содержимое первого операнда (делимое) делится на упакованное десятичное содержимое второго операнда (делитель). Частное и остаток полностью замещают делимое в первом операнде. Знак частного определяется знаками делимого и делителя в соответствии с правилами алгебры; знак остатка принимает то же значение, что и знак делимого.

При организации рабочих областей и формировании команды десятичного деления нужно следовать приведенным ниже правилам:

1. Длина делителя не должна превышать 8 байтов.
2. Должно быть подготовлено поле QUOREM [1]) (поле частного и остатка), в которое перед выполнением команды DP должно быть помещено делимое.

---

[1]) От слов QUOtient — частное и REMainder — остаток. — *Прим. ред.*

3. Длина поля QUOREM должна быть равна сумме длин делимого и делителя.
4. Длина области QUOREM не может превышать 16 байтов.
5. Длина делителя должна быть по крайней мере на 1 байт меньше длины поля делимого.
6. Длина поля делимого не должна превышать 15 байтов.
7. В обоих операндах должны находиться правильные упакованные десятичные числа со знаком.

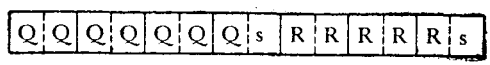
Следование этим правилам предотвратит любое возможное при выполнении команды DP программное прерывание. Нарушение любого из правил предшествующего списка с большой вероятностью приведет к неправильной спецификации или к некорректности десятичного деления и, как следствие, к программному прерыванию.

Как и в случае команды MP, для пересылки делимого в поле QUOREM желательно использовать команду ZAP. Это не только дает уверенность в том, что значение делимого выравнено по правой границе поля QUOREM, но также и в том, что старшие шестнадцатеричные разряды этого поля установлены в нуль.

После выполнения команды DP частное и остаток полностью занимают поле QUOREM. Частное располагается в самых левых байтах поля; его длина определяется указателем длины делимого. Остаток располагается в самых правых байтах поля QUOREM, занимая длину, определяемую указателем длины делителя. Знак как частного, так и остатка находится в младшем полубайте каждой из соответствующих частей поля QUOREM. Это можно проиллюстрировать следующим образом:

```
DIVIDEND    DC    PL4'0'
DIVISOR     DC    PL3'0'
QUOREM      DC    PL7'0'
```

После выполнения команды DP, работающей с этими полями, интерпретация формата поля результата QUOREM будет следующей:



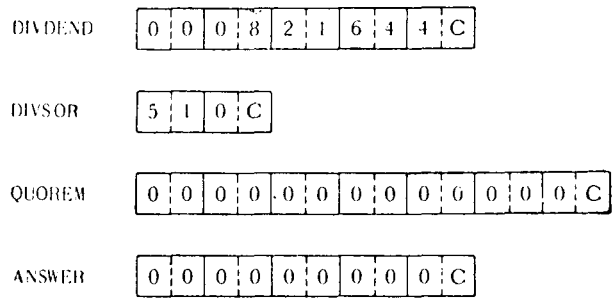
где Q = Частное (длина делимого, включая знак),
R = Остаток (длина делителя, включая знак),
s = Знак для каждого подполя.

Часто применение арифметических операций деления требует использования понятия десятичной точки. Анализ этого метода

применительно к команде DP можно найти в следующем разделе этой главы.

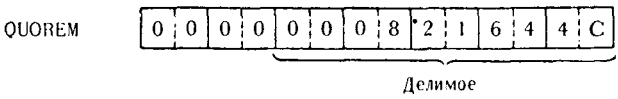
Подробный анализ следующих примеров будет способствовать правильному пониманию команды DP и соответствующих ей форматов полей.

Пример 1. Поля и рабочие области, которые должны быть использованы в этом примере, имеют следующее содержимое:



На рис. 10.2 приводятся предложения, определяющие эти области, а также последовательность команд, выполняемых в этом примере.

Предложение 001 для записи упакованного десятичного числа DIVDEND в младшие байты QUOREM использует команду ZAP.



Затем выполняется предложение 002 — команда DP, после чего содержимое QUOREM станет равным

Заметьте, что, так как длина делимого задана равной 5 байтам, старшие 5 байтов поля QUOREM после выполнения команды деления содержат частное. И так как длина делителя равна 2 байтам, то в 2 младших байтах поля QUOREM теперь находится остаток.

Предложение 003 пересылает частное из поля QUOREM в поле ANSWER посредством команды ZAP. В связи с тем что частное занимает только первые 5 байтов поля QUOREM, во

IBM System/360 Assembler Coding Form

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE | CARD ELECTRO NUMBER

STATEMENT

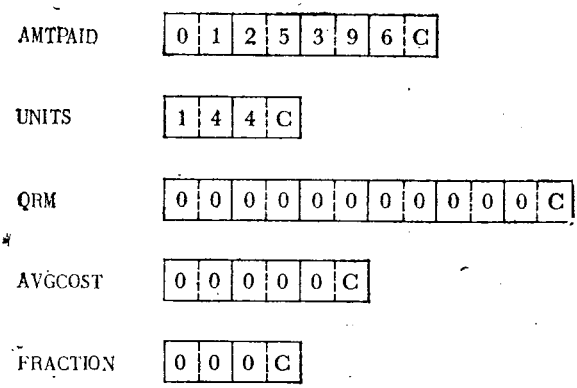
Name	Operation	Operand	Comments	Identification-Sequence
*				
DIVSUB	ZAP	QUOREM,DIVDEND		001
	DP	QUOREM,DIVSOR		002
	ZAP	ANSWER,QUOREM(5)		003
	- -	- - - - - - - -		
	- -	- - - - - - - -		
	- -	- - - - - - - -		
*				
DIVDEND	DC	PL5'821644'		072
DIVSOR	DC	PL2'510'		073
QUOREM	DC	PL7'0'		074
ANSWER	DC	PL5'0'		075
*				

Рис. 10.2.

втором операнде команды (определяющем адрес поля, откуда идет пересылка) указана длина в 5 байтов.

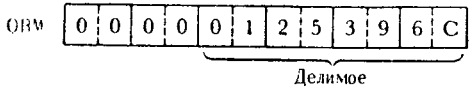
Если эта длина не будет задана, то выполнение команды приведет к попытке переслать все 7 байтов поля QUOREM в поле ANSWER. Это в свою очередь закончится программным прерыванием либо по причине переполнения, либо из-за того, что в младшем полубайте пятого байта слева, который не является младшим байтом всего семибайтного поля, находится код знака.

Пример 2. В этом примере используется как частное, так и остаток, полученные в результате операции деления. Содержимое рабочих областей и полей до выполнения команды следующее:

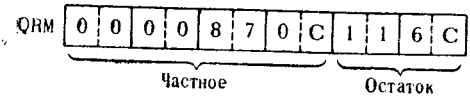


Запись команд приведена на рис. 10.3.

В предложении 001 для размещения делимого (AMTPAID) в младших 4 байтах поля QRM используется команда ZAP. После этого шестнадцатеричным содержимым QRM будет



Предложение 002 выполняет деление упакованного десятичного содержимого QRM на упакованное десятичное содержимое поля UNITS. После завершения этой операции частное и остаток можно найти в поле QRM в следующем виде:



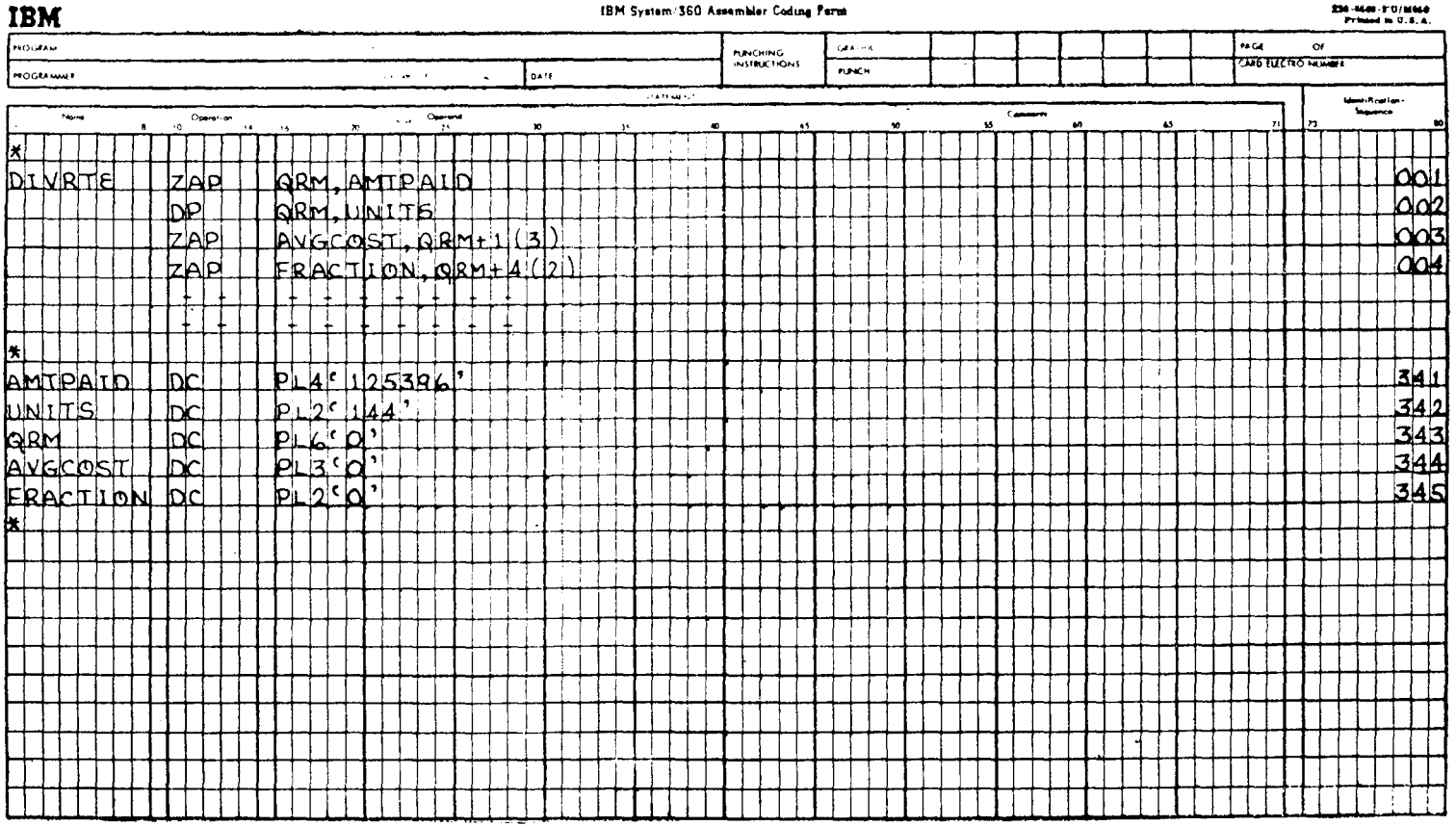
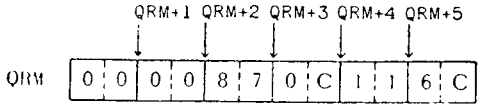

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
PUNCH
PAGE OF
CARD ELECTRO NUMBER
Name
Operation
Operand
Comments
Identification-Sequence
*
DIVRTE ZAP QRM,AMTPAID 001
DP QRM,UNITS 002
ZAP AVGCOST,QRM+1(3) 003
ZAP FRACTION,QRM+4(2) 004
*
AMTPAID DC PL4'125396' 341
UNITS DC PL2'144' 342
QRM DC PL6'0' 343
AVGCOST DC PL3'0' 344
FRACTION DC PL2'0' 345
*

Рис. 10.3.

В предложении 003 для пересылки младших 3 байтов частного в поле результата AVGCOST используется команда ZAP. Второй операнд этой команды указывает адрес QRM+1 следующего байта за QRM и задает длину пересылаемых данных, равную 3 байтам.

Расположенные в порядке возрастания адреса байтов поля QRM могут быть изображены следующим образом:

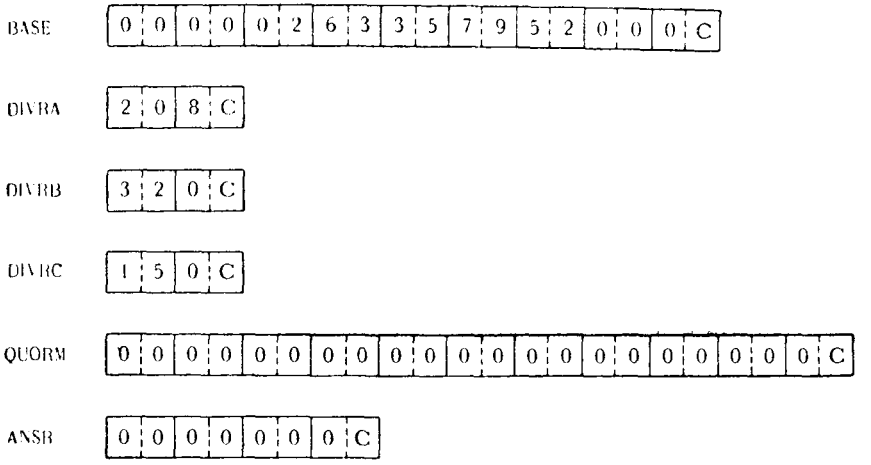


В этом случае программист знает, что делимое делится на трехразрядный делитель, поэтому он может быть уверен, что в частном должен быть по крайней мере 1 байт нулевых старших разрядов. В соответствии с этим длина поля AVGCOST, в котором должен находиться ответ, была задана на 1 байт меньше длины поля делимого. После выполнения этой команды в поле AVGCOST будет находиться число:

В предложении 004 для пересылки остатка в некоторую область памяти используется команда ZAP. Предполагается, что программист может пожелать использовать этот остаток в другом месте программы. Обратите внимание на способ, каким во втором операнде указывается адрес остатка, — начальный адрес значения остатка отстоит на 4 байта от начала поля QRM, следовательно, его адресом является QRM+4. Также указана была двухбайтовая длина остатка, вычисляемая как число оставшихся в поле QRM байтов. Теперь содержимым поля FRACTION будет

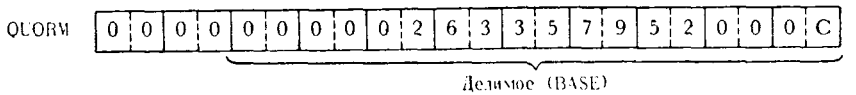
Пример 3. Этот пример немного сложнее и потребует полного внимания в связи с использованием программистом явных указателей длины и приращений адреса. Выполняемая задача заключается в последовательном делении заданного числа и последующих частных на три различных числа. Шестнадцатеричное содержимое областей и полей, используемых в этом при-

мере, имеет вид

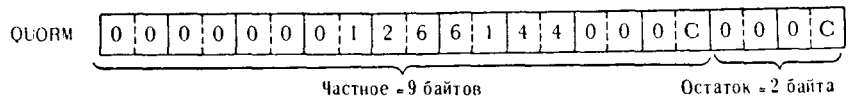


Программа для этого примера приведена на рис. 10.4.

Предложение 001 STARTUP выполняет пересылку упакованного десятичного содержимого BASE в рабочую область QUORM с предварительной очисткой последней. После этого содержимым QUORM становится



Предложение 002 посредством команды DP выполняет деление содержимого QUORM на упакованное десятичное содержимое DIVRA (+208), после чего в поле QUORM будет находиться частное и остаток:



В предложении 003 для пересылки 9 байтов частного в младшие байты этого же поля QUORM используется команда ZAP. Это необходимо для выравнивания частного по правой границе поля QUORM, так как следующая команда деления будет использовать это частное в качестве делимого.

Второй операнд содержит указатель длины в 9 байтов для того, чтобы команда ZAP выполнила пересылку только старших 9 байтов, т. е. частного. Каждая последующая команда DP

IBM

IBM System/360 Assembler Coding Form

X28-6500-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

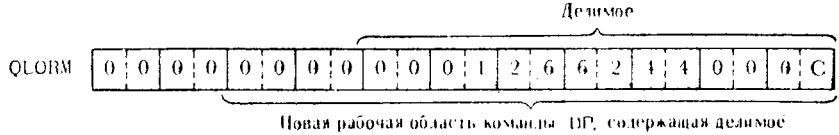
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
STARTUP	ZAP	QUORM,BASE		001
	DP	QUORM,DIVRA		002
	ZAP	QUORM,QUORM(9)		003
	DP	QUORM+2(9),DIVRB		004
	ZAP	QUORM,QUORM+2(7)		005
	DP	QUORM+4(7),DIVRC		006
	ZAP	ANSR,QUORM+5(4)		007
	- -	- - - - - - -		008
	- -	- - - - - - -		
	- -	- - - - - - -		
*				
*				250
BASE	DC	PL9'263357952000'		251
DIVRA	DC	PL2'208'		252
DIVRB	DC	PL2'320'		253
DIVRC	DC	PL2'150'		254
QUORM	DC	PL11'0'		255
ANSR	DC	PL4'0'		256
*				257
*				

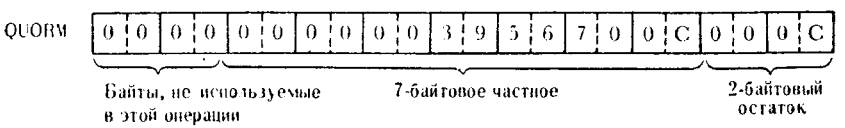
Рис. 10.4.

задана таким образом, что длина обрабатываемой части поля QUORM каждый раз уменьшается на 2 байта. Поэтому в этих командах предусмотрена настройка адреса и указание длины. Исходя из требований проблемной программы, в этом месте алгоритма заранее предусмотрено уменьшение длины поля QUORM до 9 байтов, после чего его шестнадцатеричное представление принимает вид



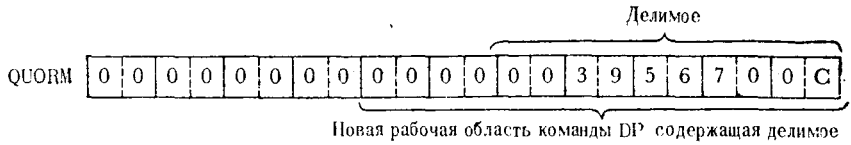
Несмотря на то что второй операнд команды ZAP указывает, что нужно пересылать только 9 байтов данных, в первом операнде по-прежнему подразумевается длина QUORM, вследствие чего в 2 старших байтах QUORM также будут помещены шестнадцатеричные нули. В этом примере они уже содержали нули, но если бы в них были значащие цифры, то можно было бы обнаружить эффект неявной длины первого операнда.

Предложение 004 теперь выполняет деление младших 9 байтов поля QUORM на 2 байта делителя DIVRB. Для обработки именно этих байтов в команде DP указывается адрес поля QUORM в виде QUORM+2 и длина в 9 байтов. В виду того что делитель содержит 2 байта, а заданная длина используемой в этой команде части поля QUORM равна 9 байтам, подразумевается что делимое состоит только из 7 байтов. Поэтому и частное будет состоять только из 7 байтов. После выполнения этой команды поле QUORM будет иметь следующий формат:

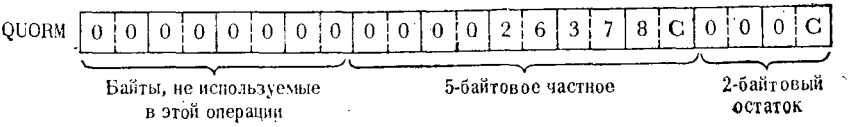


В предложении 005 для пересылки 7 байтов частного, расположенных начиная с байта QUORM+2, в младшие байты этого же поля используется команда ZAP. Таким образом, частное выравнивается по правой границе поля, что является приготовлением к следующей команде DP. В связи с тем что во втором операнде указывается длина в 7 байтов, начиная с QUORM+2, произойдет пересылка только 7 байтов, содержащих частное. Оставшаяся часть поля QUORM (старшие 4 байта) будет установлена в нуль по той причине, что в первом операнде используется неявная длина QUORM (11 байтов).

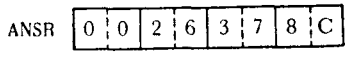
Теперь шестнадцатеричным представлением QUORM будет



В предложении 006 будет выполнено деление младших 7 байтов поля QUORM на двухбайтовый делитель DIVRC. В команде DP указывается адрес этой семибайтовой рабочей области в виде QUORM+4 и содержится указатель длины в 7 байтов. Это обеспечивает выполнение операции деления с использованием выделенной части поля QUORM, а не всех 11 байтов этого поля. В связи с тем что в этой точке программы программистом установлена длина используемой части поля QUORM в 7 байтов, а поле делителя содержит 2 байта, то подразумевается, что делимое, а следовательно, и частное должны состоять только из 5 байтов. По завершении выполнения этой команды шестнадцатеричное содержимое QUORM будет



В предложении 007 для пересылки частного, полученного в результате выполнения последней команды, из поля QUORM в область памяти, адрес которой задается меткой ANSR, используется команда ZAP. Понятно, что поле частного должно занимать 5 байтов, начиная с байта QUORM+4, но заранее известно, что по крайней мере в одном старшем байте частного будут находиться нули. Поэтому программист указал адрес второго операнда в виде QUORM+5 и задал длину в 4 байта. В соответствии с этим 4 младших байта частного (байты 6, 7, 8 и 9 поля QUORM) пересылаются в поле ANSR, после чего содержимое последнего выглядит так:



В этом примере указание длины поля QUORM для каждой последующей операции деления не было абсолютно необходимым. Программист мог бы принять, что длина делимого всегда 9 байтов, а делителю всегда требуется 2 байта, чем и была бы определена общая длина рабочей области QUORM в 11 байтов для каждой команды DP. Однако использованная форма записи

программы позволяет продемонстрировать настройку адресов и присвоение полям указателей длины.

Пересылка со сдвигом — MVO. Хотя операция, выполняемая этой командой, не является арифметической в строгом смысле этого слова, в связи с тем что она очень часто используется в арифметических программах, целесообразно обсудить ее в этом разделе.

Данные, указанные вторым операндом, пересылаются по адресу, указанному в первом операнде, но внутри поля первого операнда размещаются со сдвигом влево на четыре двоичных разряда относительно младшего разряда поля.

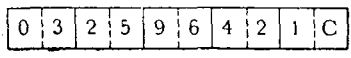
Если длина поля первого операнда больше длины поля второго операнда, то старшие полубайты поля первого операнда будут установлены в нуль. Если длина поля первого операнда меньше длины поля второго операнда, то избыточные полубайты поля второго операнда теряются.

Пересылаемые данные обрабатываются справа налево — от младших к старшим. Смещение на полбайта происходит только в поле первого операнда.

Наиболее часто эта команда применяется для сдвига десятичных чисел в упакованном формате внутри своего поля с целью изменения подразумеваемой длины дробной части числа. Это применение будет проиллюстрировано в следующем разделе этой главы.

Пример 1. Обрабатываемое упакованное десятичное поле определяется как

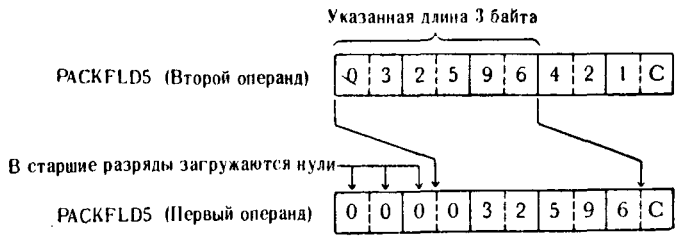
```
PACKFLD5    DC    PL5'32596421'
```



Должна быть выполнена команда

```
MVO    PACKFLD5(5),PACKFLD5(3)
```

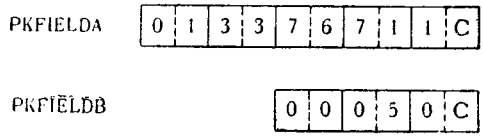
С помощью этой команды выполняется следующая операция:



Первые 3 байта PACKFLD5, как указано во втором операнде, пересылаются в PACKFLD5 с выравниванием по правой границе и смещением на полубайт. Так как длина поля второго операнда короче длины принимающего поля, то старшие разряды принимающего поля заполняются нулями.

Пример 2. Упакованные десятичные поля, обрабатываемые в этом примере, имеют вид

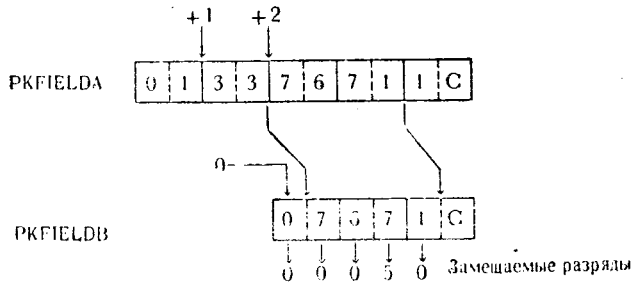
```
PKFIELDA    DC    PL5'13376711'
PKFIELDB    DC    PL3'50'
```



Выполняется команда

```
MVO     PKFIELDB,PKFIELDA+2(2)
```

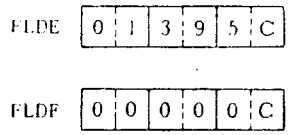
Пересылка данных, указанных в этой команде, будет выполнена следующим образом:



В этом примере 2 байта упакованных десятичных данных из поля PKFIELDA, расположенных начиная с третьего байта этого поля (PKFIELDA+2), пересылаются в поле PKFIELDB, выравниваются по правой границе этого поля и сдвигаются относительно младшего разряда на полубайт. В соответствии с этим 2 байта данных, содержащих шестнадцатеричные цифры 7671, занесены в PKFIELDB и сдвинуты влево на полбайта. В поле PKFIELDB сформировано правильное упакованное десятичное число +7671 со знаком, так как находившийся в младшем полубайте стандартный код знака плюс C не искажается при выполнении команды.

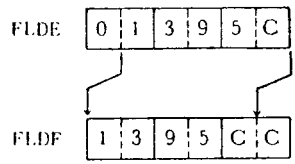
Пример 3. В этом примере показан результат пересылки всего поля упакованных десятичных данных (вместе с знаковым разрядом) в другое упакованное десятичное поле, в котором уже содержится код знака. Используются следующие поля данных:

```
FLDE    DC    PL3'1395'
FLDF    DC    PL3'00'
```



Выполняемая команда и результат ее выполнения следующие:

```
MVO     FLDF,FLDE
```



В связи с тем что пересылается все поле, а младший полубайт принимающего поля не изменяется, результатом является непригодное для использования в операциях десятичной арифметики поле FLDF. Кроме самого правого полубайта, еще в одной позиции этого поля находится шестнадцатеричная цифра, значение которой больше 9. Правильным способом пересылки FLDE в FLDF в этом случае было бы использование команды ZAP.

Г. ПРИМЕНЕНИЕ ПОДРАЗУМЕВАЕМОЙ ДЕСЯТИЧНОЙ ТОЧКИ

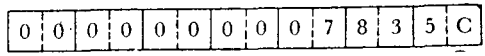
Программисту часто приходится учитывать положение десятичной точки и использовать правила округления при выполнении операций десятичной арифметики. Как и в случае арифметических операций с фиксированной точкой, дополнительные десятичные разряды могут быть включены или удалены из упакованного десятичного поля путем умножения или деления на 10, 100, 1000 и т. д. Необходимо заметить, что при работе с упакованными десятичными числами применение команды MVO обеспечивает эффективное выполнение многих из этих задач.

Имеется несколько способов увеличения или уменьшения числа позиций после десятичной точки (дробной части числа), с помощью которых могут быть получены одинаковые

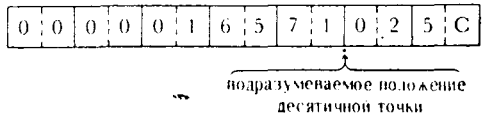
результаты. Ответственность за решение вопроса, какой метод больше подходит к данному случаю, лежит на программисте. Желательно, конечно, чтобы это решение основывалось на эффективности выполнения операций, а не на симпатиях или антипатиях пользователя. В последующих примерах приводится сравнение этих методов на основе анализа текстов программ. Действительная их эффективность может быть определена только с учетом характеристик вычислительной системы, где они применяются, и требований конкретной проблемной программы.

Пример 1. На рис. 10.5 приведены команды программы и определены используемые поля и рабочие области. Если необходимо, подразумеваемое положение десятичной точки указывается в виде комментария в той же строке, где определяется поле или область.

В предложении 001 для пересылки содержимого поля UNITS в младшие байты WORK используется команда ZAP. После этого шестнадцатеричное содержимое WORK станет равным

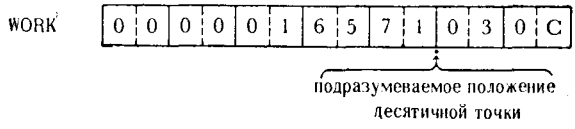


В предложении 002 происходит умножение числа WORK (множимого) на число COSTPER (множитель). Результат умножения помещается в WORK в следующем виде:



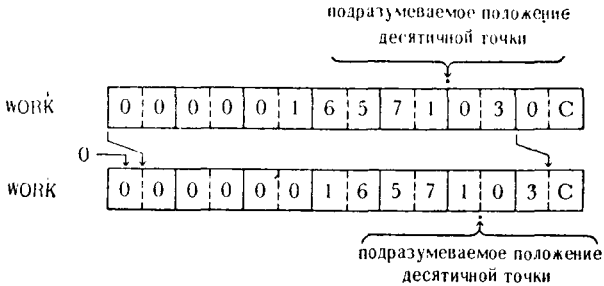
Так как подразумевается, что число в поле COSTPER содержит три разряда после десятичной точки, то и результат имеет ту же подразумеваемую длину дробной части. Поэтому произведение выражает число +16571.025. В этом месте программы требуемый ответ должен быть округлен до второго знака после точки, что и выполняется следующей командой.

Предложение 003 добавляет число +5 в младший полубайт поля WORK, таким образом округляя число WORK до ближайшей сотой. Теперь содержимым WORK является



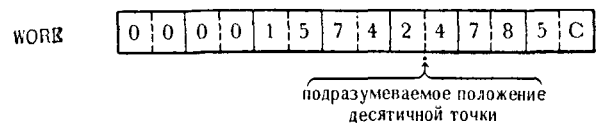
В предложении 004 для удаления одной позиции после десятичной точки из числа WORK используется команда MVO.

Это связано с тем, что следующей командой это число пересылается в поле GROSS, предназначенное для чисел с двумя разрядами после десятичной точки. Команда MVO пересылает первые 6 байтов поля WORK в младшие позиции WORK со сдвигом на четыре двоичных разряда. Выполнение команды может быть представлено следующим образом:



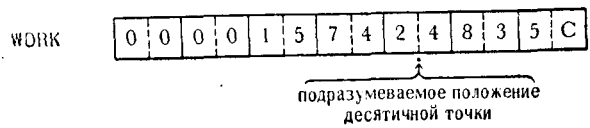
В предложении 005 для пересылки младших 4 байтов WORK, указанных операндом WORK+3(4), в поле GROSS используется команда ZAP. После этого содержимым GROSS будет

В предложении 006 произойдет умножение содержимого WORK на величину DISCOUNT. В связи с тем что в поле WORK подразумеваются две позиции после десятичной точки, так же как и в поле DISCOUNT, результат умножения в поле WORK расматривается как имеющий четырехразрядную дробную часть. По завершении выполнения этой команды шестнадцатеричное содержимое WORK будет иметь следующий вид:



По существу величина в поле WORK уменьшена на 5%, и результат (95% прежнего значения) должен быть теперь округлен до ближайшей сотой и помещен в поле NET.

В предложении 007 для округления этого числа до второго знака после десятичной точки к WORK прибавляется +50. Теперь поле WORK имеет вид



IBM

IBM System/360 Assembler Coding Form

X28-6509-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

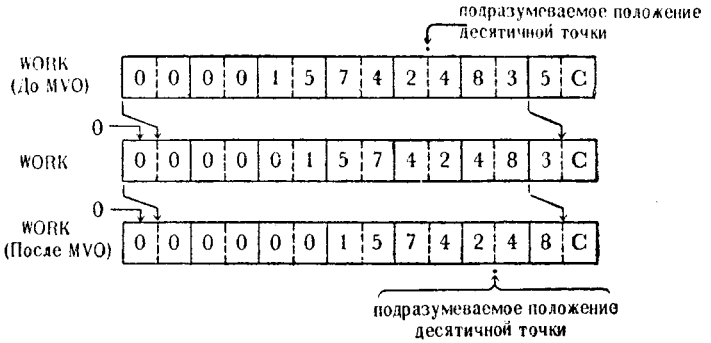
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
UNITS	DC	PL3'7835'	(XXXXX. ASSUMED)	
COSTPER	DC	PL3'2115'	(X.000 ASSUMED)	
GROSS	DC	PL4'0'	(XXXXX.00 ASSUMED)	
DISCOUNT	DC	PL2'95'	(X.00 ASSUMED)	
NET	DC	PL4'0'	(XXXXX.00 ASSUMED)	
WORK	DC	PL7'0'		
ROUND1	DC	PL1'5'		
ROUND2	DC	PL2'50'		
*				
*				
STARTER	ZAP	WORK,UNITS		001
	MP	WORK,COSTPER		002
	AP	WORK,ROUND1		003
	MVO	WORK(7),WORK(6)		004
	ZAP	GROSS,WORK+3(4)		005
	MP	WORK,DISCOUNT		006
	AP	WORK,ROUND2		007
	MVO	WORK(7),WORK(6)		008
	MVO	WORK(7),WORK(6)		009
	ZAP	NET,WORK+3(4)		010
				011

Рис. 10.5.

Предложения 008 и 009 являются одинаковыми командами пересылки со сдвигом. Их выполнение приведет к следующим изменениям поля WORK:

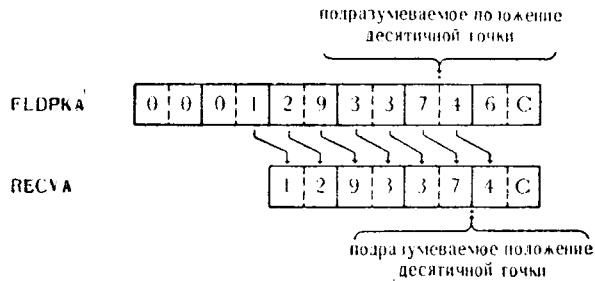


Можно было бы исключить две младшие позиции после десятичной точки путем деления на +100. Последовательность из двух команд MVO применена для того, чтобы показать работу этих команд в случае пересылки четного числа позиций.

В предложении 010 для пересылки младших 4 байтов поля WORK в поле результата NET используется команда ZAP. После этого шестнадцатеричное содержимое поля NET примет вид

Пример 2. Этот пример (рис. 10.6) представляет собой просто ряд команд MVO, оперирующих с различными полями. Он демонстрирует адресацию, требуемую для удаления нечетного числа младших позиций после десятичной точки.

В предложении 001 для исключения из числа, находящегося в поле FLDPKA, одной младшей позиции после десятичной точки при его пересылке в RECVA используется команда MVO. Эта пересылка выполняется следующим образом:



IBM

IBM System/360 Assembler Coding Form

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE | OF

PROGRAMMER | DATE | CARD ELECTRO NUMBER

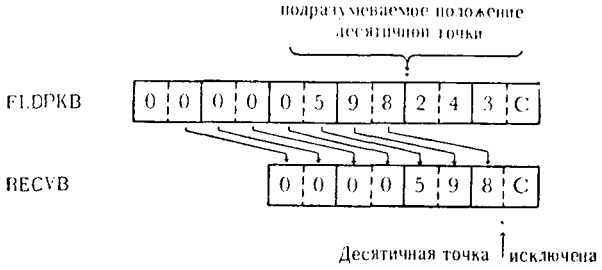
STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
FLDPKA	DC	PL6'12933746'	(XXXXXXX.00 ASSUMED)	
FLDPKB	DC	PL6'598243'	(XXXXXX.000 ASSUMED)	
FLDPKC	DC	PL6'216597718'	(XXX.000000 ASSUMED)	
FLDPKD	DC	PL6'1112490003'	(.0000000000 ASSUMED)	
RECVA	DC	PL4'0'		
RECVB	DC	PL4'0'		
RECVC	DC	PL3'0'		
RECVD	DC	PL2'0'		
*				
MODRTE	MVO	RECVA,FLDPKA(5)		001
	MVO	RECVB,FLDPKB(4)		002
	MVO	RECVC,FLDPKC(3)		003
	MVO	RECVD,FLDPKD(2)		004

Рис. 10.6.

Произошла пересылка содержимого FLDPKA в RECVA и одновременное изменение значения содержимого с +129337.46 на + 129337.4.

В предложении 002 производится уменьшение дробной части числа FLDPKB на три позиции при его пересылке в RECVB. Это может быть изображено так:



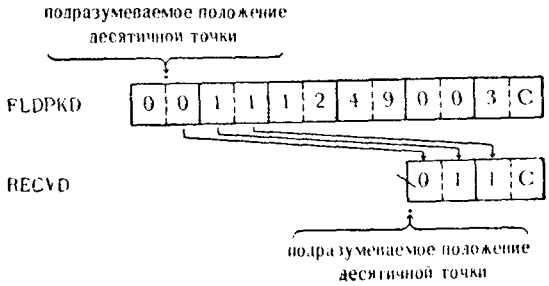
Выполнена пересылка упакованного десятичного содержимого FLDPKB в RECVB и изменено его значение с +598.243 на +598.

В предложении 003 при помощи команды MVO производится пересылка числа FLDPKC в RECVC, но при этом дробная часть числа укорачивается на пять позиций.

Упакованное десятичное содержимое FLDPKC теперь частично переписано в RECVC, но при пересылке значение числа изменено с +216.597718 на +216.5.

В предложении 004 для пересылки части содержимого FLDPKD в RECVD с одновременным уменьшением дробной части пересылаемого числа на семь позиций используется

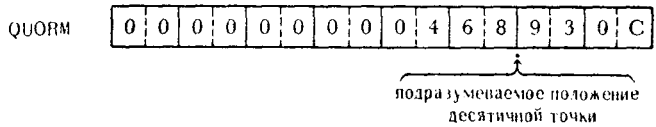
команда MVO. После пересылки число примет вид



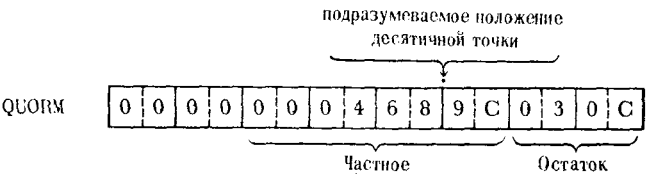
При пересылке из FLDPKD в RECVD число укорочено на семь десятичных позиций, что привело к уменьшению значения рассматриваемого числа с +.0111249003 до +.011.

Пример 3. В этом примере демонстрируется метод уменьшения числа на четное число младших позиций. Это можно сделать с помощью команды DP или пары команд MVO. Будут показаны оба способа (рис. 10.7).

В предложении 001 для пересылки упакованного десятичного числа AFLD в младшие байты QUORM в целях подготовки к операции деления используется команда ZAP. Теперь шестнадцатеричное содержимое QUORM имеет следующий вид:



В предложении 002 происходит деление упакованного десятичного числа QUORM на упакованное десятичное число +100. Эта операция укоротит число QUORM на два младших разряда. В результате выполнения этой команды содержимое поля примет вид



Первоначальное значение делимого уменьшено с 468.930 до 468.9.

В предложении 003 три младших байта частного пересылаются из QUORM в поле HOLDA. В поле HOLDA теперь

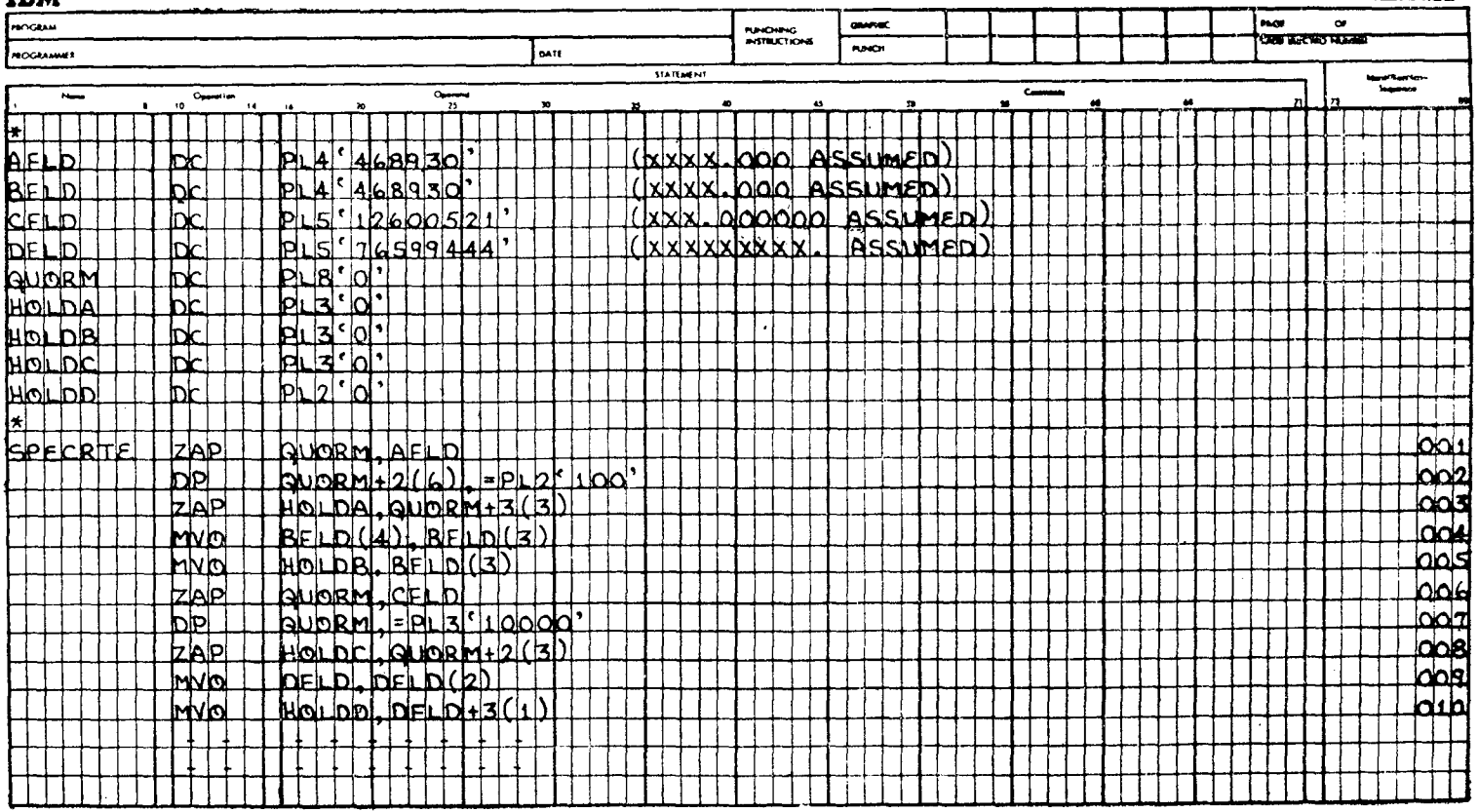
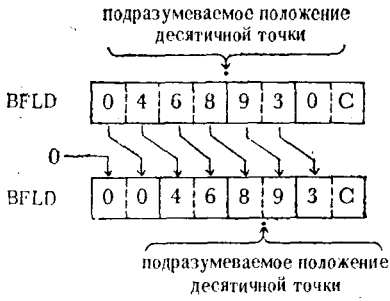

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
*
AFLD DC PL4'4689930' (XXXX.000 ASSUMED)
BFLD DC PL4'4689300' (XXXX.000 ASSUMED)
CFLD DC PL5'1260052 1' (XXX.000000 ASSUMED)
DFLD DC PL5'7459944 4' (XXXXXXXXX. ASSUMED)
QUORM DC PL8'0'
HOLDA DC PL3'0'
HOLDB DC PL3'0'
HOLDC DC PL3'0'
HOLDD DC PL2'0'
*
SPECRTE ZAP QUORM,AFLD 001
DP QUORM+2(6),=PL2'100' 002
ZAP HOLDA,QUORM+3(3) 003
MVO BFLD(4),BFLD(3) 004
MVO HOLDB,BFLD(3) 005
ZAP QUORM,CFLD 006
DP QUORM,=PL3'10000' 007
ZAP HOLDC,QUORM+2(3) 008
MVO DFLD,DFLD(2) 009
MVO HOLDD,DFLD+3(1) 010

Рис. 10.7.

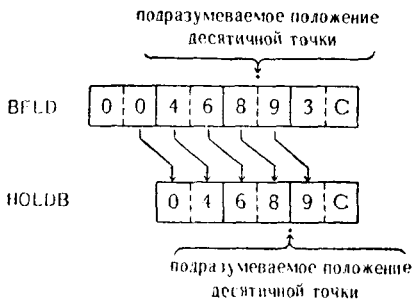
будут находиться следующие данные:

В предложении 004 выполняется команда MVO для уменьшения числа BFLD на один младший разряд. После этого в поле BFLD окажется число +468.93, и поле примет вид



Это предложение было первым шагом удаления двух младших разрядов из числа BFLD. Удаление второго разряда выполняется следующим предложением.

В предложении 005 происходит пересылка числа BFLD в поле HOLDB с одновременным удалением еще одного младшего разряда.



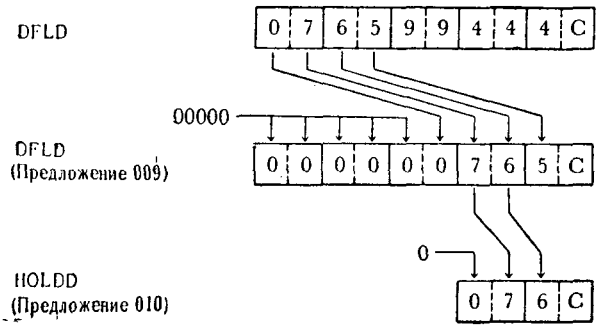
При сравнении результатов выполнения предложений 004 и 005 и предложений 001, 002 и 003 можно заметить, что они сформировали одинаковые числа. Как в AFLD, так и в BFLD находились одинаковые исходные числа, а теперь в HOLDA и HOLDB также содержатся одинаковые числа.

В предложении 006 содержимое CFLD командой ZAP заносится в младшие байты QUORM.

В предложении 007 для уменьшения числа QUORM на четыре младших разряда производится деление QUORM на упакованное десятичное число +10000. Поле QUORM после выполнения этой команды примет вид

В предложении 008 три младших байта частного пересылаются при помощи команды ZAP в HOLDC. Теперь число CFLD уменьшено на четыре младших разряда, что изменило его значение с +12.600521 на +12.60, и это новое число помещается в HOLDC:

В предложениях 009 и 010 для уменьшения упакованного десятичного числа DFLD на шесть младших разрядов используются две команды MVO. Одновременно с этим в предложении 010 происходит пересылка результата в поле HOLDD.

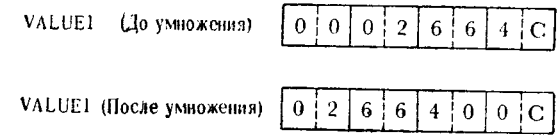


Из рисунка видно, что шесть младших разрядов DFLD удалены.

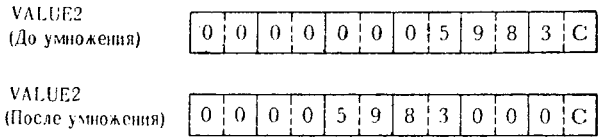
Пример 4. Добавление нулевых разрядов в младшие позиции упакованного десятичного числа может быть легко выполнено с помощью команды MP. Следующая запись является иллюстрацией сказанного:

```
VALUE1,   DC    PL4'2664'
VALUE2,   DC    PL6'5983'
VALUE3,   DC    PL6'7717'
VALUE4,   DC    PL7'4092'
*
RTE1      MP    VALUE1,=PL2'100'      0001
          MP    VALUE2,=PL3'1000'     0002
          MP    VALUE3,=PL3'10000'    0003
          MP    VALUE4,=PL4'100000'   0004
```

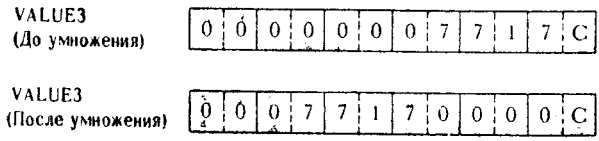
Предложение 0001 увеличит число VALUE1 путем добавления двух младших нулевых разрядов. Это достигается умножением его текущего значения на упакованное десятичное число +100. Поле VALUE1 будет иметь следующий вид:



Предложение 0002 добавит к числу в поле VALUE2 три младших нулевых разряда путем умножения значения величины в этом поле на упакованное десятичное число +1000.



Предложение 0003 увеличит число в поле VALUE3 путем умножения его на упакованное десятичное число +10000. К первоначальному числу добавится четыре младших нулевых разряда.



Предложение 0004 увеличит число в поле VALUE4, добавив пять младших нулевых разрядов.

VALUE4 (До умножения)	0	0	0	0	0	0	0	0	0	4	0	9	2	C
VALUE4 (После умножения)	0	0	0	0	4	0	9	2	0	0	0	0	0	C

Приведенные примеры дают достаточно материала для работы с подразумеваемыми десятичными точками упакованных десятичных чисел. Этот метод может быть эффективно использован для уменьшения или увеличения значений величин не обязательно в связи с положением десятичной точки. При помощи команд MP, DP, MVO и ZAP упакованные десятичные числа могут быть сформированы или преобразованы в соответствии с требованиями проблемной программы.

Упражнения

Следующие задачи приводятся либо в форме вопросов, либо в форме команд, которые должны быть выполнены. В вопросах вставьте пропущенное слово или слова; в командах занесите в незаполненные поля данные в том виде, в каком они должны получиться после выполнения команды.

1. Данные, состоящие из цифровых символов кода EBCDIC, преобразуются в упакованный десятичный формат командой _____.

2. Числа с фиксированной точкой преобразуются в упакованные десятичные числа командой _____.

3. Отличительной чертой команд десятичной арифметики является присутствие в обоих операндах в явной или неявной форме указателя _____.

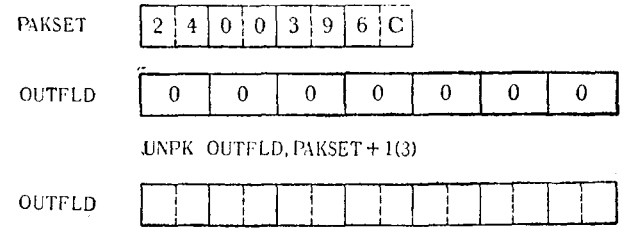
4.

INFLD	3	2	5	9	6			
PAKFLD	1	7	0	6	2	3	9	D

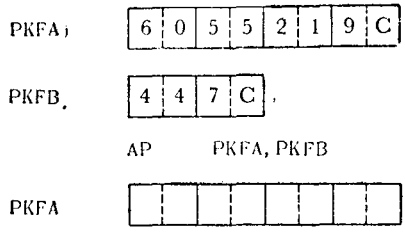
PACK PAKFLD,INFLD+1(3)

PAKFLD								

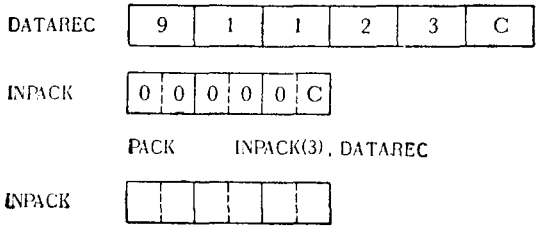
5.


PAKSET
2 4 0 0 3 9 6 C
OUTFLD
0 0 0 0 0 0 0
UNPK OUTFLD, PAKSET+1(3)
OUTFLD

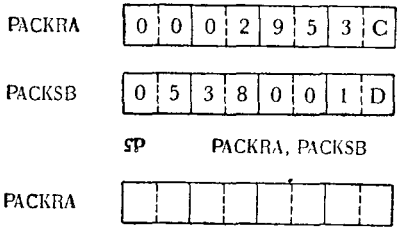
6.


PKFA
6 0 5 5 2 1 9 C
PKFB
4 4 7 C
AP PKFA, PKFB
PKFA

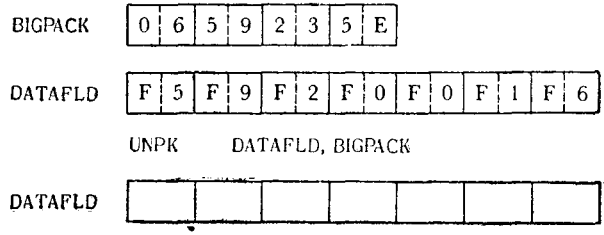
7.


DATAREC
9 1 1 2 3 C
INPACK
0 0 0 0 0 C
PACK INPACK(3), DATAREC
INPACK

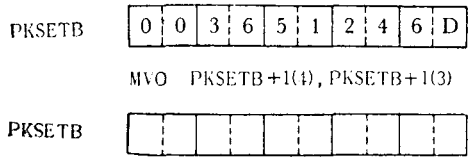
8.


PACKRA
0 0 0 2 9 5 3 C
PACKSB
0 5 3 8 0 0 1 D
SP PACKRA, PACKSB
PACKRA

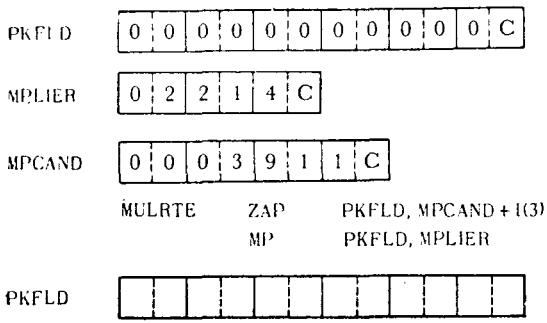
9.


BIGPACK
0 6 5 9 2 3 5 E
DATAFLD
F 5 F 9 F 2 F 0 F 0 F 1 F 6
UNPK DATAFLD, BIGPACK
DATAFLD

10.



11.



12. Максимальная длина делителя в команде DP равна _____ байтам.

13. Если длина поля множимого 6 байтов, а длина поля множителя 4 байта, то длина поля результата соответствующей команды MP должна быть равна _____ байтам.

14. Преобразуемые в число с фиксированной точкой упакованные данные должны находиться в поле из _____ байтов и быть выравнены в нем по _____ границе поля.

15. Если буквенные символы кода EBCDIC ABCDEF упаковываются, а затем немедленно распаковываются, то будут получены символы кода EBCDIC _____.

16. Длина рабочей области команды DP, называемой QUOREM, должна быть равна суммарной длине делимого и _____.

17. В команде ZAP только _____ операнд должен представлять правильное поле упакованного формата со знаком.

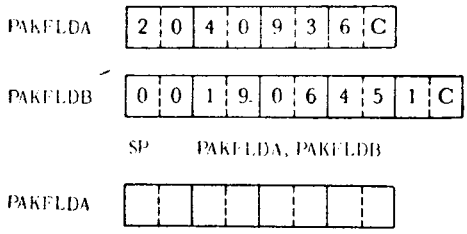
18. Числа с фиксированной точкой могут быть преобразованы в цифровые символы кода EBCDIC при помощи совместного использования команд _____ и _____.

19. Если упаковываемое поле зонного формата содержит больше цифровых символов, чем может поместиться в поле упакованного формата, то _____ разряды поля зонного формата теряются.

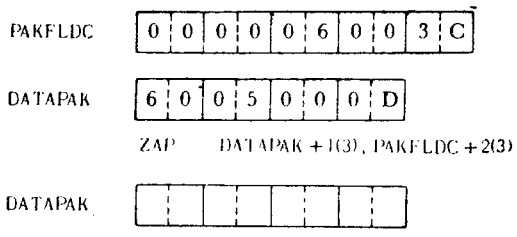
20. Команда MVO пересылает содержимое одного поля в другое поле, в пределах которого сдвигает его _____ на полубайт.

21. Если трехбайтовое поле упакованного формата со знаком распаковывается в восьмибайтовое поле, то в каждый из 3 старших байтов принимающего поля будет загружена шестнадцатеричная константа_____.

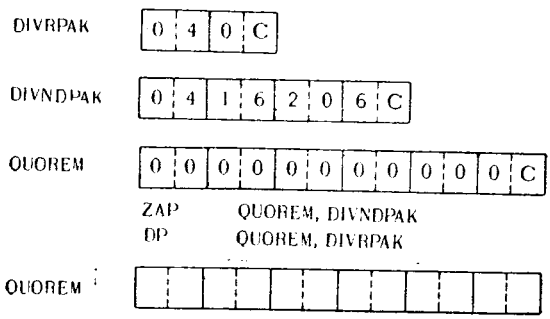
22.



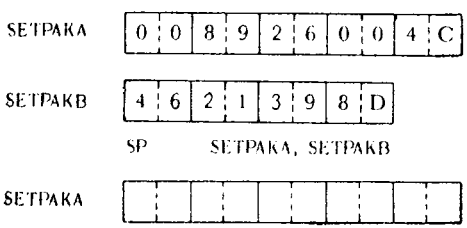
23.



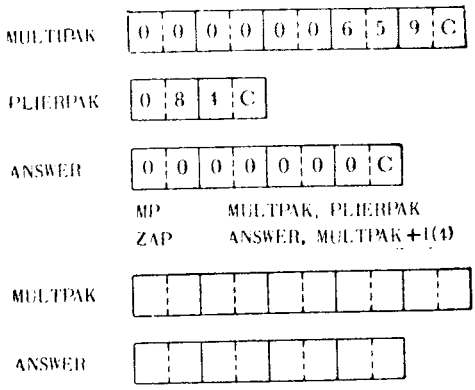
24.



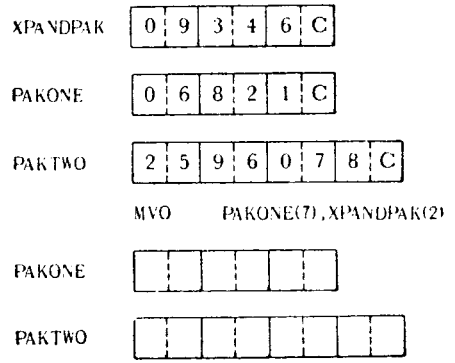
25.



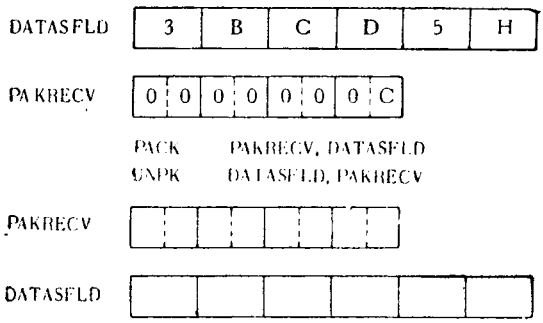
26.


MULTPAK
0 0 0 0 0 0 6 5 9 C
PLIERPAK
0 8 1 C
ANSWER
0 0 0 0 0 0 0 C
MP MULTPAK, PLIERPAK
ZAP ANSWER, MULTPAK+1(4)
MULTPAK
ANSWER

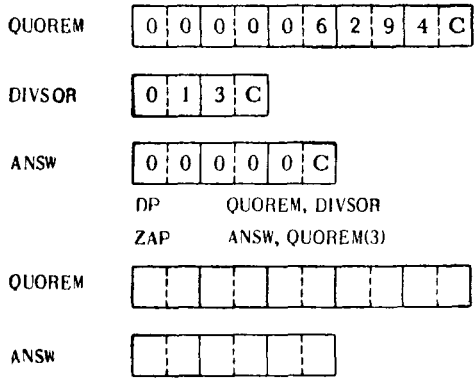
27.


XPANDPAK
0 9 3 4 6 C
PAKONE
0 6 8 2 1 C
PAKTWO
2 5 9 6 0 7 8 C
MVO PAKONE(7),XPANDPAK(2)
PAKONE
PAKTWO

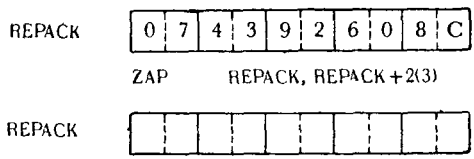
28.


DATASFLD
3 B C D 5 H
PAKRECV
0 0 0 0 0 0 0 C
PACK PAKRECV, DATASFLD
UNPK DATASFLD, PAKRECV
PAKRECV
DATASFLD

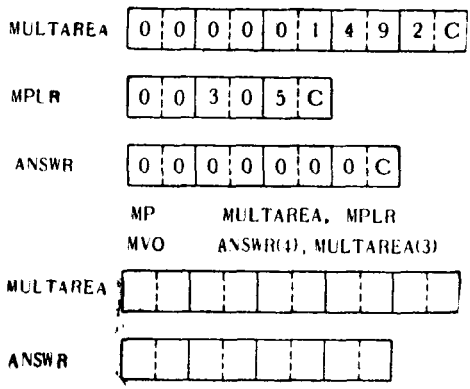
29.


QUOREM
0 0 0 0 0 6 2 9 4 C
DIVSOR
0 1 3 C
ANSW
0 0 0 0 0 C
DP QUOREM, DIVSOR
ZAP ANSW, QUOREM(3)
QUOREM
ANSW

30.


REPACK
0 7 4 3 9 2 6 0 8 C
ZAP REPACK, REPACK+2(3)
REPACK

31.


MULTAREA
0 0 0 0 0 1 4 9 2 C
MPLR
0 0 3 0 5 C
ANSWR
0 0 0 0 0 0 0 C
MP MULTAREA, MPLR
MVO ANSWR(4), MULTAREA(3)
MULTAREA
ANSWR

Глава 11

Арифметические действия над числами с фиксированной точкой

А. КОМАНДЫ АРИФМЕТИКИ С ФИКСИРОВАННОЙ ТОЧКОЙ

Команда Add — A (Сложение)

Мнемоника	Код операции	Формат операндов
A	5A	$R_1,D_2(X_2,B_2)$

По этой команде 32-разрядное значение второго операнда (1 бит знака и 31 бит числа) складывается с содержимым регистра первого операнда. Второй операнд должен быть расположен на границе слова. Результат сложения помещается в общий регистр первого операнда.

CC	BC	Условие
0	8	Регистр первого операнда содержит 0
1	4	Регистр первого операнда содержит отрицательную величину
2	2	Регистр первого операнда содержит положительную величину
3	1	Произошло переполнение

Команда Add Halfword — AH (Сложение полуслова)

Мнемоника	Код операции	Формат операндов
AH	4A	$R_1,D_2(X_2,B_2)$

Сначала 16-битовое значение второго операнда извлекается и расширяется до полного слова путем добавления 16 старших двоичных разрядов, причем каждый дополнительный бит имеет то же значение, что и знаковый бит первоначального значения второго операнда. Затем это расширенное значение складывается с числом с фиксированной точкой (1 бит знака и 31 бит числа), находящимся в регистре первого операнда, и сумма помещается в регистр первого операнда. Второй операнд должен быть размещен в памяти на границе полуслова.

CC	BC	Условие
0	8	Регистр первого операнда содержит 0
1	4	Регистр первого операнда содержит отрицательную величину
2	2	Регистр первого операнда содержит положительную величину
3	1	Произошло переполнение

Команда Add Registers — AR (Сложение)

Мнемоника	Код операции	Формат операндов
AR	1A	R_1,R_2

Число с фиксированной точкой из регистра первого операнда складывается с числом с фиксированной точкой из регистра второго операнда и результат сложения помещается в регистр первого операнда. Оба общих регистра содержат величины, состоящие из знакового бита и 31 бита числа.

CC	BC	Условие
0	8	Регистр первого операнда содержит 0
1	4	Регистр первого операнда содержит отрицательную величину
2	2	Регистр первого операнда содержит положительную величину
3	1	Произошло переполнение

Команда Add Logical — AL (Сложение кодов)

Мнемоника	Код операции	Формат операндов
AL	5E	$R_1,D_2(X_2,B_2)$

Производится сложение 32-битового представления величины второго операнда с 32 битами регистра первого операнда и результат помещается в регистр первого операнда. В сложении принимают участие все биты, включая биты знака без последующего изменения знакового бита результата. Условие переполнения не вызывает программного прерывания. Второй операнд должен размещаться в памяти на границе слова. Признак результата, устанавливаемый после выполнения команды, указывает, произошел или нет перенос из знакового разряда.

CC	BC	Условие
0	8	Первый операнд равен 0; переноса из знакового разряда не произошло
1	4	Первый операнд не равен 0; переноса не произошло
2	2	Первый операнд равен 0; произошел перенос
3	1	Первый операнд не равен 0; произошел перенос

Команда Add Logical Registers — ALR (Сложение кодов)

Мнемоника	Код операции	Формат операндов
ALR	1E	R_1,R_2

Производится сложение 32-битового представления регистра второго операнда с 32 битами регистра первого операнда и сумма помещается в регистр первого операнда. Все биты, включая знаковые, складываются по правилам сложения двоичных кодов без последующего изменения знакового бита результата. *Условие переполнения не вызывает программного прерывания.* Значение признака результата показывает, произошел или нет перенос из знакового разряда.

CC	BC	Условие
0	8	Первый операнд равен 0; перенос не произошел
1	4	Первый операнд содержит ненулевое значение; перенос не произошел
2	2	Первый операнд равен 0; произошел перенос
3	1	Первый операнд содержит ненулевое значение; произошел перенос

Команда Subtract — S (Вычитание)

Мнемоника	Код операции	Формат операндов
S	5B	$R_1,D_2(X_2,B_2)$

Значение числа с фиксированной точкой из слова, указанного вторым операндом, вычитается из содержимого регистра первого операнда. Вычитание производится путем сложения содержимого второго операнда в дополнительном коде с содержимым первого операнда. Результат помещается в общий регистр первого операнда. Если в результате выполнения этой команды произойдет переполнение, будет установлено

соответствующее значение признака результата, что обычно приводит к программному прерыванию. Второй операнд должен быть расположен в памяти на границе слова.

CC	BC	Условие
0	8	Первый операнд равен 0
1	4	Первый операнд содержит отрицательное число
2	2	Содержимое первого операнда больше 0
3	1	Произошло переполнение

Команда Subtract Halfword — SH (Вычитание полуслова)

Мнемоника	Код операции	Формат операндов
SH	4B	$R_1,D_2(X_2, B_2)$

Число со знаком из полуслова, указанного вторым операндом, расширяется до целого слова путем добавления 16 старших двоичных разрядов со значением, равным значению знакового разряда первоначального содержимого полуслова. Затем это расширенное значение вычитается из содержимого регистра первого операнда. Вычитание производится путем сложения расширенного значения в дополнительном коде со значением, содержащимся в общем регистре первого операнда. Второй операнд должен располагаться на границе полуслова. Если во время выполнения команды произойдет переполнение, то установится соответствующее значение признака результата и, как правило, произойдет программное прерывание.

CC	BC	Условие
0	8	Первый операнд равен 0
1	4	Первый операнд содержит отрицательную величину
2	2	Первый операнд содержит положительную величину
3	1	Произошло переполнение

Команда Subtract Registers — SR (Вычитание)

Мнемоника	Код операции	Формат операндов
SR	1B	R_1,R_2

Из содержимого регистра первого операнда вычитается содержимое регистра второго операнда. Вычитание производится путем сложения содержимого второго операнда в дополнительном коде с содержимым регистра первого операнда. Результат помещается в регистр первого операнда. Если при выполнении

этой команды происходит переполнение, устанавливается соответствующее значение признака результата.

CC	BC	Условие
0	8	Первый операнд равен 0
1	4	Первый операнд содержит число меньше 0
2	2	Первый операнд содержит число больше 0
3	1	Произошло переполнение

Команда Subtract Logical — SL (Вычитание кодов)

Мнемоника	Код операции	Формат операндов
SL	5F	$R_1,D_2(X_2, B_2)$

Из содержимого регистра первого операнда вычитается содержимое 32 битов полного слова, адресуемого вторым операндом. Это производится путем сложения содержимого первого операнда с дополнением до 2 содержимого второго операнда; результат помещается в регистр первого операнда. В вычитании участвуют все 32 бита операндов, включая знаковый бит. Если возникает переполнение, программного прерывания не происходит. Вместо этого значение признака результата после выполнения команды укажет, произошел или нет перенос из знакового разряда.

CC	BC	Условие
1	4	Первый операнд содержит ненулевое значение; переноса не произошло
2	2	Первый операнд содержит нулевое значение; произошел перенос
3	1	Первый операнд содержит ненулевое значение; произошел перенос

Команда Subtract Logical Registers — SLR (Вычитание кодов)

Мнемоника	Код операции	Формат операндов
SLR	1F	R_1,R_2

Из содержимого регистра первого операнда вычитается содержимое регистра второго операнда. Это производится путем суммирования дополнения до 2 содержимого регистра второго операнда с содержимым регистра первого операнда; результат помещается в регистр первого операнда. В вычитании принимают участие все 32 бита обоих операндов, включая бит знака. Если возникает условие переполнения, программного прерывания

не происходит. Значение признака результата показывает, произошел или нет перенос из знакового разряда при выполнении команды.

CC	BC	Условие
1	4	Содержимое первого операнда ненулевое; переноса не произошло
2	2	Первый операнд содержит нулевое значение; произошел перенос
3	1	Первый операнд содержит ненулевое значение; произошел перенос

Команда Multiply — M (Умножение)

Мнемоника	Код операции	Формат операндов
M	5C	$R_1,D_2(X_2,B_2)$

Содержимое первого операнда (представляющее множимое, хранящееся в паре общих регистров, номер первого из которых четный) умножается на содержимое второго операнда (множитель). Оба операнда представляют собой 32-разрядные двоичные числа со знаком. Первый операнд должен всегда указывать четный регистр пары общих регистров. Поле множителя, определяемое вторым операндом, должно быть расположено в памяти на границе слова. Результат умножения представляет собой 64-битовое число со знаком, размещенное в паре регистров, указанной первым операндом. Знаки множимого и множителя определяют знак произведения по правилам алгебры.

Признак результата не изменяется.

Команда Multiply Registers — MR (Умножение)

Мнемоника	Код операции	Формат операндов
MR	1C	R_1,R_2

Содержимое первого операнда (представляющее множимое, хранящееся в паре общих регистров, номер первого из которых четный) умножается на содержимое регистра второго операнда (множитель). Первый операнд всегда должен указывать четный регистр пары общих регистров. Произведение помещается в пару регистров, указанную первым операндом. Знак произведения определяется знаками сомножителей по правилам алгебры.

Признак результата не изменяется.

Команда Multiply Halfword — MH (Умножение полуслова)

Мнемоника	Код операции	Форма операндов
MH	4C	$R_1,D_2(X_2,B_2)$

Хотя, судя по названию этой команды, можно предположить что множитель и множимое являются полусловами, множимое тем не менее может иметь значение, большее максимальной величины числа длиной в полуслово, но при условии, что результат умножения не превосходит максимального значения, которое можно записать в полное слово. Сначала содержимое полуслова, указанного вторым операндом, извлекается и расширяется до размера слова путем добавления 16 старших битов того же значения, что и знаковый бит первоначального полуслова. Затем содержимое регистра первого операнда (множимое) умножается на 32-битовый расширенный множитель. Результат умножения, представляющий собой 32-разрядное число с фиксированной точкой, помещается в общий регистр первого операнда. Знаки множителя и множимого определяют знак произведения в соответствии с правилами алгебры. Второй операнд должен располагаться в памяти на границе полуслова.

Признак результата не изменяется.

Команда Divide — D (Деление)

Мнемоника	Код операции	Формат операндов
D	5D	$R_1,D_2(X_2,B_2)$

Содержимое пары общих регистров, номер первого из которых четный, представляющее делимое, делится на 32-разрядное число с фиксированной точкой, указанное вторым операндом. В предложении для этой команды необходимо указывать в качестве первого операнда регистр с четным номером пары общих регистров, содержащих делимое. Само делимое представляет собой 64-битовое число со знаком, занимающее пару этих регистров. После завершения операции деления остаток запоминается в регистре с четным номером, а частное — в регистре с нечетным номером, причем каждое число со знаком занимает 32 бита. Второй операнд должен быть расположен в памяти на границе слова.

Признак результата не изменяется

Команда Divide Registers — DR (Деление)

Мнемоника	Код операции	Формат операндов
DR	1D	R_1,R_2

Содержимое пары общих регистров, представляющее делимое, делится на содержимое регистра второго операнда, представляющее 32-битовое число со знаком. При записи предложения команды необходимо указывать в качестве первого операнда регистр с четным номером из пары общих регистров,

содержащих делимое. Делимое представляется внутри этих регистров в виде 64-битового числа со знаком. После завершения операции деления остаток хранится в регистре с четным номером, а частное — в регистре с нечетным номером, и каждое из этих чисел со знаком занимает 32 бита.

Признак результата не изменяется.

Б. ВЫПОЛНЕНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ С ФИКСИРОВАННОЙ ТОЧКОЙ

Для того чтобы сделать понятным представление чисел с фиксированной точкой, нам следует рассмотреть некоторые общие понятия. Вместе с тем нам важно разобраться в конфигурациях положительных и отрицательных чисел с фиксированной точкой.

Положительное число с фиксированной точкой представляется нулевым значением знакового бита, расположенного в старшем разряде, и прямым двоичным кодом целого числа в остальных позициях. Отрицательное число с фиксированной точкой представляется единичным значением знакового бита, занимающего старший разряд, и значением целого числа в дополнительном коде в остальных позициях. Две таблицы, приведенные в конце книги, таблица значений двоичных разрядов и таблица преобразования шестнадцатеричных чисел в десятичные, помогут разобраться в представлении чисел с фиксированной точкой.

При выполнении арифметических операций с фиксированной точкой используется по крайней мере один общий регистр и в зависимости от конкретного вида команды требуется, чтобы поля данных, содержащие арифметические числа с фиксированной точкой, были расположены в памяти на границе слова или полуслова.

Ниже приведены рисунки, иллюстрирующие двоичное представление для положительных и отрицательных чисел с фиксированной точкой.

1. Слово, содержащее число +33974:

	S							
С фикс. точкой	0000	0000	0000	0000	1000	0100	1011	0110
Шестн.	0	0	0	0	8	4	B	6

2. Полуслово, содержащее число +29878:

	S			
С фикс. точкой	0111	0100	1011	0110
Шестн.	7	4	B	6

3. Полуслово, содержащее число —987:

	S															
С фикс. точкой	1	1	1	1	1	1	0	0	0	0	1	0	1	0	1	1
Шестн.	F				C				2				5			

4. Слово, содержащее число +4593211:

	S																															
С фикс. точкой	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	1	1				
Шестн.	0				0				4				6				1				6				3				B			

5. Слово, содержащее число —3915228:

	S																															
С фикс. точкой	1	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0				
Шестн.	F				F				C				4				4				2				2				4			

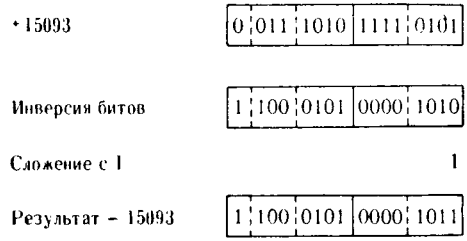
6. Полуслово, содержащее число —1:

	S																															
С фикс. точкой	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1																
Шестн.	F				F				F				F																			

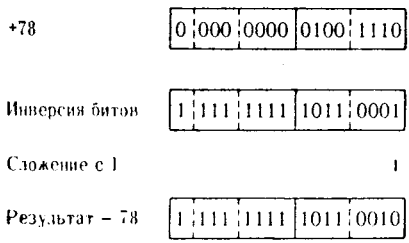
Следует отметить, что представление отрицательных чисел с фиксированной точкой внешне не согласуется с обычным представлением о значащих двоичных цифрах, определяющих величину числа. Если воспользоваться последним примером, где представлено число —1 в дополнительном коде, и сложить соответствующие степени двойки для всех значащих (единичных) битов, исключая знаковый, то мы получим число —32767. Чтобы правильно определить величину отрицательного числа с фиксированной точкой, следует просуммировать двоичные веса значащих единичных битов, а затем вычесть эту сумму из максимального отрицательного числа, которое можно записать в поле данной длины; полуслово имеет максимальную отрицательную величину —32768, а слово — величину —2147483648. Если обратиться к последнему примеру с числом —1, сумма двоичных весов значащих единичных битов с учетом знака дает число —32767. Вычитая это число из максимальной отрицательной величины для полуслова (—32768), получаем правильное отрицательное значение полуслова: —1.

Перевод числа, записанного в прямом двоичном коде (положительное число), в дополнительный двоичный код (отрицательное число) несложен. Содержимое каждого бита инвертируется: единичные биты заменяются нулевыми, а нулевые — единичными. После инвертирования всех битов к младшему разряду (самому правому биту) следует прибавить единицу, и на этом перевод в двоичный дополнительный код завершается. Например:

1. Содержимое полуслова +15093 перевести в —15093:

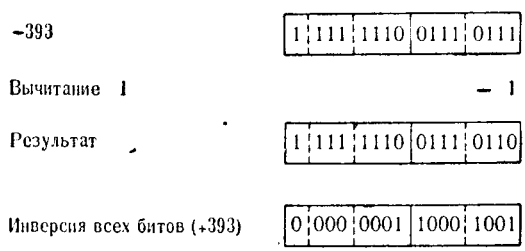


2. Содержимое полуслова +78 перевести в —78:

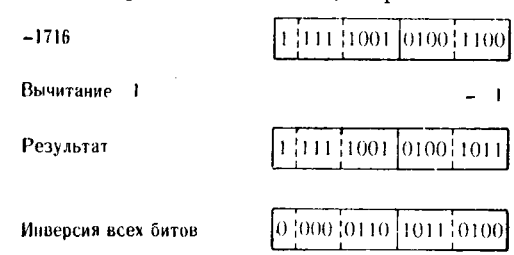


Обратное преобразование производится так же легко. Из двоичного дополнительного кода сначала вычитается единица из младшего разряда, а затем значения всех битов инвертируются. В результате мы переходим от записи числа в двоичном дополнительном коде к записи положительного числа в прямом коде. Например:

1. Содержимое полуслова —393 перевести в +393:



2. Полуслово, содержащее —1716, перевести в +1716



Приведенные пояснения и иллюстрации дают представление о формате чисел с фиксированной точкой, достаточные для понимания команд арифметики с фиксированной точкой.

В последующем тексте этой главы конфигурации областей памяти и общих регистров будут представлены в следующей форме:

0110	0000	1101	0111
6	0	D	7

числа 0 и 1 в верхней строке будут отображать двоичную конфигурацию полей, а символы от 0 до 9 и от A до F в нижней строке будут представлять конфигурацию каждых четырех двоичных позиций в шестнадцатеричном коде.

1. Команды сложения с фиксированной точкой

Сложение — A. К числу с фиксированной точкой, содержащемуся в общем регистре, прибавляется число с фиксированной точкой, содержащееся в области памяти длиной в 4 байта. После выполнения сложения проверяется, не произошло ли переполнение.

Первый операнд в предложении команды должен быть общим регистром или символическим именем, приравненным номеру какого-либо общего регистра. Второй операнд должен представлять слово и размещаться в памяти на границе слова. Второй операнд складывается с первым и сумма помещается в общий регистр первого операнда.

Пример 1.

```
A    6,FULLWRD1
```

	S								
Регистр 6	0000	0000	0000	1001	0000	1111	0110	1001	+593769
(До)	0	0	0	9	0	F	6	9	
FULLWRD1	S								
	0000	0000	0000	0000	0001	0100	1011	0101	+5301
	0	0	0	0	1	4	B	5	
Сумма в	S								
регистре 6	0000	0000	0000	1001	0010	0100	0001	1110	+599070
	0	0	0	9	2	4	1	E	

В этом примере число с фиксированной точкой из области памяти FULLWRD1 (+5301) складывается с числом с

фиксированной точкой из общего регистра 6 и сумма помещается в регистр 6. Шестнадцатеричная конфигурация указанных операндов может быть интерпретирована так:

Регистр 6	+ 9 0 F 6 9	или	+ 593769
FULLWRD1	+ 1 4 B 5	или	+ 5301
Регистр 6 (сумма)	+ 9 2 4 1 E		+ 599070

Правильность выкладок можно проверить с помощью таблицы степеней 2 и таблицы перевода шестнадцатеричных чисел в десятичные.

Пример 2.

A 8,=F'75963'

	S								
Регистр 8 (До)	0000	0000	0000	0000	0000	0000	1111	1111	+255
	0	0	0	0	0	0	F	F	
Литерал F'75963'	S								
	0000	0000	0000	0001	0010	1000	1011	1011	+75 963
	0	0	0	1	2	8	B	B	
Сумма в регистре 8	S								
	0000	0000	0000	0001	0010	1001	1011	1010	+76 218
	0	0	0	1	2	9	B	A	

В этом примере литерал, представленный словом, содержащим число с фиксированной точкой +75963, прибавляется к содержимому общего регистра 8 — числу +255. Здесь удовлетворено требование, чтобы второй операнд, представляющий собой полное слово, располагался на границе слова, так как литерал задан в форме полного слова и транслятор обеспечит необходимое выравнивание.

В шестнадцатеричном представлении числа этого примера складываются так:

Регистр 8	+ F F	или	+ 255
Литерал	+ 1 2 8 B B	или	+ 75963
Регистр 8 (Сумма)	+ 1 2 9 B A	или	+ 76218

Сложение полуслова — AH. К числу с фиксированной точкой, содержащемуся в общем регистре, прибавляется число с фиксированной точкой, являющееся содержимым двухбайтовой области памяти. Первый операнд должен быть общим регистром или

символическим именем, приравненным номеру какого-либо общего регистра. Второй операнд должен представлять собой полуслово и располагаться на границе полуслова. Число, содержащееся в полуслове, извлекается и расширяется до размера полного слова. Это достигается добавлением к числу 16 старших двоичных разрядов, имеющих значение, равное значению знакового бита полуслова. Значение числа с фиксированной точкой в результате такого расширения не изменяется, не изменяется и содержимое полуслова в памяти. Расширенное число затем прибавляется к числу, содержащемуся в регистре первого операнда, и результирующая сумма помещается в этот регистр. При этом проверяется, не возникло ли переполнение.

Пример 1.

```
AH      3,HAFWRD2
```

Перед выполнением команды AH содержимое области имело следующий вид:

	0000	1011	1100	1100	
HAFWRD2	0	B	C	C	+3020

Эта конфигурация расширяется до размера полного слова, так что при сложении операнды и результат выглядят следующим образом:

	S								
HAFWRD2	0000	0000	0000	0000	0000	1011	1100	1100	+3 020
(Расширенное)	0	0	0	0	0	B	C	C	

	S								
Регистр 3	0000	0000	0000	0110	1001	0101	0011	0111	+431 415
(До)	0	0	0	6	9	5	3	7	

	S								
	0000	0000	0000	0110	1010	0001	0000	0011	+434 435
(Сумма)	0	0	0	6	A	1	0	3	

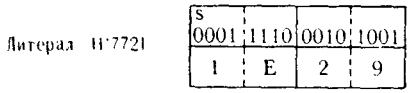
Суммирование чисел этого примера в шестнадцатеричном представлении выглядит так:

HAFWRD2	+			B	C	C		или	+	3020
Регистр 3	+	6	9	5	3	7		или	+	431415
Сумма в регистре 3	+	6	A	1	0	3		или	+	434435

Пример 2. Как и для команды A, в качестве второго операнда можно использовать литерал, если только обеспечено выравнивание по границе полуслова. Например,

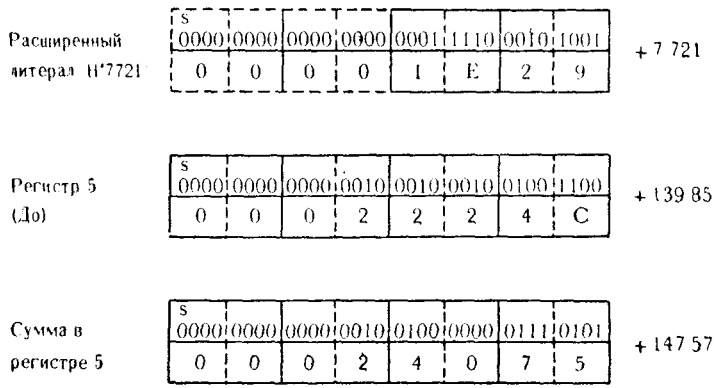
```
AH      5,=H'7721'
```

Транслятор разместит этот литерал на границе полуслова в виде следующей конфигурации:



Значение литерала извлекается и расширяется до полного слова.

Сложение будет происходить следующим образом:



Сложение чисел этого примера в шестнадцатеричном представлении выглядит так:

Литерал H'7721'	+ 1 E 2 9	или	+ 7721
Регистр 5	+ 2 2 2 4 C	или	+ 139852
Сумма в регистре 5	+ 2 4 0 7 5	или	+ 147573

Сложение — AR. Складываются два числа с фиксированной точкой, содержащиеся в двух общих регистрах. Число из регистра первого операнда складывается с числом из регистра второго операнда и сумма помещается в регистр первого операнда. По завершении сложения проверяется, не произошло ли переполнение. Оба операнда должны быть общими регистрами или символическими именами общих регистров.

Пример 1.

Общий регистр 7 содержит число +396
Общий регистр 8 содержит число +4001

Выполняемая команда

```
AR    7,8
```

	S								
Регистр 7	0000	0000	0000	0000	0000	0001	1000	1100	+396
(До)	0	0	0	0	0	1	8	C	

	S								
Регистр 8	0000	0000	0000	0000	0000	1111	1010	0001	+4001
	0	0	0	0	0	F	A	1	

	S								
Сумма в	0000	0000	0000	0000	0001	0001	0010	1101	+4397
регистре 7	0	0	0	0	1	1	2	D	

Сложение чисел этого примера в шестнадцатеричном представлении выглядит так:

Общий регистр 7	+ 1 8 C	или	+ 396
Общий регистр 8	+ F A 1	или	+ 4001
Сумма в регистре 7	+ 1 1 2 D	или	+ 4397

Находящуюся в общем регистре величину можно удвоить, складывая содержимое этого регистра с самим собой, как показано в следующем примере.

Пример 2. Выполняемая команда:

```
AR    8,8
```

Будут иметь место следующие действия:

	S								
Регистр 8	0000	0000	0000	0000	0000	1111	1010	0001	+4001
(1-й операнд)	0	0	0	0	0	F	A	1	

	S								
Регистр 8	0000	0000	0000	0000	0000	1111	1010	0001	+4001
(2-й операнд)	0	0	0	0	0	F	A	1	

	S								
Сумма в	0000	0000	0000	0000	0001	1111	0100	0010	+8002
регистре 8	0	0	0	0	1	F	4	2	

Число с фиксированной точкой в общем регистре 8 теперь удвоилось.

Вполне возможно, что программист непреднамеренно создаст ситуацию переполнения. Следующий пример показывает, как это может случиться при использовании команды AR и какой будет результирующая конфигурация.

Пример 3. Выполняется команда

AR 10,11

Число в регистре 10: + 1094788867.
Число в регистре 11: + 1057013981.

	S								
Регистр 10	0100	0001	0100	0001	0010	0111	0000	0011	+ 1 094 788 867
(До)	4	1	4	1	2	7	0	3	

	S								
Регистр 11	0011	1111	0000	0000	1100	0000	1101	1101	+ 1 057 013 981
	3	F	0	0	C	0	D	D	

	S								
Сумма в	1000	0000	0100	0001	1110	0111	1110	0000	
регистре 10	8	0	4	1	E	7	E	0	

Заметим, что в общем регистре 10 должно бы быть число +2151802848, однако значение знакового бита в позиции старшего разряда стало равным единице, что соответствует знаку минус. Поскольку сумма оказалась больше, чем максимально возможное положительное число с фиксированной точкой, помещающееся в общем регистре, переполнение цифровых битов изменило значение бита знака. Это возбуждает сигнал переполнения с фиксированной точкой, значение признака результата устанавливается соответствующим образом, и возникнет прерывание по переполнению с фиксированной точкой.

Сложение кодов — AL. Команда AL подобна команде A, за исключением того, что данные, содержащиеся в операндах, необязательно должны представлять собой правильные числа с фиксированной точкой. Двоичная конфигурация, содержащаяся в общем регистре первого операнда, складывается с двоичной конфигурацией, определенной вторым операндом. Все 32 бита операндов участвуют в сложении без учета особой роли знако-

вого бита, и сумма помещается в регистр первого операнда. Второй операнд обязательно должен представлять четырехбайтовую область памяти, размещенную на границе слова. В отличие от команды A в команде AL переполнение с фиксированной точкой не вызывает программного прерывания. Значение признака результата, устанавливаемое этой командой, индицирует перенос знакового бита, если он возникнет, а не переполнение с фиксированной точкой.

Следующий пример иллюстрирует выполнение команды AL для случая, когда знаковый бит в результате сложения не изменяется.

Пример 1. Выполняемая команда:

```
AL      7,FULLWD9
```

	S								
Регистр 7 (До)	0000	0000	0000	1001	0001	1110	0110	0000	+597 060
	0	0	0	9	1	E	6	0	
FULLWD9	S								
	0000	0000	0000	0000	0000	0000	1111	1111	+255
	0	0	0	0	0	0	F	F	
Сумма в регистре 7	S								
	0000	0000	0000	1001	0001	1111	0101	1111	+597 315
	0	0	0	9	1	F	5	F	

Сложение чисел в шестнадцатеричном коде выглядит следующим образом:

Общий регистр 7	+ 9 1 E 6 0	или	+ 597060
FULLWD9	+ F F	или	+ 255
Сумма в регистре 7	+ 9 1 F 5 F	или	+ 597315

В результате выполнения команды установится значение признака результата, указывающее, что сумма не равна нулю и переноса из знакового бита не произошло.

Пример 2. Этот пример показывает выполнение команды AL в ситуации, когда возникает перенос из старшего разряда целого числа, изменяющий значение знакового бита суммы. Выполняется команда:

```
AL      11,ONEFWRD
```

	S								
Регистр 11	0111	0001	0000	0000	0011	1100	0000	1111	+1 895 840 783
(До)	7	1	0	0	3	C	0	F	

	S								
ONEFWRD	0000	1111	0011	0111	0001	0001	1001	0000	+255 267 216
	0	F	3	7	1	1	9	0	

	S								
Регистр 11	1000	0000	0011	0111	0100	1101	1001	1111	−2 143 859 297
(Сумма кодов)	8	0	3	7	4	D	9	F	

В результате сложения знаковый бит приобрел новое значение, так как произошел перенос старшего разряда числа в позицию бита знака.

Поэтому в регистре 11 оказывается число, имеющее в формате с фиксированной точкой значение —2143859297. После выполнения команды признак результата укажет, что сумма не равна нулю и переноса из знакового бита не было.

Сложение кодов — ALR. Команда ALR имеет такое же сходство с командой AR, как команда AL с командой A. Двоичная конфигурация данных, содержащихся в регистре второго операнда, складывается с двоичной конфигурацией данных, содержащихся в регистре первого операнда. Команда не вызывает прерывания в случае возникновения ситуации, обычно рассматриваемой как переполнение с фиксированной точкой. Появление переноса знакового бита отражается установкой соответствующего значения признака результата. Сумма помещается в общий регистр первого операнда. Оба операнда должны быть общими регистрами или символическими именами общих регистров.

Пример 1. Выполняемая команда:

```
ALR     2,7
```

Общий регистр 2 содержит число —3596. Общий регистр 7 содержит число —398541.

	S								
Регистр 2	1111	1111	1111	1111	1111	0001	1111	0100	−3 596
(До)	F	F	F	F	F	1	F	4	

	S								
Регистр 7	1111	1111	1111	1001	1110	1011	0011	0011	−398 541
	F	F	F	9	E	B	3	3	

Сумма в регистре 2

1111	1111	1111	1001	1101	1101	0010	0111
F	F	F	9	D	D	2	7

−402 137

Заметим, что двоичная конфигурация в регистрах представляет отрицательные числа в дополнительном коде. В этом легко убедиться, применив для проверки рассмотренную в этой главе методику.

Обзор команд сложения с фиксированной точкой. Хотя пять рассмотренных команд сложения имеют некоторые индивидуальные отличия, в основном выполняемые ими действия весьма близки. Следующий пример, использующий все пять команд, дает материал для их сравнения. Программа для этого примера приведена на рис. 11.1.

Предложение 001 вычитает содержимое общего регистра 10 из самого себя с помощью команды SR. В результате содержимое регистра 10 становится равным нулю.

Регистр 10

S							
0000	0000	0000	0000	0000	0000	0000	0000
0	0	0	0	0	0	0	0

+0

Предложение 002 загружает в регистр 7 содержимое области памяти FULLWD4, после чего регистр содержит число с фиксированной точкой +9000.

Регистр 7

S							
0000	0000	0000	0000	0010	0011	0010	1000
0	0	0	0	2	3	2	8

+9000

Предложение 003 складывает содержимое общих регистров 7 (+9000) и 10 (+0) с помощью команды AR. Теперь регистр 10 содержит число +9000, что видно из следующих пояснений:

Регистр 10 (До)

S							
0000	0000	0000	0000	0000	0000	0000	0000
0	0	0	0	0	0	0	0

+0

Регистр 7

S							
0000	0000	0000	0000	0010	0011	0010	1000
0	0	0	0	2	3	2	8

+9000

Регистр 10 (После)

S							
0000	0000	0000	0000	0010	0011	0010	1000
0	0	0	0	2	3	2	8

+9000

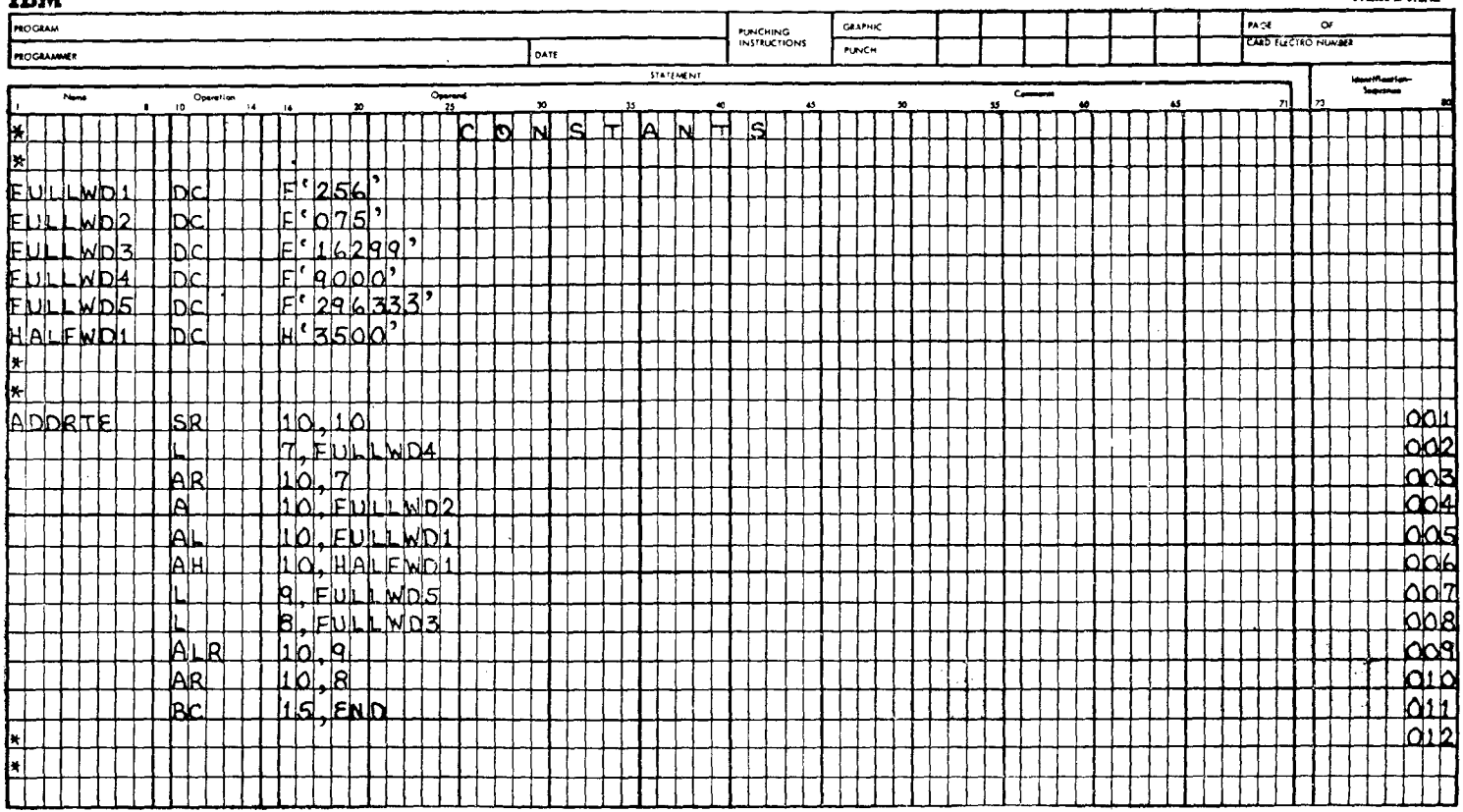

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
CONSTANTS
*
*
FULLWD1 DC F'256'
FULLWD2 DC F'075'
FULLWD3 DC F'16299'
FULLWD4 DC F'9000'
FULLWD5 DC F'296333'
HALFWD1 DC H'3500'
*
*
ADDRTE SR 10,10 001
L 7,FULLWD4 002
AR 10,7 003
A 10,FULLWD2 004
AL 10,FULLWD1 005
AH 10,HALFWD1 006
L 9,FULLWD5 007
L 8,FULLWD3 008
ALR 10,9 009
AR 10,8 010
BC 15,END 011
* 012
*

Рис. 11.1.

Предложение 004 использует команду A, чтобы прибавить содержимое FULLWD2 (+75) к содержимому общего регистра 10 (+9000).

Теперь регистр 10 содержит число +9075.

	S								
Регистр 10	0000	0000	0000	0000	0010	0011	0010	1000	+9000
(До)	0	0	0	0	2	3	2	8	
	S								
FULLWD2	0000	0000	0000	0000	0000	0000	0100	1011	+75
	0	0	0	0	0	0	4	B	
	S								
Регистр 10	0000	0000	0000	0000	0010	0011	0111	0011	+9075
(Сумма)	0	0	0	0	2	3	7	3	

Предложение 005 прибавляет содержимое FULLWD1 (+256) к содержимому регистра 10 (+9075), используя команду AL. Поскольку складываемые числа не так велики, чтобы повлиять на бит знака, эта команда работает так же, как обычная команда A. Общий регистр 10 содержит теперь число +9331.

	S								
Регистр 10	0000	0000	0000	0000	0010	0011	0111	0011	+9075
(До)	0	0	0	0	2	3	7	3	
	S								
FULLWD1	0000	0000	0000	0000	0000	0001	0000	0000	+256
	0	0	0	0	0	1	0	0	
	S								
Сумма в	0000	0000	0000	0000	0010	0100	0111	0011	+9331
регистре 10	0	0	0	0	2	4	7	3	

Предложение 006 использует команду AH для того, чтобы прибавить к содержимому регистра 10 (+9331) содержимое HALFWD1 (+3500). Теперь регистр 10 содержит число +12831.

Регистр 10 (До)

S								
0000	0000	0000	0000	0010	0100	0111	0011	+9331
0	0	0	0	2	4	7	3	

HALFWD1

S								
0000	0000	0000	0000	0000	1101	1010	1100	+3500
0	0	0	0	0	D	A	C	

Сумма в регистре 10

S								
0000	0000	0000	0000	0011	0010	0001	1111	+12 831
0	0	0	0	3	2	1	F	

Предложение 007 загружает регистр 9 содержимым области FULLWD5 (+296333).

Регистр 9

S								
0000	0000	0000	0100	1000	0101	1000	1101	+296 333
0	0	0	4	8	5	8	D	

Предложение 008 загружает общий регистр 8 содержимым FULLWD3 (+16299).

Регистр 8

S								
0000	0000	0000	0000	0011	1111	1010	1011	+16 299
0	0	0	0	3	F	A	B	

Предложение 009 прибавляет к содержимому общего регистра 10 (+12831) содержимое общего регистра 9 (+296333), используя для этого команду ALR. Регистр 10 теперь содержит число с фиксированной точкой +309164.

Регистр 10 (До)

S								
0000	0000	0000	0000	0011	0010	0001	1111	+12 831
0	0	0	0	3	2	1	F	

Регистр 9

S								
0000	0000	0000	0100	1000	0101	1000	1101	+296 333
0	0	0	4	8	5	8	D	

Регистр 10 (Сумма)

S								
0000	0000	0000	0100	1011	0111	1010	1100	+309 164
0	0	0	4	B	7	A	C	

Предложение 010 вновь использует команду AR, на этот раз для того, чтобы сложить содержимое общего регистра 10 (+309164) с содержимым общего регистра 8 (+16299). Общий регистр 10 теперь содержит число +325463.

	S								
Регистр 10	0000	0000	0000	0100	1011	0111	1010	1100	+309 164
(До)	0	0	0	4	B	7	A	C	

	S								
Регистр 8	0000	0000	0000	0000	0011	1111	1010	1011	+16 299
	0	0	0	0	3	F	A	B	

	S								
Сумма в	0000	0000	0000	0100	1111	0111	0101	0111	+325 463
регистре 10	0	0	0	4	F	7	5	7	

Предложение 011 — команда безусловного перехода к подпрограмме END.

Хотя этот пример демонстрирует применение различных команд сложения чисел с фиксированной точкой, здесь показан не самый эффективный путь достижения намеченной цели. Этого же можно добиться различными способами, зачастую более эффективными с точки зрения количества команд и скорости выполнения. Вот пример более выгодной кодировки:

```
ADDRTE   LM    6,10,FULLWD1     001
         AR    6,7              002
         AR    6,8              003
         AR    6,9              004
         AR    10,6             005
         AH    10,HALFWD1       006
         BC    15,END           007
```

Здесь предложение 001 загружает содержимое FULLWD1 в общий регистр 6, содержимое FULLWD2 в общий регистр 7, содержимое FULLWD3 в общий регистр 8, содержимое FULLWD4 в общий регистр 9 и содержимое FULLWD5 в общий регистр 10. Назначение остальных предложений вполне очевидно. Ясно также, что окончательный результат вычислений будет по-прежнему находиться в общем регистре 10.

2. Команды вычитания с фиксированной точкой

Вычитание — S. Число, содержащееся во втором операнде, вычитается из числа, содержащегося в регистре первого

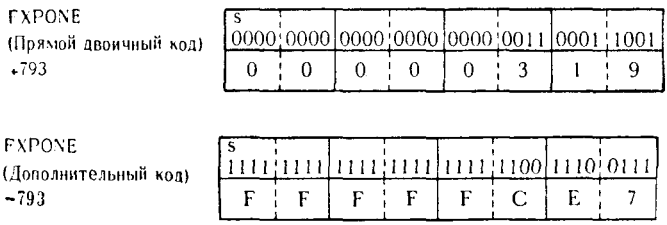
операнда. Второй операнд длиной в слово должен размещаться на границе полного слова. Вычитание производится путем сложения числа с фиксированной точкой первого операнда с дополнением до 2 числа с фиксированной точкой второго операнда.

Результирующая сумма такой операции, представляющая собой разность между первоначальными представлениями чисел, помещается в общий регистр, указанный первым операндом. При выполнении команды S проверяется, не произошло ли переполнение. В случае переполнения признак результата устанавливается соответствующим образом, и, как правило, происходит программное прерывание.

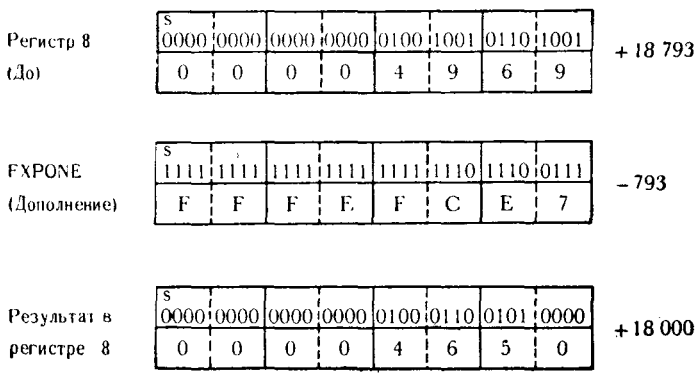
Чтобы проиллюстрировать это действие, предположим, что общий регистр 8 содержит число с фиксированной точкой +18793, а слово FXPONE содержит число с фиксированной точкой +793. Выполняется команда

```
S       8,FXPONE
```

В соответствии с правилами вычитания содержимое FXPONE преобразуется в дополнительный код. Это преобразование выглядит так:



Операция завершается сложением содержимого общего регистра 8 с дополнением до 2 содержимого FXPONE. Сложение выглядит следующим образом:



Если из числа с фиксированной точкой вычитается отрицательное число с фиксированной точкой, второй операнд преобразуется вновь в свое прямое двоичное представление (дополнение до 2 отрицательной конфигурации) и затем складывается со значением первого операнда. Чтобы продемонстрировать эти действия, предположим, что общий регистр 5 содержит число —695236 и полное слово памяти FXPTWO содержит число —38523. Выполняется команда

```
S     5,FXPTWO
```

Содержимое FXPTWO сначала преобразуется следующим образом:

FXPTWO (До)	s							
	1111	1111	1111	1111	0110	1001	1000	0101
−38523	F	F	F	F	6	9	8	5

FXPTWO (Дополнение)	s							
	0000	0000	0000	0000	1001	0110	0111	1011
+38523	0	0	0	0	9	6	7	B

Вычитание, заданное командой, выполняется путем сложения содержимого FXPTWO с числом, хранящимся в общем регистре 5.

Регистр 5 (До)	s								−659 236
	1111	1111	1111	0101	1111	0000	1101	1010	
	F	F	F	5	F	0	D	A	

FXPTWO	s								+38 523
	0000	0000	0000	0000	1001	0110	0111	1011	
	0	0	0	0	9	6	7	B	

Результат в регистре 5	s								−620 713
	1111	1111	1111	0110	1000	0111	0101	0101	
	F	F	F	6	8	7	5	5	

Если в команде задано вычитание из положительного числа, содержащегося в регистре, отрицательного числа, содержащегося в основной памяти, то отрицательное число будет преобразовано из дополнительного кода в прямой и сложению подвергнутся два положительных числа.

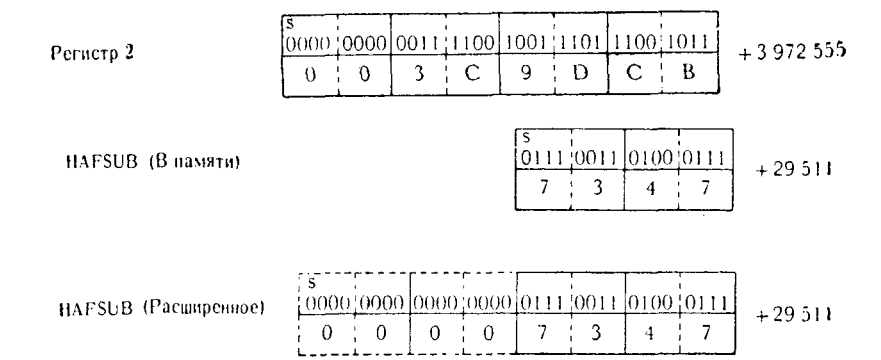
Вычитание полуслова SH. Из числа, содержащегося в общем регистре, вычитается число, содержащееся в полуслове памяти. Первым операндом должен быть общий регистр или

символическое имя, присвоенное общему регистру. Второй операнд должен представлять полуслово — двухбайтовую область памяти, размещенную на границе полуслова. Сначала содержимое полуслова извлекается и расширяется до размера полного слова путем распространения значения знакового разряда на дополнительные 16 старших двоичных разрядов полуслова. Такое расширение не изменяет значение числа с фиксированной точкой, не изменяется также и содержимое самого полуслова. Вычитание выполняется затем с помощью сложения содержимого регистра и двоичного дополнения расширенного значения, взятого из полуслова. Результат этой операции помещается в общий регистр первого операнда. При выполнении команды проверяется, не произошло ли переполнение. Если произошло, устанавливается соответствующее значение признака результата и возникает запрос на программное прерывание.

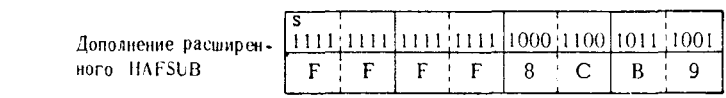
В качестве примера рассмотрим команду:

```
SH      2,HAFSUB
```

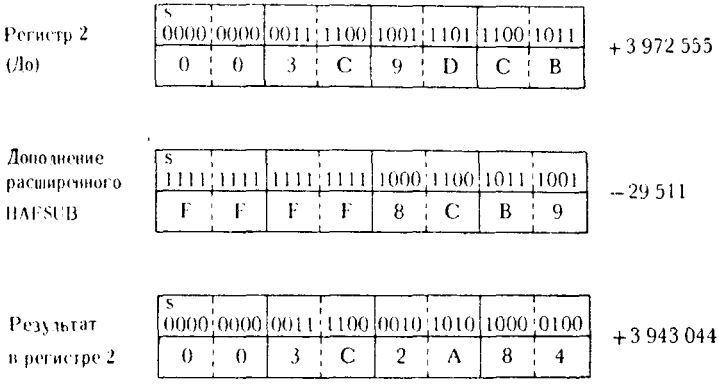
Предположим, что общий регистр 2 содержит число с фиксированной точкой +3972555, а полуслово, адресуемое меткой HAFSUB, содержит число с фиксированной точкой +29511. Битовые конфигурации общего регистра 2, полуслова HAFSUB и расширенного значения HAFSUB выглядят следующим образом:



Дополнение до 2 расширенного значения HAFSUB, формируемое перед выполнением командой SH арифметической операции, выглядит следующим образом:

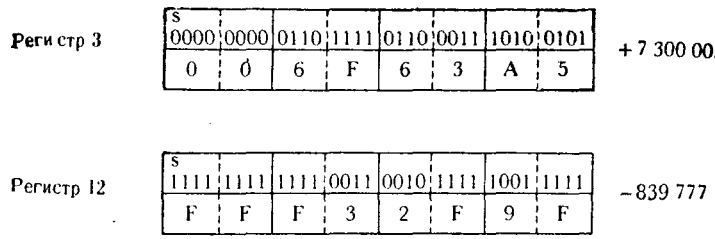


Процесс вычитания заканчивается сложением числа, содержащегося в общем регистре 2, с дополнительным кодом расширенного значения HAFSUB. Это выглядит так:



Вычитание SR. Содержимое регистра второго операнда, рассматриваемое как число с фиксированной точкой, вычитается из числа с фиксированной точкой, содержащегося в регистре первого операнда. Как и при выполнении команд S и SH, значение числа с фиксированной точкой, содержащегося во втором операнде, преобразуется в дополнительный код и складывается с числом, хранящимся в регистре первого операнда. Этот процесс дает в результате разность двух операндов, которая помещается в регистр первого операнда. При возникновении переполнения с фиксированной точкой устанавливается соответствующее значение признака результата и происходит программное прерывание[1]). Первоначальное содержимое регистра второго операнда всегда инвертируется. Положительное число преобразуется в отрицательное число в дополнительном коде; отрицательное число преобразуется в дополнение до 2, т. е. в положительное число в прямом коде.

Для иллюстрации предположим, что общий регистр 3 содержит число +7300005, а общий регистр 12 содержит число —839777. Двоичная конфигурация содержимого этих регистров такова:



[1]) Как и при выполнении других команд, программное прерывание по причине переполнения с фиксированной точкой не происходит, если соответствующий бит маски программы в PSW установлен в нуль. — *Прим. ред.*

Выполняется команда:

```
SR      3,12
```

Это предложение сначала инвертирует отрицательное значение числа в общем регистре 12 таким образом:

	S								
Регистр 12	0000	0000	0000	1100	1101	0000	0110	0001	+839 777
(Дополнение)	0	0	0	C	D	0	6	1	

Затем складываются содержимое регистра 12 в дополнительном коде и содержимое регистра 3, и сумма помещается в общий регистр 3. Это можно представить так:

	S								
Регистр 3	0000	0000	0110	1111	0110	0011	1010	0101	+7 300 005
(До)	0	0	6	F	6	3	A	5	

	S								
Регистр 12	0000	0000	0000	1100	1101	0000	0110	0001	+839 777
	0	0	0	C	D	0	6	1	

	S								
Результат в	0000	0000	0111	1100	0011	0100	0000	0110	+8 139 782
регистре 3	0	0	7	C	3	4	0	6	

Команда SR часто используется для очистки общего регистра. При вычитании содержимого регистра из самого себя все 32 бита этого регистра устанавливаются равными нулю. В качестве примера рассмотрим команду

```
SR      11,11
```

Выполнение этой команды выглядит следующим образом:

	S								
Регистр 11	0000	0000	0100	0000	0111	0110	0101	1011	+4 224 603
(1-й операнд)	0	0	4	0	7	6	5	B	

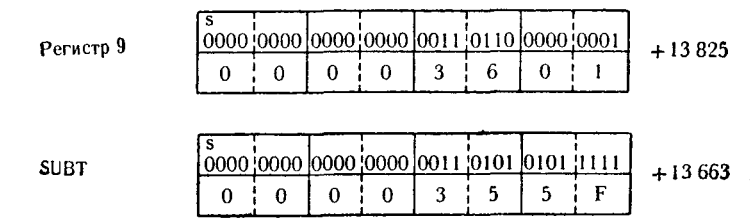
	S								
Регистр 11	1111	1111	1011	1111	1000	1001	1010	0101	−4 224 603
(Дополнение	F	F	B	F	8	9	A	5	
2-й операнд)									

	S								
Сумма в	0000	0000	0000	0000	0000	0000	0000	0000	+0
регистре 11	0	0	0	0	0	0	0	0	

Так как некоторые команды затрагивают лишь часть содержимого общего регистра, только что приведенное использование команды SH с целью очистки регистра гарантирует, что старшие разряды содержат нули. Например, при использовании команды IC, изменяющей только младший байт регистра, программисту может понадобиться поместить в регистр 1 байт данных для последующего использования значения, задаваемого содержимым этого байта как двоичным числом без знака. Если старшие биты или байты регистра не установлены в нуль, то значение, хранящееся в регистре, не соответствует значению, записанному в младший байт. Такую ситуацию можно предотвратить, выполнив сначала команду SR с указанием в обоих операндах этого регистра.

Вычитание кодов — SL. Эта команда выполняется почти так же, как и команда S. Двоичная конфигурация второго операнда вычитается из двоичной конфигурации регистра первого операнда. Вычитание выполняется путем преобразования двоичной конфигурации второго операнда в дополнительный код и сложения с двоичной конфигурацией регистра первого операнда. Результат помещается в регистр первого операнда. Второй операнд должен представлять четырехбайтовую область памяти, размещенную на границе полного слова. Команда SL отличается от команды S тем, что в операции вычитания участвуют все 32 бита первого операнда и 32 бита второго операнда; она не вызывает программного прерывания, если произойдет переполнение с фиксированной точкой. Признак результата не фиксирует переполнение; его значение указывает на перенос из знакового разряда, если таковой имел место. Вообще говоря, если выполнение команды SL не вызвало переноса в знаковый бит или из него, то ее действие в точности совпадает с действием команды S. Единственное различие в этом случае заключается в интерпретации значения признака результата.

Предположим, что общий регистр 9 содержит число с фиксированной точкой +13825; полное слово, адресуемое меткой SUBT, содержит число с фиксированной точкой +13663. Содержимое регистра и слова памяти будет следующим:



Выполняется команда

```
SL     9,SUBT
```

Первое действие команды состоит в преобразовании значения второго операнда в дополнение до двух. Это выглядит так:

	S								
SUBT	1111	1111	1111	1111	1100	1010	1010	0001	−13 663
(Дополнение)	F	F	F	F	C	A	A	1	

Затем производится сложение двоичной конфигурации дополнения SUBT с содержимым общего регистра 9, и сумма помещается в регистр 9. Это действие изменяет поля операндов следующим образом:

	S								
Регистр 9	0000	0000	0000	0000	0011	0110	0000	0001	+13 825
(До)	0	0	0	0	3	6	0	1	
SUBT	1111	1111	1111	1111	1100	1010	1010	0001	−13 663
(Дополнение)	F	F	F	F	C	A	A	1	
Результат в	0000	0000	0000	0000	0000	0000	1010	0010	+162
регистре 9	0	0	0	0	0	0	A	2	

В этом примере признак результата получит значение, указывающее, что разность в общем регистре 9 не равна нулю и перенос из знакового бита имел место.

Вычитание кодов — SLR. Содержимое регистра второго операнда вычитается из содержимого регистра первого операнда. Вычитание выполняется таким же способом, как и для всех команд вычитания, описанных в этой главе: второй операнд преобразуется в свое дополнение до 2 и складывается с первым операндом.

Сумма помещается в регистр первого операнда. Очевидно, что оба операнда должны указывать общие регистры или быть символическими именами общих регистров. Команда SLR не вызывает программного прерывания, если произойдет переполнение с фиксированной точкой. Принятая для операций над кодами интерпретация признака результата позволяет судить о том, произошел или нет перенос из знакового бита.

Для иллюстрации работы этой команды предположим, что общий регистр 9 содержит число с фиксированной точкой +2147863600, а общий регистр 12 содержит число с фиксиро-

ванной точкой —486. Содержимое этих регистров выглядит следующим образом:

	S								
Регистр 9	0111	1111	1111	1111	1111	1111	1101	0000	+2 147 483 600
	7	F	F	F	F	F	D	0	
Регистр 12	S								
	1111	1111	1111	1111	1111	1110	0001	1010	−486
	F	F	F	F	F	E	1	A	

Выполняется команда:

SLR 9,12

Как и при выполнении остальных команд вычитания с фиксированной точкой, первое действие, выполняемое командой SLR, состоит в преобразовании второго операнда в форму дополнения до 2. После этого регистр 12 имеет следующий вид:

	S								
Регистр 12	0000	0000	0000	0000	0000	0001	1110	0110	+486
(Дополнение)	0	0	0	0	0	1	E	6	

Теперь остается сложить двоичную конфигурацию содержимого регистра 9 с дополнением двоичной конфигурации содержимого регистра 12. Это выглядит следующим образом:

	S								
Регистр 9	0111	1111	1111	1111	1111	1111	1101	0000	+2 147 483 600
(До)	7	F	F	F	F	F	D	0	
Регистр 12	S								
	0000	0000	0000	0000	0000	0001	1110	0110	+486
(Дополнение)	0	0	0	0	0	1	E	6	
Результат в	S								
	1000	0000	0000	0000	0000	0001	1011	0110	−2 147 483 210
регистре 9	8	0	0	0	0	1	B	6	

Обратим внимание, что в результате получилась не та величина, которую следует ожидать в случае обычного вычитания. При выполнении команды произошел перенос в знаковый бит, изменивший знак результата. В этом случае значение признака результата будет указывать, что разность не равна нулю и перенос из знакового бита не имел места.

Обзор команд вычитания с фиксированной точкой. Рассмотрев действие и способы использования пяти команд вычитания с фиксированной точкой, перейдем к разбору программы, включающей эти команды. Это поможет увидеть сходные черты и различия этих команд. Алгоритм, описываемый этой программой, вряд ли имеет практический смысл и приведен только в качестве учебного примера (рис. 11.2).

Предложение 001 — команда LM. Она загружает содержимое области LESS2 (+376) в общий регистр 7 и содержимое области LESS3 (—49377) в общий регистр 8. После выполнения этой команды регистры 7 и 8 будут иметь следующий вид:

Регистр 7

S								
0000	0000	0000	0000	0000	0001	0111	1000	+376
0	0	0	0	0	1	7	8	

Регистр 8

S								
1111	1111	1111	1111	0011	1111	0001	1111	−49 377
F	F	F	F	3	F	1	F	

Предложение 002 — команда SLR, с помощью которой из числа, содержащегося в регистре 7 (+376), вычитается число, содержащееся в регистре 8 (—49377). Результат вычитания (+49753) помещается в общий регистр 7. Содержимое общего регистра 8 сначала преобразуется в форму дополнения до 2 своего первоначального значения, а затем складывается с содержимым общего регистра 7. Содержимое этих регистров во время выполнения команды, когда в общем регистре 8 находится дополнение до 2 его прежнего содержимого, будет следующим:

Регистр 7 (До)

S								
0000	0000	0000	0000	0000	0001	0111	1000	+376
0	0	0	0	0	1	7	8	

Регистр 8 (Дополнение)

S								
0000	0000	0000	0000	1100	0000	1110	0001	+49 377
0	0	0	0	C	0	E	1	

Результат в регистре 7

S								
0000	0000	0000	0000	1100	0010	0101	1001	+49 753
0	0	0	0	C	2	5	9	

IBM System/360 Assembler Coding Form

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
LESS1	DC	F'2590'		
LESS2	DC	F'376'		
LESS3	DC	F'-49377'		
HAFSUB	DC	H'-747'		
LOGWRD	DC	F'20000'		
WORD	DC	F'32596'		
*				
*				
SUBTEX	LM	7,8,LESS2		001
	SLR	7,8		002
	SR	3,3		003
	L	3,WORD		004
	S	3,LESS1		005
	SR	3,7		006
	SH	3,HAFSUB		007
	SL	3,LOGWRD		008
	BC	15,FINSUBT		009
*				010

Рис. 11.2.

Предложение 003 — команда SR. Содержимое общего регистра 3 вычитается из самого себя. После выполнения этого предложения регистр 3 содержит число с фиксированной точкой +0.

Предложение 004 загружает общий регистр 3 содержимым слова WORD, представляющим число +32596 в формате с фиксированной точкой. После выполнения этого предложения содержимое общего регистра 3 будет таким:

	S								
Регистр 3	0000	0000	0000	0000	0111	1111	0101	0100	+32 596
	0	0	0	0	7	F	5	4	

Предложение 005 вычитает содержимое слова с меткой LESS1 (+2590) из содержимого общего регистра 3 (+32596). Величина LESS1 сначала преобразуется в дополнение до 2, после чего имеет вид

	S								
LESS1	1111	1111	1111	1111	1111	0101	1110	0010	−2 590
(Дополнение)	F	F	F	F	F	5	E	2	

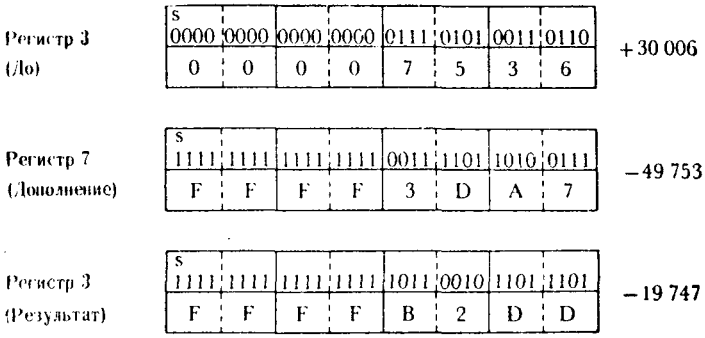
Эта величина затем прибавляется к содержимому общего регистра 3. Содержимое регистра 3 до и после выполнения команды показано ниже:

	S								
Регистр 3	0000	0000	0000	0000	0111	1111	0101	0100	+32 596
(До)	0	0	0	0	7	F	5	4	

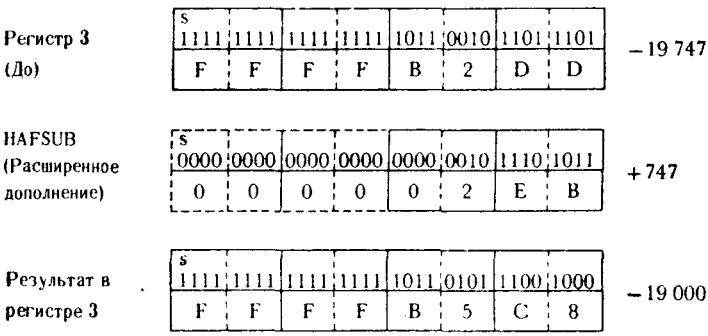
	S								
LESS1	1111	1111	1111	1111	1111	0101	1110	0010	−2 590
(Дополнение)	F	F	F	F	F	5	E	2	

	S								
Результат в	0000	0000	0000	0000	0111	0101	0011	0110	+30 006
регистре 3	0	0	0	0	7	5	3	6	

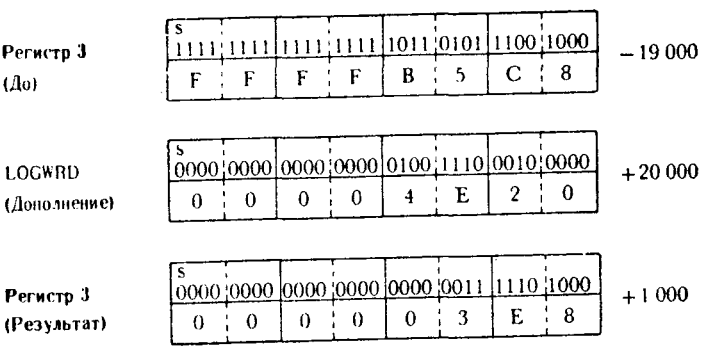
Предложение 006 — команда SR. Число, содержащееся в общем регистре 7 (+49753), вычитается из числа с фиксированной точкой, содержащегося в общем регистре 3. Ниже показано содержимое общего регистра 3 до и после выполнения этой команды, а также конфигурация дополнения общего регистра 7, используемая командой SR.



Предложение 007 вычитает из содержимого общего регистра 3 (—19747) число с фиксированной точкой, содержащееся в полуслове HAFSUB. После расширения до полного слова битовая конфигурация HAFSUB преобразуется в дополнение до 2 и затем складывается с числом из общего регистра 3, как показано ниже:



Предложение 008 — команда SL. Число с фиксированной точкой, содержащееся в слове LOGWRD (—20000), вычитается из числа с фиксированной точкой, содержащегося в общем регистре 3 (—19000). Изменение содержимого общего регистра 3 при сложении с дополнением содержимого слова LOGWRD будет таким:



Предложение 009 является командой безусловного перехода к подпрограмме FINSUBT.

После выполнения всех команд, описанных в этом примере, в регистре 3 будет содержаться число с фиксированной точкой, равное +1000. Если заданные значения констант не будут изменяться перед каждым выходом в эту подпрограмму, то каждый раз будет получаться один и тот же результат, что делает повторение вычислений бессмысленным.

3. Команды умножения с фиксированной точкой

Умножение — M. Для выполнения команды M требуется пара соседних общих регистров, номер первого из которых должен быть четным, используемых в качестве рабочей области при выполнении команды, а также для запоминания результатов умножения. Программист должен загрузить в регистр с нечетным номером число, являющееся множимым. При выполнении команды M число с фиксированной точкой, содержащееся в нечетном регистре пары, умножается на множитель, число с фиксированной точкой, содержащееся в полном слове памяти. Произведение занимает пару регистров в виде 64-разрядного двоичного числа со знаком (1 знаковый бит и 63 бита числа), причем младшие разряды результата занимают младшие позиции регистра с нечетным номером. Номер регистра первого операнда должен быть четным; при несоблюдении этого правила возникнет программное прерывание по неправильной спецификации. Если известно, что произведение достаточно мало и может поместиться в одном общем регистре, то результат умножения можно извлекать из регистра с нечетным номером, в противном случае его следует извлекать из совокупности общих регистров.

Для иллюстрации работы команды и ее влияния на содержимое регистров, используемых в качестве рабочей области, предлагаются следующие примеры.

Пример 1. Числа с фиксированной точкой, подлежащие умножению, находятся в полных словах памяти. Вот их содержимое:

```
FULLWD1    DC    F'32973'
FULLWD2    DC    F'7851'
```

Выполняются команды:

```
L     9,FULLWD1      001
M     8,FULLWD2      002
```

Предложение 001 загружает в общий регистр 9 число с фиксированной точкой, содержащееся в слове FULLWD1. Хотя про-

изведение должно занимать общие регистры 8 и 9, загрузка множимого затрагивает только один регистр, с номером 9. Содержимое регистра 8 на этом этапе неизвестно и непредсказуемо. После выполнения команды загрузки содержимое общих регистров, используемых в примере, примет следующий вид:

Общий регистр 8 Общий регистр 9 (+32973)

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	1000	0000	1100	1101
								0	0	0	0	8	0	C	D

Предложение 002 умножает множимое (предполагается, что это число находится в общем регистре 9) на число с фиксированной точкой из FULLWD2, т. е. множитель. Результат умножения появится в совокупности общих регистров 8 и 9 в виде 64-разрядного двоичного числа со знаком. Конфигурации множимого, множителя и произведения будут такими:

Множимое (Перед выполнением команды M)

Общий регистр 8 Общий регистр 9

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	1000	0000	1100	1101
								0	0	0	0	8	0	C	D

Множитель FULLWD2 (+7851)

S 0000	0000	0000	0000	0001	1110	1010	1011
0	0	0	0	1	E	A	B

Произведение

Общий регистр 8 (+258871023) Общий регистр 9

S 0000	0000	0000	0000	0000	0000	0000	0000	0000	1111	0110	1110	0000	1110	1110	1111
0	0	0	0	0	0	0	0	0	F	6	E	0	E	E	F

В этом примере множимое целиком умещается в регистре с нечетным номером. Поэтому результат умножения (произведение) можно поместить для сохранения в некоторое слово памяти с помощью команды ST, применив ее к регистру 9. Знаковый бит в регистре 9 имеет такое же значение, как и знаковый бит в регистре 8. Это происходит потому, что по завершении действий умножения знаковый бит, значение которого определяется правилами алгебры, распространяется вправо до тех пор, пока не встретится первый значащий разряд произведения. Для

положительного числа первым значащим битом будет единичный бит, для отрицательного числа таким битом явится нулевой бит. Эти правила можно свести к одному: значащим разрядом произведения является бит, противоположный по значению знаковому биту произведения.

Пример 2. Этот пример показывает результат умножения, в котором произведение превосходит длину регистра с нечетным номером. Используются следующие числа и области памяти:

```
MPCAN    DC    F'75968322'
MPLIER   DC    F'59312'
ANSWER   DS    D
```

Ниже приведена последовательность выполняемых команд:

```
L        3,MPCAN        001
M        2,MPLIER       002
STM      2,3,ANSWER     003
```

Предложение 001 загружает общий регистр 3 числом с фиксированной точкой из MPCAN, равным +75968322. Содержимое регистра 3 выглядит теперь следующим образом:

Регистр 3

S								
0000	0100	0111	0111	1110	1101	0000	0010	+75 968 322
0	4	7	7	E	D	0	2	

Предложение 002 умножает число с фиксированной точкой, содержащееся в общем регистре 3, на число с фиксированной точкой из области MPLIER, и результат умножения помещается в совокупность общих регистров 2 и 3. Содержимое регистров, используемых командой M предложения 002, выглядит следующим образом:

Множимое (Перед выполнением команды)

Общий регистр 3 (+75968322)

?	?	?	?	?	?	?	?	S 0000	0100	0111	0111	1110	1101	0000	0010
								0	4	7	7	E	D	0	2

Множитель (MPLIER)

(+59312)

S							
0000	0000	0000	0000	1110	0111	1011	0000
0	0	0	0	E	7	B	0

Произведение

Общий регистр 2 (+45058331144464) Общий регистр 3

S															
0000	0000	0000	0000	0000	0100	0001	1001	0001	1000	1001	0101	0000	1011	0110	0000
0	0	0	0	0	4	1	9	1	8	9	5	0	B	6	0

Заметим, что произведение, получившееся в результате выполнения предложения 002, выходит за рамки общего регистра 3 и занимает еще 2 младших байта общего регистра 2.

Предложение 003 отсылает содержимое регистров 2 и 3 в двойное слово ANSWER. Двойное слово понадобилось для запоминания потому, что значение произведения превышает максимальное число, которое может поместиться в одном регистре.

Умножение — MR. Команда MR подобна команде M, за одним исключением: второй операнд предложения должен указывать общий регистр. Для формирования произведения требуется пара общих регистров, первый из которых должен иметь четный номер. Обычно множимое должно быть загружено в общий регистр с нечетным номером из пары регистров, а множитель — в общий регистр второго операнда. Исключение из этого правила будет обсуждено при дальнейшем рассмотрении. При выполнении команды MR число с фиксированной точкой, содержащееся в регистре с нечетным номером, умножается на число с фиксированной точкой, содержащееся в регистре второго операнда. Произведение помещается в паре общих регистров в виде 64-битового числа со знаком: 1 бит знаковый и 63 бита числа. Младшие разряды произведения занимают младшие позиции нечетного общего регистра. Первый операнд должен всегда указывать регистр с четным номером из пары регистров, иначе произойдет программное прерывание по неправильной спецификации. Если известно, что величина произведения достаточно мала, чтобы поместиться в одном регистре, то можно использовать в качестве результата содержимое одного нечетного регистра; в противном случае следует пользоваться содержимым пары регистров.

Пример 1. Полные слова памяти содержат числа с фиксированной точкой

```
POINTA   DC    F'3276'
POINTB   DC    F'991'
ANSR     DC    F'0'
```

Выполнению подлежат команды

```
LM      5,6,POINTA      001
MR      4,6             002
ST      5,ANSR          003
```

Предложение 001 представляет собой команду LM, которая загружает слово POINTA в общий регистр 5, а слово POINTB в регистр 6. Регистр 5, являясь нечетным регистром пары регистров, содержит теперь множимое. Значение, содержащееся в регистре 6, в этом примере рассматривается как множитель. На этой стадии содержимое трех участвующих в операции регистров будет иметь следующий вид:

Общий регистр 4 Общий регистр 5 (+3276)

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	0000	1100	1100	1100
								0	0	0	0	0	C	C	C

Общий регистр 6 (+991)

S 0000	0000	0000	0000	0000	0011	1101	1111
0	0	0	0	0	3	D	F

Как показано на рисунке, общий регистр 4 до выполнения команды MR содержит непредсказуемые данные.

Предложение 002 содержит команду MR. Число с фиксированной точкой из общего регистра 6 умножается на число, содержащееся в общем регистре 5. Произведение занимает совокупность общих регистров 4 и 5. Конфигурация произведения такова:

Общий регистр 4 Общий регистр 5

S 0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0011	0001	1000	1001	1011	0100
0	0	0	0	0	0	0	0	0	0	3	1	8	9	B	4

(+3 246 516)

Предложение 003 записывает число, содержащееся в общем регистре 5, в полное слово памяти ANSR. Как предполагается, программисту известно, что результат умножения умещается в одном общем регистре. После выполнения команды ST содержимое слова ANSR будет следующим:

ANSR

S 0000	0000	0011	0001	1000	1001	1011	0100
0	0	3	1	8	9	B	4

+3 246 516

Пример 2. Команду умножения можно использовать для возведения числа в квадрат. Это можно сделать, загрузив число в нечетный регистр пары регистров и выполнив команду MR, в которой в качестве первого операнда указан четный регистр, а в качестве второго операнда — нечетный регистр пары. Предположим, что мы хотим возвести в квадрат число с фиксированной точкой +1006 с помощью следующей программы:

```
SQRIT    DC     F'1006'
*
RTE      L      3,SQRIT     001
         MR     2,3         002
```

Предложение 001 загружает число +1006 в общий регистр 3. Содержимое общих регистров 2 и 3 будет следующим:

Общий регистр 2 Общий регистр 3 (+1006)

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	0000	0011	1110	1110
								0	0	0	0	0	3	E	E

Предложение 002 предписывает умножить содержимое общего регистра 3 (в качестве первого операнда указан общий регистр 2) на содержимое общего регистра 3. В результате число, содержащееся в регистре 3, возводится в квадрат и произведение помещается в пару регистров 2 и 3. После выполнения команды содержимое общих регистров 2 и 3 будет иметь следующий вид:

Общий регистр 2 (+1012036) Общий регистр 3

S 0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1111	0111	0001	0100	0100
0	0	0	0	0	0	0	0	0	0	0	F	7	1	4	4

Пример 3. Используя прием, показанный в примере 2, программист может загрузить множимое и множитель в область произведения и произвести над ними операцию умножения. Предположим, что определены следующие поля и их содержимое:

```
QUANTITY   DC     F'395'
UNITCOST   DC     F'75'
ANSWER     DC     F'0'
```

Выполняются такие команды:

```
LM    6,7,QUANTITY    001
MR    6,6             002
ST    7,ANSWER        003
```

Предложение 001 выполняет групповую загрузку, в результате чего содержимое слова QUANTITY загружается в общий регистр 6, а содержимое слова UNITCOST — в общий регистр 7. По завершении операции содержимое этих регистров будет следующим:

Общий регистр 6 (+395) Общий регистр 7 (+75)

S															
0000	0000	0000	0000	0000	0001	1000	1011	0000	0000	0000	0000	0000	0000	0100	1011
0	0	0	0	0	1	8	B	0	0	0	0	0	0	4	B

Предложение 002 умножает содержимое общего регистра 6 на содержимое общего регистра 7, хотя оба операнда указывают общий регистр 6.

Предложение 003 записывает результат умножения в полное слово памяти ANSWER. В этом примере предполагается, что произведение умещается в одном регистре.

Умножение полуслова — MH. Эта команда используется для того, чтобы умножить число с фиксированной точкой, содержащееся в полуслове, на содержимое общего регистра. Для размещения произведения этой команде требуется лишь один общий регистр. Число, подлежащее умножению, сначала следует загрузить в тот регистр, где предполагается получить произведение, затем это число умножается на содержимое полуслова памяти. При этом может возникнуть переполнение, однако в этом случае прерывание не имеет места. Необходимо тщательно следить за тем, чтобы используемые значения не вызвали переполнения, поскольку в результате истинное значение знакового бита может исказиться.

Пример 1. Заданы области памяти и их значения:

```
MPLIER      DC    H'352'
MPCAND      DC    F'49365'
PRODANS     DC    F'0'
```

Выполняются следующие команды:

```
L     6,MPCAND      001
MH    6,MPLIER      002
ST    6,PRODANS     003
```

Предложение 001 загружает в общий регистр 6 число с фиксированной точкой, содержащееся в MPCAND. Содержимое общего регистра 6 будет таким:

Регистр 6

s							
0000	0000	0000	0000	1100	0000	1101	0101
0	0	0	0	C	0	D	5

+49 365

Предложение 002 умножает число, содержащееся в общем регистре 6 (+49365), на содержимое полуслова MPLIER (+352). Сначала из полуслова MPLIER извлекается содержащееся в нем число и расширяется до размера полного слова. Содержимое общего регистра умножается на это расширенное значение, и произведение помещается вновь в общий регистр. Содержимое области MPLIER, ее расширения и значение произведения, полученное в общем регистре 6, будут следующими:

Регистр 6 (До)

s							
0000	0000	0000	0000	1100	0000	1101	0101
0	0	0	0	C	0	D	5

+49 365

MPLIER (До)

s			
0000	0001	0110	0000
0	1	6	0

+352

MPLIER (Расширенное)

s							
0000	0000	0000	0000	0000	0001	0110	0000
0	0	0	0	0	1	6	0

+352

Регистр 6 (Произведение)

s							
0000	0001	0000	1001	0010	0100	1110	0000
0	1	0	9	2	4	E	0

+17 376 480

Предложение 003 записывает число с фиксированной точкой, содержащееся в общем регистре 6, в полное слово памяти PRODANS.

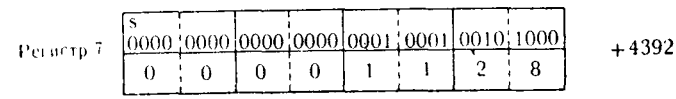
Пример 2. В этом примере выполняется умножение двух чисел, каждое из которых находится в полуслове памяти. Перемножаются числа, заданные следующим образом:

```
HAF1    DC    H'4392'
HAF2    DC    H'68'
ANSW    DC    F'0'
```

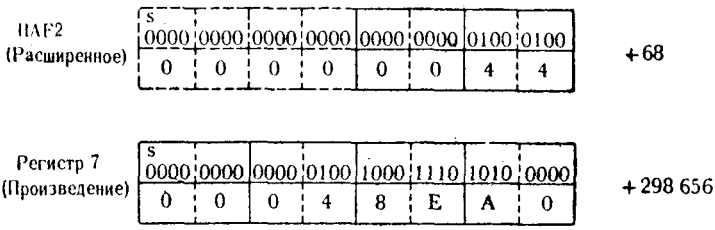
Выполняются следующие команды:

```
LH      7,HAF1      001
MH      7,HAF2      002
ST      7,ANSW      003
```

Предложение 001 загружает число из полуслова HAF1 в общий регистр 7. Перед загрузкой команда расширяет содержимое полуслова до полного слова, заполняя 16 дополнительных старших битов значением, равным значению знакового бита исходного полуслова. Поэтому нет необходимости очищать общий регистр 7 перед загрузкой в него HAF1. Регистр 7 теперь содержит следующие данные:



Предложение 002 умножает число, содержащееся в общем регистре 7, на число, содержащееся в области HAF2. Область HAF2 расширяется до полного слова, после чего выполняется операция умножения. Конфигурации расширенного значения HAF2 и произведения, полученного в общем регистре 7, имеют следующий вид:



Предложение 003 записывает содержимое общего регистра 7 в полное слово памяти ANSW. Если бы было заведомо известно, что произведение в общем регистре 7 никогда не превысит максимального значения, умещающегося в полуслове, в качестве предложения 003 можно было бы использовать команду STH.

Обзор команд умножения с фиксированной точкой. После изучения трех команд умножения с фиксированной точкой становится очевидным, что действия, выполняемые этими командами, различаются очень незначительно. Использование той или другой команды зависит от вида обрабатываемых данных, диапазона их изменения и при прочих равных условиях от личных вкусов программиста.

Для иллюстрации разобранных команд предлагается следующий пример. Допустим, что программист хочет определить общее количество заклепок, необходимых для соединения узлов некоторого агрегата. Хотя приводимая последовательность команд умножения не является лучшим средством достижения поставленной цели, она демонстрирует сходство в выполнении всех этих команд. Обычно количество используемых заклепок и узлов зависит от конкретного вида собираемого агрегата. Предположим, что в некоторый момент времени выполняется фрагмент проблемной программы, приведенный на рис. 11.3, и поля данных содержат указанные величины.

Предложение 001 устанавливает содержимое общего регистра 11 равным нулю путем вычитания его из самого себя. Регистр 11 предполагается использовать для накопления общего количества требуемых заклепок, которое вычисляется с помощью команд умножения.

Предложение 002 загружает общий регистр 3 содержимым полного слова памяти ITEM1QTY (+593). Поскольку общий регистр 3 является одним из пары общих регистров, предназначенной для размещения произведения, содержимое области произведения будет иметь следующий вид:

Общий регистр 2 Общий регистр 3 (+593)

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	0000	0010	0101	0001
								0	0	0	0	0	2	5	1

Предложение 003 умножает содержимое множимого (+593) из общего регистра 3 на число с фиксированной точкой, содержащееся в RIVET1 (+15), что дает в результате +8895 — число, достаточно маленькое, чтобы поместиться в общем регистре 3, несмотря на то что для произведения отведена совокупная область, состоящая из общих регистров 2 и 3. Содержимое регистров 2 и 3 будет теперь таким:

Общий регистр 2 (+8895) Общий регистр 3

S 0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0010	0010	1011	1111
0	0	0	0	0	0	0	0	0	0	0	0	2	2	B	F

Предложение 004 складывает число, содержащееся в общем регистре 3 (+8895), с текущим значением числа с фиксированной точкой в общем регистре 11 (+0), в результате чего в регистре 11 оказывается новое значение +8895.

Предложение 005 содержит команду групповой загрузки, которая загружает содержимое области ITEM2QTY в общий

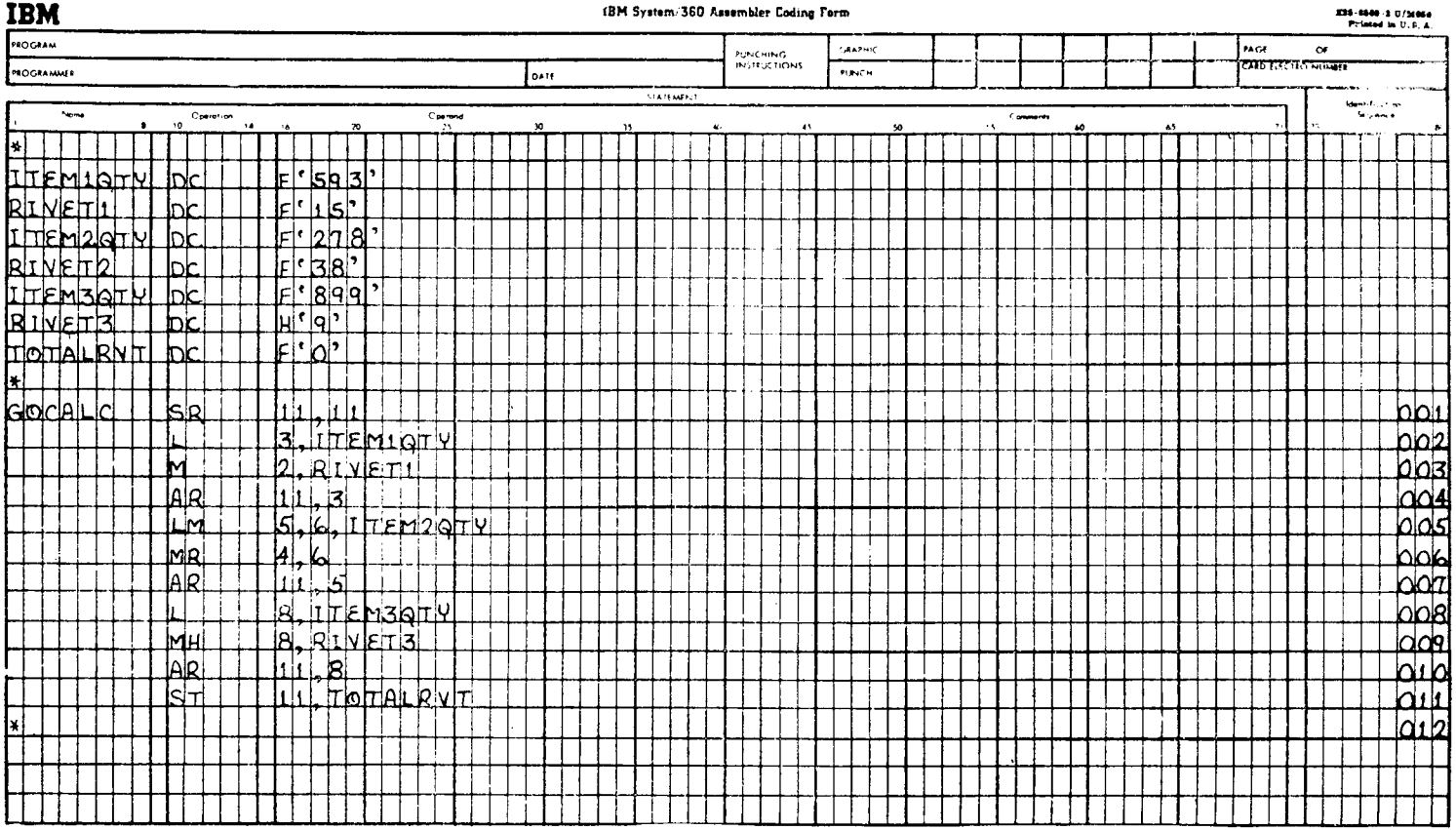

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE
OF
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
*
ITEM1QTY DC F'593'
RIVET1 DC F'15'
ITEM2QTY DC F'278'
RIVET2 DC F'38'
ITEM3QTY DC F'899'
RIVET3 DC H'9'
TOTALRVT DC F'0'
*
GOCALC SR 11,11 001
L 3,ITEM1QTY 002
M 2,RIVET1 003
AR 11,3 004
LM 5,6,ITEM2QTY 005
MR 4,6 006
AR 11,5 007
L 8,ITEM3QTY 008
MH 8,RIVET3 009
AR 11,8 010
ST 11,TOTALRVT 011
* 012

Рис. 11.3.

регистр 5 и содержимое области RIVET2 в общий регистр 6. Тем самым множимое загружается в общий регистр 5 пары, а множитель — в общий регистр 6. После выполнения этой команды регистры будут иметь следующее содержимое:

Общий регистр 4 Общий регистр 5 (+278)

?	?	?	?	?	?	?	?	S 0000	0000	0000	0000	0000	0001	0001	0110
								0	0	0	0	0	1	1	6

Регистр 6

S 0000	0000	0000	0000	0000	0000	0010	0110
0	0	0	0	0	0	2	6

+38

Предложение 006 умножает содержимое общего регистра 5 (+278) на содержимое общего регистра 6 (+38) и помещает полученное произведение в совокупность общих регистров 4 и 5 (+10564). Произведение имеет следующий вид:

Общий регистр 4 Общий регистр 5 (+10564)

S 0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0010	1001	0100	0100
0	0	0	0	0	0	0	0	0	0	0	0	2	9	4	4

Предложение 007 прибавляет число, содержащееся в общем регистре 5 (+10564), к содержимому общего регистра 11 (+8895). Регистр 11 теперь содержит новое число с фиксированной точкой +19459:

Регистр 11

S 0000	0000	0000	0000	0100	1100	0000	0011
0	0	0	0	4	C	0	3

+19459

Предложение 008 загружает содержимое ITEM3QTY (+899) в общий регистр 8. Так как следующей должна выполняться команда MH, необходимо загрузить это значение только в один регистр, причем его номер может быть как четным, так и нечетным. Регистр 8 будет иметь следующее содержимое:

Регистр 8

S 0000	0000	0000	0000	0000	0011	1000	0011
0	0	0	0	0	3	8	3

+899

Предложение 009 содержит команду MH. Содержимое общего регистра 8 (+899) умножается на содержимое полуслова

RIVET3 (+9). После этого регистр 8 будет содержать число +8091.

Регистр 8

S							
0000	0000	0000	0000	0001	1111	1001	1011
0	0	0	0	1	F	9	B

+8091

Предложение 010 содержит команду AR. Сумма значений величин, содержащихся в общих регистрах 8 (+8091) и 11 (+19459), помещается в общий регистр 11:

Регистр 11

S							
0000	0000	0000	0000	0110	1011	1001	1110
0	0	0	0	6	B	9	E

+27 550

Предложение 011 помещает содержимое регистра 11 в полное слово памяти TOTALRVT.

Итак, с помощью этой программы, содержащей несколько команд умножения, определено общее количество заклепок (27550), необходимых для сборки агрегата.

4. Команды деления с фиксированной точкой

Деление — D. Команда D, подобно команде M, для выполнения требует пару соседних общих регистров, которые используются в качестве областей для хранения делимого, частного и остатка. Делимое перед выполнением команды должно быть загружено в эту пару регистров. Команда, загружающая делимое, может адресовать нечетный регистр пары, если только программист не собирается установить знак делимого с помощью операции сдвига. Если делимое загружается непосредственно в нечетный регистр, содержимое четного регистра непредсказуемо, а нечетный регистр будет содержать делимое в виде числа с фиксированной точкой. Делимое можно загрузить из полного слова памяти или из любого другого общего регистра. При выполнении команды деления число, содержащееся в регистре с нечетным номером, будет разделено на число с фиксированной точкой, содержащееся в операнде делителя. После завершения операции частное будет находиться в регистре с нечетным номером в формате числа с фиксированной точкой. Остаток, имеющий знак делителя, будет содержаться в регистре с четным номером. Знак частного определяется правилами алгебры. Поскольку делимое рассматривается как 64-битовое число, содержащееся в паре регистров, то предполагается, что четный регистр заведомо имеет знак, равный знаку нечетного регистра. Это легко выполнить различными путями, в чем мы убедимся на примерах, следующих ниже.

Пример 1. В этом примере используются следующие константы:

```
DIVDEND   DC   F'48933'
DIVSOR    DC   F'37'
QUOTNT    DC   F'0'
REMNDR    DC   F'0'
```

Выполняются следующие команды:

```
ROUTE1    SR   8,8          001
          L    9,DIVDEND    002
          D    8,DIVSOR     003
          ST   8,REMNDR     004
          ST   9,QUOTNT     005
```

Предложение 001 вычитает содержимое общего регистра 8 из самого себя, в результате чего регистр принимает значение +0. Для этого примера заведомо известно, что все числа, выступающие в роли делимого, имеют положительные значения, поэтому мы можем быть уверены, что 64-битовое делимое имеет знак, совпадающий со знаком числа, загружаемого в младшие 32 бита области.

Предложение 002 загружает число с фиксированной точкой из DIVDEND в общий регистр 9 — нечетный регистр пары. После выполнения этой команды эти регистры будут иметь следующее содержимое:

Общий регистр 8 Общий регистр 9 (+48933)

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1011	1111	0010	0101
0	0	0	0	0	0	0	0	0	0	0	0	B	F	2	5

Предложение 003 делит содержимое общих регистров 8 и 9 на число, содержащееся в DIVSOR (+37). После выполнения этой команды частное оказывается в общем регистре 9, а остаток — в общем регистре 8. Содержимое этих регистров имеет следующий вид:

Общий регистр 8 (Остаток +19) Общий регистр 9 (Частное +1322)

S								S							
0000	0000	0000	0000	0000	0000	0001	0011	0000	0000	0000	0000	0000	0101	0010	1010
0	0	0	0	0	0	1	3	0	0	0	0	0	5	2	A

Предложение 004 записывает остаток, находящийся в общем регистре 8, в полное слово памяти REMNDR.

Предложение 005 записывает частное, находящееся в общем регистре 9, в полное слово памяти QUOTNT.

Пример 2. В этом примере берется общая сумма денег, вырученных от продажи различных вещей, и эта сумма делится на число заказов, с тем чтобы определить среднюю стоимость покупки.

Программа закодирована следующим образом:

```
AMTPD      DC      F'3927869'
UNITS      DC      F'8346'
REMAINDR   DC      F'0'
AVGCOST    DC      F'0'
*
DIVRTE     L       4,AMTPD          001
           SRDA    4,32             002
           D       4,UNITS          003
           STM     4,5,REMAINDR     004
```

Предложение 001 загружает общий регистр 4 содержимым слова AMTPD. Оно представляет общую сумму вырученных денег. После выполнения этой команды содержимое регистров 4 и 5 выглядит следующим образом:

Общий регистр 4								Общий регистр 5							
S 0000	0000	0011	1011	1110	1111	0011	1101	?	?	?	?	?	?	?	?
0	0	3	B	E	F	3	D								

Предложение 002 — это команда двойного арифметического сдвига вправо с длиной сдвига 32 бита. В результате содержимое регистра 4 целиком сдвинется в регистр 5 и освободившиеся при сдвиге биты регистра 4 приобретут значение, равное значению знакового бита регистра 4. Теперь мы можем быть уверены, что знаковый бит 64-битового числа, занимающего общие регистры 4 и 5, имеет такое же значение, как и бит знака первоначального делимого, находящегося в слове AMTPD. После сдвига общие регистры 4 и 5 имеют следующее содержимое:

Общий регистр 4								Общий регистр 5							
S 0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0011	1011	1110	1111	0011	1101
0	0	0	0	0	0	0	0	0	0	3	B	E	F	3	D

Предложение 003 делит содержимое регистров 4 и 5 (+3927869) на общее число сделанных заказов, представленное

числом в UNITS (+8346). После выполнения этого предложения общий регистр 4 содержит остаток, а общий регистр 5 — частное, причем это число представляет среднюю стоимость одной покупки. Содержимое этих регистров выглядит следующим образом:

Общий регистр 4 (Остаток +5249) Общий регистр 5 (Частное +470)

S								S							
0000	0000	0000	0000	0001	0100	1000	0001	0000	0000	0000	0000	0000	0001	1101	0110
0	0	0	0	1	4	8	1	0	0	0	0	0	1	D	6

Предложение 004 выполняет групповую запись в память содержимого регистров. Оно записывает содержимое регистра 4 в слово REMAINDR и содержимое общего регистра 5 в слово AVGCOST.

Деление — DR. Команда DR выполняется аналогично команде D, за одним исключением: делитель должен находиться в общем регистре, а не в памяти. Эта команда также использует пару соседних общих регистров в качестве области размещения делимого.

Пример 1. Константы и команды этого примера закодированы следующим образом:

```
SETSIGN   DC     F'0'
DIVDEN    DC     F'72833117'
DIVSR     DC     F'442'
REMDR     DC     F'0'
ANSR      DC     F'0'
*
BEGINDV   LM     6,8,SETSIGN      001
          DR     6,8              002
          STM    6,7,REMDR        003
```

Предложение 001 выполняет групповую загрузку регистров. Число с фиксированной точкой из слова SETSIGN загружается в общий регистр 6, число из DIVDEN — в общий регистр 7 и число из DIVSR — в общий регистр 8. В этой программе предполагается, что все числа, выступающие в качестве делимого, — положительные. Поэтому достаточно загрузить в регистр 6 число +0 (из SETSIGN), чтобы быть уверенным в том, что 64-битовое число со знаком (занимающее совокупность регистров 6 и 7) будет иметь знаковый бит, равный знаковому биту делимого (DIVDEN), загружаемого в регистр 7. Одновременно с

установкой знакового бита и загрузкой делимого в общий регистр 8 загружается делитель (DIVSR). Эта команда заменяет три команды L, с помощью которых можно было бы выполнить все эти действия. После завершения операции содержимое общих регистров 6, 7 и 8 имеет такой вид:

Общий регистр 6 (+72833117) Общий регистр 7

s															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0100	0101	0111	0101	1000	0101	1101
0	0	0	0	0	0	0	0	0	4	5	7	5	8	5	D

Общий регистр 8

s								
0000	0000	0000	0000	0000	0001	1011	1010	(+442)
0	0	0	0	0	1	B	A	

Предложение 002 содержит команду DR. В данном случае 64-битовое число с фиксированной точкой, представленное регистрами 6 и 7, делится на число, содержащееся в регистре 8. После выполнения этой операции содержимое регистров 6 и 7 примет следующий вид:

Регистр 6 (Остаток +357) Регистр 7 (Частное +1647801)

s								s							
0000	0000	0000	0000	0000	0001	0110	0101	0000	0000	0000	0010	1000	0011	1010	1100
0	0	0	0	0	1	6	5	0	0	0	2	8	3	A	C

Предложение 003 — команда групповой записи в память. Число из регистра 6 (остаток) записывается в слово REMDR, а число из регистра 7 (частное) — в слово ANSR.

Так как команд деления с фиксированной точкой только две, нет необходимости делать обзор их функционирования. Следующий раздел, посвященный применению команд с фиксированной точкой для действий над десятичными числами, дает достаточно полное представление о применении команд D и DR.

B. ПРИМЕНЕНИЕ КОМАНД С ФИКСИРОВАННОЙ ТОЧКОЙ ДЛЯ ОПЕРАЦИЙ НАД ЧИСЛАМИ С ПОДРАЗУМЕВАЕМОЙ ДЕСЯТИЧНОЙ ТОЧКОЙ

Реализация арифметических действий над числами с подразумеваемой десятичной точкой достаточно проста. В основу метода положено промежуточное умножение или деление различных чисел с фиксированной точкой на коэффициенты 10, 100, 1000, 10 000 и т. д. Для того чтобы приписать к числу две позиции после подразумеваемой десятичной точки, достаточно умножить его на 100. Для того чтобы удалить из числа дробную часть, содержащую два десятичных разряда после точки, достаточно

разделить это число на 100 и взять значение целой части результата. Округление десятичных величин и последующее выделение целочисленной части выполняется аналогично.

Например, предположим, что программист хочет умножить стоимость одного предмета на общее количество этих предметов и получить общую стоимость с точностью до двух десятичных знаков. Используя данные, закодированные ниже, эту задачу можно выполнить по крайней мере двумя путями:

```
UNITCOST    DC    F'95'       (XXXX. подразумеваемый
                                формат)
UNITQTY     DC    F'863'      (XXXX. подразумеваемый
                                формат)
TOTLCOST    DC    F'0'        (XXXXXXXXX.00
                                подразумеваемый формат)
XPANDIT     DC    F'100'      (X.00 подразумеваемый
                                формат)
*
DORTE       SR    4,4                                  001
            L     5,UNITCOST                           002
            M     4,XPANDIT                            003
            M     4,UNITQTY                            004
            ST    5,TOTLCOST                           005
```

Предложение 001 очищает общий регистр 4 до значения +0, вычитая содержимое регистра из самого себя.

Предложение 002 загружает в общий регистр 5 число с фиксированной точкой из UNITCOST. Здесь принимается, что стоимость равна 95 долл. Содержимое общих регистров 4 и 5 будет таким:

Общий регистр 4 (+0) Общий регистр 5 (+95)

S								S							
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0101	1111
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	F

Предложение 003 умножает содержимое общих регистров 4 и 5 на 100 — число, содержащееся в слове XPANDIT. Это изменит содержимое общих регистров 4 и 5 следующим образом:

Общий регистр 4 (+9500) Общий регистр 5

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0010	0101	0001	1100
0	0	0	0	0	0	0	0	0	0	0	0	2	5	1	C

Предложение 004 умножает содержимое общих регистров 4 и 5 на общее число предметов (+863), которое содержится в UNITQTY. Регистры имеют теперь такую конфигурацию:

Общий регистр 4 (+8198500 – подразумеваемая стоимость) Общий регистр 5 (81985.00 долларов)

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0111	1101	0001	1001	0110	0100
0	0	0	0	0	0	0	0	0	0	7	D	1	9	6	4

Предложение 005 записывает содержимое общего регистра 5 в слово памяти TOTLCOST. По замыслу программиста запоминаемые данные имеют формат дробного десятичного числа 81985.00, хотя на самом деле десятичной точки нет и она только подразумевается. Программист обязан помнить о предполагаемом месте десятичной точки, с тем чтобы в дальнейших действиях правильно пользоваться этими данными.

Того же результата можно достигнуть, выполнив следующие команды:

```
DORTE    SR    4,4              001
         L     5,UNITCOST       002
         M     4,UNITQTY        003
         M     4,XPANDIT        004
         ST    5,TOTLCOST       005
```

В этом примере стоимость одного предмета сначала умножается на общее число предметов, а затем результат умножается на 100. Содержимое регистров при выполнении первых четырех команд выглядит следующим образом:

После предложений 001 и 002:

Общий регистр 4 (+0) Общий регистр 5 (+95)

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0101	1111
0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	F

После предложения 003:

Общий регистр 4 (+81985) Общий регистр 5

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0100	0000	0100	0001
0	0	0	0	0	0	0	0	0	0	0	1	4	0	4	1

После предложения 004:

Общий регистр 4 Общий регистр 5

(+8198500 – подразумеваемая стоимость 8198500 долларов)

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0111	1101	0001	1001	0110	0100
0	0	0	0	0	0	0	0	0	0	7	D	1	9	6	4

Как и в предыдущем примере, предложение 005 выполняет запись в память.

Предварительное умножение величины на постоянный коэффициент часто используется для получения процентного отношения двух величин. Рассмотрим этот способ на примере:

```
SQUARES   DC   F'893'    (XXXX. подразумеваемый
                         формат)
CIRCLES   DC   F'995'    (XXXX. подразумеваемый
                         формат)
MULT      DC   F'1000'   (X.000 подразумеваемый
                         формат)
RATIO     DC   F'0'      (XXXX0% подразуме-
                         ваемый формат)
*
PCTRTE    SR   6,6                          001
          L    7,SQUARES                    002
          M    6,MULT                       003
          D    6,CIRCLES                    004
          ST   7,RATIO                      005
```

Предложение 001 помещает в общий регистр 6 значение +0, вычитая содержимое регистра из самого себя.

Предложение 002 загружает общий регистр 7 содержимым SQUARES. Содержимое регистров 6 и 7 будет таким:

Общий регистр 6 (+893) Общий регистр 7

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0011	0111	1101
0	0	0	0	0	0	0	0	0	0	0	0	0	3	7	D

Предложение 003 умножает содержимое общих регистров 6 и 7 (+893) на +1000 — число с фиксированной точкой, содержащееся в MULT. В результате к числу в общем регистре 7 добавились три десятичные позиции после подразумеваемой десятичной точки. Теперь регистры будут иметь следующее содержимое:

Общий регистр 6 Общий регистр 7
(+893000 – подразумеваемое число 893.000)

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1101	1010	0000	0100	1000
0	0	0	0	0	0	0	0	0	0	0	D	A	0	4	8

Предложение 004 делит содержимое общих регистров 6 и 7 (+893000) на число с фиксированной точкой, содержащееся в слове CIRCLES (+995). После выполнения этой команды содержимое регистров будет таким:

Общий регистр 6 Общий регистр 7 (Частное —+897)
(Остаток —+458) (подразумеваемое значение .897 или 89.7%)

S								S							
0000	0000	0000	0000	0000	0001	1110	0101	0000	0000	0000	0000	0000	0011	1000	0001
0	0	0	0	0	1	E	5	0	0	0	0	0	3	8	1

Предложение 005 записывает число с фиксированной точкой, содержащееся в общем регистре 7, в слово RATIO. Программист считает, что данные в этом слове представляют собой величину 0.897 или 89.7%.

Таким же способом выполняется округление десятичного числа до ближайшего целого или до меньшего числа позиций после десятичной точки. Ниже приведены два примера, представляющие подпрограммы умножения и деления с округлением.

Пример 1. Умножается несколько смешанных чисел и произведение затем округляется до ближайшего целого числа. Если первый десятичный разряд дробной части больше или равен 5, то значение целой части следует увеличить на 1.

На рис. 11.4 приведены команды и константы, используемые в этом примере.

Предложение 001 устанавливает регистр 2 в нуль, вычитая его содержимое из самого себя.

Предложение 002 загружает в общий регистр 3 число с фиксированной точкой из слова CASES. Теперь регистры 2 и 3 имеют такое содержимое:

Общий регистр 2 (+4331) Общий регистр 3

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0000	1110	1011
0	0	0	0	0	0	0	0	0	0	0	0	1	0	E	C

Предложение 003 умножает содержимое общего регистра 3 (+4331) на число, хранящееся в UNITS. Хотя в этом слове содержится число +293, программист принимает, что значение

IBM

IBM System/360 Assembler Coding Form

X28-6648-2 U/360S
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE | OF | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
*				
CASES	DC	F'4331'	(XXXX. ASSUMED FORMAT)	
UNITS	DC	F'293'	(.000 ASSUMED FORMAT)	
ROUNDER	DC	F'500'	(.000 ASSUMED FORMAT)	
REDUCE	DC	F'1000'	(X.000 ASSUMED FORMAT)	
ANSWER	DC	F'0'	(XXXXXX. ASSUMED FORMAT)	
*				
*				
SAMPRTE	SR	2,2		001
	L	3,CASES		002
	M	2,UNITS		003
	A	3,ROUNDER		004
	D	2,REDUCE		005
	ST	3,ANSWER		006

Рис. 11.4.

UNITS равно 0.293. После выполнения команды умножения содержимое общих регистров примет вид

Общий регистр 2 Общий регистр 3

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0011	0101	1100	1111	0111
0	0	0	0	0	0	0	0	0	0	1	3	5	C	F	7

+1268983 (Подразумеваемое значение +1268.983)

Предложение 004 прибавляет число с фиксированной точкой +500 из ROUNDER к содержимому общего регистра 3. Алгоритм таков, что если первая цифра справа от подразумеваемой десятичной точки равна или больше 5, то целую часть числа следует увеличить на 1. Добавляя .500 к цифрам, расположенным справа от подразумеваемой десятичной точки, программа автоматически прибавляет единицу к разряду десятичных единиц, если первая цифра справа от десятичной точки равна или больше 5. После выполнения этого предложения содержимое регистров будет иметь вид

Общий регистр 2 Общий регистр 3

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0011	0101	1110	1110	1011
0	0	0	0	0	0	0	0	0	0	1	3	5	E	E	B

+1269483 (Подразумеваемое значение +1269.483)

Предложение 005 делит число, содержащееся в регистрах 2 и 3, на содержимое REDUCE (+1000). В результате делимое уменьшается до величины подразумеваемой целой части, так как при делении на +1000 из дробной части числа исключаются как раз все три подразумеваемые позиции после десятичной точки. Теперь содержимое регистров будет выглядеть следующим образом:

Общий регистр 2 Общий регистр 3

S								S							
0000	0000	0000	0000	0000	0001	1110	0011	0000	0000	0000	0000	0000	0100	1111	0101
0	0	0	0	0	1	E	3	0	0	0	0	0	4	F	5

(Остаток +483) (Частное +1269)

Предложение 006 записывает содержимое общего регистра 3 (+1269) в слово ANSWER.

Пример 2. В этом примере вычисляется процентное отношение двух чисел; оно получается в виде подразумеваемого выражения 0.0000 или 000.00%. Это значение затем округляется до

числа вида 0.000 или 000.0% и записывается в память. Используются команды и константы, представленные на рис. 11.5.

Предложение 001 устанавливает общий регистр 8 в нуль, вычитая его содержимое из самого себя.

Предложение 002 загружает в общий регистр 9 число с фиксированной точкой из RECEIPTS (+2395). Теперь регистры имеют такое содержимое:

Общий регистр 8 (+2395) Общий регистр 9

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1001	0101	1011
0	0	0	0	0	0	0	0	0	0	0	0	0	9	5	C

Предложение 003 умножает содержимое этих регистров на 10000 — число, содержащееся в XPAND. Конфигурация регистров имеет следующий вид:

Общий регистр 8 Общий регистр 9

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0110	1101	0111	0010	1011	0000
0	0	0	0	0	0	0	0	0	1	6	D	7	2	B	0

+ 23950000 (Подразумеваемое значение 2395.0000)

Предложение 004 делит содержимое общих регистров 8 и 9 на число, содержащееся в SALES. Частное, получающееся в регистре 9, имеет формат процентного отношения (0.0000). Теперь содержимое регистров следующее:

Общий регистр 8 Общий регистр 9

S								S							
0000	0000	0000	0000	0000	1100	0010	1000	0000	0000	0000	0000	0001	0110	1011	1010
0	0	0	0	0	C	2	8	0	0	0	0	1	6	B	A

(Остаток +3112) (Частное +5818)
(Подразумеваемое значение .5818 или 58.18%)

Предложение 005 прибавляет число +5 (подразумеваемое значение .0005 или .05%) к содержимому общего регистра 9 с тем, чтобы округлить процентное отношение до ближайшего числа вида .0%. После выполнения команды регистры будут иметь следующее содержимое:

Общий регистр 8 Общий регистр 9

S								S							
0000	0000	0000	0000	0000	1100	0010	1000	0000	0000	0000	0000	0001	0110	1011	1111
0	0	0	0	0	C	2	8	0	0	0	0	1	6	B	F

(+3112) (5823 — подразумеваемое значение 5823 или 58.23%)

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M068
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
RECEIPTS	DC	F'2395'	(XXXX. ASSUMED FORMAT)	
SALES	DC	F'4116'	(XXXX. ASSUMED FORMAT)	
XPAND	DC	F'10000'	(X.0000 ASSUMED FORMAT)	
ROUND	DC	F'5'	(.0000 ASSUMED FORMAT)	
CUTOFF	DC	F'10'	(X.0 ASSUMED FORMAT)	
HOLDIT	DC	F'0'	(XXX.0% ASSUMED FORMAT)	
*				
*				
NOWRTE	SR	8,8		001
	L	9,RECEIPTS		002
	M	8,XPAND		003
	D	8,SALES		004
	A	9,ROUND		005
	SR	8,8		006
	D	8,CUTOFF		007
	ST	9,HOLDIT		008

Рис. 11.5.

Предложение 006 устанавливает значение регистра 8 равным нулю, подготовляя его для выполнения следующей команды деления. Регистры теперь имеют следующее содержимое:

Общий регистр 8 Общий регистр 9

S															
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	0110	1011	1111
0	0	0	0	0	0	0	0	0	0	0	0	1	6	B	F

(+0) (+5823 — подразумеваемое значение .5823 или 58.23%)

Предложение 007 делит содержимое общих регистров 8 и 9 на число, содержащееся в CUTOFF (+10). В результате общее количество десятичных разрядов числа, содержащегося в общем регистре 9, уменьшается на единицу. Теперь регистры имеют такое содержимое:

Общий регистр 8 Общий регистр 9

S								S							
0000	0000	0000	0000	0000	0000	0000	0011	0000	0000	0000	0000	0000	0010	0100	0110
0	0	0	0	0	0	0	3	0	0	0	0	0	2	4	6

(Остаток = +3) Частное = +582)
(Подразумеваемое значение .582 или 58.2%)

Предложение 008 записывает содержимое общего регистра 9 в слово памяти, адресуемое меткой HOLDIT. Запоминаемое значение +582 выражает подразумеваемое программистом значение 58.2% и в таком формате используется в других частях программы.

Приведенные примеры и иллюстрации должны дать хорошее представление о применении и обработке подразумеваемых разрядов дробной части десятичных чисел. В общем случае для удаления лишних разрядов дробной части числа необходимо лишь разделить их на 10, 100, 1000, 10 000 и т. д. Для того чтобы добавить необходимое количество десятичных разрядов после точки, надо умножить число на 10, 100, 1000, 10 000 и т. д. Комбинация этих двух приемов дает программисту средства для обработки десятичных чисел с помощью арифметических команд с фиксированной точкой.

Упражнения

1. При выполнении команды AH содержимое полуслова, указанного вторым операндом, прибавляется к содержимому двух младших байтов общего регистра первого операнда. (Правильно) (Неправильно).

2. SLR — это мнемонический код для команды Ассемблера _____.

3. При выполнении команд с фиксированной точкой _____ и _____ значение признака результата в PSW не изменяется.

4. Отрицательное число с фиксированной точкой всегда имеет знаковый бит, значение которого равно _____.

5. При выполнении команды S содержимое второго операнда _____ и эта новая конфигурация алгебраически _____ с содержимым первого операнда.

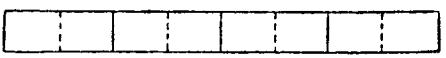
6. При написании команды M или MR первый операнд предложения должен всегда указывать регистр с _____ номером пары общих регистров.

7. Если выполняется команда SR, в которой регистр первого операнда содержал +2048, а регистр второго операнда содержал —2048, то регистр первого операнда получит значение _____.

8. В команде MH можно указывать в качестве регистра первого операнда общий регистр с нечетным номером. (Правильно) (Неправильно).

9. В команде AR результат сложения помещается в регистр (первого) (второго) операнда.

10. Если полуслово памяти, содержащее число с фиксированной точкой —32768, прибавляется к содержимому общего регистра с помощью команды AH, то расширение этого полуслова до размера полного слова можно представить в виде следующего шестнадцатеричного числа:



11. В результате выполнения команды M может получиться число, не умещающееся в рамках одного регистра. (Правильно) (Неправильно).

12. Команда Сложение — _____ прибавляет двоичную конфигурацию слова памяти к двоичной конфигурации общего регистра первого операнда независимо от значения знакового бита.

13. Хотя для размещения первого операнда команды MH используется только один общий регистр, в результате выполнения этой команды не может произойти переполнения с фиксированной точкой. (Правильно) (Неправильно).

Используя числа, содержащиеся в указанных общих регистрах и областях памяти (см. следующие ниже примеры), напи-

шите двоичные и шестнадцатеричные конфигурации полей, получающиеся в результате выполнения предложений.

14. Регистр 3 содержит +23975
Регистр 9 содержит —14210

```
                ADDFLDS     AR      3,9
Регистр 3 (Шестн.)
          (Двоичн.)
```

15. Регистр 11 содержит +392417
HAFWD содержит +32519

```
                SETADDRT    AH      11,HAFWD
Регистр 11 (Шестн.)
           (Двоичн.)
```

16. Регистр 7 содержит 32416817
FWD9 содержит —27941

```
                LOGADD      AL      7,FWD9
Регистр 7  (Шестн.)
           (Двоичн.)
```

17. Регистр 3 содержит +31965423
Регистр 9 содержит +7592087

```
                GOLCADD     ALR     9,3
Регистр 9 (Шестн.)
          (Двоичн.)
```

18. Регистр 10 содержит +159375
Регистр 9 содержит +2899

```
                SUBRTE      SLR     10,9
Регистр 10 (Шестн.)
           (Двоичн.)
```

19. Регистр 4 содержит —39541782

```
                SUBRT2      SR      4,4
Регистр 4 (Шестн.)
          (Двоичн.)
```

20. Регистр 12 содержит —389510
FULLWD6 содержит +426877

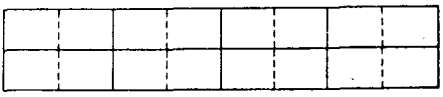
```
SUMRTE    A     12,FULLWD6
```

Регистр 12 (Шестн.)
(Двоичн.)

21. Регистр 7 содержит +319586
HALFWDB содержит —7853

```
JOBRTE7   SH    7,HALFWDB
```

Регистр 7 (Шестн.)
(Двоичн.)



22. Регистр 10 содержит +21583910
FULLWRDX содержит +30009

```
CLRTE5    SL    10,FULLWRDX
```

Регистр 10 (Шестн.)
(Двоичн.)

23. Регистр 3 содержит +325198
FWORD содержит —2768447

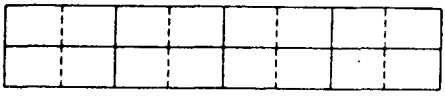
```
SETCALC   S     3,FWORD
```

Регистр 3 (Шестн.)
(Двоичн.)

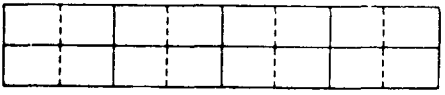
24. Регистр 2 содержит +83915
Регистр 3 содержит —5176933
Регистр 4 содержит +213847
FVALUE1 содержит +15006
FVALUE2 содержит +8092

```
SETVALS   SR    2,2
          LM    3,4,FVALUE1
          MR    2,4
```

Регистр 2 (Шестн.)
(Двоичн.)

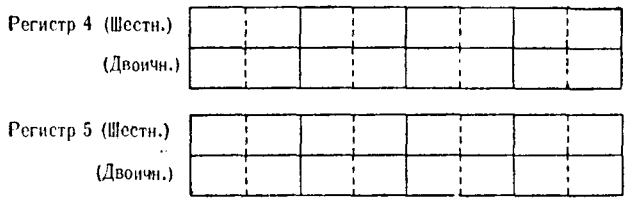


Регистр 3 (Шестн.)
(Двоичн.)



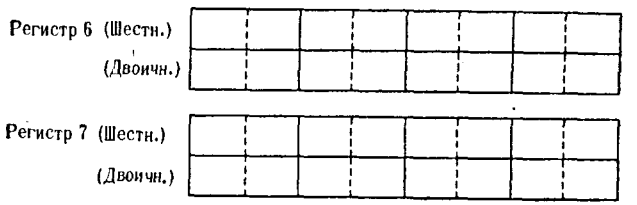
25. Регистр 4 содержит +3975
Регистр 5 содержит +17842931
FULLWORD содержит +800216
FWMP содержит +83

```
MULTRTE   L      4, FULLWORD
          SRDL   4,32
          M      4,FWMP
```



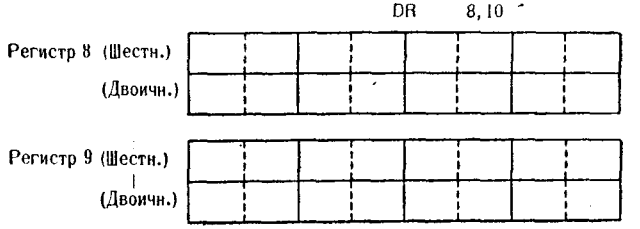
26. Регистр 6 содержит +0
Регистр 7 содержит +385910
HAFWD9 содержит +16694

```
MULRTE8   MH     6, HAFWD9
```



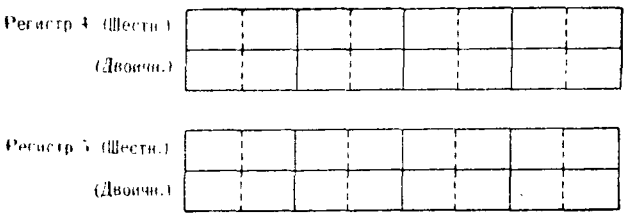
27. Регистр 8 содержит +315876
Регистр 9 содержит +2977310
Регистр 10 содержит —15900
FWD1 содержит +0
FWD2 содержит + 26985
FWD3 содержит +204

```
DIVRTE3   LM     8, 10, FWD1
          DR     8, 10
```



28. Регистр 4 содержит +3906608
Регистр 5 содержит —21504
FULLWA содержит +612983
FULLWB содержит +716

```
DIVROUT4    L     4,FULLWA
            SRDA  4,32
            D     4,FULLWB
```



Дополнительные упражнения к главам 3—11

Предлагаемые упражнения состоят из предложений, в которых могут быть ошибки. Может встретиться неверный формат предложений, неверные операнды и неправильные спецификации длины и регистров. Отметьте неверно составленные предложения.

1.	____	DATARTE	CLI	DATA3+9,Z'6'
2.	____		AP	DATPAK(5),7(4,5)
3.	____	BIN4SET	DC	63BL1'11000001'
4.	____	LOADEM	L	2,ADCON1
5.	____		MVC	BIGFLD+1500(6),PADDATA
6.	____	LOADSET	LM	13,2,4(8)
7.	____		SL	5,15(0,8)
8.	____	BRANCHA	BALR	3,6
9.	____	SETLARGE	DC	6000XL2'4040'
10.	____		M	10,0(0,3)
11.	____		D	3,FULLWORD
12.	____	PUTTEM	STM	10,9,8(7)
13.	____		AP	DATAPAK,=XL3'004593'
14.	____		C	DATA1(6),DATA2
15.	____	MULTEM	MH	3,=H'328'
16.	____		SH	11,=H'1'
17.	____		LH	10,FULLWRD9
18.	____	ADDUP	A	6,=H'12396'
19.	____	COMPRTE	CLC	0+260(260,3),0(4)
20.	____		AR	6(4),8

```
21. ____          SLDA  5,12
22. ____          CLC   DATA1+900(15),INDATA(15,5)
23. ____          LCR   10,10
24. ____ CONVRTER CVB   DUBLPAK,6
25. ____ ODDSET   DS    0CL393
26. ____          MVI   DATAFLD+56(4,4),C'DATE'
27. ____          SR    10,1
28. ____          SP    0+19(15,12),0+45(12,8)
29. ____          SL    3,4,FULLWORDS
30. ____ CALCIT   AP    0+5(3,9),8(2,10)
31. ____          MH    5,HAFWRD1
32. ____          L     6,8FULWD3
33. ____ REROUTE  A     10,FLWD(0,9)
34. ____ SETCHAR  STC   5,ONEBYTE
35. ____          S     3,F'820015916'
36. ____ DIVREG   DR    10,4
37. ____          ALR   4,=F'9831'
38. ____ NULLAREA DC    0CL6'123123'
39. ____ LTROUTE  LTR   9,0(10)
40. ____ SETTLERTE SR   10,9
41. ____          M     7,=F'99210'
42. ____          MVZ   B8(17),CONST1
43. ____ S3468F   MVC   G,A(2)
44. ____          SP    DATAPAK(3),LONGPACK(4)
45. ____          CP    PACKBYTE(1),PL1'4'
46. ____          AR    5,5(5)
47. ____ CHEXIT   CH    4,=H'15002'
48. ____          MVC   DATA+1(255),DATA
49. ____          SH    6(2),=H'32765'
50. ____          SRA   8,8
51. ____          SP    7(7,8),DATA3+2(1)
52. ____          A     11,4(3,5)
53. ____ PAKITUP  PACK  PAKIT(9),0+400(12,5)
54. ____          MVC   0+256(256),0(8)
55. ____          DP    QUOREM(7),=P'51'
56. ____          MR    6,5
57. ____          SRL   4,1
58. ____ CLEARIT  SR    11,11
```

```
59. _____            BAL   8,GOBACK
60. _____            SH    H'523',HAFWRD
61. _____            LA    6,6
62. _____ LEFTITE    SLDL  8,60
63. _____            ST    6,FULLWORDS
64. _____            SR    5,5
65. _____ DIVIT      D     7,11
66. _____            BC    5,NEWLOOP
67. _____            M     3,FULLWRD7
68. _____            L     FULLWD,9
69. _____            SRDR  8,32
70. _____            LM    5,WORD1,WORD2
71. _____ SUMUP      AH    9,2(8,9)
72. _____            DR    6,5
73. _____            ST    11,FULLWORDS
74. _____ TOTALS     AL    6,=F'392566'
75. _____            S     9,10(0,4)
76. _____            LH    3,HAFWRDA
77. _____            MVZ   DATAHOLD(3,5),
                           CHANGE(3,6)
78. _____            AR    3,=F'5932'
79. _____ LEFTMUV    SLL   2,2
80. _____            LTR   8,8
81. _____            MH    5,0(0,5)
82. _____ DIVQRM     DR    3,8
83. _____            C     6,4(0,9)
84. _____ VALCONST   DC    PL10'+15356983.333'
85. _____            MR    8,10
86. _____            BXLE  9,10,12
87. _____ OUTRTE     BCT   7,RELOOP
88. _____            L     8,DUBLWD
89. _____            D     6,FULLWRD9
90. _____            AL    8,FULLWD1
91. _____ SETTAB     CLI   FIELDA(6,1),=CL1'X'
92. _____            M     11,0(0,8)
93. _____            SH    HAFWORD,2
94. _____            UNPK  DATAFLD(20),
                           PACKFLD+20(8)
```

```
95.  ____          AL      2,36(1,11)
96.  ____          D       12,14(0,8)
97.  ____          LM      6,15,WORDS
98.  ____  SAVE    STH     3,FULLSWT
99.  ____          A       8,5(6,8)
100. ____          SRDA    11,48
101. ____          MR      10,=F'939'
102. ____          CL      DATAFIELD,=F'155220'
103. ____  SETPAK  PACK    PACKER(18),DATA(18)
104. ____  BCTEXIT BCT     10,0(0,8)
105. ____          C       10,FULLWD5
106. ____          IC      6,6(6,12)
107. ____          LNR     13,13
108. ____          MP      MULTPAK(12),MPLIER(10)
109. ____          BC      0,NEXTRTE
110. ____          CH      HAFWRD1,HAFWRD2
111. ____          ST      FULLWD,7
112. ____          MVC     DATASET+8(1),=C'A'
113. ____          CVD     10,PACK8
114. ____          BCTR    8,8
115. ____  MATCH   CR      9,10
116. ____          LM      FULLWORD,6,9
117. ____          STC     DATABYTE,=CL1'X'
118. ____          BAL     6,3,NEWROUTE
119. ____          AP      DPACK(12),EPACK(11)
120. ____          CLR     10,0(3,4)
121. ____          LA      7,1932659
122. ____          BXH     5,9,10(11)
123. ____  VALCONS DS      8CL2'00'
124. ____  SETREDY DC      CL9'IN && OUT'
125. ____  H       DH      8,HALFWRD
```

Глава 12

Применение булевой логики

А. КОМАНДЫ БУЛЕВОЙ АЛГЕБРЫ ЛОГИКИ

Команда And Immediate — NI (И непосредственное)

Мнемоника	Код операции	Формат операндов
NI	94	$D_1(B_1),I_2$

Команда NI выполняет операцию логического умножения, используя 1 байт непосредственных данных второго операнда в качестве маски для байта данных, указанного первым операндом. Команда изменяет биты в соответствии со следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	0
0	0	0
0	1	0

Результат операции замещает первый операнд.
Эта команда воздействует только на 1 байт данных.

CC	BC	Условие
0	8	Первый операнд равен 0 (все биты нулевые)
1	4	Первый операнд не равен 0 (1 или больше битов ненулевые)

Команда And — N (И)

Мнемоника	Код операции	Формат операндов
N	54	$R_1,D_2(X_2,B_2)$

Команда N выполняет операцию логического умножения, используя содержимое второго операнда в качестве маски для содержимого регистра первого операнда. Результат воздействия

этой команды на содержимое первого операнда определяется следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	0
0	0	0
0	1	0

Результат операции помещается в регистр первого операнда. Второй операнд должен располагаться в памяти на границе слова.

Оба операнда должны иметь одинаковую длину, равную 4 байтам.

CC	BC	Условие
0	8	Регистр первого операнда содержит 0 (все биты нулевые)
1	4	Регистр первого операнда имеет ненулевое содержимое (1 или больше битов ненулевые)

Команда And — NC (И)

Мнемоника	Код операции	Формат операндов
NC	D4	$D_1(L,B_1),D_2(B_2)$

Команда NC выполняет операцию логического умножения, используя данные из памяти, указанные вторым операндом, в качестве маски для данных первого операнда. Результирующие значения битов первого операнда определяются следующим образом:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	0
0	0	0
0	1	0

Результат замещает первый операнд. Длина поля каждого операнда команды NC может быть до 256 байтов.

CC	BC	Условие
0	8	Содержимое первого операнда нулевое (все биты нулевые)
1	4	Содержимое первого операнда ненулевое (1 или больше битов ненулевые)

Команда And — NR (И)

Мнемоника	Код операции	Формат операндов
NR	14	R_1,R_2

Эта команда выполняет операцию логического умножения, используя содержимое регистра второго операнда в качестве маски для содержимого регистра первого операнда. Результат определяется следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	0
0	0	0
0	1	0

Результат помещается в регистр первого операнда.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно 0 (все биты нулевые)
1	4	Содержимое регистра первого операнда не равно 0 (1 или более битов не равны 0)

Команда OR Immediate — OI (ИЛИ непосредственное)

Мнемоника	Код операции	Формат операндов
OI	96	$D_1(B_1),I_2$

Команда логического сложения OI использует 1 байт непосредственных данных, заданный вторым операндом, в качестве

маски для 1 байта данных, указанного первым операндом. Биты первого операнда изменяются по следующим правилам:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	1
0	1	1
0	0	0

Результат выполнения команды помещается в область первого операнда.

Эта команда воздействует только на 1 байт данных.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0 (все биты нулевые)
1	4	Содержимое первого операнда не равно 0 (1 или более битов не равны 0)

Команда OR — O (ИЛИ)

Мнемоника	Код операции	Формат операндов
O	56	$R_1,D_2(X_2,B_2)$

Эта команда логического сложения использует содержимое полного слова второго операнда в качестве маски для содержимого регистра первого операнда в соответствии со следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	1
0	1	1
0	0	0

Результат помещается в общий регистр первого операнда. Второй операнд должен располагаться на границе слова.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0 (все биты равны 0)
1	4	Первый операнд имеет ненулевое содержимое (1 или более битов ненулевые)

Команда OR — OC (ИЛИ)

Мнемоника	Код операции	Формат операндов
OC	D6	$D_1(L,B_1),D_2(B_2)$

Эта команда логического сложения использует область памяти, указанную вторым операндом, в качестве маски для данных, расположенных в области памяти, указанной первым операндом. Команда изменяет биты первого операнда по следующим правилам:

Значение битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	1
0	1	1
0	0	0

Результат выполнения этой команды помещается в область первого операнда. Максимальная длина поля операнда составляет 256 байтов.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0 (все биты нулевые)
1	4	Первый операнд имеет ненулевое содержимое (1 или несколько битов ненулевые)

Команда OR — OR (ИЛИ)

Мнемоника	Код операции	Формат операндов
OR	16	R_1,R_2

Эта команда логического сложения использует содержимое регистра второго операнда в качестве маски для содержимого регистра первого операнда. Биты изменяются по следующим правилам:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	1
1	0	1
0	1	1
0	0	0

Результат операции помещается в общий регистр первого операнда. Оба операнда должны находиться в общих регистрах.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно 0 (все биты нулевые)
1	4	Содержимое регистра первого операнда не равно 0 (1 или более битов не равны 0)

Команда Exclusive OR Immediate — XI
(Исключающее ИЛИ непосредственное)

Мнемоника	Код операции	Формат операндов
XI	97	$D_1(B_1),I_2$

Команда XI использует 1 байт непосредственных данных, заданный вторым операндом, в качестве маски для одного байта данных, указанного вторым операндом. Биты первого операнда изменяются в соответствии со следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	0
1	0	1
0	1	1
0	0	0

Результат выполнения этой команды помещается в область первого операнда.

Эта команда воздействует только на 1 байт памяти.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0 (все биты нулевые)
1	4	Первый операнд не равен 0 (1 или несколько битов не равны 0)

Команда Exclusive OR — X (Исключающее ИЛИ)

Мнемоника	Код операции	Формат операндов
X	57	$R_1,D_2(X_2,B_2)$

Команда X использует содержимое полного слова, указанного вторым операндом, в качестве маски для содержимого регистра первого операнда. Результат выполнения этой команды определяется следующей таблицей:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	0
1	0	1
0	1	1
0	0	0

Результат помещается в общий регистр первого операнда. Второй операнд должен располагаться в памяти на границе полного слова.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0 (все биты нулевые)
1	4	Содержимое первого операнда не равно 0 (1 или несколько битов не равны 0)

Команда Exclusive OR — XC (Исключающее ИЛИ)

Мнемоника	Код операции	Формат операндов
XC	D7	$D_1(L,B_1),D_2(B_2)$

Эта команда использует содержимое области памяти, указанной вторым операндом, в качестве маски для данных, указанных первым операндом. Биты первого операнда изменяются по следующим правилам:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	0
1	0	1
0	1	1
0	0	0

Результат замещает первый операнд.
Максимальная длина поля операнда составляет 256 байтов.

CC	BC	Условие
0	8	Содержимое первого операнда равно 0
1	4	Содержимое первого операнда не равно 0

Команда Exclusive OR — XR (Исключающее ИЛИ)

Мнемоника	Код операции	Формат операндов
XR	17	R_1,R_2

Эта команда использует содержимое регистра второго операнда в качестве маски для содержимого регистра первого операнда. Выполнение команды изменяет биты следующим образом:

Значения битов		Результат в первом операнде
второй операнд	первый операнд	
1	1	0
1	0	1
0	1	1
0	0	0

Результат выполнения команды помещается в общий регистр первого операнда. Оба операнда команды должны находиться в общих регистрах.

CC	BC	Условие
0	8	Содержимое регистра первого операнда равно 0 (все биты нулевые)
1	4	Содержимое регистра первого операнда не равно 0 (1 или несколько битов ненулевые)

Б. ПРИМЕНЕНИЕ КОМАНД БУЛЕВОЙ ЛОГИКИ

Команды булевой, или двоичной, логики оперируют с отдельными битами или группами битов. Данные, обрабатываемые этими командами, как правило, не следует рассматривать как состоящие из символов. В языке Ассемблера имеются три группы команд булевой логики — И, ИЛИ и Исключающее ИЛИ. Каждая из этих групп состоит из нескольких команд, которые выполняют одинаковые действия и отличаются друг от друга лишь типом и длиной полей обрабатываемых данных.

Все эти команды имеют одну общую черту — они используют *маску* для того, чтобы определить, каким образом следует изменить биты исходного поля. Эту маску было бы логично рассматривать как некий набор битов, причем каждый из этих наборов вызывает свойственные только ему действия над одним и тем же исходным полем. Вид маски может быть каким угодно, это зависит от интерпретации значения отдельных битов, а также от вида групповых или индивидуальных действий над ними.

Предложения кодируются таким образом, что исходное поле указывается в качестве первого операнда, а маска — в качестве второго операнда. Исходная битовая конфигурация изменяется в том случае, если маска порождает какие-либо действия, а это совсем не обязательно должно случиться.

1. Команды группы И

Команды группы И выполняют операцию логического умножения в соответствии с правилами: любая величина, умноженная на нуль, равняется нулю и нуль, взятый сколько угодно раз, равен нулю. Поскольку команды булевой логики оперируют только с битами, следует понимать, что умножение каждого бита исходного поля на соответствующий бит маски ограничивается случаями 1×0, 0×0, 1×0 и 1×1, так как бит может принимать лишь два значения: 0 и 1. Результат в исходном поле в зависимости от его первоначального содержимого и значения маски будет следующим:

Бит маски		Соответствующий бит исходного поля		Бит результата в исходном поле
1	×	1	=	1
1	×	0	=	0
0	×	1	=	0
0	×	0	=	0

После выполнения любой команды группы И признак результата будет установлен в одно из двух состояний — одно указывает, что исходное поле теперь равно нулю (т. е. состоит из одних нулевых битов), другое указывает, что это поле не равно нулю, (т. е. содержит хотя бы 1 бит, имеющий единичное значение).

CC	BC	Условие
0	8	Результат нулевой
1	4	Результат ненулевой

Команды группы И часто применяются для того, чтобы установить битовый переключатель в состояние «выключено» [1]).

Три из четырех команд типа И имеют одинаковое название, однако мнемонические коды их различны и указывают тип используемых полей данных.

И непосредственное — NI. Команда NI использует однобайтовую маску непосредственных данных для того, чтобы вызвать определенные действия над 1 байтом из памяти. Для большей ясности принято выражать содержимое байта непосредственных данных в шестнадцатеричной форме, что позволяет легко определить битовую конфигурацию этого байта. Как и другие команды группы И, эта команда наилучшим образом подходит для установки двоичных (битовых) переключателей в состояние «выключено». 1 байт непосредственного операнда команды NI вмещает восемь битовых переключателей, и каждый нулевой бит маски устанавливает соответствующий бит в исходном байте в нулевое состояние. Если программист хочет оставить некоторые позиции битов в прежнем состоянии, он должен установить соответствующие биты маски равными единице.

Пусть байт, адресуемый меткой CHEXBITS, используется в качестве набора битовых переключателей:

	Один байт							
CHEXBITS	0	0	0	0	0	0	0	0
Номера переключателей	1	2	3	4	5	6	7	8

Будем считать, что единичный бит означает состояние «включено», а нулевой бит означает состояние «выключено». Тогда

[1]) Далее для краткости наряду с наименованиями состояния битового переключателя «включено» и «выключено» будут применяться наименования «Да» и «Нет» соответственно. — *Прим. ред.*

значение X'B6' в CHEXBITS соответствует следующим состояниям переключателей:

	Один байт							
CHEXBITS	1	0	1	1	0	1	1	0
Номера переключателей	1	2	3	4	5	6	7	8

В следующих примерах предполагается, что исходное поле содержит X'B6' и эта конфигурация изменяется под воздействием разных значений маски.

Пример 1. Требуется установить в нуль битовые переключатели 1, 4, 5 и 8 перед входом в некоторую подпрограмму. Чтобы установить эти биты в нуль и оставить при этом остальные биты без изменения, необходимо построить такую маску, в которой битовые позиции 1, 4, 5 и 8 содержат нулевые значения, а остальные позиции содержат единичные значения. Эта маска имеет конфигурацию X'66'

0	1	1	0	0	1	1	0
1	2	3	4	5	6	7	8

Используем команду

```
NI      CNEXBITS,X'66'
```

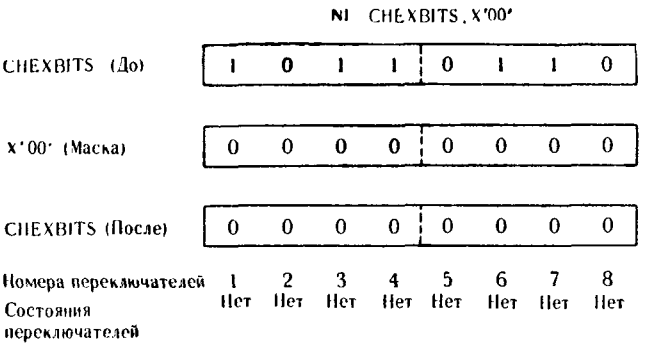
которая производит следующие действия:

	Один байт								
CHEXBITS (До)	1	0	1	1	0	1	1	0	X'B6'
X'66' (Маска)	0	1	1	0	0	1	1	0	X'66'
CHEXBITS (После)	0	0	1	0	0	1	1	0	X'26'
Номера переключателей	1	2	3	4	5	6	7	8	
Состояния переключателей	Нет	Нет	Да	Нет	Нет	Да	Да	Нет	

Как и требовалось, переключатели 1, 4, 5 и 8 установлены в состояние «выключено» или остались в этом состоянии. Переключатели 2, 3, 6 и 7 остались в том состоянии, в каком находились ранее.

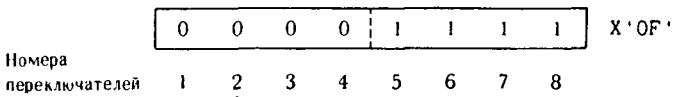
Пример 2. В программе требуется, чтобы все битовые переключатели, содержащиеся в CHEXBITS, были установлены в

состояние «выключено». Как нам известно, нулевой бит маски обеспечивает установку соответствующего бита исходного поля в нуль. Поэтому достаточно установить все биты в маске равными нулю (X'00'), чтобы командой NI сбросить все переключатели поля CHEXBITS.



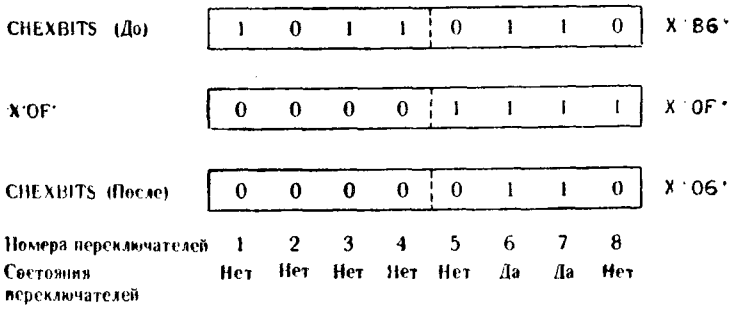
Теперь все переключатели поля CHEXBITS установлены в состояние «выключено» или остались в этом состоянии.

Пример 3. В этом примере производится проверка определенных битов с разрушением некоторой информации. Требуется определить, находятся ли переключатели 5, 6, 7 и 8 в состоянии «выключено». Если это условие выполняется, программа передает управление подпрограмме AUDIT; если один или несколько переключателей находятся в состоянии «включено», управление передается подпрограмме ERROR. Поскольку переключатели 1, 2, 3 и 4 не проверяются и в соответствии с алгоритмом программы желательно, чтобы они в этом месте программы были установлены в нуль, можно использовать такую маску:



Эта маска сбросит переключатели 1, 2, 3, 4, если они находились в состоянии «включено», и в то же время оставит переключатели 5, 6, 7 и 8 в состоянии «включено», если они до этого находились в этом состоянии. Значение признака результата укажет, каким получился результат: нулевым или ненулевым. Если результат нулевой, то ни один из последних переключателей не находился в состоянии «включено»; если ненулевой, это означает, что один или несколько из этих переключателей были в состоянии «включено». Команды этого примера выглядят так:

```
                NI      CHEXBITS,X'0F'          001
                BC      8,AUDIT                 002
                BC      4,ERROR                 003
```



Предложение 001 выполняет команду NI и дает результат, представленный в поле CHEXBITS (После) на предыдущем рисунке.

Предложение 002, если результат оказался нулевым, передает управление подпрограмме AUDIT. В этом примере переход не выполняется.

Предложение 003, если результат ненулевой, передает управление подпрограмме ERROR. В этом месте нашего примера произойдет переход.

Рассмотренные в приведенных примерах действия характерны для всех команд группы И. Аналогично тому, как команда NI использует маску непосредственных данных для обработки 1 байта исходного поля, другие команды группы И оперируют с масками и полями других типов.

И — N. Команда N использует полное слово памяти в качестве маски для того, чтобы изменить значения битов, содержащихся в общем регистре. Длина операндов ограничивается 32 битами. Маска должна находиться в области памяти, расположенной на границе слова, а первый операнд должен указывать исходное поле в общем регистре непосредственно или символически.

Эта команда может быть использована для манипуляций с битовыми переключателями подобно тому, как было показано в примерах к команде NI. Ее также можно применить для установки в нуль содержимого общего регистра.

Для иллюстрации работы команды N предположим, что нам следует написать программу, выполняющую такие действия:

1. Общий регистр 8 содержит число в младших 3 байтах и некоторый код в старшем байте.
2. Старший байт необходимо очистить, используя команду N.

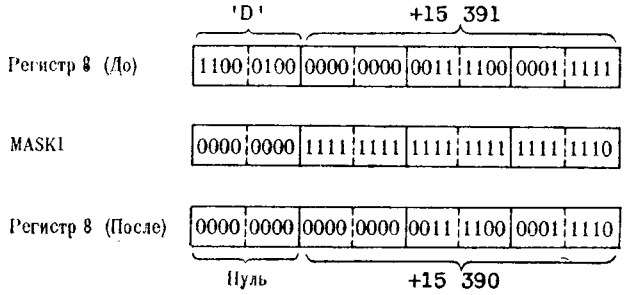
3. Число, содержащееся в 3 младших байтах, должно быть четным; если оно не является таковым, его следует уменьшить до ближайшего меньшего четного числа.

Перед выполнением команд общий регистр 8 содержит число +15391 и код символа D в старшем байте.

Ниже приведены выполняемая команда и задана маска в слове памяти:

```
         DS    0F
MASK1    DC    X'00FFFFFE'
*
         N     8,MASK1
```

При выполнении команды значения битов изменяются следующим образом:



Старший байт маски (X'00') установил все биты старшего байта общего регистра 8 в нуль. Единственный нулевой младший бит маски обеспечивает установку единичного младшего бита регистра в нуль, и поэтому общий регистр 8 всегда будет содержать четное число.

Заметим, что для единичных битов маски соответствующие битовые позиции общего регистра 8 не изменяются, а там, где в маске появляется нулевой бит, соответствующий ему бит общего регистра приобретает нулевое значение.

И — NC. Команда NC для просмотра и изменения содержимого исходного поля, находящегося в области памяти, отведенной программе, использует маску, занимающую область той же длины. Длина поля маски и соответственно исходного поля может достигать 256 байтов. Любой из операндов команды может быть представлен либо символическим, либо действительным адресом.

Особым случаем использования этой команды является ее применение для отображения всех буквенных символов кода EBCDIC в буквенные символы диапазона от A до I. Это преобразование является возможным вследствие определенной

конфигурации четырех старших двоичных разрядов буквенных символов кода EBCDIC.

При исследовании таблицы символов кода EBCDIC можно обнаружить сходство между указанными наборами символов.

Первоначальные символы	Преобразованные символы
A или J	A (X'C1')
B или K или S	B (X'C2')
C или L или T	C (X'C3')
D или M или U	D (X'C4')
E или N или V	E (X'C5')
F или O или W	F (X'C6')
G или P или X	G (X'C7')
H или Q или Y	H (X'C8')
I или R или Z	I (X'C9')

Поле данных, которое должно быть преобразовано (в той части, в какой это окажется необходимым), представляет собой пятибайтовую область памяти, содержащую символы RAQWS кода EBCDIC. Маска, используемая для выполнения указанного преобразования исходного поля, определена в виде пятибайтовой константы с меткой BIGMASK. Ее содержимое имеет следующий вид:

```
BIGMASK    DC      XL5'CFCFCFCFCF'
```

или

BIGMASK	1100 1111	1100 1111	1100 1111	1100 1111	1100 1111

Должна быть выполнена команда

```
           NC      DATAFLD,BIGMASK
```

Результатом выполнения этой команды явится следующее преобразование данных поля DATAFLD:

DATAFLD (Символьн.)	R	A	Q	W	S
(Шестн.)	D 9	C 1	D 8	E 6	E 2
(Двоичн.)	1101 1001	1100 0001	1101 1000	1110 0110	1110 0010

BIGMASK	1100 1111	1100 1111	1100 1111	1100 1111	1100 1111

DATAFLD (Двоичн.)	1100 1001	1100 0001	1100 1000	1100 0110	1100 0010
(Шестн.)	C 9	C 1	C 8	C 6	C 2
(Символьн.)	I	A	H	F	B

Указанное в задаче преобразование выполнено. Эта же маска может быть использована для выполнения преобразования независимо от того, какие символы содержатся в DATAFLD, при условии что этими символами являются буквы кода EBCDIC.

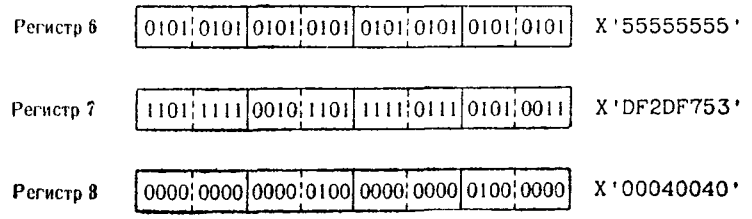
И — NR. Эта команда является командой И формата RR — как маска, так и исходное поле должны находиться в общих регистрах. Следовательно, при выполнении команды обрабатываются точно 32 бита, или 4 байта, — длина общего регистра. Некоторые случаи применения, проиллюстрированные в примерах, приведенных для других команд группы И, могут быть отнесены и к этой команде, но при том условии, что учтена ограниченная длина регистров.

Интересен метод применения команды NR для формирования маски в одном из регистров для последующего ее использования с третьим регистром. Другими словами, содержимое общего регистра X (маска) логически умножается на содержимое общего регистра Y, а затем выполняется логическое умножение содержимого регистра Y (маска) на содержимое регистра Z. Смысл такого технического приема может заключаться в следующем:

1. Содержимое общего регистра Y было сформировано в результате выполнения ряда подпрограмм, которые использовали отдельные биты этого регистра в качестве переключателей. Некоторые позиции битов были установлены в 1.
2. Содержимое регистра Z также было частично сформировано при помощи некоторых подпрограмм. Программа должна сравнить часть регистра Z, содержащую определенные позиции битов, с соответствующими позициями битов общего регистра Y. Дальнейший ход выполнения программы зависит от того, были ли все сравниваемые позиции битов в противоположных состояниях перед логическим умножением или нет.

В приведенном ниже примере позиции нумеруются слева направо от 1 до 32. В регистре Z в качестве переключателей используются только биты с четными номерами. Следовательно, маска, находящаяся в регистре X, должна удалить из регистра Y любой из единичных битов, который может появиться в позиции с нечетным номером, ввиду того что регистр Y будет использован в качестве маски при логическом умножении его содержимого на содержимое регистра Z. Вместо регистров, условно обозначенных X, Y и Z, будут использованы регистры 6, 7 и 8. Содержимое этих общих регистров перед выполнением

команд NR имеет следующий вид:



Выполняется следующая программа:

```
NROUTE   NR    7,6          001
         NR    8,7          002
         BC    4,BADMATCH   003
         BC    8,VALID      004
```

В предложении 001 происходит логическое умножение содержимого регистров 6 и 7, конечный результат операции располагается в регистре 7. Все единичные биты, занимавшие в регистре 7 позиции с нечетными номерами, замещаются нулями. Оставшиеся единичные биты занимают позиции с четными номерами, и теперь они будут использованы в предложении 002 в качестве значащих разрядов маски.

В предложении 002 новая конфигурация регистра 7 используется как маска при логическом умножении на содержимое общего регистра 8. Только те позиции регистра 7, которые содержат единичные биты, будут сопоставляться с соответствующими позициями регистра 8. Все остальные позиции общего регистра 8 считаются не принимающими участия в операции и загружаются нулями.

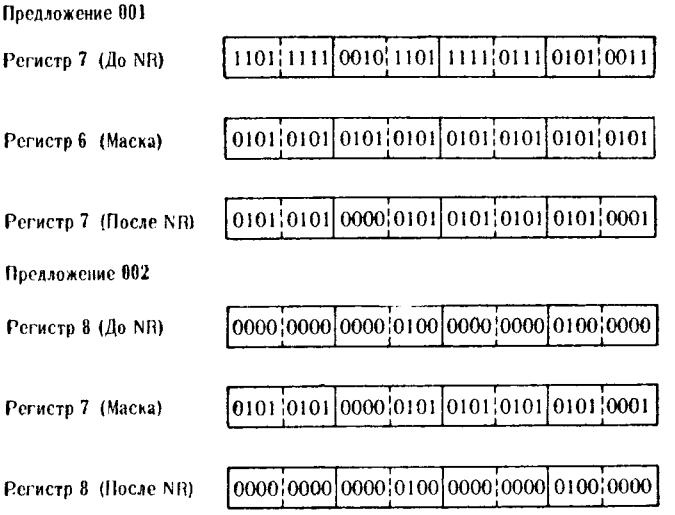
В предложении 003 анализируется признак результата, полученный при выполнении предложения 002. Если 1 или более единичных битов остались в регистре 8 после завершения логического умножения, признак результата покажет, что результат отличен от нуля, и произойдет переход к BADMATCH.

В предложении 004 предполагается, что если в предложении 003 перехода не произошло, то результатом логического умножения должен быть нуль. В соответствии с этим это предложение анализирует признак результата и осуществляет переход к подпрограмме VALID. Поскольку в результате выполнения предложения 002 могут быть получены только два условия (либо результат равен нулю, либо не равен), то предложение 004 могло бы быть записано как безусловный переход, т. е.

```
         BC    15,VALID.
```

Содержимое регистров по мере их участия в выполнении двух команд логического умножения будет изменяться следую-

щим образом:



Нет необходимости в том, чтобы читатель сам нашел конкретное применение только что рассмотренному техническому приему. Достаточно того, что он разобрался в использовании значащих и незначащих разрядов в маске логического умножения.

2. Команды группы ИЛИ

Существует четыре команды ИЛИ — столько же, сколько и команд И. Команды группы ИЛИ реализуют функцию логического сложения, складывая значения битов маски со значениями соответствующих битов исходного поля. В этой операции поразрядного двоичного сложения каждая позиция трактуется как самостоятельная и перенос из одной позиции в следующую отсутствует. Это можно более ясно изложить, перечислив правила сложения: $1 + 1 = 1$, $0 + 1 = 1$, $1 + 0 = 1$ и $0 + 0 = 0$. Команда ИЛИ, основываясь на первоначальной конфигурации исходного поля и маски, обрабатывает исходное поле следующим образом:

Бит маски		Соответствующий бит исходного поля		Бит исходного поля после выполнения команды
1	+	1	=	1
1	+	0	=	1
0	+	1	=	1
0	+	0	=	0

Обратите внимание, что 1 + 1 равно 1 без переноса в следующую старшую позицию.

После завершения выполнения команды ИЛИ значение признака результата в PSW будет указывать на одно из двух возможных состояний: в исходном поле находится нуль (все биты нулевые); в исходном поле содержится 1 единичный бит или более.

CC	BC	Условие
0	8	Результат равен 0
1	4	Результат не равен 0

В то время как команды И использовались для установки битов в нуль, команды ИЛИ используются для установки битов в единицу. Если считается, что позиция бита исходного поля находится в состоянии «включено» (содержит единицу), то и единичный бит, и нулевой бит в соответствующей позиции маски не изменят состояния этого бита. Если позиция бита исходного поля находится в состоянии «выключено» (содержит нулевой бит), то единичный бит маски изменит ее состояние на «включено», а нулевой бит маски оставит ее в том же состоянии. Только у одной из команд ИЛИ есть свое особое название (ИЛИ непосредственное), хотя у каждой из четырех команд есть свой собственный мнемонический код. Выбор той или иной команды ИЛИ связан с типом полей данных (области памяти или регистры), которые используются в качестве исходного поля и маски.

ИЛИ непосредственное — OI. Команда OI воздействует на 1 байт исходного поля, расположенного в памяти, используя в качестве маски непосредственный символ, определенный в качестве второго операнда команды. Каждый бит восьмибитовой маски логически складывается с соответствующим битом байта области памяти первого операнда. Сумма битов представляет собой результат выполнения команды ИЛИ. Признак результата в PSW указывает, равна сумма нулю или нет.

Конфигурация маски определяется посредством оценки действия, которое должно быть выполнено над байтом исходного поля. Нулевой бит маски не изменит конфигурацию соответствующего бита исходного поля; единичный бит маски обеспечивает установку соответствующего бита исходного поля в единицу.

Для иллюстрации этой команды предположим, что перед началом выполнения подпрограммы во всех битах одного байта области памяти (BITSET) содержатся нули. В первом цикле программы встречаются две команды OI. Каждая из этих команд устанавливает один битовый переключатель в состояние

«включено». Хотя может показаться, что эти две команды расположены относительно близко друг к другу, в программе они отделены друг от друга значительным числом других команд.

```
OI      BITSET,X'20'     0011
        . . . . . .
        . . . . . .
OI      BITSET,X'08'     0132
```

В предложении 0011 происходит логическое сложение битов непосредственного символа X'20' с содержимым BITSET. К моменту выполнения этой команды в BITSET находился код X'00'.

	Один байт							
BITSET (До OI)	0	0	0	0	0	0	0	0
X'20' (Маска)	0	0	1	0	0	0	0	0
BITSET (После OI)	0	0	1	0	0	0	0	0

В предложении 0132 происходит логическое сложение маски, или непосредственного символа X'08' с текущим содержимым BITSET

BITSET (Перед OI)	0	0	1	0	0	0	0	0
X'08' (Маска)	0	0	0	0	1	0	0	0
BITSET (После OI)	0	0	1	0	1	0	0	0

Рассматривая BITSET как восьмиразрядный переключатель, можно сказать, что переключатели 3 и 5 находятся в состоянии «включено».

BITSET	0	0	1	0	1	0	0	0
Номера переключателей	1	2	3	4	5	6	7	8
Состояния переключателей	Нет	Нет	Да	Нет	Да	Нет	Нет	Нет

Команда OI может быть использована в тех случаях, когда требуется поместить единицу в одной или более позициях.

ИЛИ — О. Команда О производит логическое сложение маски, представляющей собой полное слово памяти, с исходным полем, расположенным в общем регистре. Все 32 бита четырехбайтовой маски используются для вычисления конечного состояния битов регистра. Первый операнд команды указывает общий регистр, а второй операнд представляет собой действительный или символический адрес полного слова памяти.

В качестве примера применения этой команды рассмотрим следующий случай. Программист написал модульную программу, состоящую из восьми модулей-подпрограмм и управляющего модуля. В каждой из подпрограмм некоторое значение адреса загружается в 3 младших байта регистра 10. После возврата управления управляющему модулю необходимо иметь информацию, какая из подпрограмм загрузила в регистр 10 величину, которая находится там в настоящий момент. Задача может быть легко решена, если старшие 8 битов регистра 10 использовать в качестве переключателей. После того как один из модулей выполнит загрузку регистра 10, тот же модуль может немедленно установить двоичный переключатель в состояние «включено» в разряде, соответствующем идентификатору модуля; так, например, модуль 6 может «включить» 6-й разряд двоичного переключателя. Тогда при передаче значения адреса через регистр 10 в управляющий модуль в этом регистре будет также содержаться идентификатор модуля, загрузившего адрес. Пусть следующие команды являются частью подпрограммы 6:

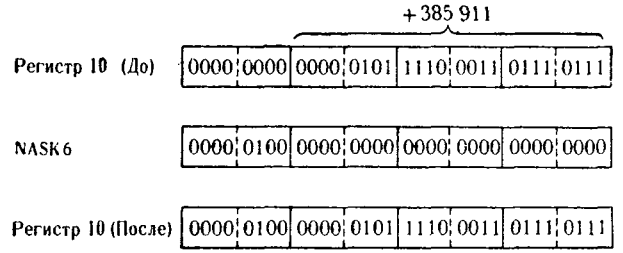
```
— —   — — — — — —
L     10,NEWVALUE     010
O     10,MASK6        011
BC    15,CONTROL      012
```

В этом случае MASK6 является константой длиной в слово с шестнадцатеричным содержимым 04000000.

Предложение 010 загружает в регистр 10 значение адреса, содержащееся в NEWVALUE. Допустим, что это значение равно +385911.

В предложении 011 происходит логическое сложение полного слова MASK6 с содержимым регистра 10. В связи с тем что в 3 младших байтах MASK6 не содержится значащих разрядов, значение адреса в регистре 10 не будет искажено. Выполнение предложения 011 вызывает следующее изменение содержимого

регистра 10:



Теперь в 3 младших байтах регистра 10 содержится адрес, а в старшем байте указатель (двоичный переключатель), который сообщает управляющему модулю, какая подпрограмма загрузила адрес.

Предложение 012 — безусловный переход к управляющему модулю.

ИЛИ — ОС. Команда ОС представляет собой команду группы ИЛИ формата память—память. Как маска, так и исходное поле располагаются в областях памяти, длина каждой из которых может достигать 256 байтов. Если маска воздействует на всю область первого операнда, то области, указываемые первым и вторым операндами, должны иметь равную длину, заданную в явной или неявной форме.

В примере выполнения команды NC было показано, как преобразовать любой буквенный символ кода EBCDIC в один из буквенных символов в диапазоне от A до I. Для демонстрации выполнения команды OC в следующем примере будет показано преобразование любого буквенного символа кода EBCDIC в соответствующий цифровой символ этого кода. Так как в младших четырех двоичных разрядах каждого буквенного символа кода EBCDIC содержится шестнадцатеричный код из диапазона 0—9, то команда OC используется для изменения 4 старших битов каждого символа на единицы, таким образом преобразуя буквенный символ в цифровой.

В этом примере в качестве исходного поля используется шестибайтовая область CONVRSN и шестибайтовая маска NUMASK. Перед выполнением команды в CONVRSN содержатся следующие данные:

CONVRSN (Символьн.)	N		U		M		B		E		R	
(Шестн.)	D	5	E	4	D	4	C	2	C	5	D	9
(Двоичн.)	1101	0101	1110	0100	1101	0100	1100	0010	1100	0101	1101	1001

Должна быть выполнена команда:

```
OC      CONVRSN(6),NUMASK
```

Полагая, что конфигурация поля NUMASK в шестнадцатеричном формате имеет вид F0F0F0F0F0F0, выполнение команды приведет к следующему результату:

CONVRSN (До OC)	1101	0101	1110	0100	1101	0100	1100	0010	1100	0101	1101	1001
NUMASK	1111	0000	1111	0000	1111	0000	1111	0000	1111	0000	1111	0000
CONVRSN (После OC) (Шестн.)	1111	0101	1111	0100	1111	0100	1111	0010	1111	0101	1111	1001
(Двоичн.)	F	5	F	4	F	4	F	2	F	5	F	9
(Символьн.)	5		4		4		2		5		9	

Путем логического сложения маски X'F0' с любым буквенным символом кода EBCDIC получаем цифровой символ с тем же значением в правом шестнадцатеричном разряде, что и у исходного буквенного символа.

ИЛИ — OR. Мнемонический код этой команды чисто случайно совпадает с обозначением выполняемой этой командой функции. Название команды OR является обозначением группы булевых команд, а мнемоника OR указывает, что это код операции команды ИЛИ формата регистр — регистр.

Ввиду того что оба операнда команды (исходное поле и маска) должны находиться в общих регистрах, длина логически складываемых полей равна 4 байтам.

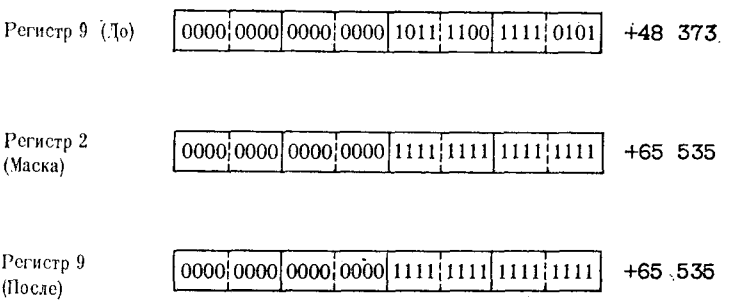
В примере использования этой команды программисту необходимо обеспечить в регистре 9 число с фиксированной точкой, минимальное значение которого +65535, для последующего его использования некоторой подпрограммой. Любое число, даже большее чем +65535, преобразуется путем введения единиц во все 16 младших двоичных разрядов исходного поля. С этой целью выполняется логическое сложение 2 младших байтов общего регистра 9 с числом X'FFFF', так как это число является конфигурацией требуемого минимального значения +65535. Необходимо также заметить, что программист должен обеспечить в регистре 9 положительное значение, тогда как ранее в нем могло находиться отрицательное число. Перед выполнением команд в регистре 9 содержится число +48373, а маска распо-

лагается в регистре 2. Должны быть выполнены следующие команды:

```
— —     — — — — — —
LPR     9,9         061
OR      9,2         062
```

Предложение 061 содержит команду LPR, преобразующую число, находящееся в регистре 9, в прямой код, если оно до этого было в дополнительном коде.

Предложение 062 устанавливает единицы во все младшие 16 битов регистра 9. Теперь в регистре 9 находится положительное число, не меньшее чем +65535, что и требовалось в соответствии с алгоритмом программы:



В связи с тем что необходимость применения этого алгоритма может быть не вполне очевидна, дадим некоторые пояснения, чтобы читатель при необходимости мог воспользоваться этим приемом. В рассматриваемой программе содержится подпрограмма, которая берет число из регистра 9 и выделяет память блоками по 65536 байтов, одновременно вычитая из содержимого регистра 9 размер блока до тех пор, пока значение в регистре 9 не уменьшится до величины, меньшей размера одного блока. После этого распределяется последний блок памяти, длина которого в байтах равна 65535. В байт памяти, следующий непосредственно за коротким (65535) блоком, заносится символ-ограничитель для указания того, что это последний выделенный блок памяти. Если число в регистре 9 к моменту выполнения этой подпрограммы было меньше числа +65535, то оно при помощи команды логического сложения будет увеличено до этого минимально допустимого значения. Полученный при этом блок памяти будет единственным и к тому же коротким блоком со следующим за ним символом-ограничителем. Выполнение команды OR является гарантией того, что в последнем, или единственном, блоке будет содержаться точно 65535 байтов.

3. Команды группы Исключающее ИЛИ

Группа команд Исключающее ИЛИ состоит из четырех сходных команд, каждая из которых выполняет аналогичные функции, но пользуясь различными типами данных. Функция, реализуемая любой из разновидностей команды Исключающее ИЛИ представляет собой модифицированное сложение. Результат модифицированного сложения определяется следующими правилами: 2 равных по значению бита дают в результате нуль; 2 разных бита дают в результате единицу. Ниже приведены результаты сложения всевозможных конфигураций исходного поля и маски:

Бит маски		Соответствующий бит исходного поля		Результирующий бит исходного поля
1	+	1	=	0
1	+	0	=	1
0	+	0	=	0
0	+	1	=	1

В связи с тем что каждая позиция бита рассматривается как самостоятельная и независимая от других, то при выполнении модифицированного сложения двух единиц перенос отсутствует.

После выполнения команд Исключающее ИЛИ признак результата в PSW устанавливается так, как и после выполнения других команд булевой логики, указывая, был ли результат нулевой или нет.

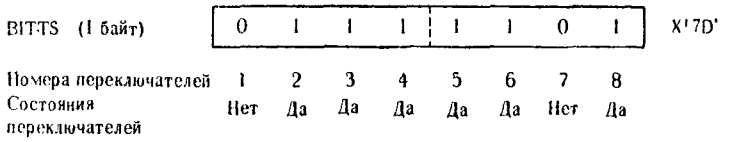
CC	BC	Условие
0	8	Результат равен 0
1	4	Результат не равен 0

У команд Исключающее ИЛИ есть несколько практических применений, одно из которых заключается в изменении состояния двоичных переключателей на обратное. Если двоичный переключатель находится в состоянии «включено», команда Исключающее ИЛИ может его «выключить», и наоборот. Можно дать такое объяснение действиям этих команд: единичный бит маски изменяет значение соответствующего бита исходного поля на обратное, а нулевой бит маски оставляет его неизменным.

Исключающее ИЛИ непосредственное — XI. В этой команде для изменения содержимого однобайтовой области памяти используется однобайтовая маска — непосредственный символ.

Каждый значащий бит (единичный бит) маски изменяет состояние соответствующего бита исходного байта на обратное. Нулевой бит маски не изменяет соответствующего бита исходного поля.

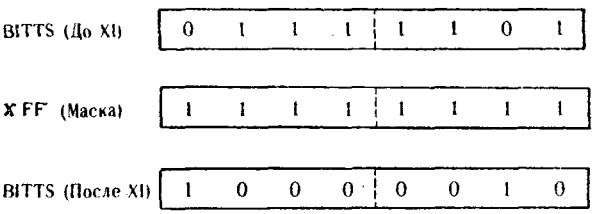
Обратимся еще раз к применению двоичных переключателей и предположим, что в байте памяти BITTS находится шестнадцатеричная величина 7D.



Команда Исключающее ИЛИ с непосредственным символом X'FF' должна быть выполнена с полем BITTS в качестве второго операнда, а полученный признак результата должен быть проверен, чтобы установить, все ли двоичные переключатели находятся теперь в состоянии «выключено».

```
XI      BITTS,X'FF'     001
BC      8,ALLOFF        002
BC      15,NOTOFF       003
```

Предложение 001 содержит команду XI, использующую маску X'FF' и исходное поле BITTS. Команда выполняется следующим образом:



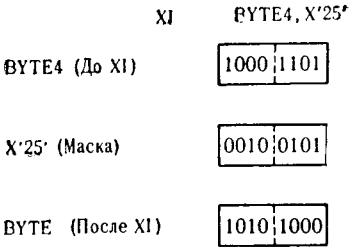
Признак результата теперь указывает, что содержимое исходного поля не равно нулю.

В предложении 002 указывается, что если бы признак результата соответствовал равенству содержимого поля BITTS нулю, то произошел бы переход к подпрограмме ALLOFF. Однако в рассматриваемом примере в этом месте перехода не произойдет.

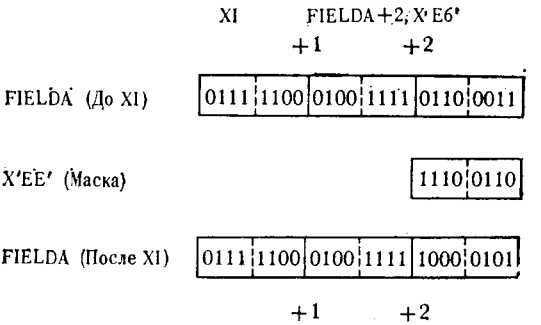
Предложение 003 является безусловным переходом к подпрограмме NOTOFF. Из того, что в предложении 002 перехода не произошло, следует, что признак результата соответствует случаю, когда результат не равен нулю. Команда BC 4,NOTOFF привела бы к выполнению такого же перехода.

Следующие примеры, приводимые без подробного анализа, служат иллюстрацией выполнения команды XI.

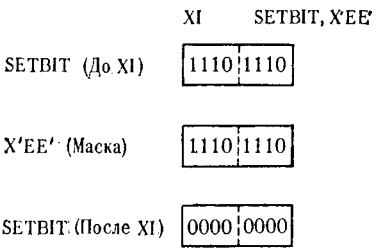
Пример 1.



Пример 2.



Пример 3.



Исключающее ИЛИ — X. В команде X в качестве первого операнда указывается регистр, а в качестве второго операнда — полное слово памяти. Предполагается, что в полном слове второго операнда должна находиться маска, которая участвует в модифицированном сложении с содержимым общего регистра первого операнда. Длина обрабатываемых командой X данных фиксированная и равна длине регистра (исходное поле) или длине слова (маска).

В полном слове маски, расположенном по адресу FULLMASK, находится шестнадцатеричная конфигурация FF000000. Кроме того, в команде используется общий регистр 3. Во время выполнения этого примера в регистре 3 находилась величина X'EE783D01'. Команда и содержимое регистра 3 имеют следующий вид:

X 3,FULLMASK

Регистр 3 (До)	1110 1110	0111 1000	0011 1101	0000 0001
FULLMASK	1111 1111	0000 0000	0000 0000	0000 0000
Регистр 3 (После)	0001 0001	0111 1000	0011 1101	0000 0001

Значащие биты старшего байта маски вызвали изменение значения всех соответствующих битов регистра 3 на обратное. Нулевые биты маски не повлияли на значение соответствующих битов регистра 3.

Исключающее ИЛИ — XC. Команда XC имеет формат память — память. Как маска, так и поле данных являются областями памяти, длина которых не превосходит 256 байтов, причем оба поля должны иметь равные указатели длины, выраженные в явной или неявной форме.

Пример выполнения этой команды представляет собой программу преобразования алфавитно-цифровых символов кода EBCDIC. Любая буква от A до I преобразуется в цифровой символ от 1 до 9 или любая цифра кода EBCDIC от 1 до 9 преобразуется в буквенный символ от A до I.

Таблица преобразований выглядит следующим образом:

Из	A	в	1	или	из	1	в	A
Из	B	в	2	или	из	2	в	B
Из	C	в	3	или	из	3	в	C
Из	D	в	4	или	из	4	в	D
Из	E	в	5	или	из	5	в	E
Из	F	в	6	или	из	6	в	F
Из	G	в	7	или	из	7	в	G
Из	H	в	8	или	из	8	в	H
Из	I	в	9	или	из	9	в	I

Если обрабатывается буквенный символ, отличный от символов из диапазона от A до I, то он будет преобразован сле-

дующим образом:

Из	J	в	«пробел» (X'E1')					
Из	K	в.	S	или	из	S	в	K
Из	L	в	T	или	из	T	в	L
Из	M	в	U	или	из	U	в	M
Из	N	в	V	или	из	V	в	N
Из	O	в	W	или	из	W	в	O
Из	P	в	X	или	из	X	в	P
Из	Q	в	Y	или	из	Y	в	Q
Из	R	в	Z	или	из	Z	в	R

Маска, которая должна быть использована для преобразования, хранится в шестибайтовой области памяти KEYBITS, а содержимым ее является X'303030303030'.

KEYBITS	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000

Преобразуемые в этом примере данные представляют собой два шестибайтовых поля с метками TRANSA и TRANSB:

TRANSA	A	H	I	J	2	5

TRANSB	1	3	8	J	K	L

Выполняются следующие команды:

```
XC      TRANSA,KEYBITS
XC      TRANSB,KEYBITS
```

Команды XC будут преобразовывать данные исходного поля следующим образом:

TRANSA (Символьн.) (До)	A		H		I		J		2		5	
(Шестн.)	C	1	C	8	C	9	D	1	F	2	F	5
(Двоичн.)	1100	0001	1100	1000	1100	1001	1101	0001	1111	0010	1111	0101

KEYBITS (Маска)	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000

TRANSA (После)	(Двоичн.)	1111	0001	1111	1000	1111	1001	1110	0001	1100	0010	1100	0101
	(Шестн.)	F	1	F	8	F	9	E	1	C	2	C	5
	(Символьн.)	1		8		9		пробел		B		E	

TRANSB (Символьн.) (До)

	1		3		8		J		K		L	
(Шестн.)	F	1	F	3	F	8	D	1	D	2	D	3
(Двоичн.)	1111	0001	1111	0011	1111	1000	1101	0001	1101	0010	1101	0011

KEYBITS (Маска)	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000	0011	0000

TRANSB (Двоичн.) (После)

	1100	0001	1100	0011	1100	1000	1110	0001	1110	0010	1110	0011
(Шестн.)	C	1	C	3	C	8	E	1	E	2	E	3
(Символьн.)	A		C		H		пробел		S		T	

Если теперь соединить два поля данных, TRANSA и TRANSB, то их объединенным содержимым будет

1	8	9	¢	B	E	A	C	H	¢	S	T

Многие организации имеют массивы информации, которые считаются секретными. Создаются различные системы, от очень сложных до крайне простых, для шифровки данных, содержащихся в этих массивах. Если не уполномоченный на то персонал случайно получает доступ к этим массивам, то информация предстает перед ним в виде бесполезных массивов байтов данных, пока к ним не будет предоставлен специальный «ключ». Только что приведенный пример мог бы быть использован в такого рода задаче — произошло не только изменение состояния третьего и четвертого битов каждого байта на обратное, но также единая логическая запись были разделена на сегменты и расположена в разных местах памяти.

Исключающее ИЛИ — XR. Команда XR имеет формат регистр — регистр. Все 32 бита регистра маски складываются с 32 битами регистра исходного поля. Биты исходного поля данных изменяются, когда в соответствующей позиции бита маски содержится единичный бит. Оба операнда команды должны находиться в общих регистрах.

В следующей иллюстрации рассматриваемой команды исходное поле представлено регистром 8, содержимое которого X'007A91BF', а маска находится в регистре 12, содержащем

X'96969696'. Выполняется команда:

```
                                  XR      8,12

Регистр 8  (Шестн.)          | 0 | 0 | 7 | A | 9 | 1 | B | F |
(До)       (Двоичн.)         |0000|0000|0111|1010|1001|0001|1011|1111|

Регистр 12 (Маска)           |1001|0110|1001|0110|1001|0110|1001|0110|

Регистр 8  (Двоичн.)         |1001|0110|1110|1100|0000|0111|0010|1001|
(После)    (Шестн.)          | 9 | 6 | E | C | 0 | 7 | 2 | 9 |
```

B. ПОИСК ДАННЫХ С ПОМОЩЬЮ КОМАНД БУЛЕВОЙ ЛОГИКИ

Так как машинное хранение, обработка и выборка информации в настоящее время применяются во все увеличивающихся масштабах, роль правильного выбора носителя информации постоянно возрастает. Даже только в промышленности размеры памяти, требуемой для записи информации о каждом сотруднике, могут составлять тысячи байтов. Уже давно принято хранить данные в символьной форме кода EBCDIC, причем для каждого раздела информации предусмотрено специальное поле, которое идентифицируется по типу хранимой информации. В некоторых полях может находиться один символ, представляющий информацию в явном виде. Например, для определения семейного положения используется однобайтовое поле, в котором может содержаться символ M для женатых или S для одиноких [1]). Даже эта сокращенная форма записи данных требует значительных размеров памяти.

Манипуляции с битами посредством команд булевой алгебры создают новые возможности как для проектировщиков систем, так и для программистов. Информация, которая первоначально требовала 1 байт (или даже больше) памяти на магнитной ленте или диске, теперь может быть отражена в одной позиции бита. В одном восьмиразрядном байте данных можно отразить до 16 различных состояний, или ответов.

Для иллюстрации такой возможности в следующих примерах используется байт данных опроса. Записываемые данные каждого принимаемого на работу компанией «К» представляют собой один байт, в котором содержится следующая информация:

---

[1]) Первые буквы слов married — женатый и single — одинокий. — *Прим. ред.*

Позиция бита	Информация	Единичный бит означает	Нулевой бит означает
1	Пол	Мужской	Женский
2	Семейное положение	Женат	Холост
3	Возрастная группа	Больше 25 лет	До 25 лет
4	Средняя школа	Окончена	Не окончена
5	Колледж (4 года)	Есть диплом	Нет диплома
6	Водительские права	Есть	Нет
7	Домовладелец	Да	Нет
8	Гражданство	С рождения	Получил гражданство

Если в таком массиве нужно найти сотрудников, удовлетворяющих некоторому набору критериев, то это можно сделать за гораздо меньшее время по сравнению с обработкой множества полей символов по каждой отдельной записи. При этом не только используется поле минимальной длины (1 байт), но также имеется возможность проверить набор условий в этом 1 байте с помощью одной команды. Тип информации, приведенный в этом 1 байте данных опроса, представляется весьма общим. При необходимости он может быть изменен и расширен так, что будет удовлетворять требованиям практически любой справочной системы.

В каждом приводимом в настоящей главе примере отыскивается соответствующая группа лиц, отвечающая предъявленным требованиям. Данные о каждом подходящем человеке вместе с другими необходимыми данными извлекаются из системы или используются для накопления статистики для последующего анализа. В связи с тем что команды булевой логики при опросе байта разрушают его, этот байт сначала пересылается в рабочую область, а только потом обрабатывается.

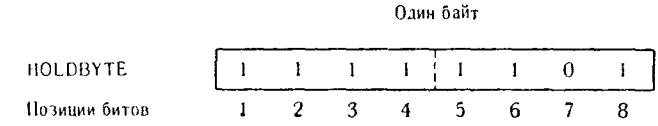
Пример 1. Администрации компании «К» требуется список всех сотрудников мужского пола 25 лет или старше, окончивших школу, но не имеющих дипломов колледжей. Компания планирует использовать список для извещения этого персонала о получении новой программы для помощи обучающимся в колледже. Комбинация состояния битов, по которой должен проводиться поиск, следующая:

Бит 1	Единица	Мужчина
Бит 3	Единица	25 лет и более
Бит 4	Единица	Средняя школа закончена
Бит 5	Нуль	Нет диплома об окончании колледжа

В этом примере используются следующие команды:

```
MVC     HOLDBYTE(1),DATA+3      001
NI      HOLDBYTE,X'B8'          002
XI      HOLDBYTE,X'B0'          003
BC      8,HIT                   004
BC      15,MISS                 005
```

Для того чтобы показать результат выполнения этих команд, предположим, что в проверяемом байте содержится следующая конфигурация:

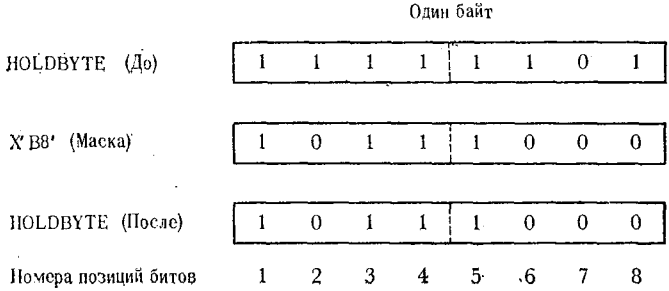


Эти состояния битов сообщают о сотруднике следующее:

Бит 1	Единица	Мужчина
Бит 2	Единица	Женат
Бит 3	Единица	25 лет или старше
Бит 4	Единица	Окончил школу
Бит 5	Единица	Имеет диплом колледжа
Бит 6	Единица	Есть права водителя
Бит 7	Нуль	Не является домовладельцем
Бит 8	Единица	Гражданство с рождения

Предложение 001 пересылает опрашиваемый байт из области хранения в рабочую область HOLDBYTE.

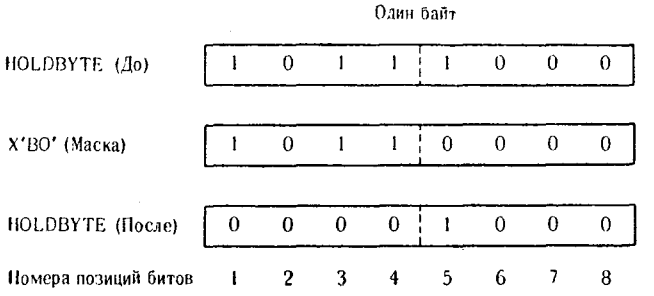
Предложение 002 содержит команду NI. В маске единичные биты находятся в позициях, соответствующих проверяемым позициям опрашиваемого байта.



Команда засылает нули во все позиции битов, опрос которых не представляет интереса. Маска X'B8' сохраняет состояние

битов исходного поля, позициям которых соответствовали единицы в маске.

Предложение 003 выполняет команду XI, используя в качестве маски X'B0'.



Значение маски X'B0' явится причиной того, что результатом будет нуль, если рассматриваемые позиции битов поля HOLDBYTE, до выполнения команды XI были установлены в искомое состояние. Сказанное выше основывается на следующем:

Позиции битов	Должны быть	Означает	Маска	Желаемый результат после XI
1	1	Мужчина	1	0
3	1	25 лет или больше	1	0
4	1	Окончил школу	1	0
5	0	Нет диплома колледжа	0	0

Однако в этом примере результат не нулевой — сотрудник, чья запись данных проверялась, имеет диплом об окончании колледжа. Поэтому при выполнении команды XI пятый бит поля HOLDBYTE и соответствующий нулевой бит маски дадут в результате единичный бит.

В предложении 004 проверяется полученный при выполнении команды XI признак результата. Если результатом является нуль в HOLDBYTE, то происходит переход.

Предложение 005 является безусловным переходом к программе MISS; при этом подразумевается, что личная запись не удовлетворяет этому специальному набору критериев.

Пример 2. В этом случае администрация компании «К» выясняет общее число всех неженатых служащих компании в

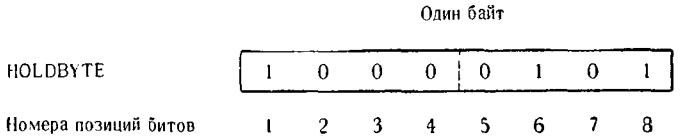
возрасте до 25 лет, не окончивших школу и всегда являвшихся гражданами данной страны. Объединенные вместе эти характеристики потребуют просмотра следующих состояний битов:

Бит 2	Нуль	Холост
Бит 3	Нуль	Меньше 25 лет
Бит 4	Нуль	Не закончил школу
Бит 8	Единица	Подданство не менял

Выполняются следующие команды:

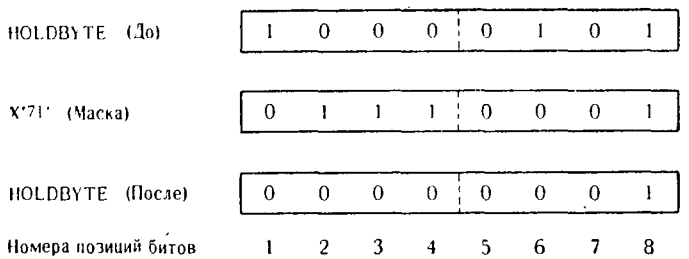
```
        MVC   HOLDBYTE(1),DATA+3    001
        NI    HOLDBYTE,X'71'        002
        XI    HOLDBYTE,X'01'        003
        BC    4,EXIT                004
        AP    COUNT,=PL1'1'         005
EXIT    BC    15,NEXTCHEK           006
```

В этом примере в поле HOLDBYTE пересылается байт со следующим содержимым:

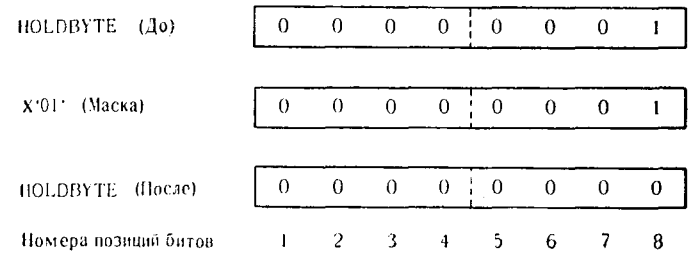


Предложение 001 пересылает опрашиваемый байт из четвертого байта поля DATA в рабочую область HOLDBYTE.

Предложение 002 содержит команду NI. Конфигурация маски обеспечит занесение нулей в позиции 1, 5, 6 и 7 исходного поля, но не изменит значения битов в позициях 2, 3, 4 и 8.



Предложение 003 содержит команду XI. Маска X'01' вызовет следующее изменение содержимого рабочей области:



Алгоритмом предусмотрена проверка позиций 2, 3 и 4 на нуль. Для выявления этого состояния в эти позиции маски были помещены нули. Если в одной из этих позиций в исходном поле содержалась единица, то единица появится и в результате. Так как в соответствии с формулировкой задачи должна проверяться и восьмая позиция HOLDBYTE на наличие в ней единицы, то и в маске в этой позиции также находится единица. Если единица присутствовала в исходном байте, то после выполнения команды XI в эту позицию результата будет помещен нуль. По завершении команды XI в HOLDBYTE содержатся только нули, что указывает на выполнение необходимых условий.

Предложение 004 содержит команду условного перехода. Если в поле результата содержится ненулевое значение, происходит переход к предложению EXIT. В этом примере поле результата нулевое и переход не происходит.

В предложении 005 предполагается, что так как переход в предложении 004 не был сделан, то в HOLDBYTE должен был находиться нуль. Это правильное предположение. Поэтому команда AP прибавляет к счетчику упакованную десятичную константу +1 для указания того, что отыскиваемая запись обнаружена.

Предложение 006 является безусловным переходом к точке программы с меткой NEXTCHEK. Эта подпрограмма, вероятно, загрузит другую запись данных и снова вернется к выполнению предложений от 001 до 006. Алгоритм программы может привести к предложению 006 одним из двух путей — либо будет сделан переход из предложения 004 непосредственно в предложение 006, но в этом случае значение в поле COUNT не будет увеличено на число +1, либо будут последовательно выполняться предложения 004 и 005.

Можно привести еще много примеров такого рода. Для хранения информации в двоичной поразрядной форме могут быть использованы 2, 3, 4 или еще больше байтов. Посчитайте эффективность экономии размеров хранимых данных, если 4 байта (32 двоичных разряда) могут содержать столько же информации, сколько содержится в 32 байтах, — экономится 28 байтов на каждую запись. Представьте, что на предприятии работает

15 000 служащих. В этом случае экономия составит 420 000 (28 × 15 000) байтов памяти только на 1 файл. Даже хотя некоторые носители и методы хранения данных относительно дешевы, машинное время стоит дорого. Фактическая экономия времени процессора, затраченного в приведенных примерах, приблизительно определяется отношением 1 к n, где n — число команд сравнения, необходимых для выполнения той же задачи, но с использованием команд сравнения данных в символьной форме кода EBCDIC.

Упражнения

1. Принято считать, что команды булевой логики наиболее подходят для работы с отдельными _____ или группами _____.

2. Команда _____ применяется для установки двоичных переключателей в состояние «выключено».

3. Группа команд ИЛИ реализует _____, при котором 1 + 1 = 1 и 1 + 0 = 1.

4. Выполнение команд Исключающее ИЛИ формирует результат, в котором _____ биты получены при условии, что в соответствующих позициях маски и поля источника находятся одинаковые биты, а разные по значению биты в соответствующих позициях маски и поля источника сформировали _____ биты.

5. До 2048 позиций битов могут быть изменены при использовании одного из вариантов команды Исключающее ИЛИ, мнемоника которой _____.

6. Во всех командах булевой логики под содержимым второго операнда подразумевается _____ независимо от того, что это символ, регистр или область памяти.

7. При выполнении любой из команд ИЛИ в поле результата можно обеспечить присутствие единичного бита при помощи записи бита, равного _____, в соответствующую позицию маски, представляемой вторым операндом.

8. Команда, мнемоника которой X, воздействует на каждую из _____позиций битов первого операнда.

9. Выполняется команда И: единичный бит маски и соответствующий единичный бит исходного поля дадут в результате _____ бит в исходном поле, а нулевой бит маски и соответствующий единичный бит исходного поля дадут в результате _____ бит.

10. Для команды O необходимо, чтобы первый операнд команды определялся как _____, а второй операнд являлся правильно выравненным _____ памяти.

11. Мнемоника XI указывает, что первый операнд является одним байтом памяти, а вторым операндом является _____ символ.

12. Команды группы И выполняют операцию логического _____ над битами данных.

13. Команда NC, одна из команд группы _____ использует в качестве обоих операндов заданные области памяти и может работать с полями данных от 1 до _____ байтов.

14. Формат команды NR — _____ регистр.

15. Мнемоника OR указывает, что первым операндом команды должен быть _____ и вторым операндом команды должен быть _____.

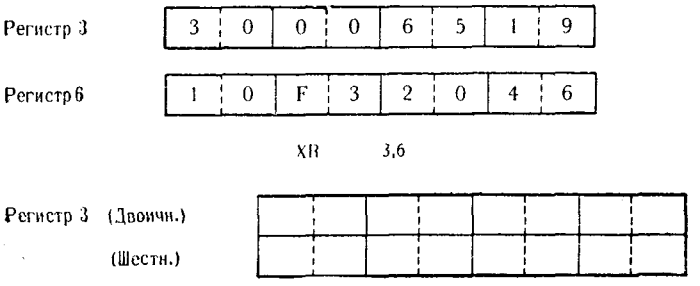
16. Логические команды группы ИЛИ используются при работе с двоичными переключателями, так как они могут обеспечить установку битов двоичного переключателя в состояние _____.

17. При использовании команды ИЛИ для выполнения некоторой операции _____ бит маски (второй операнд) обеспечивает неизменность состояния соответствующей позиции исходного поля.

18. При выполнении команды Исключающее ИЛИ _____ бит маски (второй операнд) вызывает изменение состояния соответствующей позиции исходного поля на противоположное; единичный бит изменится на нулевой, нулевой — на единичный.

В каждой из следующих задач присутствует одно или несколько исходных полей и задано значение маски. Примените к исходному полю маску, используя логические команды, и заполните исходные поля как в шестнадцатеричном, так и в двоичном формате содержимым, получаемым при выполнении команд.

19.



20.

Регистр 8	0	F	3	9	2	F	F	F
FULLWRD3	1	1	3	1	0	0	6	C
FULLWRD4	0	0	4	2	7	7	D	3

```
L     8,FULLWRD4
O     8,FULLWRD3
```

Регистр 8 (Двоичн.)

(Шестн.)

21.

DATAFLD	D	1	D	2	E	3

```
NI    DATAFLD+1,X'CF'
```

DATAFLD (Двоичн.)

(Шестн.)

22.

DATATEST	D	C	E	7	E	0	3	C	E	9
MASK22	4	0	2	4	1	8	1	4	2	4

```
XC    DATATEST(3),MASK22+2
```

DATATEST (Двоичн.)

(Шестн.)

23.

Регистр 12	0	0	C	3	5	6	9	8
WORDTWO	0	0	0	4	0	4	0	4

```
O     12,WORDTWO
```

Регистр 12 (Двоичн.)

(Шестн.)

24.

DATASET

C	7	C	8	D	7	D	9	C	1

MASK1

0	2	0	2	0	2

```
NC      DATASET+2(3),MASK1
```

DATASET (Двоичн.)

(Шестн.)

25.

CHKFLD

2	D	8	C	C	1	E	6	F	3

```
          LA      6,3
DATALOOP  LA      7,CHKFLD+1
          XI      0(7),X'33'
          AH      7,=H'1'
          BCT     6,DATALOOP
```

(Двоичн.)

(Шестн.)

26.

FLDSET

C	3	6	B	4	B	9	C

```
          OI      FLDSET,X'21'
          OI      FLDSET+2,X'12'
```

FLDSET (Двоичн.)

(Шестн.)

27.

Регистр 4

F	F	3	9	2	6	7	8

FWORD

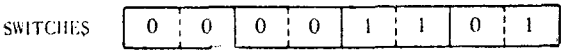
0	0	F	F	F	F	F	F

```
          N       4,FWORD
```

Регистр 4 (Двоичн.)

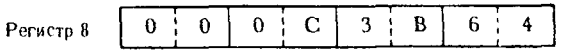
(Шестн.)

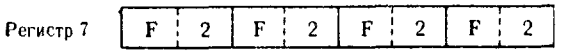
28.



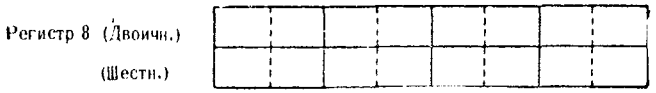

```
OI      SWITCHES+1, X'2A'
OI      SWITCHES+2, X'5E'
```

29.

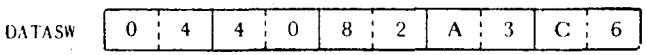





```
NR      8, 7
```

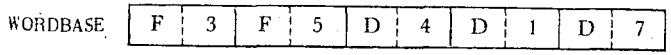


30.




```
OC      DATASW(2), DATASW+2
```

31.




```
NC      WORDBASE+2(2), SETCHNGE+1
```

32.

Регистр 6	8	2	8	7	8	5	8	8
Регистр 10	4	0	4	0	4	0	4	0

OR 6, 10

Регистр 6 (Двоичн.)								
(Шестн.)								

33.

Регистр 5	F	F	F	F	F	F	F	3
FULLWD3	F	F	F	F	F	F	F	F

X 5, FULLWD3

Регистр 5 (Двоичн.)								
(Шестн.)								

Глава 13

Использование переключателей и индикаторов в программировании

А. КОМАНДА «ПРОВЕРИТЬ ПО МАСКЕ»

Команда: Test under Mask — TM (Проверить по маске)

Мнемоника	Код операции	Формат операндов
TM	91	$D_1(B_1),I_2$

При выполнении этой команды в качестве второго операнда используется непосредственный байт данных. Его применяют для анализа конфигурации нулей и единиц в байте данных, который адресован первым операндом. Конфигурация нулей и единиц в анализируемом байте не изменяется. Программист формирует второй операнд, играющий роль маски, по следующим правилам:

1. Если бит данных подлежит проверке, то соответствующий ему бит маски должен быть установлен в единицу.
2. Если бит данных проверке не подлежит, бит маски должен быть установлен в нуль.

Команда проверяет лишь те биты поля данных, которым соответствуют единичные биты маски. При выполнении команды формируется рабочий байт по образцу адресованного байта данных. Каждый единичный бит маски вызывает проверку бита, занимающего такую же позицию в байте данных (в рабочем байте). Если при этом оказывается, что проверяемый бит равен единице, то соответствующий бит в рабочем байте устанавливается в единицу, т. е. остается в прежнем состоянии. Если же оказывается, что проверяемый бит равен нулю, то бит рабочего байта устанавливается в нуль. В завершение анализа команда устанавливает признак результата, значение которого зависит от окончательного содержимого рабочего байта. Данная команда позволяет программисту анализировать состояние «включен/выключен» любого одноразрядного переключателя или конфигурации нулей и единиц целого байта данных, не изменяя ее при этом.

CC	BC	Условие
0	8	Все проверенные биты равны 0 [1])
1	4	Среди проверенных битов имеются как равные 0, так и равные 1
3	1	Все проверенные биты равны 1

Б. ПОНЯТИЕ О ПЕРЕКЛЮЧАТЕЛЯХ И ИНДИКАТОРАХ

В условиях применения вычислительной техники, которая предшествовала системам третьего поколения, упоминание о переключателе или индикаторе скорее всего воспринималось как относящееся к физическому устройству. Работа проблемной программы могла зависеть от положения переключателя на пульте управления машиной. Только при определенном положении переключателя программа могла выполнить некоторые свои функции. Конечно, физические переключатели существуют и в Системе/360, но здесь они чаще связаны с функционированием периферийного оборудования, как, например, двухканальный переключатель на блоке управления устройством с прямым доступом или лентопротяжкой. Проблемная программа не может управлять состоянием переключателей этого типа, более того, информация о их состоянии непосредственно не влияет на работу такой программы.

Можно считать, что рассматриваемые в этом разделе переключатели и индикаторы существуют в форме символов или данных в памяти и что состояния этих переключателей представляют собой выбор одной из двух возможностей, таких, как «да» или «нет», «включено» или «выключено», «переход» или «продолжение», а также и других альтернатив в случаях, когда имеются вопросы, требующие одного ответа из двух возможных.

К сожалению, по поводу практического применения переключателей в программах часто обнаруживаются значительные расхождения во мнениях. Небрежное использование логических переключателей может вызвать нарушения в проблемной программе и довольно часто до такой степени, что становится крайне трудно указать неверную или пропущенную команду. Отсутствие в нужном месте одной команды, по которой переключатель должен быть установлен в состояние «включено» или «выключено», может повлечь за собой полное искажение задуманного алгоритма. С другой стороны, при соблюдении соответствующих программных соглашений и точном составлении блок-схемы можно было бы вместо целой подпрограммы или поля данных использовать один переключатель.

[1]) Или все биты маски равны нулю. — *Прим. ред.*

Установка переключателя в состояние «включено» или «выключено» сама по себе не вызывает ожидаемого в этом случае действия проблемной программы. Состояние переключателя устанавливается для последующей его проверки и выбора решения в зависимости от результата проверки и алгоритма программы. В данном разделе будут подробно рассмотрены и проиллюстрированы способы установки и проверки переключателей и индикаторов.

Как правило, переключатели и индикаторы бывают двух типов: символьные переключатели и битовые переключатели.

Каждому типу присущи свои характеристики и функции, хотя области их применения могут часто пересекаться. В соответствии с целями этой книги каждый тип рассматривается в связи с областью, в которой он чаще всего применяется. Сходство в особенностях применения будет само по себе, без дополнительного обсуждения свидетельствовать о их взаимозаменяемости.

1. Символьные переключатели и индикаторы

Символьный переключатель или индикатор занимает 1 байт данных, поэтому его можно называть байтовым переключателем. Поскольку 1 байт данных может представлять собой любую из 256 различных конфигураций нулей и единиц, постольку можно считать, что байтовый или символьный переключатель имеет 256 различных состояний. Однако на практике обычно считается, что байтовый переключатель имеет только два состояния. Из этого не следует, что пользоваться переключателем с большим числом состояний нецелесообразно, но обычно оказывается, что достаточно двух состояний.

Так как символьный переключатель занимает, по определению, 1 байт данных, то для установки и проверки переключателя логично использовать команды, которые обрабатывают только 1 байт. Такие команды называются командами с непосредственным операндом. Для установки байтового переключателя в нужное состояние идеально подходит команда MVI (Пересылка непосредственная), а для проверки байтового переключателя удобно использовать команду CLI (Сравнение непосредственное). Обе эти команды, MVI и CLI, значительно удобнее, чем обычные команды пересылки и сравнения. Второй операнд этих команд представляет собой самоопределенную величину, и команда в целом выполняется значительно быстрее, чем если бы второй операнд определял адрес, который в свою очередь служил бы для обращения к символу.

Обычно для использования переключателя в первую очередь задают константу, которой присваивают имя. Так как исходное

значение переключателя, прежде чем он будет использован, должно быть нулевым, целесообразно формировать переключатель как однобайтовую константу, содержащую код пробела. Это может быть сделано следующим образом:

```
SWITCH1    DC    CL1'␣'
```

или

```
SWITCH1    DC    XL1'40'
```

Символ ␣ здесь, как и в других местах книги, обозначает пробел, кодируемый в шестнадцатеричном представлении как 40.

Имея в виду, что переключатель обычно используется для представления состояний «включено» и «выключено», можно условиться, что символ X будет обозначать состояние «включено», а пробел — состояние «выключено». Чтобы по желанию изменить состояние переключателя, будут использоваться две команды:

```
MVI    SWITCH1,C'X'
```

или

```
MVI    SWITCH1,C'␣'
```

Первая команда пересылает символ X в однобайтовое поле SWITCH1. Это действие рассматривается как включение переключателя. Вторая команда пересылает пробел (шестнадцатеричное 40) в SWITCH1, выключая переключатель.

Хотя программа, представленная на рис. 13.1, является сама по себе искусственным примером, тем не менее она демонстрирует применение байтового переключателя: включение, проверку состояния и выключение. Согласно алгоритму программы, переключатель сначала устанавливается в состояние «включено», затем выполняется некоторое количество команд, после чего производится проверка, находится ли переключатель в состоянии «включено», и, если это так, производится переход к подпрограмме анализа вычисленного значения в некотором упакованном десятичном поле. Если это значение равно +2999, то сохраняется естественный порядок выполнения команд, в соответствии с которым затем выполняется предложение OFFIT. Эта команда устанавливает переключатель в состояние «выключено», после чего производится возврат к REPEAT, обработка, анализ переключателя командой с именем ONOFF и обнаруживается, что переключатель выключен (не равен X). В этом случае сохраняется естественный порядок выполнения команд, в соответствии с которым выполняется команда, вызывающая переход к предложению END. В этом частном примере можно было бы избежать применения переключателя: предложение CHECKIT могло бы заменить предложение ONOFF, устраняя необходимость использования всей подпрограммы CHECKIT,

IBM

IBM System/360 Assembler Coding Form

X28-6606-3 U/M360
Printed in U.S.A.

PROGRAM		PUNCHING INSTRUCTIONS	GRAPHIC								PAGE OF
PROGRAMMER	DATE		PUNCH								CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
TURNON	MVI	SWITCH1,C'X'		0040
REPEAT	AP	FIELD,=PL1'1'		0041
	. .	. .		
	. .	. .		
	. .	. .		
ONOFF	CLI	SWITCH1,C'X'		0063
EQUALB	BC	8,CHECKIT		0064
	BC	15,END		0065
*				
*				
*				
CHECKIT	CP	FIELD,=PL3'2999'		0066
	BC	7,REPEAT		0067
OFFIT	MVI	SWITCH1,C'0'		0068
	BC	15,REPEAT		0069
*				
*				

Рис. 13.1.

Пример в этом случае выглядел бы следующим образом:

```
REPEAT   AP      FIELD,=PL1'1'
         — —     — — — — — —
         — —     — — — — — —
         — —     — — — — — —
         CP      FIELD,=PL3'3000'
         BC      7,REPEAT
         BC      15,END
```

Выше упоминалось, что байтовый переключатель может принимать более чем два состояния. Рассмотрим пример, в котором проблемная программа осуществляет ввод записей трех типов: записей типа 1, записей типа 2 и записей типа 3. Когда проблемная программа вводит каждую новую запись в память, в переключателе устанавливается конфигурация нулей и единиц, соответствующая числам 1, 2 или 3. Согласно алгоритму, позднее возникает необходимость по значению величины, содержащейся в переключателе, определить тип обрабатываемой записи и перейти к одной из трех подпрограмм. Команда сравнения анализирует переключатель следующим образом:

```
         CLI     SWITCH,C'2'
         BC      4,RECORD1   (число в переключателе
                             меньше чем 2)
         BC      2,RECORD3   (число в переключателе
                             больше чем 2)
RECORD2  MVC     FLDB,FLDA   (число в переключателе
                             равно 2)
```

Первая команда сравнивает байт поля SWITCH с символом 2. Следующая команда выполняет переход к подпрограмме RECORD1, если сравниваемое содержимое поля SWITCH меньше чем 2. Если сравниваемое содержимое поля SWITCH больше чем 2, следующая команда вызывает переход к подпрограмме RECORD3. Если переход не произошел при выполнении второй и третьей команд, это означает, что при сравнении обнаружилось равенство (поле SWITCH содержит 2), и тогда дальнейшая обработка производится программой RECORD2.

Иное применение переключателей имеет место в многоцелевой программе, которая предоставляет пользователю возможность выбирать средства вывода информации. Пусть в этом примере проблемная программа производит вывод на все три устройства: накопитель на магнитной ленте, устройство перфорации карт и печатающее устройство — или на любое из них. При вводе проблемной программы в машину пользователь должен

также ввести управляющую карту, указывающую желаемый тип вывода. На этой управляющей карте должен стоять символ X в колонке 5 — для вывода на магнитную ленту, X в колонке 6 — для вывода на перфокарты и X в колонке 7 — для вывода на печать. Одна из первых подпрограмм должна считать эту управляющую карту и установить переключатели вывода, которые указаны пользователем. Эта подпрограмма может быть составлена так, как показано на рис. 13.2.

Предложение 001 является стандартной макрокомандой метода доступа QSAM, которая считывает управляющую карту в область данных CARD.

Предложение 002 проверяет, имеется ли символ X на карте в колонке 5 (CARD+4). Если X отсутствует, то предложение 003 вызывает переход к предложению 005 (PASS1).

Предложение 004 устанавливает переключатель, указывающий, что требуется вывод на ленту.

Предложение 005 проверяет, имеется ли символ X на карте в колонке 6 (CARD+5). Если X отсутствует, то предложение 006 вызывает переход к предложению 008 (PASS2) и предложение 007 не выполняется.

Предложение 007 устанавливает переключатель, указывающий, что требуется вывод на перфокарты.

Предложение 008 проверяет, имеется ли символ X на карте в колонке 7 (CARD+6). Если X отсутствует, то предложение 009 вызывает переход к предложению 011, PASS3. Предложение 010 в этом случае не выполняется.

Предложение 010 устанавливает признак вывода данных на печать.

Теперь, когда переключатели установлены, программа выполняется до тех пор, пока не наступит пора производить вывод. Команды для проверки переключателей и для перехода к соответствующим подпрограммам вывода могут быть составлены так, как показано на рис. 13.3.

Предложения 001, 004 и 007 проверяют три переключателя, чтобы определить, в какой форме должен производиться вывод. Если переключатель не установлен в состояние «включено», последующая команда вызывает переход к предложению, которое проверяет следующий переключатель. Условие, при котором должен произойти переход, выражается следующим образом:

```
BC      7, . . .
```

что означает: «Если предшествующая команда сравнения обнаруживает неравенство, то происходит переход...». Если команда CLI обнаруживает равенство, означающее, что для данного типа вывода переключатель установлен, то сохраняется естественный порядок команд и после команды BC выполняется команда

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT

Name	Operation	Operand	Comments	Sequence
*				
READCON	GET	CARDER,CARD		001
	CLI	CARD+4,C'X'		002
	BC	7,PASS1		003
	MVI	SWTAPE,C'T'		004
PASS1	CLI	CARD+5,C'X'		005
	BC	7,PASS2		006
	MVI	SWPUNCH,C'P'		007
PASS2	CLI	CARD+6,C'X'		008
	BC	7,PASS3		009
	MVI	SWPRINT,C'W'		010
PASS3	- -	- - - - - -		011
	- -	- - - - - -		

Рис. 13.2.

IBM

IBM System 360 Assembler Coding Form

X28-6506-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
WHATOUT	CLI	SWTAPE,C'T'		001
	BC	7,SKIP1		002
	BAL	10,MAGTAPE		003
SKIP1	CLI	SWPUNCH,C'P'		004
	BC	7,SKIP2		005
	BAL	10,PUNCHER		006
SKIP2	CLI	SWPRINT,C'W'		007
	BC	7,SKIP3		008
	BAL	10,PRINTER		009
SKIP3	- -	- - - - - -		010
	- -	- - - - - -		011

Рис. 13.3.

BAL (Переход с возвратом). Команда BAL, в свою очередь, обеспечивает возврат к команде, следующей за BAL, после выполнения подпрограммы, к которой осуществляется переход.

Другой способ загрузки, или установки, переключателей этого типа состоит в том, что содержимое управляющей карты пересылается прямо в область, отведенную для переключателей. Пусть, как в предыдущем примере, для указания способа вывода колонки 5, 6 и 7 управляющей карты содержат некоторую комбинацию символов X и пробелов. Соответственно три переключателя располагаются в памяти один за другим, а именно,

```
SWTAPE    DC    CL1'Ь'
SWPUNCH   DC    CL1'Ь'
SWPRINT   DC    CL1'Ь'
```

Чтобы в соответствии с содержимым управляющей карты установить переключатель в состояние «включено» или оставить его в состоянии «выключено», необходимы следующие предложения:

```
READCON   GET   CARDER,CARD
          MVC   SWTAPE(3),CARD+4
```

Второе из этих предложений пересылает содержимое колонок 5, 6 и 7 карты (X или пробел в каждой колонке) в поля SWTAPE, SWPUNCH и SWPRINT. Следует отметить, что в этом случае мы обошлись двумя предложениями по сравнению с десятью предложениями, потребовавшимися для установки переключателей в предыдущем примере. Еще одно и последнее отличие заключается в том, что когда получает управление программа, проверяющая переключатели, то проверяется равенство каждого переключателя не своему особому символу, а одному и тому же для всех переключателей символу X.

По мере роста мастерства программиста он найдет много дополнительных приемов использования байтовых переключателей. Пожалуй, наиболее слабым местом у программистов, пытающихся пользоваться переключателями, является небрежность при их очистке, т. е. при выключении переключателей в нужное время и на нужном этапе выполнения программы.

2. Битовые переключатели и индикаторы

В то время как символьный переключатель занимает целый байт и может находиться во многих состояниях, битовый переключатель занимает только один двоичный разряд и может находиться самое большее в двух состояниях. С другой стороны, 1 байт данных может содержать восемь отдельных битовых переключателей — по одному на каждый разряд.

IBM

IBM System 360 Assembler Coding Form

X28-6500-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
WHATOUT	CLI	SWTAPE,C'T'		001
	BC	7,SKIP1		002
	BAL	10,MAGTAPE		003
SKIP1	CLI	SWPUNCH,C'P'		004
	BC	7,SKIP2		005
	BAL	10,PUNCHER		006
SKIP2	CLI	SWPRINT,C'W'		007
	BC	7,SKIP3		008
	BAL	10,PRINTER		009
SKIP3	- -	- - - - - -		010
	- -	- - - - - -		011

Рис. 13.3.

BAL (Переход с возвратом). Команда BAL, в свою очередь, обеспечивает возврат к команде, следующей за BAL, после выполнения подпрограммы, к которой осуществляется переход.

Другой способ загрузки, или установки, переключателей этого типа состоит в том, что содержимое управляющей карты пересылается прямо в область, отведенную для переключателей. Пусть, как в предыдущем примере, для указания способа вывода колонки 5, 6 и 7 управляющей карты содержат некоторую комбинацию символов X и пробелов. Соответственно три переключателя располагаются в памяти один за другим, а именно,

```
SWTAPE    DC    CL1'Ь'
SWPUNCH   DC    CL1'Ь'
SWPRINT   DC    CL1'Ь'
```

Чтобы в соответствии с содержимым управляющей карты установить переключатель в состояние «включено» или оставить его в состоянии «выключено», необходимы следующие предложения:

```
READCON   GET   CARDER,CARD
          MVC   SWTAPE(3),CARD+4
```

Второе из этих предложений пересылает содержимое колонок 5, 6 и 7 карты (X или пробел в каждой колонке) в поля SWTAPE, SWPUNCH и SWPRINT. Следует отметить, что в этом случае мы обошлись двумя предложениями по сравнению с десятью предложениями, потребовавшимися для установки переключателей в предыдущем примере. Еще одно и последнее отличие заключается в том, что когда получает управление программа, проверяющая переключатели, то проверяется равенство каждого переключателя не своему особому символу, а одному и тому же для всех переключателей символу X.

По мере роста мастерства программиста он найдет много дополнительных приемов использования байтовых переключателей. Пожалуй, наиболее слабым местом у программистов, пытающихся пользоваться переключателями, является небрежность при их очистке, т. е. при выключении переключателей в нужное время и на нужном этапе выполнения программы.

2. Битовые переключатели и индикаторы

В то время как символьный переключатель занимает целый байт и может находиться во многих состояниях, битовый переключатель занимает только один двоичный разряд и может находиться самое большее в двух состояниях. С другой стороны, 1 байт данных может содержать восемь отдельных битовых переключателей — по одному на каждый разряд.

Эти переключатели играют особую роль в случаях, когда логический ход выполнения программы зависит от ряда факторов, которые могут быть представлены группой переключателей или индикаторов. Например, байт может содержать ряд переключателей состояния, которые могут быть подвергнуты все сразу или по отдельности следующей проверке:

(Да или Нет) И/ИЛИ (Да или Нет) И/ИЛИ
(Да или Нет) И/ИЛИ (Да или Нет) И/ИЛИ
(Да или Нет) И/ИЛИ (Да или Нет) И/ИЛИ
(Да или Нет) И/ИЛИ (Да или Нет)

Проверка состояний (да/нет или включено/выключено) восьми битовых переключателей может производиться, например, по следующей логической схеме:

«Если битовый переключатель 1 включен И битовый переключатель 2 включен ИЛИ битовый переключатель 3 выключен, то...»

Задание байта данных, который должен содержать от одного до восьми переключателей, производится примерно так же, как и для байтового переключателя. Различие заключается в том, что задаваемое поле битовых переключателей должно содержать вместо пробела нуль, представленный как число с фиксированной точкой. Нуль как значение однобайтовой величины с фиксированной точкой представляется восемью нулевыми битами, а пробел (шестнадцатеричное 40) представляется сочетанием битов 01000000. Константа, используемая для задания битового переключателя, записывается следующим образом:

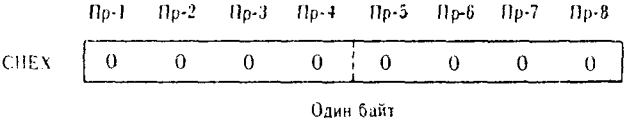
```
SWITCHES    DC    X'00'
```

Для действий над конфигурацией битовых переключателей удобно пользоваться командами, выполняющими операции булевой алгебры. Для включения в байте отдельных переключателей может быть использована команда OI (ИЛИ непосредственное). Команда NI (И непосредственное) используется для выключения битовых переключателей в байте. Использование этих двух команд дает программисту возможность установить любой бит или любую их комбинацию в состояние «включено» или «выключено».

Используя команду Проверить по маске, можно подвергнуть анализу выбранные биты, не изменяя их содержимого. В дальнейшем будут описаны и проиллюстрированы приемы использования этих трех команд.

Как было указано, команда «ИЛИ непосредственное» применяется для того, чтобы установить битовый переключатель в состояние «включено», т. е. чтобы занести единицу в определен-

ный двоичный разряд, отведенный под этот переключатель. Если использовать однобайтовую область памяти с именем CHEX, можно считать, что каждый из восьми двоичных разрядов этой области содержит битовый переключатель. Графически это выглядит так:



Каждый двоичный разряд содержит отдельный переключатель, способный находиться в состоянии «да» или «нет»: если 0, то «нет», если 1, то «да». Применяя команду ИЛИ непосредственное, можно установить в состояние «да» любой переключатель или их группу. Ниже следуют несколько примеров.

Пример 1. Установить переключатель 5 (Пр-5) в состояние «да» (или «включено»).

Первое, что следует предпринять, это определить маску для команды OI. Это можно сделать, рассматривая последовательность переключателей в байте.

Номер переключателя	Должен ли быть данный переключатель включен? — 0, если нет или изменения не требуется; 1, если да
Пр-1	0...нет
Пр-2	0...нет
Пр-3	0...нет
Пр-4	0...нет
Пр-5	1...да
Пр-6	0...нет
Пр-7	0...нет
Пр-8	0...нет

Оказывается, что в данном примере требуется маска, представляющая собой следующий набор нулей и единиц: 0-0-0-0-1-0-0-0. Этот набор можно представить шестнадцатеричным числом 08, т. е. маску, используемую в команде OI, следует кодировать как X'08', а сама команда должна быть записана следующим образом:

```
OI      CHEX,X'08'
```

При выполнении этой команды поле CHEX преобразуется следующим образом:

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
CHEX (До)	0	0	0	0	0	0	0	0
X'08' (Маска)	0	0	0	0	1	0	0	0
CHEX (После)	0	0	0	0	1	0	0	0

Переключатель 5 успешно установлен в состояние «да» (или «включено»).

Пример 2. Требуется включить переключатели Пр-2, Пр-4 и Пр-8. В этом примере маска может быть найдена следующим образом:

Номер переключателя	Должен ли быть данный переключатель включен? — 0, если нет или изменения не требуется; 1, если да
Пр-1	0...нет
Пр-2	1...да
Пр-3	0...нет
Пр-4	1...да
Пр-5	0...нет
Пр-6	0...нет
Пр-7	0...нет
Пр-8	1...да

Это означает, что маску следует кодировать как набор нулей и единиц 0-1-0-1-0-0-0-1, т. е. шестнадцатеричное 51. Сама команда записывается следующим образом:

```
OI      CHEX,X'51'
```

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
CHEX (До)	0	0	0	0	0	0	0	0
X'51' (Маска)	0	1	0	1	0	0	0	1
CHEX (После)	0	1	0	1	0	0	0	1

Пример 3. В данном случае задача заключается в том, чтобы обеспечить в программе установку переключателей Пр-1, Пр-2, Пр-6, Пр-7 и Пр-8 в состояние «да», т. е. включить их, если они

выключены, а если они уже включены, то оставить их в этом состоянии. В данном примере предполагается, что Пр-1 и Пр-8 уже включены. Однако при составлении маски это обстоятельство не известно, поэтому следует предусмотреть установку всех нужных переключателей. Маска будет представлять собой набор нулей и единиц 1-1-0-0-0-1-1-1, т. е. X'C7', а команда запишется следующим образом:

```
OI      CHEX,X'C7'
```

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
CHEX (До)	1	0	1	0	0	0	0	1
X'C7' (Маска)	1	1	0	0	0	1	1	1
CHEX (После)	1	1	1	0	0	1	1	1

В этом примере проявляются две особенности рассматриваемой операции, которые не были проиллюстрированы в предыдущих примерах. Во-первых, если единице в маске соответствует единица в разряде переключателя, то результатом операции оказывается единица. Тем самым удовлетворяется требование устанавливать состояние «включено», если даже переключатель уже находится в этом состоянии. Во-вторых, если нуль в маске соответствует единице в разряде переключателя, то эта единица сохраняется. Тем самым удовлетворяется требование: «Если переключатель уже включен, то следует оставить его в этом состоянии». Так сохранено состояние Пр-3: он находился в состоянии «включено» до выполнения команды и остался включенным после того, как нуль в маске был сопоставлен с его состоянием.

Команда И непосредственное используется, наоборот, чтобы выключать переключатели, т. е. устанавливать их в состояние «нет». Поэтому подход к формированию маски для команды NI противоположен подходу к формированию маски для команды OI. Если битовый переключатель должен быть выключен, то в соответствующем разряде маски должен быть записан нуль. Если состояние переключателя не должно измениться, независимо от того, включен он или нет, в соответствующий разряд маски ставится единица. Следующие примеры демонстрируют эти приемы:

Пример 1. В данном случае команда выполняется для того, чтобы выключить Пр-3. В этом примере маска строится по следующей логической схеме:

Номер переключателя	Должен ли быть данный переключатель выключен? — 0, если да; 1, если нет или изменения не требуется
Пр-1	1...нет
Пр-2	1...нет
Пр-3	0...да
Пр-4	1...нет
Пр-5	1...нет
Пр-6	1...нет
Пр-7	1...нет
Пр-8	1...нет

Из этого следует, что маска должна представлять собой следующую конфигурацию нулей и единиц: 1-1-0-1-1-1-1-1, или в шестнадцатеричном представлении DF. Команду следует кодировать как

```
NI      CHEX,X'DF'
```

Если Пр-3 является единственным включенным переключателем, то выполнение команды вызывает следующее преобразование:

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
CHEX (До)	0	0	1	0	0	0	0	0
X'DF (Маска)	1	1	0	1	1	1	1	1
CHEX (После)	0	0	0	0	0	0	0	0

Пример 2. Требуется выключить Пр-2, Пр-5 и Пр-7 независимо от того, включены они или выключены в настоящий момент. В этом примере маска должна представлять собой набор нулей и единиц 1-0-1-1-0-1-0-1, чему соответствует B5 в шестнадцатеричном представлении. Команда записывается следующим образом:

```
NI      CHEX,X'B5'
```

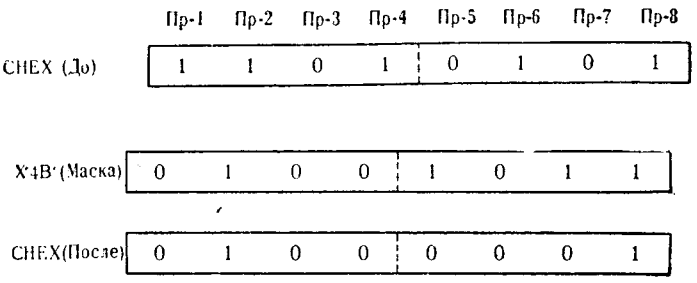
Если бы в этом примере Пр-2 и Пр-5 оказались включенными, то произошло бы следующее:

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
CHEX (До)	0	1	0	0	1	0	0	0
X'B5'	1	0	1	1	0	1	0	1
CHEX (После)	0	0	0	0	0	0	0	0

Теперь Пр-2 и Пр 5 выключены, что и требовалось. Так как Пр-7 был уже выключен, соответствующий разряд в маске, установленный в нуль, не вызвал изменения этого состояния.

Пример 3. Задача заключается в том, чтобы обеспечить в программе установку Пр-1, Пр-3, Пр-4 и Пр-6 в состояние «нет», или «выключено». Если переключатели выключены, то они и должны остаться в этом состоянии, если они включены, то их следует выключить. Изменять состояние других переключателей не следует. В этом случае маска должна представлять собой набор битов со следующими значениями: 0-1-0-0-1-0-1-1, или X'4B', а команда записывается следующим образом:

```
NI      CHEX,X'4B'
```



Как это и требовалось в программе, выполнение этой команды обеспечило то, что Пр-1, Пр-3, Пр-4 и Пр-6 находятся теперь в состоянии «выключено». Пр-2 и Пр-8, которые до начала выполнения команды находились в состоянии «включено», остались в этом же состоянии, потому что соответствующие биты маски были установлены в единицу.

Теперь, когда показаны способы установки битовых переключателей в положение «включено» или «выключено», остается неосвещенным способ проверки состояния каждого переключателя, а именно, анализ значения всех или какого-то одного бита посредством команды TM (Проверить по маске). Так же как и в командах, устанавливающих битовые переключатели в состояния «включено» или «выключено», для проверки состояния битов используется маска как часть команды Проверить по маске. Так как эта команда обрабатывает только 1 байт данных, маска задается как однобайтовый самоопределенный непосредственный символ. Биты в составе байта данных, которые следует проверять как битовые переключатели, соответственно обозначаются в маске единичными битами. Это означает, что для каждого проверяемого битового переключателя соответствующий ему бит маски будет установлен в единицу, а для тех

битовых переключателей, которые или не нужно проверять, или состояние которых несущественно, соответствующие биты маски устанавливаются в нуль. При выполнении команды TM нулевой бит маски не вызовет проверки соответствующего битового переключателя и состояние последнего не повлияет на результат. Единичный бит маски вызовет установку признака результата в соответствии с результатом сравнения с единицей состояния соответствующего битового переключателя. Если единичному биту маски соответствует единица в поле битового переключателя, то фиксируется единичный промежуточный результат; если единичному биту маски соответствует нуль в поле битового переключателя, то фиксируется промежуточный нулевой результат. После того как проверены все битовые переключатели, которым соответствуют единичные биты маски, устанавливается признак результата, отражающий наличие одного из трех возможных условий. Эти условия могут быть представлены следующим образом:

CC	BC	Условия
0	8	Все зафиксированные промежуточные результаты равны 0
1	4	Среди зафиксированных промежуточных результатов имеются как 1, так и 0
3	1	Все зафиксированные промежуточные результаты равны 1

Первое условие указывает на то, что все проверенные битовые переключатели были выключены; второе условие указывает на то, что некоторые были выключены, а некоторые — включены; третье — на то, что все битовые переключатели были включены.

Критерий, по которому в данном случае задается шестнадцатеричное содержимое маски, подобен критерию составления маски для команды OI. Если, например, команда TM применяется для проверки состояний Пр-3, Пр-5 и Пр-8, то маску можно построить согласно следующей таблице:

Номер переключателя	Маска 0, если проверять не следует 1, если этот переключатель следует проверять
Пр-1	0...нет
Пр-2	0...нет
Пр-3	1...да
Пр-4	0...нет
Пр-5	1...нет
Пр-6	0...нет
Пр-7	0...нет
Пр-8	1...да

Таким образом, маска должна представлять собой 0-0-1-0-1-0-0-1, или 29 шестнадцатеричное. Если затем команда будет записана как

```
TM      CHEX,X'29'
```

то в этом случае проверяться будут только переключатели Пр-3, Пр-5 и Пр-8. Результат проверки этих трех битовых переключателей, и только их, обусловит состояние битов признака результата.

Пример 1. Следует проверить байт данных, содержащий восемь битовых переключателей. Состояние этих переключателей в данный момент времени может быть представлено шестнадцатеричным числом 9D. В этом примере должны проверяться только переключатели Пр-3 и Пр-7, для чего требуется маска X'22'. Команда может быть записана как

```
TM      BITSWIT,X'22'
```

При выполнении этой команды производится следующее преобразование:

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
BITSWIT (До)	1	0	0	1	1	1	0	1
X'22' (Маска)	0	0	1	0	0	0	1	0
Биты промежуточного результата			0				0	

Следует отметить, что, хотя во включенном состоянии находилось несколько переключателей, только некоторые были проверены в соответствии с единичными битами маски. В данном примере признак результата показывает, что все биты промежуточного результата были равны нулю.

Пример 2. В этом примере должны быть проанализированы битовые переключатели, составляющие байт, содержимое которого X'FF'. Следует проверить состояние переключателей Пр-1, Пр-2 и Пр-4.

```
TM      BITSWITB,X'D0'
```

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
BITSWITB (До)	1	1	1	1	1	1	1	1
X'D0' (Маска)	1	1	0	1	0	0	0	0
Биты промежуточного результата	1	1		1				

В данном примере признак результата показывает, что все биты промежуточного результата равны единице. Следовательно, можно сделать вывод, что все три переключателя были во включенном состоянии.

Пример 3. В этом примере должен быть проанализирован байт данных, содержащий шестнадцатеричное число 27. Команда должна проверить, включены ли переключатели Пр-2, Пр-3, Пр-5 и Пр-8.

```
         TM      BITSWITC,X'69'
```

	Пр-1	Пр-2	Пр-3	Пр-4	Пр-5	Пр-6	Пр-7	Пр-8
BITSWITC (До)	0	0	1	0	0	1	1	1
X'69' (Маска)	0	1	1	0	1	0	0	1
Биты промежуточного результата		0	1		0			1

После выполнения этой команды признак результата в PSW будет свидетельствовать о том, что среди промежуточного результата были как единичные, так и нулевые биты.

В. ОБЛАСТИ ПРИМЕНЕНИЯ ПЕРЕКЛЮЧАТЕЛЕЙ

Можно выделить несколько основных областей, охватывающих все многообразие конкретных случаев рационального применения символьных и битовых переключателей. Различия в применении переключателей в пределах каждой такой области зависят, главным образом, от алгоритма программы. Некоторые из этих основных областей подробно рассматриваются в следующих параграфах.

1. Обработка нескольких групп состояний конца файла

Если в процессе выполнения программы обмен информацией производится параллельно с несколькими накопителями на магнитной ленте и содержимое каждой ленты рассматривается как самостоятельный набор данных, то может оказаться не так-то просто составить подпрограмму для закрытия этих наборов. Примером может служить программа, которая обновляет или производит объединение значительных по размеру наборов данных. Пусть при этом несколько наборов данных, размещенных на различных магнитных лентах, объединяются, образуя на магнитной ленте сводный файл. В этом случае обычно неизвестно, на какой ленте раньше обнаружится состояние EOF (End of File) (Конец файла). Поэтому для того, чтобы программа могла

проверить, закрыты ли уже определенные ленты, часто используют некоторые виды переключателей или индикаторов. Если, например, данные с магнитных лент TAPE1, TAPE2 и TAPE3 должны последовательно объединяться с данными с магнитной ленты MASTER1, то программа при выполнении должна располагать информацией о том, содержимое каких лент исчерпывается прежде: ленты MASTER1, одной из лент, содержимое которой используется для модификации ленты MASTER1, или же всех этих лент. Если состояние EOF обнаружилось на ленте TAPE2 раньше, чем на остальных трех лентах, программа должна продолжать слияние TAPE1 и TAPE3 с MASTER1, не обращая внимание на ленту TAPE2. Так как любые дальнейшие попытки читать с TAPE2 привели бы к программному прерыванию в системе, то ясно, как можно использовать переключатель в данной ситуации. После того как закончена работа с TAPE2, получает управление подпрограмма EOF, которая устанавливает переключатель в состояние, свидетельствующее, что содержимое TAPE2 исчерпано. Любые дальнейшие попытки читать с TAPE2 предотвращаются в результате проверки переключателя непосредственно перед входом в подпрограмму, производящую чтение.

Аналогично можно составить подпрограмму, закрывающую наборы данных. Специальные подпрограммы сообщают основной программе, что та может попытаться закрыть определенные наборы данных на магнитной ленте. После того как программа завершается, необходимо знать, имеются ли незакрытые наборы данных на ленте, и если имеются, то какие. Если в процессе выполнения основной программы набор данных на ленте был закрыт, то этот факт очень легко отметить с помощью переключателя. Поэтому при завершении программы подпрограмма, предназначенная для окончательного закрытия наборов данных, проверяет переключатель для каждого набора данных на ленте. Если переключатель включен, то это укажет подпрограмме, что набор данных уже закрыт, и подпрограмма не предпримет никаких действий; если переключатель выключен, то это укажет подпрограмме, что набор данных еще открыт и можно выдать необходимые макрокоманды, чтобы закрыть его.

Описанные методы применимы не только к наборам данных на магнитных лентах, но и к наборам данных любых типов независимо от того, размещены ли они на магнитных дисках, барабанах или других носителях.

2. Выполнена ли подпрограмма?

Выполнение любой программы может распадаться на несколько этапов, и при выполнении различных этапов может использоваться один и тот же программный модуль. Этот модуль

может работать по-разному в зависимости от того, на каком этапе находится выполнение управляющей подпрограммы. Производя установку переключателя в различных точках управляющей подпрограммы и затем проверяя состояния этого переключателя при выполнении общего модуля, можно направлять выполнение этого модуля по разным путям.

3. Какая подпрограмма передала управление?

Эта разновидность применения подобна только что описанной. Проблемная программа может в различных точках своего алгоритма использовать один и тот же модуль, передавая ему управление с помощью команды BAL. В каждой такой точке выхода из основной подпрограммы от общего модуля может требоваться выполнение одной или нескольких специфичных для этой точки функций, однако не настолько специфичных, чтобы для различных точек требовались различные модули. Для того чтобы идентифицировать выходную точку, в которой управление от основной подпрограммы передается общему модулю, используется переключатель. Прежде чем выполнить команду BAL, передающую управление общему модулю, этот переключатель устанавливается в состояние, идентифицирующее точку выхода. Модуль в свою очередь может проанализировать этот переключатель и выполнить те функции, которые определяются его состоянием.

4. Особые случаи при обработке

В пределах данной области применения переключателей имеется столько возможностей, что их трудно описать в общем виде; некоторое представление об этой области может дать уже рассматривавшийся пример модификации сводного файла. В процессе модификации может случиться, что входные данные совпадают с уже имеющейся на сводном файле записью. В этом случае некоторые действия могут быть опущены, так как управляющая информация уже входит в состав записи сводного файла. Если во входных данных обнаружена запись, для которой не существует соответствующей записи в сводном файле, то может потребоваться создание таковой. Создание сводной записи представляет собой реализацию побочной ветви алгоритма программы. В этом случае снова может быть использован переключатель. Как только в результате сравнения входной записи и сводного файла выясняется, что в сводном файле такой записи нет, может быть установлен переключатель, состояние которого служит индикатором данного особого случая. Впоследствии этот переключатель проверяется в соответствующей точке: если он

выключен, управление передается в обход определенных подпрограмм, если переключатель включен, то сохраняется естественный порядок выполнения команд, согласно которому выполняются части программы, формирующие дополнительные данные для создания записи на сводном файле.

5. Переключатели в теле записей данных

Использование переключателей в теле записей часто является главным средством управления файлами данных, содержимое и использование которых изменяется во времени. При таком управлении записи помечаются путем установки переключателя, входящего в качестве данных в состав записи. Один из способов использования таких записей состоит в следующем. Определенные записи помечаются, а затем печатается список помеченных записей. Если к моменту следующей модификации не будет предпринято никаких действий, чтобы подтвердить необходимость сохранения этих записей, все они устраняются из данного файла.

6. Групповые операции

В данном случае программа на основе анализа двух или более переключателей производит выбор, какие действия ей предпринять. При этом используются, главным образом, битовые переключатели. Если рассмотреть в качестве примера систему снабжения через стол заказов, то при появлении на входе проблемной программы записи данных устанавливаются определенные битовые переключатели, и некоторые из них могут свидетельствовать о наличии следующих условий:

		Состояние переключателя	
		Да	Нет
Пр-1	Получен ли заказ?	1	0
Пр-2	Возвращен ли заказ?	1	0
Пр-3	Возвращен ли заказ спустя 60 дней?	1	0
Пр-4	Получена ли накладная?	1	0
Пр-5	Отличалась ли цена более чем на 5% от цены в заказе?	1	0
Пр-6	Была ли отправка неполной?	1	0
Пр-7	Обнаружены ли дефекты при приемке заказанного товара?	1	0
Пр-8	Должна ли быть задержана оплата по накладной?	1	0

В зависимости от состояния этих переключателей для каждой записи данных некоторые записи могут быть выделены и

использованы особым образом. Например, отдел оплаты счетов может пожелать получить перечень всех записей, для которых переключатели Пр-1, Пр-4, Пр-5 и Пр-8 находятся в состоянии «включено». Эти записи указывают, что материал получен, накладная получена от продавца, продажная цена отличается более чем на 5% от цены в заказе и оплата по накладной до сих пор не произведена. Конечно, наличие любого из этих условий могло бы быть проверено путем анализа переключателей или индикаторов в теле соответствующих записей. Однако в системах данного типа многие запросы вызывают сходные между собой проверки. Устанавливая во входной подпрограмме группы битовых переключателей, можно в ответ на каждый запрос анализировать наличие соответствующих условий просто посредством одной команды TM (Проверить по маске). В приведенном примере с системой снабжения по заказам необходимая проверка обеспечивается маской X'99'.

Упражнения

1. В соответствии с определением битового переключателя 1 байт в памяти может представлять собой совокупность _____ различных переключателей или индикаторов.

2. Если необходимо сравнить содержимое символьного переключателя с данным сочетанием нулей и единиц, то наиболее удобно сделать это с помощью команды _____.

3. Состояние битовых переключателей, не изменяя их, можно проверить с помощью команды _____.

4. Байтовый переключатель называется также _____ переключателем.

5. Состояние бита в составе исходного байта может быть проверено путем кодирования _____ в соответствующем разряде маски в команде TM.

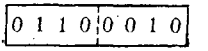
6. Команда Ассемблера _____ является наиболее эффективной для установки конфигурации нулей и единиц байтового или символьного переключателя.

7. Символьный переключатель может устанавливаться так, чтобы представлять _____ различных состояний.

В каждой из следующих задач задан набор переключателей, входящих в состав байта. Действия, которые требуется произвести, определены в терминах разрядов, перенумерованных слева направо от первого до восьмого. Требуется написать мнемонику и маску в шестнадцатеричной форме команды булевой логики, которая выполняет требуемое действие. Предполагается,

что содержимое исходного байта до выполнения команды неизвестно. Затем требуется заполнить исходный байт в соответствии с результатом выполненной команды.

8.

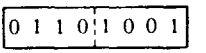
ABITS 

Задача: включить переключатели 1, 3 и 5.

_____ ABITS,X'_____

ABITS 

9.

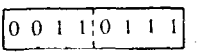
KBITS 

Задача: определить содержимое KBITS после выполнения следующих команд:

```
NI    KBITS,X'C3'
XI    KBITS,X'17'
OI    KBITS,X'4D'
NI    KBITS,X'A0'
```

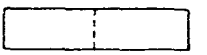
KBITS 

10.

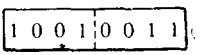
HBITS 

Задача: выключить все переключатели, кроме четвертого.

_____HBITS,X'_____'

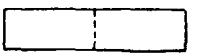
HBITS 

11.

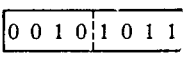
BBITS 

Задача: включить переключатели: 1, 4, 6, 7 и 8.

_____BBITS,X'_____'

BBITS 

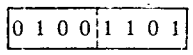
12.



Задача: выключить переключатели 1, 3 и 8, включить переключатели 2 и 5.

```
_____EBITS,X'_____'
_____EBITS,X'_____'
```

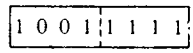
13.



Задача: выключить переключатели 2, 4 и 6.

```
_____CBITS,X'_____'
```

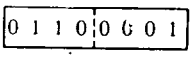
14.



Задача: изменить состояние каждого из переключателей 3, 4, 5 и 7 на противоположное; те, которые были включены, выключить, те, которые были выключены, включить.

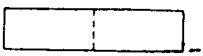
```
_____FBITS,X'_____'
```

15.

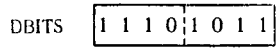


Задача: проверить состояние переключателей 2, 5, 7 и 8, не изменяя их.

```
TM    GBITS,X'_____'
```

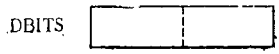


16.



Задача: выключить переключатели 1, 2, 6, 7 и 8.

```
_______DBITS,X'_______'
```



Глава 14

Циклы и поиск в таблицах

В этой главе рассматриваются основы управления циклами и использования прямой адресации при поиске в таблицах. Суть техники поиска в таблицах заключается в приращении адресов внутри непрерывного поля или блока данных, которое регулируется счетчиком циклов или проверкой некоторого условия путем сравнения. В некоторых случаях оказывается возможным использовать значение ключа искомого сегмента таблицы для формирования действительного адреса, по которому этот сегмент расположен в данной таблице.

Прежде чем приступить к подробному обсуждению самой техники поиска в таблицах, важно тщательно разобраться в приемах управления циклами.

А. УПРАВЛЕНИЕ ЦИКЛАМИ

Число последовательных прохождений по заданному циклу регулируется с помощью команд перехода по счетчику и условного перехода. Чаще всего используются следующие команды: BCT, BCTR, BC и BCR. Реже, но столь же эффективно используются команды BXH и BXLE. Так как все эти команды похожи одна на другую и используются сходным образом, для рассмотрения способов управления таблицами будут использованы только команды перехода по счетчику и условного перехода. Анализ возможностей остальных упомянутых команд показывает, что и они могут быть использованы в указанных целях.

Если общая длина поля или таблицы является неизменной, то можно считать, что управление циклом заключается в соблюдении того, чтобы не была превышена некоторая постоянная величина. В частности, команды перехода по счетчику можно использовать для прекращения прохождений по циклу, в котором производится просмотр поля или таблицы, как только общее количество циклов станет равным заданной величине. Если длина просматриваемой области является переменной, изменяющейся от одного использования программы к другому, то обычно применяют команду условного перехода. В конце области

данных помещают несколько специальных символов (специальными символами, в частности, может быть заполнен последний сегмент таблицы), и затем при каждом продвижении адреса внутри области производится проверка, не обнаружены ли эти специальные символы. Обнаружение этих символов означает, что область просмотрена полностью и искомых данных внутри нее не найдено.

Б. ПОСЛЕДОВАТЕЛЬНЫЙ ПОИСК

Последовательный поиск состоит в последовательной проверке примыкающих друг к другу сегментов данных независимо от того, соответствует ли расположение этих сегментов логической последовательности их значений. По отношению к значениям поля идентификатора (ключа) такие группы сегментов могут иметь или последовательную, или случайную организацию. Слово «последовательный» в словосочетании «последовательный поиск» означает, что сегменты таблицы рассматриваются один за другим в порядке их физического расположения независимо от того, соответствуют ли значения сегментов их расположению. Ниже приведен пример физически последовательного табличного массива, который организован логически последовательно.

/001/002/003/005/006/007
/009/010/012/013/017/018
/019/020/021/023/025/026

Приведем пример физически последовательного табличного массива, который в отношении значений элементов организован случайно.

/006/005/047/001/012/017
/075/065/066/063/014/019
/003/008/010/009/088/027

Последовательный поиск применим при любом формате данных. В некоторых случаях, однако, может оказаться удобнее использовать другую разновидность поиска в таблице, а именно поиск, который является последовательным как физически, так и логически. Таковым является, в частности, двоичный, или частично-последовательный, поиск.

Последовательный поиск выполняется путем многократного прохождения цикла; в каждом цикле производится сравнение текущего адресуемого сегмента таблицы, на основе сравнения принимается решение и выполняется команда, увеличивающая текущий адрес так, чтобы он указывал на следующий сегмент в таблице. Поиск по такой схеме обычно начинается от начала

таблицы, и проверке подвергаются все сегменты один за другим. Возможны многочисленные варианты этой схемы. В одном случае поиск может начинаться от конца таблицы и идти к ее началу, при этом в одном цикле проверяется один сегмент. В другом случае поиск может вестись в двух направлениях одновременно. Ниже приведены примеры, иллюстрирующие некоторые из этих вариантов поиска.

Пример 1. Данная программа (рис. 14.1а и 14.1б) выполняет поиск в таблице с целью преобразования трехбайтового числа в коде EBCDIC в однобайтовый алфавитный символ в том же коде. Таблица построена в физическом отношении последовательно и в логическом — случайно. Каждый сегмент таблицы состоит из четырех байтов — трехбайтового числа и однобайтового символа. Таблица состоит из 50 сегментов и ограничителя в конце таблицы. Если нужный сегмент найден, то выполняется переход к подпрограмме, производящей выборку символа из этого сегмента таблицы. Если нужный сегмент не найден, таблица просматривается полностью и сохраняется естественный порядок выполнения команд программы.

Смысл отдельных предложений разъясняется ниже.

Предложение 0101 вызывает переход с возвратом к предполагаемой подпрограмме, которая считывает одну карту в область CARDIN.

Предложение 0102 пересылает третий, четвертый и пятый символы из поля CARDIN (предполагается, что эти байты содержат числовой ключ) в поле HOLDKEY, которое теперь содержит трехбайтовый код (аргумент), который используется для поиска в таблице.

Предложение 0103 загружает адрес первого байта поля TABLE в общий регистр 6. Теперь к этой величине (адресу) можно добавить приращения для последовательного продвижения по таблице.

Предложение 0104 — команда LA загружает величину +50 в общий регистр 7. Регистр 7 будет использоваться в команде перехода по счетчику для управления числом циклов. Так как в таблице имеется всего 50 сегментов, не считая ограничителя, величина +50 задает максимальное число приращений к адресу начала таблицы.

Предложение 0105 — первая команда в цикле LOOP. Она сравнивает 3 байта поля HOLDKEY с данными, текущий адрес которых находится в общем регистре 6. Каждый раз, когда к содержимому общего регистра 6 добавляется величина приращения, данная команда сравнивает содержимое поля HOLDKEY с новым сегментом таблицы — тем, на который указывает содержимое регистра 6.

IBM System/360 Assembler Coding Form

X28-6509-3 U/M050
Printed in U.S.A.

PROGRAM: TABLE LOOK-UP ROUTINE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
*		PROGRAM WORK AREAS		
*				
CARDIN	DC	CL80' '	-CARD INPUT AREA	
HOLDKEY	DC	CL3' '	-HOLD AREA FOR RECORD KEY	
CONVKEY	DC	CL1' '	-HOLD AREA FOR 1-BYTE CHARACTER	
*				
*				
*				
*		TABLE		
*				
TABLE	DC	CL28'010A015D020E031B045A04EC242C'		
	DC	CL28'026B053G056A096B100C177D316G'		
	DC	CL28'049A098E072C039D176E134A410F'		
	DC	CL28'032C121B165H073A016F265C259H'		
	DC	CL28'071D041A083F030A088B085G047F'		
	DC	CL28'110E378D059A237B103A025A196H'		
	DC	CL28'051F283B075E066A217C291D019J'		
	DC	CL8'035C999X'		
*				

Рис. 14.1а.

IBM

IBM System 360 Assembler Coding Form

PROGRAM: TABLE LOOK-UP ROUTINE

PROGRAMMER:

DATE:

PAGE 2 OF 2

```
*
*                    R O U T I N E   L O G I C
*
GO          BAL     8,GETCARD                                    0101
            MVC     HOLDKEY(3),CARDIN+3                          0102
            LA      6,TABLE                                      0103
            LA      7,50                                         0104
LOOP        CLC     HOLDKEY(3),0(6)                              0105
            BC      8,FOUND                                      0106
            AH      6,=H'4'                                      0107
            BCT     7,LOOP                                       0108
            BC      15,GO                                        0109
*
*
*
FOUND       MVC     CONVKEY(1),0+3(6)                            0
            BAL     8,PUTDATA                                    012
            BC      15,GO
*
*
*
```

Рис. 14. 1б.

Предложение 0106 — команда BC вызывает переход к подпрограмме FOUND, если содержимое поля HOLDKEY равно содержимому первых 3 байтов сегмента таблицы, который адресован регистром 6.

Предложение 0107 выполняет сложение числа +4 с содержимым регистра 6. Так как при выполнении предложения 0106 перехода не произошло, что означает несовпадение содержимого HOLDKEY и текущего сегмента таблицы, который адресован содержимым регистра 6, адрес в регистре 6 увеличивается на длину одного сегмента таблицы (4 байта). Если в дальнейшем будет выполняться предложение 0105, поле HOLDKEY будет сравниваться со следующим сегментом таблицы.

Предложение 0108 — команда BCT управляет циклом. Каждый раз, когда это предложение выполняется, содержимое общего регистра 7 уменьшается на число +1. Пока содержимое регистра 7 больше чем нуль, производится переход к LOOP. Как только содержимое регистра 7 уменьшается до нуля, что случается, когда вся таблица просмотрена, переход не производится и выполняется предложение 0109.

Выполнение предложения 0109 вызывает безусловный переход к предложению GO. Так как ключ, содержащийся в поле HOLDKEY, не найден, в программе производится переход к GO, ввод новой карты, перенос из нее нового значения ключа, установка заново регулятора цикла и указателя на начало таблицы TABLE, после чего производится вход в цикл LOOP для проверки, находится ли в таблице TABLE новый ключ.

Предложение 0110 находится в начале подпрограммы, которая получает управление, если содержимое текущего сегмента таблицы равно текущему значению ключа. Данное предложение пересылает четвертый байт текущего сегмента таблицы TABLE в однобайтовую область CONVKEY. На этот сегмент указывает адрес, содержащийся в общем регистре 6, а для выборки байта данных, являющегося четвертым байтом сегмента, этот адрес увеличивается на число +3.

Выполнение предложения 0111 вызывает переход с возвратом к предполагаемой подпрограмме, которая производит операцию вывода, используя символ, пересланный в CONVKEY.

Выполнение предложения 0112 вызывает безусловный переход к GO и повторное выполнение последовательности предложений, обеспечивающее ввод новой карты, пересылку аргумента, используемого при просмотре, установку заново регулятора цикла и т. д.

В данном примере для обнаружения конца таблицы использовалась команда BCT. Так как таблица состояла всего из 50 сегментов, содержимое которых имело смысл проверять, в регистр команды BCT загружалось число 50. На самом деле,

таблица состояла из 51 сегмента, последний из которых представлял собой ограничитель. В рассмотренной программе предполагалось, что таблица всегда состоит из 50 сегментов, подлежащих проверке. Если использовать сегмент-ограничитель, то общая длина таблицы окажется несущественной. Сравните программу, приведенную на рис. 14.2, с программой предыдущего примера.

В данной программе число сегментов таблицы никак не используется. Просмотр таблицы продолжается, пока не будет найден нужный сегмент таблицы или пока при выполнении предложения 0107 не обнаружится сегмент-ограничитель. Как только при выполнении предложения 0107 в результате сравнения будет обнаружено равенство, т. е. вся таблица просмотрена и искомый сегмент не обнаружен, при выполнении предложения 0108 перехода не произойдет и сохранится естественный порядок следования команд, а именно следующим будет выполняться предложение 0109. Хотя в данный вариант программы включены дополнительные предложения 0107 и 0108, предложения, входившие в первый вариант программы под номерами 0104 и 0108, устранены, и в результате для достижения цели требуется то же число предложений. Допустимо использовать оба эти метода, однако второй метод имеет определенное преимущество, если общая длина таблицы не является постоянной во всех случаях использования программы.

Во многих случаях таблица, используемая проблемной программой, существует лишь как определенная область в программе и фактически не содержит данных. Данные, которые должны быть занесены в эту таблицу, недоступны до тех пор, пока программа не начнет выполняться. Они существуют как дополнение к объектному модулю или как входной набор данных и загружаются в область, отведенную для таблицы, самой проблемной программой. Обычно такая таблица называется загружаемой. Если, например, проблемная программа вводится через устройство чтения перфокарт, даные, предназначаемые для таблицы, могут быть представлены на дополнительных картах, которые помещаются после ограничителя — последней карты объектного модуля. Когда проблемная программа начинает выполняться, то в качестве одного из первых своих действий она инициирует считывание карт, содержащих данные, предназначенные для заполнения таблицы, и помещает эти данные в отведенную для этой таблицы область. В этом случае опять можно использовать сегмент-ограничитель таблицы, но на этот раз с двоякой целью.

В процессе считывания карта-ограничитель будет извещать о конце ввода таблицы; когда же производится поиск, эта карта будет извещать, что достигнут конец таблицы. Рассмотренный

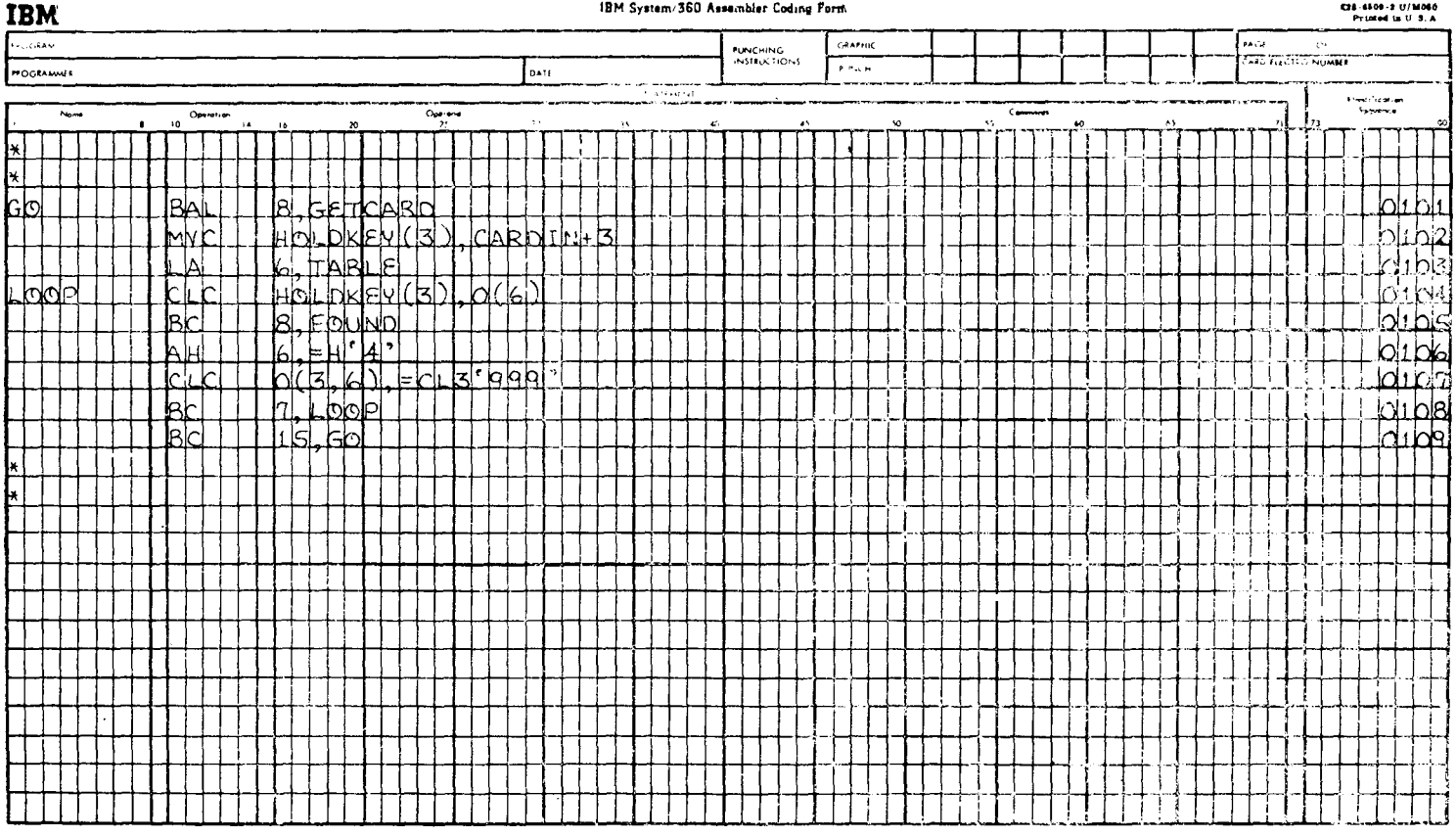

IBM
IBM System/360 Assembler Coding Form
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
CARD ELECTRO NUMBER
Name
Operation
Operand
Comments
Identification Sequence
*
*
GO BAL 8,GETCARD 0101
MVC HOLDKEY(3),CARDIN+3 0102
LA 6,TABLE 0103
LOOP CLC HOLDKEY(3),0(6) 0104
BC 8,FOUND 0105
AH 6,=H'4' 0106
CLC 0(3,6),=CL3'999' 0107
BC 7,LOOP 0108
BC 15,GO 0109
*
*

Рис. 14.2.

случай представляет собой хороший пример применения понятия таблицы переменной длины. При заполнении таблицы данными, вводимыми с карт, можно регулировать циклы с использованием как сегмента-ограничителя таблицы, так и команд перехода по счетчику. Сегмент-ограничитель таблицы укажет, когда будет считана последняя карта, а команда перехода по счетчику, применяемая для регулирования цикла, не позволит совокупности считанных данных выйти за пределы отведенной для таблицы области. Если область в проблемной программе, отведенная для таблицы, допускает самое большее 500 сегментов, а программа обнаруживает ограничитель при считывании 400-го сегмента, то заполняются только 400 сегментов таблицы и лишь они подвергаются просмотру при последующем поиске, если при этом поиске для определения конца таблицы используется сегмент-ограничитель. Если проблемная программа попытается считать в данную область памяти 510 сегментов таблицы, после считывания 500-го сегмента при выполнении команды перехода по счетчику перехода не произойдет. В соответствии с естественным порядком выполнения команд управление перейдет к следующей команде, в результате выполнения которой, например, оператор вычислительной машины получит сообщение о том, что таблица переполнена. Следующий пример (рис. 14.3) иллюстрирует рассмотренный алгоритм.

Пример 2. Предложение 0041 задает метку CARDIN в качестве имени 80-байтовой области памяти, определяемой следующими предложениями. Нуль, предшествующий символам CL, означает, что данное предложение не резервирует память. К данным, вводимым в следующие 80 байтов памяти, можно обращаться, используя эту метку или метки, определенные предложениями с номерами 0042 и 0043.

Предложение 0042 фактически резервирует 4 байта памяти с меткой IDNO. Они отводятся под первые 4 байта каждой считываемой карты, содержащие идентификатор записи учета кадров, являющийся частью данных, заносимых в таблицу.

Предложение 0043 резервирует 76 последующих байтов памяти, дополняя до 80 байтов область CARDIN. Только первые 16 байтов этого поля важны, так как они содержат фамилию служащего, которая должна быть занесена в таблицу вместе с соответствующим идентификатором, находящимся в предшествующем четырехбайтовом поле.

Предложение 0044 резервирует 16 000 байтов памяти с меткой TABLE для таблицы «Идентификаторы и фамилии служащих». Предполагается, что эта таблица состоит самое большее из 800 сегментов по 20 байтов каждый: 4 байта для идентификатора и 16 байтов для фамилии, соответствующей этому

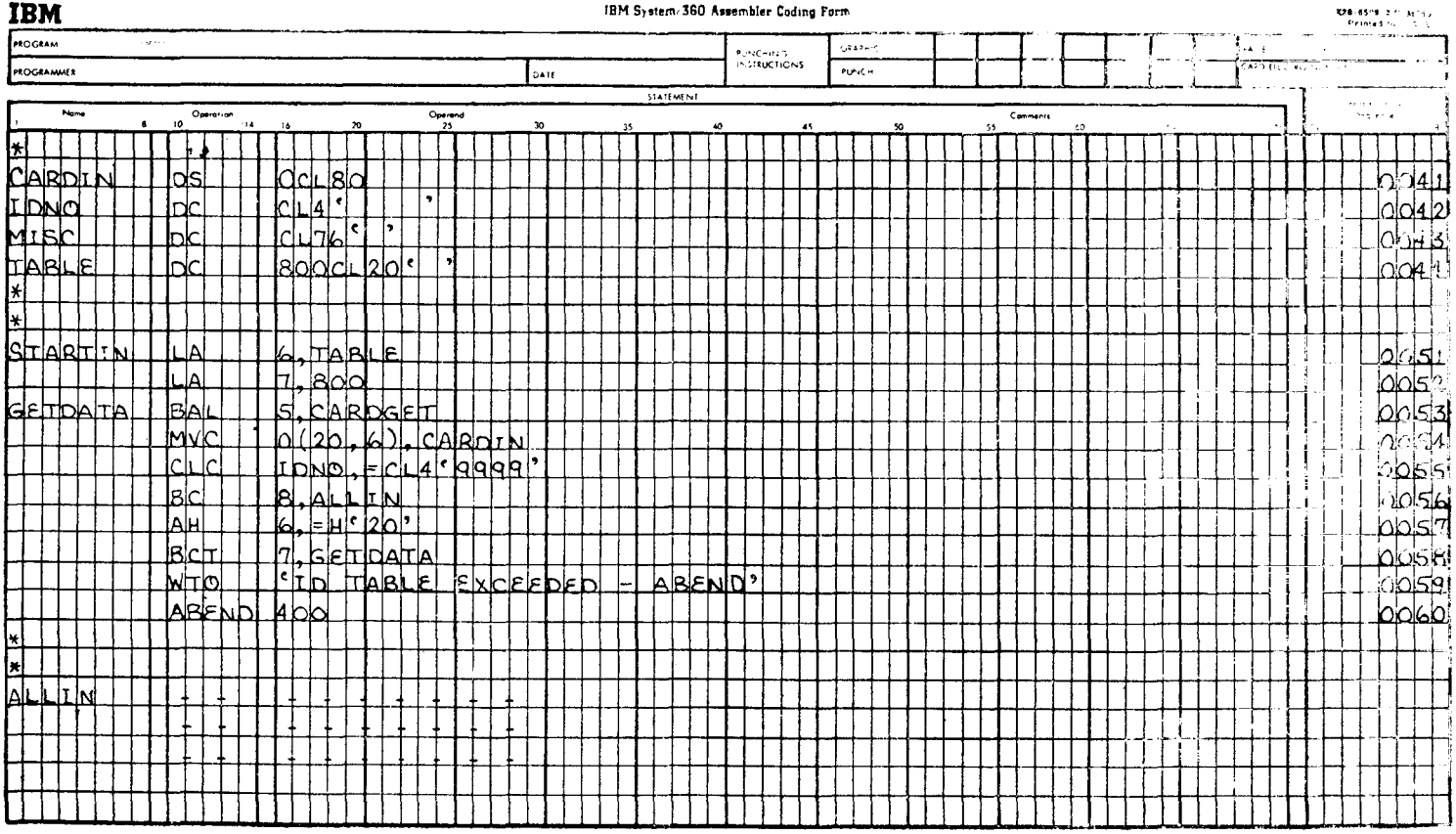

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
STATEMENT
Name
Operation
Operand
Comments
* ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
CARDIN DS 0CL80
IDNO DC CL4' '
MISC DC CL76' '
TABLE DC 800CL20' '
*
*
STARTIN LA 6,TABLE
LA 7,800
GETDATA BAL 5,CARDGET
MVC 0(20,6),CARDIN
CLC IDNO,=CL4'9999'
BC 8,ALLIN
AH 6,=H'20'
BCT 7,GETDATA
WTO 'ID TABLE EXCEEDED - ABEND'
ABEND 400
*
*
ALLIN
0041
0042
0043
0044
0051
0052
0053
0054
0055
0056
0057
0058
0059
0060

Рис. 14.3.

идентификатору. В эту таблицу можно занести любое число сегментов в пределах от 1 до 800, включая ограничитель. Так как число служащих изменяется от недели к неделе, эти изменения должны отражаться в таблице.

Остальные предложения этого примера управляют загрузкой сегментов таблицы с перфокарт в отведенную для таблицы область проблемной программы.

Предложение 0051 загружает в общий регистр 6 адрес первого байта области TABLE, длина которой составляет 16000 байтов.

Предложение 0052 загружает величину +800 в общий регистр 7; эта величина используется для управления циклом в операторе 0058, представляющем собой команду BCT. Это предложение не позволяет занести в таблицу TABLE более чем 800 сегментов.

Предложение 0053 осуществляет переход с возвратом к предполагаемой программе доступа к данным, считывающей одну карту в область CARDIN. После того как карта считана, можно обращаться к введенным данным в целом, используя имя CARDIN, или к их части, используя имя IDNO или имя MISC.

Предложение 0054 пересылает первые 20 байтов поля CARDIN в текущий сегмент таблицы, адрес которого содержится в общем регистре 6. Эти 20 байтов содержат 4-байтовый идентификатор и 16-байтовое поле, содержащее фамилию служащего, и составляют один сегмент таблицы. Когда считывается первая карта, регистр 6 содержит адрес, сформированный как TABLE + 0; когда считывается вторая карта, регистр содержит адрес TABLE + 20; когда считывается третья карта — адрес TABLE + 40; когда считывается четвертая карта — адрес TABLE + 60 и т. д.

Предложение 0055 сравнивает первые 4 байта поля CARDIN, определенные меткой IDNO и занесенные перед этим в таблицу, с 4-байтовой литеральной константой 9999. В рассматриваемой программе карта, содержащая 9999, используется как ограничитель входной информации и ограничитель таблицы. Так как при этом ограничитель таблицы должен быть частью таблицы, сравнение с целью обнаружить ограничитель входной информации не производится, пока данные не пересланы в таблицу.

Предложение 0056, содержащее команду BC, вызывает переход, если в результате сравнения сформировался признак результата, означающий «равно». В этом случае производится переход к подпрограмме проблемной программы, которая производит дополнительные действия, основываясь на допущении, что вся информация, которая должна быть помещена в таблицу, занесена в TABLE.

Предложение 0057 производит сложение полуслова со значением +20 с содержимым общего регистра 6, если предыдущее предложение не вызывает перехода. Содержимое общего регистра 6 увеличивается так, что оно становится адресом следующего сегмента таблицы, в который должны быть занесены данные.

Предложение 0058, содержащее команду BCT, управляет циклом. Регистр первого операнда этого предложения — общий регистр 7 — загружен числом +800, что позволяет заполнить самое большее 800 сегментов таблицы, прежде чем произойдет переход к предложению 0059. Если заполняются менее чем 800 сегментов таблицы, то будет обнаружена карта-ограничитель входной информации и произойдет переход в предложении 0056 прежде, чем содержимое общего регистра 7 будет уменьшено до нуля.

Предложение 0059 — макрокоманда WTO операционной системы. При выполнении этой макрокоманды системные программы управления данными печатают на пультовой пишущей машинке сообщение, текст которого, взятый в кавычки, содержится в данном предложении. В этом случае сообщение указывает оператору вычислительной машины, что число сегментов таблицы превысило 800 и программа не может выполняться дальше.

Предложение 0060 — макрокоманда ABEND [1]), которая прекращает выполнение текущей проблемной программы с кодом завершения 400. Выполнение программы прекращается, а данные, связанные с этой программой и находящиеся во входном потоке заданий, пропускаются через устройство чтения перфокарт без какой-либо обработки.

Пусть в таблице заполнено X сегментов, тогда проблемная программа может производить среди них поиск, используя второй вариант алгоритма, примененный в примере 1 данного раздела. Вместо того чтобы команда перехода по счетчику сигнализировала проблемной программе о том, что вся таблица просмотрена, выполняется команда сравнения, которая ищет код 9999, означающий, что достигнут последний из активных сегментов таблицы.

При программировании поиска в таблице, в которой данные упорядочены по значениям их содержимого, возможен иной способ выхода из цикла просмотра, позволяющий сократить время поиска. Этот способ заключается в том, что команда условного перехода вызывает переход, если результат сравнения аргумента (данных, с которыми сравниваются сегменты таблицы) с содержимым текущего сегмента говорит о том, что значение

[1]) От слов ABnormal END — аварийное окончание. — *Прим. ред.*

аргумента меньше. Такой результат показывает, что таблица последовательно просмотрена до сегмента, значение ключа которого больше, чем значение ключа, являющегося аргументом поиска, что означает отсутствие в таблице искомого сегмента. Естественно, что в этом случае следует выйти из цикла просмотра таблицы, так как дальнейший поиск в таблице бесполезен: все остальные ключи в таблице и подавно больше, чем искомый. Данный способ демонстрируется на следующем примере.

Пример 3. Алгоритм данной программы заключается в том, что поле KEYCHEX сравнивается с индексом для поиска адреса, содержащегося в 4 последних байтах каждого сегмента таблицы INDEX. Если отмечено равенство, программа пересылает соответствующий адрес из поля INDEX в поле ADDRESS. Если равенство отсутствует, цикл просмотра таблицы прекращается, как только при сравнении содержимого поля KEYCHEX с содержимым таблицы INDEX обнаружится, что ключ в KEYCHEX меньше, или как только встретится ограничитель таблицы — сегмент, содержащий 999. Предполагается, что в таблице заполнено неопределенное количество сегментов, которые упорядочены по нарастанию значений содержимого первых 3 байтов каждого сегмента (рис. 14.4).

Предложение 0061 — команда BAL — вызывает переход к предполагаемой подпрограмме для осуществления доступа к набору данных на магнитной ленте и помещения вводимых записей данных в поле с меткой TAPEIN.

Предложение 0062 загружает адрес первого байта таблицы INDEX в общий регистр 10. В этом регистре будет содержаться адрес текущего 7-байтового сегмента таблицы INDEX.

Предложение 0063 пересылает 3 байта данных из поля TAPEIN в поле KEYCHEX. Подразумевается, что эти данные представляют собой 3-байтовый ключ, с которым сравнивается содержимое ключа сегмента таблицы INDEX.

Предложение 0064 — первое предложение цикла просмотра таблицы — сравнивает 3 байта данных поля KEYCHEX с первыми 3 байтами текущего сегмента таблицы INDEX, адрес которого содержится в общем регистре 10. В результате сравнения устанавливается признак результата в Слове состояния программы, отражающий одну из возможностей: больше, меньше или равно.

Если признак результата после выполнения предложения 0064 соответствует условию равенства, предложение 0065 вызывает переход к предполагаемой подпрограмме, которая пересылает последние 4 байта текущего сегмента таблицы в поле ADDRESS. В противном случае переход не производится и сохраняется естественный порядок выполнения команд.

IBM System/360 Assembler Coding Form

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE | OF | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
*				
GONOW	BAL	5,GETAPE		0061
	LA	10,INDEX		0062
	MVC	KEYCHEX(3),TAPEIN+14		0063
LOOPCHEX	CLC	KEYCHEX(3),0(10)		0064
	BC	8,GOTADDR		0065
	BC	4,GONOW		0066
	AH	10,=H'7'		0067
	CLC	0(3,10),=CL3'999'		0068
	BC	7,LOOPCHEX		0069
NOHIT	BC	15,GONOW		0070
*				
*				
*				
TAPEIN	DC	CL25' '		0071
KEYCHEX	DC	CL3' '		0072
INDEX	DC	300CL7' '		0073
ADDRESS	DC	CL4' '		0074
*				
*				

Рис. 14.4.

Если в результате выполнения предложения 0064 обнаружено, что логическое значение поля KEYCHEX меньше значений в текущем сегменте таблицы INDEX, адрес которого содержится в регистре 10, предложение 0066 вызывает переход назад к предложению GONOW. Последнее означает, что часть таблицы INDEX, где мог быть найден ключ, соответствующий содержимому KEYCHEX, уже просмотрена, так как ключ сегмента таблицы больше по логической величине, чем ключ в поле KEYCHEX. Поэтому в таблице INDEX отсутствует сегмент, соответствующий содержимому KEYCHEX, и дальнейший поиск в таблице лишен смысла. Если перехода не происходит, то сохраняется естественный порядок выполнения команд.

Предложение 0067 складывает полуслово, содержащее число +7, с содержимым общего регистра 10. При этом содержимое общего регистра 10 увеличивается так, что оно становится адресом следующего сегмента таблицы INDEX.

Предложение 0068 сравнивает первые 3 байта сегмента таблицы, адресуемого общим регистром 10, с трехбайтовым символьным литералом с целью обнаружения конца таблицы INDEX. Последний сегмент таблицы INDEX — это ограничитель, содержащий символы 999 в первых своих 3 байтах.

Пока при выполнении предложения 0068 не обнаружится условия равенства, предложение 0069 будет вызывать передачу управления к первой команде цикла — предложению LOOPCHEX. Если при выполнении предложения 0068 обнаружено равенство, перехода не произойдет и управление переходит к предложению 0070, при этом подразумевается, что вся таблица просмотрена.

Предложение 0070 вызывает безусловный переход к GONOW. Так как для текущего содержимого KEYCHEX не обнаружено соответствия в таблице, программа не обрабатывает больше данную запись и заносит новое значение поля ключа из следующей записи на ленте.

Как было отмечено выше, программа для поиска данного типа, в которой выход из цикла производится, когда аргумент меньше, чем содержимое соответствующего сегмента таблицы, применима только тогда, когда сегменты таблицы последовательно упорядочены по возрастанию или убыванию значений ключей.

B. МЕТОДЫ ЧАСТИЧНО-ПОСЛЕДОВАТЕЛЬНОГО ПОИСКА

Методы частично-последовательного поиска основываются на логическом разделении таблицы на секторы, каждый из которых состоит из приблизительно одинакового количества сегментов. Сегменты во всей таблице упорядочены по возрастанию или

убыванию своего содержимого. Частично-последовательный поиск может начинаться с любого сектора в соответствии с выбранным алгоритмом. При последующем рассмотрении методов частично-последовательного поиска используется табл. 14.1.

Таблица 14.1

Логическая структура секторной таблицы

	+4	+8	+12	+16	+20	+24	+28	+32	+36
PTA →	0000	0008	0010	0012	0014	0018	0023	0024	0025	0027
+40	0029	0030	0033	0041	0042	0043	0045	0046	0047	0049
+80	0051	0060	0061	0062	0065	0066	0067	0068	0069	0070
+120	0072	0074	0076	0078	0079	0080	0082	0083	0085	0086
+160	0087	0088	0092	0094	0095	0096	0097	0100	0101	0103
PTB →	0105	0106	0107	0109	0110	0112	0114	0116	0117	0118
+40	0119	0120	0122	0123	0124	0127	0128	0129	0130	0132
+80	0133	0134	0135	0136	0137	0138	0139	0140	0144	0146
+120	0147	0149	0150	0151	0152	0153	0154	0155	0156	0157
+160	0158	0159	0160	0161	0162	0163	0164	0165	0166	0168
PTC →	0169	0170	0171	0172	0173	0174	0175	0176	0177	0178
+40	0180	0181	0182	0183	0184	0185	0186	0187	0189	0191
+80	0192	0193	0194	0195	0196	0197	0198	0200	0202	0203
+120	0204	0205	0206	0207	0208	0210	0211	0212	0213	0214
+160	0215	0216	0217	0218	0219	0220	0221	0222	0224	0225
PTD →	0226	0228	0230	0235	0237	0238	0239	0240	0241	0242
+40	0245	0246	0247	0248	0250	0252	0254	0256	0257	0259
+80	0260	0261	0263	0264	0265	0266	0267	0268	0269	0270
+120	0271	0272	0273	0274	0275	0276	0279	0280	0282	0284
+160	0295	0310	0311	0312	0315	0317	0321	0322	0330	0331
PTE →	0332	0335	0336	0337	0338	0339	0340	0341	0342	0343
+40	0344	0350	0351	0352	0353	0354	0355	0356	0357	0358
+80	0360	0362	0364	0365	0366	0380	0384	0385	0387	0390
+120	0391	0392	0393	0394	0395	0402	0403	0405	0407	0410
+160	0411	0412	0413	0414	0424	0425	0426	0429	0431	9999
	+164	+168	+172	+176	+180	+184	+188	+192	+196	

Эта таблица состоит из 250 элементов [1]) по 4 байта каждый и занимает в целом 1000 байтов. Таблица разбита на пять

[1]) В данном случае сегмент состоит из одного логически неделимого элемента. — *Прим. ред.*

первичных секторов, каждый из которых имеет свое имя и содержит 50 элементов. Например, элементы в секторе PTA имеют адреса PTA+0, PTA+4, PTA+8, PTA+12 и т. д. до PTA+196 включительно. За элементом PTA+196 следует первый элемент сектора PTB, который можно также адресовать как PTA+200. Следует заметить, что первый элемент сектора PTA (0000) и последний элемент сектора PTE (9999) представляют собой наименьшее и наибольшее значения ключей. Так как таблица начинается элементом 0000 и оканчивается элементом 9999, поиск в ней можно производить в порядке возрастания или убывания номеров элементов, не выходя за ее пределы. Это замечание сделано для того, чтобы подчеркнуть, что программист может использовать соответствующую команду сравнения с тем, чтобы предотвратить выход за пределы таблицы. Если элементы таблицы просматриваются от начала таблицы к ее концу и проверяется условие, что аргумент поиска KEY меньше или равен 9999, то сравнение должно вызвать прекращение поиска, как только будет достигнут последний элемент. Если элементы таблицы просматриваются от конца к ее началу и проверяется условие, что аргумент поиска KEY равен или больше 0000, то сравнение должно вызвать прекращение поиска, как только достигнут первый элемент таблицы. Степень детализации таблицы при частично-последовательном поиске зависит от искусства программиста и размера таблицы.

Чтобы лучше разъяснить основные идеи частично-последовательного поиска, некоторые методы такого поиска рассматриваются ниже на примере табл. 14.1. В каждом случае аргумент идентифицируется как KEY, а секторы таблицы — как PTA, PTB, PTC, PTD и PTE соответственно. Для удобства ссылок в дальнейшем на методы частично-последовательного поиска будем их называть метод 1, метод 2 и т. д.

Метод 1

Этот метод частично-последовательного поиска заключается в том, что элементы в пределах секторов просматриваются в порядке возрастания содержимого (рис. 14.5). Элемент с адресом PTA+0 не просматривается, так как заранее известно, что значение содержимого поля этого элемента равно 0000. В самом неблагоприятном случае такой метод поиска требует не более 54 сравнений — по одному сравнению на секторы PTB, PTC, PTD и PTE и на каждый элемент сектора PTE.

Метод 2

В данном случае поиск начинается приблизительно на середине таблицы (рис. 14.6) и требует самое большее 53 сравнений при поиске данных по табл. 14.1. В зависимости от числа сек-

торов и размера таблицы применение этого метода поиска может значительно сократить затраты времени на поиск.

Метод 3

Данный метод частично-последовательного поиска (рис. 14.7) является обобщением метода 2.

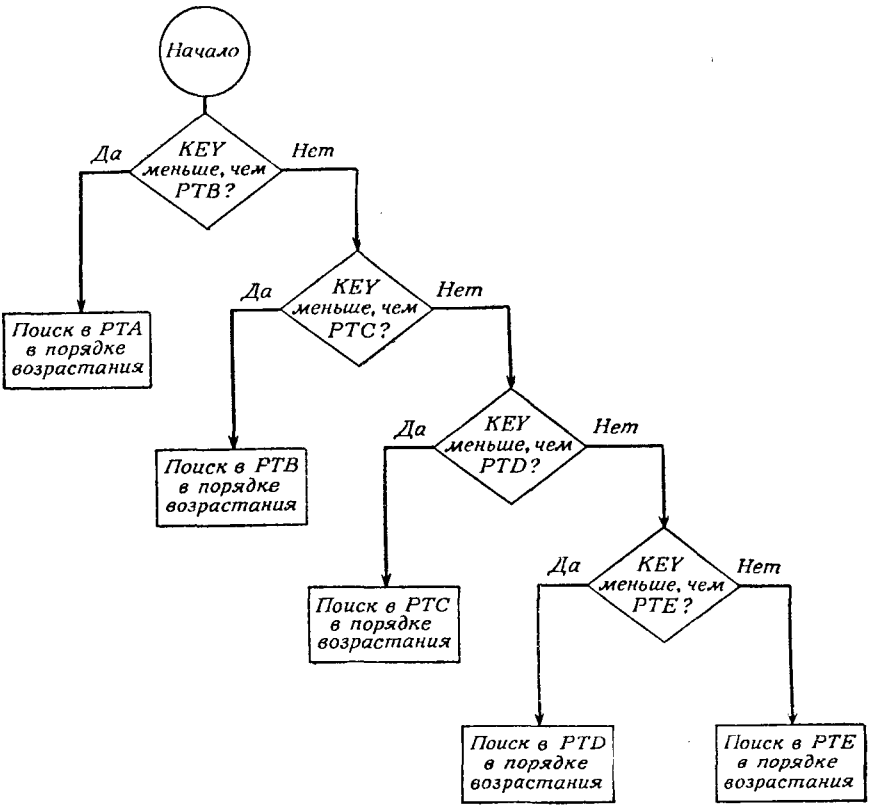


Рис. 14.5.

Можно взять любое число и проследить действие алгоритма при поиске этого числа в данной таблице. При применении этого метода частично-последовательного поиска требуется самое большее 29 сравнений, чтобы найти любой идентификатор в таблице. Этот максимум достигается только при самых неблагоприятных условиях. Следует отметить, что во всех приведенных блок-схемах при сравнениях не предусмотрено условие равенства. Несмотря на это, программист должен после каждого сравнения проверять наличие равенства и реагировать на него в программе как на достижение цели поиска.

Три приведенных метода частично-последовательного поиска несложны по своей природе и рассматриваются лишь для того, чтобы побудить программиста думать в этом направлении. Для экономии машинного времени можно разработать десятки методов частично-последовательного поиска. Алгоритмы этих методов до некоторой степени зависят от формата таблицы, в которой производится поиск, и целиком — от находчивости программиста.

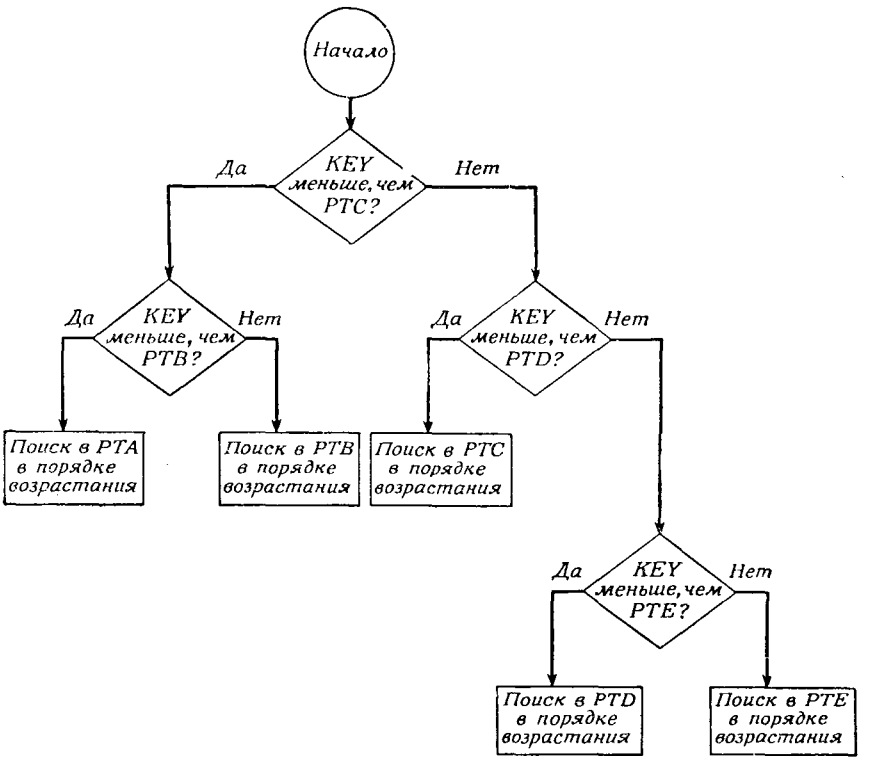


Рис. 14.6.

Как только разработан алгоритм поиска, кодирование программы сводится к следованию схеме алгоритма. Следующие фрагменты программы (рис. 14.8а и 14.8б) предназначены для обработки табл. 14.1 по методу 1 и содержат предложения, необходимые для определения элементов и секторов этой таблицы. Предполагается, что программа, содержащая эти фрагменты, уже обратилась к записям данных и поместила 4-байтовый набор числовых символов в поле с меткой KEY.

Предложение 0201 сравнивает символы в поле KEY с элементом таблицы, адресуемым меткой PTB. В табл. 14.1 имя PTB

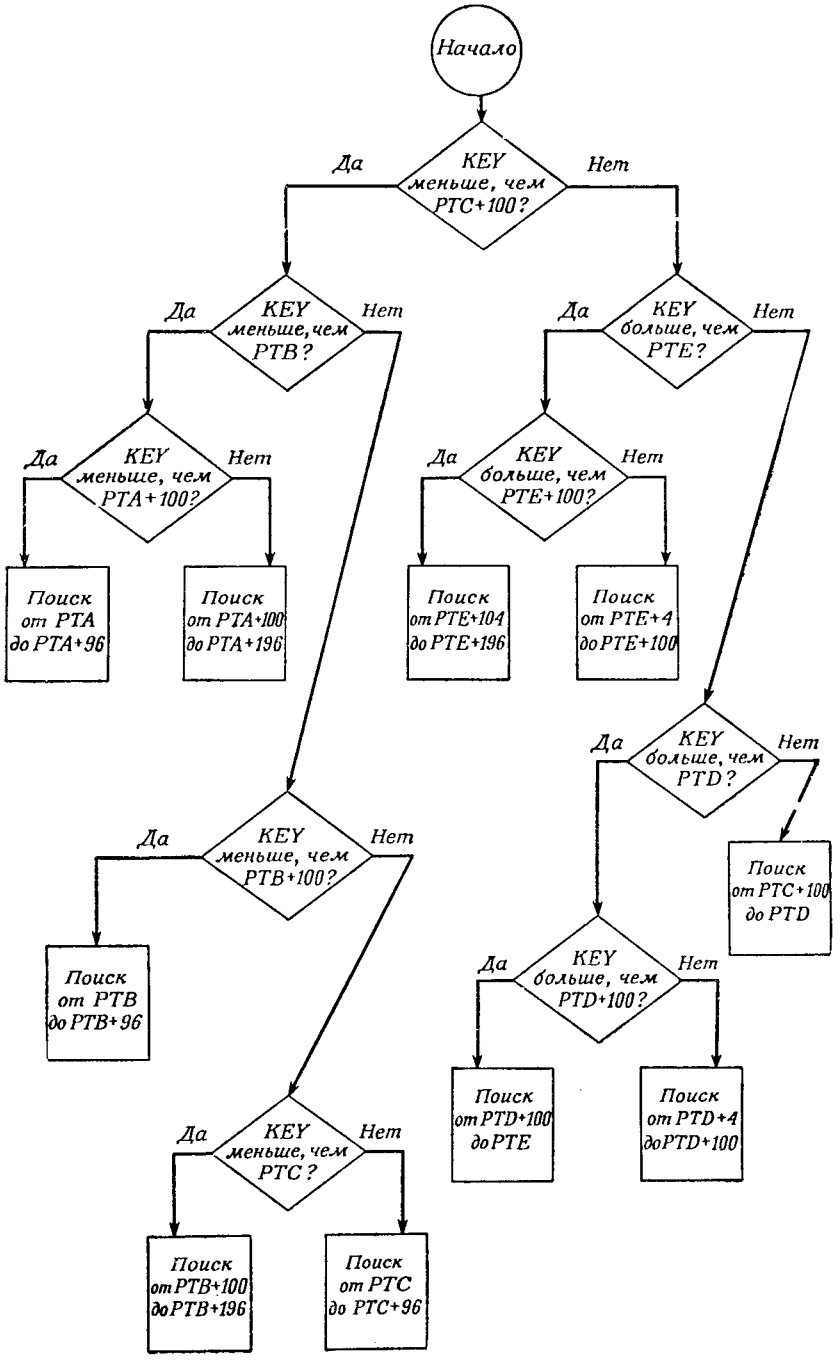

Начало
KEY меньше, чем PTC+100?
Да
Нет
KEY меньше, чем PTB?
KEY больше, чем PTE?
KEY меньше, чем PTA+100?
KEY больше, чем PTE+100?
Поиск от PTA до PTA+96
Поиск от PTA+100 до PTA+196
Поиск от PTE+104 до PTE+196
Поиск от PTE+4 до PTE+100
KEY меньше, чем PTB+100?
KEY больше, чем PTD?
Поиск от PTB до PTB+96
Поиск от PTC+100 до PTD
KEY меньше, чем PTC?
KEY больше, чем PTD+100?
Поиск от PTB+100 до PTB+196
Поиск от PTC до PTC+96
Поиск от PTD+100 до PTE
Поиск от PTD+4 до PTD+100

Рис. 14.7.

IBM

IBM System/360 Assembler Coding Form

PROGRAM: TYPE #1 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification Sequence
*				
TABLE	DS	0CL2000		0014
PTA	DC	50CL4C' '		0015
PTB	DC	50CL4C' '		0016
PTC	DC	50CL4C' '		0017
PTD	DC	50CL4C' '		0018
PTE	DC	50CL4C' '		0019
*				0100
KEY	DC	CL4C' '		0101
*				
*				
*				
GOTDATA	CLC	KEY(4),PTB		0201
	BC	2,SKIP1		0202
	LA	10,PTA		0203
	BC	15,SCANITA		0204
*				
*				
SKIP1	CLC	KEY(4),PTC		0205
	BC	2,SKIP2		0206
	LA	10,PTB		0207
	BC	15,SCANITA		0208
*				
*				

Рис. 14.8а.

IBM

IBM System/360 Assembler Coding Form

X28-6509-3 U/M050
Printed in U.S.A.

PROGRAM: TYPE #1 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 2

PROGRAMMER | DATE | CARD ELECTRO NUMBER

Name	Operation	Operand	Comments	Identification-Sequence
SKIP2	CLC	KEY(4),PTID		0209
	BC	2,SKIP3		0210
	LA	10,PTIC		0211
	BC	15,SCANITA		0212
*				
*				
SKIP3	CLC	KEY(4),PTIE		0213
	BC	2,SKIP4		0214
	LA	10,PTID		0215
	BC	15,SCANITA		0216
*				
*				
SKIP4	LA	10,PTIE		0217
	BC	15,SCANITA		0218
*				
*				
*				
SCANITA	LA	11,50		0219
SCANIT	CLC	KEY(4),0(10)		0220
	BC	8,HITKEY		0221
	BC	4,NOGO		0222
	AH	10,=H'4'		0223
	BC	15,SCANIT		0224
*				

Рис. 14.8б.

служит для обращения к 4 байтам памяти, содержащим символы 0105.

В предложении 0202 производится переход к предложению 0205, если при сравнении обнаруживается, что содержимое поля KEY больше содержимого поля PTB. Такой результат означает, что величина, соответствующая содержимому поля KEY, отсутствует среди элементов таблицы от PTA до PTB. Однако если при сравнении обнаруживается, что содержимое KEY меньше содержимого PTB, то это означает, что величина, соответствующая содержимому KEY, находится между элементами PTA и PTB. В последнем случае перехода не происходит и сохраняется естественный порядок следования команд, согласно которому следующим выполняется предложение 0203.

Предложение 0203 загружает адрес первого байта PTA в общий регистр 10. Этот адрес используется общей подпрограммой SCANITA.

Предложение 0204 вызывает безусловный переход к подпрограмме SCANITA, производящей поиск среди элементов сектора, адрес которого находится в общем регистре 10.

Если предложение 0202 вызывает переход, то выполняется предложение SKIP1 с номером 0205, сравнивающее KEY с первым четырехбайтовым элементом таблицы, меткой которого служит PTC и который содержит 0169.

Предложение 0206 вызывает переход к предложению 0209, если при сравнении обнаруживается, что содержимое поля KEY больше содержимого поля PTC. Переход означает, что величина, соответствующая содержимому поля KEY, находится не в части таблицы от PTB до PTC, а в области больших адресов. Если переход не производится, то сохраняется естественный порядок следования команд, согласно которому следующим выполняется предложение 0207; это означает, что величина, соответствующая содержимому поля KEY, находится где-то между PTB и PTC. Последнее заключение основывается на следующем: чтобы выполнялся оператор 0205, содержимое поля KEY должно быть больше или равно содержимому PTB, а чтобы выполнялся оператор 0207, содержимое поля KEY должно быть меньше, чем содержимое PTC.

Предложение 0207 загружает адрес первого байта PTB в общий регистр 10. Этот адрес используется подпрограммой SCANITA для поиска среди элементов таблицы от PTB до PTC элемента, содержимое которого совпадает с содержимым поля KEY.

Предложение 0208 вызывает безусловный переход к подпрограмме SCANITA, которая производит поиск среди элементов таблицы в секторе, адрес которого находится в общем регистре 10.

Если предложение 0206 вызывает переход, то выполняется предложение SKIP2 с номером 0209, которое сравнивает содержимое поля KEY с 0226, т. е. с первыми 4 байтами поля, имеющего метку PTD.

Если при сравнении в предложении 0209 обнаруживается, что содержимое поля KEY больше содержимого PTD, предложение 0210 вызывает переход к предложению 0213. Наличие перехода в данном месте программы означает, что аргумент поиска KEY больше содержимого PTD. Если данный переход отсутствует, это означает, что искомый элемент, если он вообще существует, заключен между PTC и PTD. Выполнение предложения 0205 и переход от предложения 0206 к предложению 0209 ранее показали, что содержимое поля KEY больше, чем содержимое PTC; так как сравнение при выполнении оператора 0209 не вызвало перехода в предложении 0210, величина аргумента поиска, содержащаяся в поле KEY, попадает в промежуток между значениями содержимого полей PTC и PTD.

Предложение 0211 загружает в общий регистр 10 адрес первого байта элемента таблицы с меткой PTC. Общий регистр 10 используется в подпрограмме SCANITA.

Предложение 0212 вызывает безусловный переход к подпрограмме SCANITA. Если вход в эту подпрограмму производится в данной точке, то подпрограмма SCANITA ищет в таблице от PTC до PTD элемент, содержимое которого совпадает с содержимым поля KEY.

Предложение SKIP3 с номером 0213 выполняется только тогда, когда выполнение предложения 0210 вызвало переход. Это означает, что содержимое поля KEY больше, чем элемент таблицы с меткой PTD. Теперь данное предложение сравнивает содержимое поля KEY с элементом таблицы, адресуемым меткой PTE.

Если при выполнении предложения 0213 обнаруживается, что содержимое поля KEY больше содержимого PTE, предложение 0214 вызывает переход к предложению 0217. Если обнаруживается, что содержимое поля KEY меньше PTE, т. е. что искомый элемент должен быть заключен между PTD и PTE, то сохраняется естественный порядок следования команд и следующим выполняется предложение 0215.

Предложение 0215 загружает адрес PTD в общий регистр 10, который после этого указывает первый байт элемента таблицы с именем PTD. В этой роли регистр 10 используется в предложении SCANIT.

Предложение 0216 вызывает безусловный переход к подпрограмме SCANITA. Если вход в эту подпрограмму производится в данной точке, подпрограмма SCANITA ищет в таблице от PTD

до PTE элемент, содержимое которого совпадает с содержимым поля KEY.

Предложение 0217 загружает в общий регистр 10 адрес PTE. Это предложение выполняется, только если при выполнении предложения 0214 имел место переход, т. е. содержимое поля KEY больше PTE. Так как PTE является последним сектором в таблице, предполагается, что искомый элемент, соответствующий содержимому поля KEY, заключен между PTE и концом таблицы. Общий регистр 10 теперь содержит адрес первого байта элемента таблицы с именем PTE и используется в подпрограмме SCANITA.

Предложение 0218 вызывает безусловный переход к предложению SCANITA с номером 0219. Если подпрограмма SCANITA выполняется при входе в данной точке, то производится поиск в таблице от PTE до конца таблицы.

Предложение 0220 является вторым предложением в подпрограмме SCANITA. Предыдущее предложение загружает в регистр максимальное число элементов в секторе для использования командой перехода по счетчику. Данное предложение сравнивает содержимое поля KEY с элементом таблицы, адрес которого находится в общем регистре 10. Если это предложение выполняется в начале цикла, общий регистр 10 содержит адрес следующих секторов:

1. PTA, если переход к SCANITA вызван предложением 0204;
2. PTB, если переход к SCANITA вызван предложением 0208;
3. PTC, если переход к SCANITA вызван предложением 0212;
4. PTD, если переход к SCANITA вызван предложением 0216;
5. PTE, если переход к SCANITA вызван предложением 0218.

Каждый раз, когда это предложение выполняется непосредственно после предложения 0223, содержимое общего регистра 10 будет увеличено, чтобы служить адресом следующего элемента таблицы в просматриваемом секторе. Подпрограмма SCANITA является общей для нескольких логических точек алгоритма, которые представлены пятью операторами безусловного перехода: 0204, 0208, 0212, 0216 и 0218.

Предложение 0221 проверяет, обнаружено ли равенство содержимого поля KEY и элемента таблицы, адрес которого находится в общем регистре 10, при выполнении предложения 0220. Если признак результата в Слове состояния программы показывает, что равенство обнаружено, данное предложение вызывает

переход к предполагаемой программе HITKEY. Эта программа может производить любые действия, какие требуются по замыслу программиста в случае, когда в таблице обнаружен искомый элемент. Если признак результата говорит о том, что равенство не обнаружено, сохраняется естественный порядок команд и следующим выполняется предложение 0222.

Предложение 0222 проверяет по текущему значению признака результата, обнаружено ли при выполнении предложения 0220, что содержимое поля KEY меньше содержимого текущего элемента. Если это так, т. е. содержимое поля KEY не совпадает ни с одним элементом таблицы, данный оператор вызывает переход к NOGO. Предполагается, что NOGO — это подпрограмма, которая производит действия, необходимые по замыслу программиста, когда искомый элемент в таблице отсутствует. Если содержимое поля KEY не меньше содержимого текущего элемента таблицы, в данной точке программы переход не производится, а сохраняется естественный порядок команд и следующим выполняется предложение 0223.

Предложение 0223 увеличивает адрес в общем регистре 10 на число +4. После этого данный регистр содержит адрес элемента таблицы, следующего за тем, который только что проверен предложением 0220.

Предложение 0224 вызывает безусловный переход назад к предложению SCANIT с номером 0220 для сравнения содержимого поля KEY с новым элементом таблицы, адрес которого содержится в регистре 10.

Чтобы нагляднее представить себе алгоритм рассмотренных программ, проследим ход их выполнения во время поиска конкретной величины в таблице.

(а) Следует найти ключ 0185.

(б) Последовательность выполнения предложений:

0201, 0202, 0205, 0206, 0209, 0210,
0211, 0212, 0219, 0220, 0221[1]), 0223,
0224, 0220, 0221, 0222, 0223, 0224,
0220, 0221, 0222, 0223, 0224, 0220,
0221, 0222, 0223, 0224, 0220, 0221,
0222, 0223, 0224, 0220, 0221, 0222,
0223, 0224, 0220, 0221, 0222, 0223,
0224, 0220, 0221, 0222, 0223, 0224,
0220, 0221, 0222, 0223, 0224, 0220,
0221, 0222, 0223, 0224, 0220, 0221,

[1]) Здесь пропущено предложение 0222. — *Прим. ред.*

0222, 0223, 0224, 0220, 0221, 0222,
0223, 0224, 0220, 0221, 0222, 0223,
0224, 0220, 0221, 0222, 0223, 0224,
0220, 0221, 0222, 0223, 0224, 0220,
0221, HITKEY.

(а) Следует найти ключ 0107.
(б) Последовательность выполнения предложений:

0201, 0202, 0205, 0206, 0207, 0208,
0219, 0220, 0221, 0222, 0223, 0224,
0220, 0221, 0222, 0223, 0224, 0220,
0221, HITKEY.

(а) Следует найти ключ 0236.
(б) Последовательность выполнения предложений:

0201, 0202, 0205, 0206, 0209, 0210,
0213, 0214, 0215, 0216, 0219, 0220,
0221, 0222, 0223, 0224, 0220, 0221,
0222, 0223, 0224, 0220, 0221, 0222,
0223, 0224, 0220, 0221, 0222, 0223,
0224, 0220, 0221, 0222, NOGO.

(а) Следует найти ключ 0332.
(б) Последовательность выполнения предложений:

0201, 0202, 0205, 0206, 0209, 0210,
0213, 0214, 0217, 0218, 0219, 0220,
0221, HITKEY.

Пример программы, алгоритм которой реализует метод 2 частично-последовательного поиска, показан на рис. 14.9а и 14.9б. При этом используется то же описание содержимого полей, что приведено в табл. 14.1.

Действие этой программы сходно с действием предыдущей и поэтому не требует специальных пояснений. Чтобы проверить алгоритм этих программ, проследим последовательность выполнения предложений при поиске приведенных ниже величин.

(а) Следует найти ключ 0014.
(б) Последовательность выполнения предложений:

0301, 0302, 0303, 0304, 0305, 0306,
0319, 0320, 0321, 0322, 0323, 0319,
0320, 0321, 0322, 0323, 0319, 0320,
0321, 0322, 0323, 0319, 0320, 0321,
0322, 0323, 0319, 0320, HITKEY.

IBM

IBM System/360 Assembler Coding Form

X28-6508-2 U/M050
Printed in U.S.A.

PROGRAM: TYPE #2 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT				Identification-Sequence
Name	Operation	Operand	Comments	
*				
GOTDATA	CLC	KEY(4),PTC		0301
	BC	10,PASS1		0302
	CLC	KEY(4),PTB		0303
	BC	10,SKIP1		0304
	LA	10,PTA		0305
	BC	15,SCANIT		0306
*				
*				
SKIP1	LA	10,PTB		0307
	BC	15,SCANIT		0308
*				
*				
PASS1	CLC	KEY(4),PTD		0309
	BC	10,PASS2		0310
	LA	10,PTC		0311
	BC	15,SCANIT		0312
*				
*				
PASS2	CLC	KEY(4),PTE		0313
	BC	10,PASS3		0314
	LA	10,PTD		0315
	BC	15,SCANIT		0316
*				

Рис. 14.9а

IBM

IBM System 360 Assembler Coding Form

X28-6500-2 U/M050
Printed in U.S.A.

PROGRAM: TYPE #2 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 2 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
PASS3	LA	10,PTE		0317
	BC	15,SCANIT		0318
*				
*				
SCANIT	CLC	KEY(4),0(10)		0319
	BC	8,HITKEY		0320
	BC	4,NOGO		0321
	AH	10,=H'4'		0322
	BC	15,SCANIT		0323
*				
*				
*				

Рис. 14.9б.

(а) Следует найти ключ 0333.
(б) Последовательность выполнения предложений:

0301, 0302, 0309, 0310, 0313, 0314,
0317, 0318, 0319, 0320, 0321, 0322,
0323, 0319, 0320, 0321, NOGO.

(а) Следует найти ключ 0226.
(б) Последовательность выполнения предложений:

0301, 0302, 0309, 0310, 0313, 0314,
0315, 0316, 0319, 0320, HITKEY.

(а) Следует найти ключ 0112.
(б) Последовательность выполнения предложений:

0301, 0302, 0303, 0304, 0307, 0308,
0319, 0320, 0321, 0322, 0323, 0319,
0320, 0321, 0322, 0323, 0319, 0320,
0321, 0322, 0323, 0319, 0320, 0321,
0322, 0323, 0319, 0320, 0321, 0322,
0323, 0319, 0320, HITKEY.

Программирование частично-последовательного поиска по методу 3 является более сложным. Чем сложнее задача программирования, тем больше вариантов возможно при ее решении. Поэтому приведенные ниже программы — это только одно из возможных применений частично-последовательного поиска по методу 3. Следует помнить, что определение полей и их содержание являются такими же, что и для приведенных выше применений частично-последовательного поиска по методу 1 и методу 2, но в некоторых предложениях допущено изменение адресов (рис. 14.10а — 14.10в). Сами программы почти не требуют пояснений. Каждая программа или подпрограмма реализует алгоритм частично-последовательного поиска в соответствии с блок-схемой, приведенной для метода 3. Ниже приведен ряд значений ключей и для каждого значения указана последовательность действий, ведущая к нахождению в таблице элемента с этим значением. Проследите очередность этих действий по номерам предложений.

(а) Следует найти ключ 0205.

IBM

IBM System 360 Assembler Coding Form

X28-6609-2 U/M040
Printed in U.S.A.

PROGRAM: TYPE #3 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 3

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
GOTDATA	CLC	KEY(4),PTC+100		0401
	BC	10,SECTB		0402
	CLC	KEY(4),PTB		0403
	BC	10,SECTA		0404
	CLC	KEY(4),PTA+100		0405
	BC	10,SECTC		0406
	LA	10,PTA		0407
	BC	15,SCANIT		0408
*				
*				
SECTC	LA	10,PTA+100		0409
	BC	15,SCANIT		0410
*				
*				
SECTA	CLC	KEY(4),PTB+100		0411
	BC	10,SECTD		0412
	LA	10,PTB		0413
	BC	15,SCANIT		0414
*				
*				
SECTD	CLC	KEY(4),PTC		0415
	BC	10,SECTE		0416
	LA	10,PTB+100		0417
	BC	15,SCANIT		0418

Рис. 14.10а.

IBM System 360 Assembler Coding Form

X28-6509-2 U/M050
Printed in U.S.A.

PROGRAM: TYPE #3 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS: GRAPHIC, PUNCH | PAGE 2 OF 3

PROGRAMMER | DATE | CARD ELECTRO NUMBER

Name	Operation	Operand	Comments	Identification-Sequence
*				
*				
SECTE	LA	10, PTC		0419
	BC	15, SCANIT		0420
*				
*				
SECTB	CLC	KEY(4), PTE		0421
	BC	4, SECTE		0422
	CLC	KEY(4), PTE+100		0423
	BC	4, SECTG		0424
	LA	10, PTE+100		0425
	BC	15, SCANIT		0426
*				
*				
SECTG	LA	10, PTE		0427
	BC	15, SCANIT		0428
*				
*				
SECTE	CLC	KEY(4), PTD		0429
	BC	4, SECTH		0430
	CLC	KEY(4), PTD+100		0431
	BC	4, SECTI		0432
	LA	10, PTD+100		0433
	BC	15, SCANIT		0434

Рис. 14.10б.

IBM System/360 Assembler Coding Form

PROGRAM: TYPE #3 SECTORING TECHNIQUE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 3 OF 3 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

Name	Operation	Operand	Comments	Identification-Sequence
*				
*				
SECTI	LA	10,PTD		0436
	BC	15,SCANIT		0436
*				
*				
SECTH	LA	10,PTC+100		0437
	BC	15,SCANIT		0438
*				
*				
SCANIT	CLC	KEY(4),0(10)		0439
	BC	8,HITKEY		0440
	BC	4,NOGO		0441
	AH	10,=H'4'		0442
	BC	15,SCANIT		0443
*				
*				

Рис. 14.10в.

(б) Поиск состоит в выполнении команд в следующем порядке:

0401, 0402, 0421, 0422, 0429, 0430,
0437, 0438, 0439, 0440, 0441, 0442,
0443, 0439, 0440, 0441, 0442, 0443,
0439, 0440, 0441, 0442, 0443, 0439,
0440, 0441, 0442, 0443, 0439, 0440,
0441, 0442, 0443, 0439, 0440, 0441,
0442, 0443, 0439, 0440, HITKEY.

(а) Следует найти ключ со значением 0066.
(б) Необходимо выполнить следующую последовательность команд:

0401, 0402, 0403, 0404, 0405, 0406, 0409, 0410,
0439, 0440, HITKEY.

(а) Следует найти ключ 0108.
(б) Поиск производится следующим образом:

0401, 0402, 0403, 0404, 0411, 0412,
0413, 0414, 0439, 0440, 0441, 0442,
0443, 0439, 0440, 0441, 0442, 0443,
0439, 0440, 0441, 0442, 0443, 0439,
0440, 0441, NOGO.

(а) Значение ключа — 0385.
(б) В процессе поиска необходимо выполнить следующую последовательность команд:

0401, 0402, 0421, 0422, 0423, 0424,
0425, 0426, 0439, 0440, 0441, 0442,
0443, 0439, 0440, 0441, 0442, 0443,
0439, 0440, HITKEY.

Хотя этот пример следует приведенной блок-схеме частично-последовательного поиска с точностью почти до символов в командах, основную программу можно закодировать значительно меньшим количеством операторов. Соответствующая последовательность предложений показана на рис. 14.11а и 14.11б, она не полностью отвечает блок-схеме частично-последовательного поиска по методу 3, но реализует алгоритм, очень близкий ему.

Для сравнения сокращенной программы с программой, рассмотренной в предыдущем примере и содержащей 43

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 1 OF 2

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
GCTDATA	LA	10,PTC+100		0501
	CLC	KEY(4),0(10)		0502
	BC	10,SECTA		0503
	SH	10,=H'200'		0504
	CLC	KEY(4),0(10)		0505
	BC	10,SECTB		0506
DOWN20	SH	10,=H'20'		0507
	CLC	KEY(4),0(10)		0508
	BC	12,SCANLT		0509
	BC	2,DOWN20		0510
*				
*				
SECTB	AH	10,=H'200'		0511
	BC	15,DOWN20		0512
*				
*				
SECTA	AH	10,=H'200'		0513
	CLC	KEY(4),0(10)		0514
	BC	2,DOWN20		0515
	LA	10,PTE+196		0516
	BC	15,DOWN20		0517
*				
*				

Рис. 14.11а.

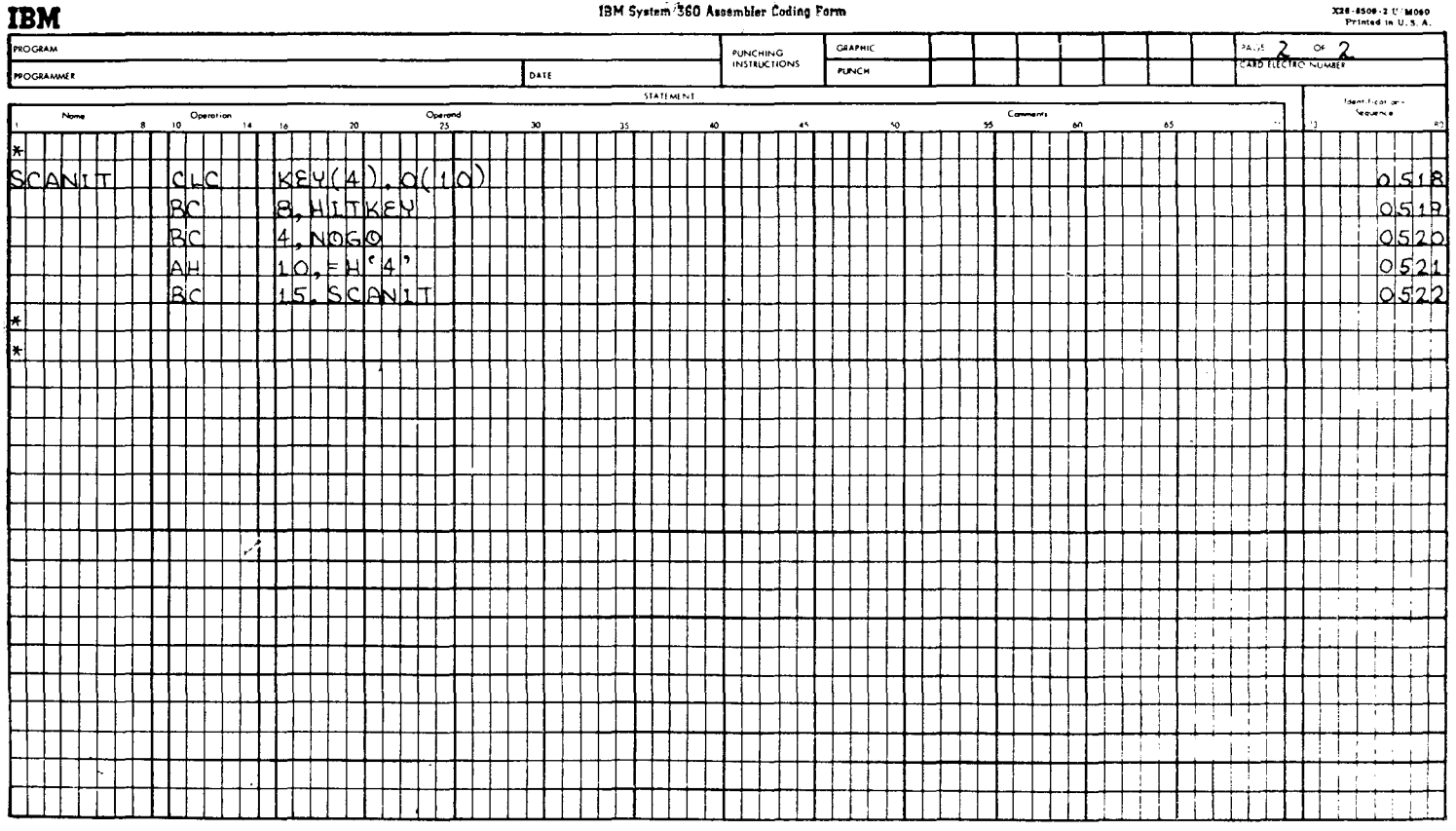

IBM
IBM System/360 Assembler Coding Form
X28-6508-2 U/M060
Printed in U.S.A.
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 2 OF 2
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
*
SCANIT CLC KEY(4),0(10) 0518
BC 8,HITKEY 0519
BC 4,NOGO 0520
AH 10,=H'4' 0521
BC 15,SCANIT 0522
*
*

Рис. 14.11б.

предложения, ниже приведена последовательность действий при поиске ключей с теми же значениями, что и выше.

(а) Значение ключа — 0205.

(б) Поиск состоит в выполнении команд в следующем порядке:

0501, 0502, 0503, 0513, 0514, 0515,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0518,
0519, 0520, 0521, 0522, 0518, 0519,
HITKEY.

(а) Значение ключа — 0066.

(б) Предложения должны выполняться в следующем порядке:

0501, 0502, 0503, 0504, 0505, 0506,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0518, 0519, HITKEY.

(а) Значение ключа — 0108.

(б) Поиск состоит в выполнении предложений в следующем порядке:

0501, 0502, 0503, 0504, 0505, 0506,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0518, 0519, 0520, 0521, 0522,
0518, 0519, 0520, 0521, 0522, 0518,
0519, 0520, 0521, 0522, 0518, 0519,
0520, NOGO.

(а) Значение ключа — 0385.

(б) Для осуществления поиска необходимо выполнить следующую последовательность команд:

```
0501, 0502, 0503, 0513, 0514, 0515,
0516, 0517, 0507, 0508, 0509, 0510,
0507, 0508, 0509, 0510, 0507, 0508,
0509, 0510, 0507, 0508, 0509, 0510,
0507, 0508, 0518, 0519, 0520, 0521,
0522, 0518, 0519, 0520, 0521, 0522,
0518, 0519, HITKEY.
```

Хотя последняя программа состоит из меньшего количества предложений, при поиске фактически выполняется большее число команд, следовательно, время счета с помощью этой программы больше. В данном случае первую программу, производящую частично-последовательный поиск по методу 3, следует считать более эффективной с точки зрения использования вычислительной машины, хотя программисту приходится тратить больше времени на кодирование. Такое бывает нередко: программист затрачивает немало времени для выделения небольших компактных подпрограмм, а в результате становится ясно, что прямой подход к проблеме был бы более эффективным.

Следует пояснить, что классификация методов частично-последовательного поиска по степени сложности введена только в целях данного изложения. Эти обозначения никоим образом не приняты среди специалистов ни для классификации этих методов, ни для оценки их сложности. Вряд ли нужно говорить, что, вероятно, имеется много больше вариантов частично-последовательного поиска, чем может придумать один человек, и, следовательно, существует мало надежды удачно классифицировать эти варианты.

Г. ПРЯМАЯ АДРЕСАЦИЯ СЕГМЕНТОВ ТАБЛИЦЫ

Существует много вариантов прямой адресации при поиске в таблице: от простого «сложения» до крайне хитроумных схем числовых преобразований. Для целей данного изложения будет рассмотрен лишь основной вариант метода прямой адресации сегментов таблицы. Этот метод в общем случае применим к таблице, значения ключей в которой непрерывно возрастают от ее начала к концу. Считается, что рассматриваемая ниже таблица имеет именно такой формат. В данной таблице должен существовать сегмент для каждого значения соответствующей числовой последовательности. Например, если значения 001, 002, 003, 005, 006, 008 и 009 — допустимые табличные аргументы, то

Таблица 14.2

Последовательно организованная таблица для обращения с применением прямой адресации

Номер сегмента таблицы	Относительный адрес	Метка	Сегмент таблицы	Адрес
		PRETAB	Фиктивная запись	32404
1	PRETAB +16	TABLE	D049586746384556	32420
2	+32		D837465847465837	32436
3	+48		D002838883746554	32454
4	+64		D922264537394058	32468
5	+80		bbbbbbbbbbbbbbb	32484
6	+96		D847263949555836	32500
7	+112		D933650003645572	32516
8	+128		D003232505926266	32532
9	+144		D225242829254836	32548
10	+160		D110584711365934	32564
11	+176		D004619375137473	32580
12	+192		D199257364563930	32596
13	+208		D887423641593211	32612
14	+224		D613800342746513	32628
15	+240		D222746143804950	32644
16	+256		D741109384125135	32660
17	+272		bbbbbbbbbbbbbbb	32676
18	+288		D001145848547522	32692
19	+304		D011466648347436	32708
20	+320		bbbbbbbbbbbbbbb	32724
21	+336		D391004216756481	32740
22	+352		D190011458374629	32756
23	+368		D700121232998414	32772
24	+384		D774038374157193	32788
25	+400		D283938474911584	32804
26	+416		D459941374620293	32820
27	+432		D120290485650481	32836
28	+448		bbbbbbbbbbbbbbb	32852
29	+464		D696370285630575	32868
30	+480		D113931146577764	32884
31	+496		D773000395877392	32900
32	+512		D033472274635443	32916
33	+528		D373747485930491	32932
34	+544		D400113843932114	32948
35	+560		bbbbbbbbbbbbbbb	32964
36	+576		bbbbbbbbbbbbbbb	32980
37	+592		D110021137894911	32996

Продолжение

Номер сегмента таблицы	Относительный адрес	Метка	Сегмент таблицы	Адрес
38	+608		D885693847513394	33012
39	+624		D775648886978411	33028
40	+640		D103478576849932	33044
41	+656		D488576829427836	33060
42	+672		D774638375856342	33076
43	+688		D933140405768113	33092
44	+704		bbbbbbbbbbbbbbb	33108
45	+720		D559045551438813	33124
46	+736		D399993374157738	33140
47	+752		bbbbbbbbbbbbbbb	33156
48	+768		D669484857611332	33172
49	+784		D390046574847339	33188
50	+800		D110200678511225	33204

таблица должна содержать сегменты, к которым можно обращаться по аргументам 001, 002, 003, 004, 005, 006, 007, 008 и 009. Значения 004 и 007 предусматриваются для обеспечения возможности прямой адресации. Табл. 14.2 представляет собой таблицу данных, к которой можно обращаться посредством прямой адресации. Следует отметить, что не все сегменты содержат правильные данные. Адреса в табл. 14.2 представлены в десятичном формате, а не в обычном шестнадцатеричном. Это сделано для того, чтобы легче было проследить алгоритм.

При составлении программы для прямой адресации сегментов таблицы необходимо учитывать размер самих сегментов. Обычно, чтобы получить смещение искомого сегмента от начала таблицы при прямой адресации, умножают значение ключа на число байтов в сегменте.

В представленной на рис. 14.12а и 14.12б программе поиск адреса сегмента таблицы выполняется по ключу сегмента с помощью команды сдвига. Метки PRETAB и TABLE используются в соответствии со структурой только что приведенной таблицы. В этом примере принимается, что во время выполнения программ метке PRETAB будет соответствовать адрес в основной памяти 32404, а метке TABLE — адрес 32420. Обычно ключ искомого сегмента заносится произвольным образом с помощью какого-либо устройства ввода-вывода. В данном примере ключи заданы как константы, чтобы продемонстрировать их преобразование в прямые адреса.

IBM

IBM System 360 Assembler Coding Form

X28-6509 2 U/M050
Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2 | CARD FORM NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
GOSEEK	PACK	DUBPAK,KEYA		0601
	CVB	6,DUBPAK		0602
	SLA	6,4		0603
	BAL	3,LOOKUP		0604
	PACK	DUBPAK,KEYB		0605
	CVB	6,DUBPAK		0606
	SLA	6,4		0607
	BAL	3,LOOKUP		0608
	PACK	DUBPAK,KEYC		0609
	CVB	6,DUBPAK		0610
	SLA	6,4		0611
	BAL	3,LOOKUP		0612
	BC	15,END		0613
*				
*				
LOOKUP	LA	8,PRETAB		0614
	AR	8,6		0615
	CLI	0(8),C'D'		0616
	BC	7,NODATA		0617
	MVC	OUTDATA(16),0(8)		0618
	BAL	5,PUTDATA		0619
NODATA	BCR	15,3		0620
*				

Рис. 14.12а.

IBM

IBM System/360 Assembler Coding Form

X28-6509-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 2 | CARD ELECTRO NUMBER

PROGRAMMER | DATE

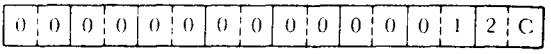
STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
PRETAB	DC	CL16' '		
TABLE	DC	50CL16' '		
KEYA	DC	CL2'12'		
KEYB	DC	CL2'24'		
KEYC	DC	CL2'44'		
DUBPAK	DC	D'0'		
OUTDATA	DC	CL16' '		

Рис. 14.12б.

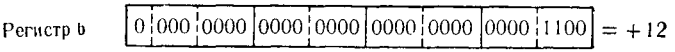
Пояснения к предложениям даются в порядке их выполнения.

Предложение 0601 — GOSEEK упаковывает значение из поля KEYA (+12) и помещает его в поле DUBPAK, содержимое которого после этого имеет следующий вид:

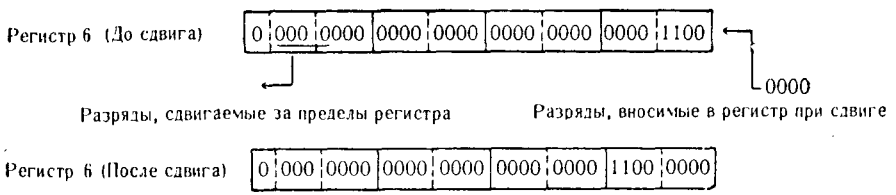


Предложение 0602 преобразует упакованную десятичную величину, содержащуюся в DUBPAK, в величину в формате числа с фиксированной точкой и помещает ее в общий регистр 6.

Число в поле KEYA было упаковано и помещено в двойное слово предложением 0601, так как в команде CVB второй операнд должен быть упакованной десятичной величиной в поле памяти размером в двойное слово. Общий регистр 6 после этого содержит следующую конфигурацию нулей и единиц:



Предложение 0603 представляет собой команду SLA. Содержимое поля KEYA — величина +12 — представляет собой номер 12-го шестнадцатибайтового сегмента таблицы. При арифметическом сдвиге на четыре двоичных разряда значение в общем регистре 6 умножается на 16, становясь таким образом относительным адресом в таблице ($12 \times 16 = 192$). Команда SLA следующим образом воздействует на общий регистр 6:



Предложение 0604 представляет собой переход с возвратом к предложению LOOKUP. После перехода общий регистр 3 содержит адрес предложения 0605.

Предложение 0614 с именем LOOKUP загружает адрес поля PRETAB в общий регистр 8. В силу условия, что адрес константы PRETAB в момент выполнения программы равен 32404, содержимое общего регистра 8 будет теперь выглядеть следующим образом:

Регистр 8 (Двоичн.)

0	000	0000	0000	0000	0111	1110	1001	0100

(Шестн.)

	0	0	0	0	7	E	9	4

Предложение 0615 производит сложение содержимого общего регистра 6 с содержимым общего регистра 8; получившаяся сумма находится в общем регистре 8. Регистр 6 содержит относительный адрес 12-го сегмента таблицы, регистр 8 содержит адрес константы PRETAB, а получившаяся после сложения сумма — это действительный абсолютный адрес 12-го сегмента таблицы. После сложения содержимое общего регистра 8 будет следующим:

Регистр 8

0	000	0000	0000	0000	0111	1111	0101	0100

= +32 596

Предложение 0616 анализирует первый байт 12-го сегмента таблицы, чтобы установить, содержит ли он правильные данные. В данном примере правильными считаются сегменты таблицы, в начале которых содержится символ D. Анализ производится сравнением.

Предложение 0617 проверяет признак результата после сравнения, произведенного предложением 0616. Если при сравнении обнаружено равенство, то сохраняется естественный порядок команд, т. е. следующим выполняется предложение 0618. Если при сравнении обнаружено неравенство, происходит переход к NODATA с тем, чтобы исключить обработку этого сегмента таблицы. В данном конкретном примере в результате сравнения будет обнаружено равенство и перехода не произойдет.

Предложение 0618 пересылает сегмент таблицы, адрес которого находится в регистре 8, в поле OUTDATA.

Предложение 0619 представляет собой переход с возвратом к подпрограмме, которая будет обрабатывать данные, содержащиеся в поле OUTDATA. После выполнения этой подпрограммы управление, согласно алгоритму, возвращается к предложению 0620.

Предложение 0620 вызывает безусловный переход по адресу, содержащемуся в регистре 3. На этом этапе выполнения программы общий регистр 3 содержит адрес предложения 0605.

Предложение 0605, упаковывает содержимое KEYB (+24) в поле DUBPAK. Поле DUBPAK будет выглядеть после этого следующим образом:

DUBPAK

0	0	0	0	0	0	0	0	0	0	0	0	2	4	C

Предложение 0606 преобразовывает величину в поле DUBPAK в величину с фиксированной точкой, помещая ее в общий регистр 6.

Теперь регистр 6 будет иметь следующее содержимое:

Регистр 6 (Двоичн.)	0 000	0000	0000	0000	0000	0000	0001	1000	= +24
(Шестн.)	0	0	0	0	0	0	1	8	

Предложение 0607 производит алгебраический сдвиг целого числа в общем регистре 6 на четыре двоичных разряда влево. В результате происходит умножение величины в регистре 6 на 16, т. е. $24 \times 16 = 384$.

Регистр 6 (Двоичн.)	0 000	0000	0000	0000	0000	0001	1000	0000	= +384
(Шестн.)	0	0	0	0	0	1	8	0	

Предложение 0608 представляет собой переход с возвратом к предложению LOOKUP с номером 0614, причем адрес возврата к предложению 0609 помещается в общий регистр 3.

Предложение 0614 загружает адрес поля PRETAB в общий регистр 8.

Предложение 0615 производит сложение относительного адреса текущего сегмента таблицы (+384), который содержится в регистре 6, с действительным текущим адресом поля PRETAB (+32404), содержащимся в регистре 8, причем получившаяся в результате сумма помещается в регистр 8. Теперь общий регистр 8 содержит действительный абсолютный адрес 24-го сегмента таблицы.

Предложения с 0616 по 0619 выполняются так, как это разъяснено выше.

Предложение 0620 вызывает безусловный переход по адресу, содержащемуся в общем регистре 3, который в данный момент выполнения программы представляет собой адрес предложения 0609.

Предложение 0609 упаковывает содержимое поля KEYC в поле DUBPAK. Поле KEYC, содержащее символы 44, представляет собой ключ 44-го сегмента таблицы. Поле DUBPAK теперь будет иметь следующий вид:

DUBPAK	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	C

Предложение 0610 преобразует упакованную десятичную величину поля DUBPAK в двоичное число с фиксированной точкой и помещает его в регистр 6.

Регистр 6 | 0 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0010 | 1100 | = +44

Предложение 0611 производит алгебраический сдвиг двоичного содержимого общего регистра 6 влево на четыре двоичных разряда. Это действие равнозначно умножению 44 на 16, что дает в результате 704 — относительный адрес 44-го сегмента таблицы.

Предложение 0612 представляет собой команду перехода с возвратом к предложению 0614, при выполнении которой адрес предложения 0613 помещается в общий регистр 3.

Предложение 0614 загружает действительный адрес поля PRETAB (+32404) в общий регистр 8.

Предложение 0615 производит сложение содержимого общего регистра 6 с содержимым общего регистра 8, причем сумма помещается в регистр 8. При этом происходит сложение действительного адреса поля PRETAB (+32404) с относительным адресом 44-го сегмента таблицы (+704) и таким образом формируется действительный абсолютный адрес сегмента таблицы (+33108).

Предложения с 0616 по 0619 выполняют обработку данных. В данном случае будет происходить переход от предложения 0617 к предложению 0620, потому что анализируемый сегмент 44 не содержит правильных данных.

Предложение 0620 вызывает переход к предложению 0613, адрес которого содержится в общем регистре 3.

Предложение 0613 вызывает безусловный переход к предполагаемой программе, завершающей работу этого модуля.

Не во всех случаях прямую адресацию элементов таблицы можно произвести посредством сдвигов. Пусть общий регистр 6 содержит номер искомого сегмента таблицы, в данном случае сегмента 5; в табл. 14.3 представлены величины, полученные при сдвиге. Коэффициент, на который должен быть умножен номер табличного элемента, представляет собой длину в байтах каждого элемента таблицы.

Если длина сегмента не равна одной из длин, приведенных в табл. 14.3, то для определения относительного адреса сегмента можно использовать обычную программу умножения чисел с фиксированной точкой или упакованных десятичных чисел. Пусть длина сегмента таблицы равна 17 байтам и программист хочет вычислить относительный адрес сегмента 19, тогда программу умножения упакованных десятичных чисел можно закодировать следующим образом:

```
PACK    DUBPAK,KEY          0001
MP      DUBPAK,=PL2'17'     0002
CVB     6,DUBPAK            0003
```

Таблица 14.3

Длина элемента таблицы (в байтах)	Команды	Окончательный относительный адрес в регистре 6	Результат сдвига
2	SLA 6,1	10	5×2 = 10
4	SLA 6,2	20	5×4 = 20
8	SLA 6,3	40	5×8 = 40
16	SLA 6,4	80	5×16 = 80
32	SLA 6,5	160	5×32 = 160
64	SLA 6,6	320	5×64 = 320
128	SLA 6,7	640	5×128 = 640
256	SLA 6,8	1280	5×256 = 1280
512	SLA 6,9	2560	5×512 = 2560

Предложение 0001 преобразует содержимое поля KEY (номер сегмента таблицы, равный 19), упаковывая его в поле DUBPAK, которое имеет теперь следующий вид:

DUBPAK

0	0	0	0	0	0	0	0	0	0	0	0	1	9	C

Предложение 0002 умножает содержимое поля DUBPAK на упакованный десятичный литерал +17, представляющий длину каждого сегмента таблицы в байтах. В поле DUBPAK содержится теперь величина +323 — относительный адрес сегмента 19.

DUBPAK

0	0	0	0	0	0	0	0	0	0	0	3	2	3	C

Предложение 0003 представляет собой команду CVB, которая и помещает упакованную десятичную величину, взятую из поля DUBPAK, в общий регистр 6 в формате с фиксированной точкой.

Регистр 6

0	000	0000	0000	0000	0000	0001	0100	0011	+323
	0	0	0	0	0	1	4	3	

Программу умножения чисел с фиксированной точкой для выполнения того же самого процесса можно закодировать следующим образом:

```
PACK    DUBPAK,KEY      0001
CVB     6,DUBPAK        0002
MH      6,=H'17'        0003
```

Предложение 0001 упаковывает содержимое поля KEY в поле DUBPAK, которое теперь будет содержать величину +19.

Предложение 0002 преобразует величину поля DUBPAK (+19) в величину в формате с фиксированной точкой в общем регистре 6.

Регистр 6

0	000	0000	0000	0000	0000	0000	0001	0011
	0	0	0	0	0	0	1	3

+19

Предложение 0003 умножает содержимое общего регистра 6 на литерал +17 длиной в полуслово, который представляет собой длину каждого сегмента таблицы. Теперь общий регистр 6 будет содержать величину с фиксированной точкой +323 — относительный адрес сегмента 19.

Регистр 6

0	000	0000	0000	0000	0000	0001	0100	0011
	0	0	0	0	0	1	4	3

+323

Как программа умножения чисел с фиксированной точкой, так и программа умножения упакованных десятичных чисел правильно нашли относительный адрес 19-го сегмента таблицы. Но следует особо подчеркнуть, что это был только относительный адрес. Эта величина должна быть суммирована с действительным адресом, доступным в процессе выполнения программы, определяя таким образом действительный абсолютный адрес сегмента таблицы.

Действительный адрес начальной точки таблицы можно получить несколькими способами. В табл. 14.2 программист установил «фиктивное» поле той же длины, что и элемент таблицы, и поместил это поле непосредственно перед самой таблицей. Сделав это, он может умножить ключ таблицы на число байтов в каждом элементе и получить относительный адрес, который должен быть суммирован с адресом поля PRETAB. Если бы он произвел сложение относительного адреса ключа с действительным адресом TABLE, результирующий абсолютный адрес в таблице был бы на 16 байтов больше желаемого, например:

$$\frac{\text{Ключ сегмента таблицы}}{(4)} \times \frac{\text{Длина сегмента}}{(16)} + \frac{\text{Адрес таблицы}}{(32420)} = 32484$$

Общий адрес 32484 указывает на начало сегмента 5 таблицы, а не сегмента 4. Если программист не желает для предварительной адресации таблицы устанавливать фиктивное поле, он

может получить правильный прямой адрес в результате следующих расчетов:

$$\frac{\text{Ключ сегмента таблицы}}{(4)} - 1 \times \frac{\text{Длина сегмента}}{(16)} + \frac{\text{Адрес таблицы}}{(32420)} = 32468$$

или

$$\frac{\text{Ключ сегмента таблицы}}{(4)} \times \frac{\text{Длина сегмента}}{(16)} - \frac{\text{Длина сегмента}}{(16)} + \frac{\text{Адрес таблицы}}{(32420)} = 32468$$

Представленные в этих расчетах величины нужно заменить действительными значениями длины сегмента таблицы и адреса по каждой из создаваемых таблиц.

Если длина сегмента таблицы равна 21 байту, два способа вычислений, о которых только что упоминалось, могут быть закодированы следующим образом:

```
PACK    DUBPAK,KEY
SP      DUBPAK,=PL1'1',
MP      DUBPAK,=PL2'21'
CVB     6,DUBPAK
LA      7,TABLE
AR      7,6
```

или

```
PACK    DUBPAK,KEY
MP      DUBPAK,=PL2'21'
SP      DUBPAK,=PL2'21'
CVB     6,DUBPAK
LA      7,TABLE
AR      7,6
```

Д. ТАБЛИЦЫ СО СМЕШАННЫМИ ДАННЫМИ

Вероятнее всего, что обычные таблицы, составляемые программистом, будут содержать данные в формате символов кода EBCDIC. Тем не менее таблица может содержать данные, представленные в шестнадцатеричном формате, двоичном формате, формате чисел с фиксированной точкой, упакованном десятичном формате или в любой комбинации этих форматов. Таблица смешанного формата может быть задана с помощью предложений DC или в процессе выполнения программы.

Для того чтобы разъяснить смысл этой концепции, нужно сначала описать предполагаемое содержание такой таблицы. Типичная таблица могла бы содержать следующие данные:

Таблица	250 сегментов
Один сегмент	4 поля
Поле 1	3 байта символьных данных
Поле 2	4 байта — величина с фиксированной точкой, выравненная по границе полного слова
Поле 3	6 байтов упакованных десятичных данных
Поле 4	3 байта символьных данных

Эта таблица инициализируется следующим образом:

1. Проблемная программа загружает в первые 3 байта каждого сегмента таблицы величины ключей. Эти 3 байта до загрузки величин ключей должны быть пробелами.

2. Поле 2, представляющее собой полное слово, содержащее величину с фиксированной точкой, должно быть инициализировано как величина с фиксированной точкой +0. С величиной этого поля можно складывать величины с фиксированной точкой и хранить их здесь для последующего использования. В некоторых случаях до попытки сложения этого поля с другой величиной неизвестно, находится ли уже в поле правильная величина с фиксированной точкой. Это и служит причиной инициализации поля занесением величины с фиксированной точкой +0.

3. Поле 3, которое должно использоваться как содержащее шестибайтовое упакованное десятичное число, должно быть инициализировано занесением упакованной десятичной величины +0 по тем же соображениям, что и поле 2. Упакованные десятичные данные не обязательно накапливаются в поле, а программа может попытаться использовать это поле для арифметических операций до того, как в него будет занесена некая величина.

4. Поле 4 содержит символьные данные в большинстве сегментов таблицы, но не во всех. Эти данные заносятся в таблицу в процессе выполнения программы. Это поле первоначально заполняется пробелами.

Относительно простой способ формирования такой таблицы состоит в использовании предложений DC, в которых данные указаны в шестнадцатеричном формате. Следует помнить о том, что первый байт второго поля в каждом сегменте должен находиться на границе полного слова.

```
ALIGN   DC    F'0'
FILL1   DC    CL1'Б'
TABLE   DC    250X'404040000000000000000000C404040'
                   Поле 1 Поле 2     Поле 3    Поле 4
```

Предложение DC с именем ALIGN задает поле с этим же именем на границе полного слова, поэтому каждый четвертый байт от начала этого поля находится на границе полного слова. В силу того что поле 2 должно начинаться на границе полного слова, а длина поля 1 равна только 3 байтам, используется однобайтовая константа FILL1 для того, чтобы сместить начало поля TABLE на 1 байт в сторону возрастания адресов. Теперь начало поля TABLE отстоит на 3 байта влево от границы полного слова, а так как первые 3 байта поля TABLE как раз заполняют этот промежуток, поле 2 начинается на границе слова. Поскольку каждый сегмент таблицы имеет длину 16 байтов (что составляет четыре полных слова), во всех последующих сегментах таблицы начало поля 2 будет приходиться на границу полного слова.

Взглянув на предложение DC для поля TABLE, можно увидеть, что теперь все поля инициализированы в соответствии с требованиями таблицы.

1. Поле 1 (3 байта, что соответствует шести шестнадцатеричным цифрам) инициализировано пробелами, представляемыми как 404040.
2. Поле 2 (4 байта, что соответствует восьми шестнадцатеричным цифрам) содержит величину с фиксированной точкой +0, т. е. X'00000000'.
3. Поле 3 (6 байтов, что соответствует двенадцати шестнадцатеричным цифрам) содержит правильную упакованную десятичную величину +0, т. е. X'00000000000C', где символ C в младшей позиции представляет собой знак плюс для этого поля.
4. Поле 4 (3 байта, что соответствует шести шестнадцатеричным цифрам) также заполняется пробелами, т. е. X'404040'.

Итак, все поля инициализированы, а в предложении DC указана общая длина таблицы со всеми ее сегментами, количество которых равно 250 и которые имеют одинаковый формат. Теперь таблица готова к загрузке.

В процессе работы проблемной программы некоторые поля этой таблицы могут использоваться как ограниченные накопители. Предположим, например, что шестибайтовое упакованное десятичное поле в каждом сегменте таблицы должно использоваться как накопитель до возникновения определенных условий. Когда происходит это событие, упакованная десятичная величина извлекается из поля 3 каждого сегмента таблицы и в поле восстанавливается исходное содержимое — величина +0. Допустим, что все величины уже извлечены, тогда стандартную подпрограмму для повторной инициализации поля 3 в каждом сегменте можно закодировать следующим образом:

```
RESETF3   LA     10,TABLE+7          00001
          LA     9,250               00002
LOOPSET   ZAP    0(6,10),=PL1'0'     00003
          AH     10,=H'16'           00004
          BCT    9,LOOPSET           00005
          BC     15,ALLSET           00006
```

Предложение 00001 загружает адрес поля TABLE+7 в общий регистр 10. Адрес, порожденный операндом TABLE+7, указывает на первый байт упакованного десятичного поля в первом сегменте таблицы.

Предложение 00002 использует команду LA для того, чтобы поместить величину +250 в общий регистр 9. Регистр 9 — это регистр, служащий для управления циклом с помощью команды BCT, и если в него загружена эта величина, то программа LOOPSET будет повторена 250 раз.

Предложение 00003 использует команду ZAP для вторичной установки в нуль упакованного десятичного поля в сегменте таблицы, текущий адрес которого содержится в общем регистре 10. Хотя в качестве второго операнда используется однобайтовый литерал, первый операнд указывает, что длина поля данных в предложении составляет 6 байтов. Команда ZAP вторично устанавливает все 6 байтов десятичных цифр в нуль, за исключением младшего шестнадцатеричного разряда, который содержит правильный код знака — С.

Предложение 00004 увеличивает содержимое общего регистра 10 на 16, с тем чтобы указать на упакованное поле следующего сегмента таблицы.

Предложение 00005 представляет собой команду BCT управления циклом. Пока содержимое общего регистра 9 не уменьшилось до нуля, будет осуществляться переход к предложению LOOPSET.

Предложение 00006 осуществляет безусловный переход к предполагаемой стандартной подпрограмме. Факт перехода означает, что все упакованные десятичные поля таблицы заново установлены в +0.

Как уже упоминалось в этом разделе по поводу таблиц и программ просмотра, есть много методов и их вариантов для решения конкретных задач. Построение таблицы и последующий поиск в ней должны отвечать требованиям соответствующей проблемной программы. Но и с учетом этого программисту предоставляются широкие возможности выбора эффективной методики решения его конкретной задачи.

Глава 15

Редактирование данных

А. КОМАНДЫ РЕДАКТИРОВАНИЯ

Команда Edit — ED (Отредактировать)

Мнемоника	Код операции	Формат операндов
ED	DE	$D_1(L,B_1),D_2(B_2)$

Команда преобразует содержимое второго операнда из упакованного десятичного формата в зонный десятичный формат, а затем модифицирует в соответствии с образцом (шаблоном). Адрес образца указывается первым операндом.

Обработка данных происходит слева направо побайтно, а длина обрабатываемого поля указывается явно либо неявно первым операндом.

В данной главе возможности редактирования раскрываются более подробно.

CC	BC	Условие
0	8	Результатом редактирования является поле нулей
1	4	Результатом редактирования является величина, меньшая 0
2	2	Результатом редактирования является величина, большая 0

Команда Edit and Mark — EDMK (Отредактировать и отметить)

Мнемоника	Код операции	Формат операндов
EDMK	DF	$D_1(L,B_1),D_2(B_2)$

В основном эта команда выполняет те же самые функции, что и команда ED. Кроме того, производятся некоторые особые действия во время обработки исходных данных, определяемых вторым операндом, в соответствии с образцом, указанным первым операндом. Эти действия состоят в том, что команда помещает в общий регистр 1 адрес первого значащего символа (символа, не равного нулю), который заносится в поле результата прежде, чем в этом поле обнаруживается символ начала значимости. Если символ начала значимости обнаруживается в образце раньше помещения в него первого значащего символа, адрес в общий регистр 1 не записывается. В следующих разде-

лах главы дается значительно более подробная информация об этой команде.

CC	BC	Условие
0	8	Результатом редактирования является поле нулей
1	4	Результатом редактирования является величина, меньшая 0
2	2	Результатом редактирования является величина, большая 0

Б. ПРИМЕНЕНИЕ КОМАНД РЕДАКТИРОВАНИЯ

1. Необходимость редактирования

В отношении своей внутренней структуры данные вообще и десятичные величины или величины с плавающей точкой в частности могут рассматриваться как имеющие двоичный, шестнадцатеричный или символьный формат. Однако не всегда имеет смысл выводить эти данные на графическое устройство непосредственно в том виде, в каком они представлены в ЭВМ, не преобразуя их внутреннюю структуру. Здесь под графическим устройством вывода понимается любое устройство вывода, которое представляет данные в виде графических символов. К таким устройствам относятся АЦПУ, пультовые пишущие машинки, дисплеи и другие подобные устройства. На этих устройствах обычное символьное представление в коде EBCDIC можно воспроизвести без какого-либо изменения внутренней структуры выводимых данных. Но упакованные десятичные величины и величины с фиксированной точкой до передачи на выводное устройство должны быть преобразованы. Частично эти изменения производятся командами редактирования и частично — другими командами в зависимости от вида представляемых данных. Так как команды редактирования применимы только к данным в упакованном десятичном формате, с их помощью можно редактировать только данные, имеющие цифровое представление. Но в процессе самого редактирования в различные части редактируемого поля могут быть внесены буквы и специальные символы.

Ниже перечислены несколько общих видов редактирования:

а) устранение старших незначащих нулей в числовом поле;

б) внесение знаков пунктуации в числовое поле;

в) задание положительных и отрицательных величин путем внесения таких описателей, как CR [1]), —, + и т. д.;

[1]) Сокращение от слова «кредит»; используется для представления отрицательных чисел наряду со знаком минус. — *Прим. ред.*

г) добавление к величинам, относящимся к денежным операциям, знака доллара ($) в фиксированную или в произвольную позицию.

Для увязки сказанного с практическими применениями ниже приводится табл. 15.1, содержащая данные, которые должны быть подвергнуты редактированию, а также различные выходные форматы, которые эти данные могут принимать в процессе редактирования.

Таблица 15.1

Содержимое поля данных	Отредактированные выходные данные
0 0 3 9 5 6 1 C	395.61
0 0 3 9 5 6 1 F	395.61+
0 0 3 9 5 6 1 F	$ 395.61
0 0 3 9 5 6 1 C	$395.61
2 5 7 8 8 1 4 D	25 788.14–
2 5 7 8 8 1 4 D	25788.14CR
2 5 7 8 8 1 4 D	(25 788.14)
0 0 0 0 0 0 6 F	6
0 0 0 0 0 0 6 F	.06
0 0 0 0 0 0 6 F	6¢

В следующих разделах подробно рассматриваются возможности составления программ редактирования. В данный же момент важно понять необходимость представления числовых данных в таком формате, чтобы он был не только легок для восприятия, но и выражал связь между исчисляемым и самим

числом. Это означает, к примеру, что если числовое поле используется для выражения количества квадратных дюймов при определении площади пола, то было бы нелогично представить эту величину как 26893; представление этой величины в виде «26893 кв. дюйм» было бы более подходящим, не говоря о том, что оно описывает само себя. Во многих случаях данные подобного рода представляются в виде столбца; поэтому заголовок столбца можно использовать для описания размерности числовых величин. Столбец может выглядеть следующим образом:

Квадратные дюймы
26893
426
5830

Величины изображены здесь правильно, а заголовок столбца характеризует их размерность. При подготовке данных в формате, пригодном для вывода на графическое устройство, очень важно, чтобы данные описывали сами себя. Они должны легко читаться, быть логично составленными и описывать смысл того, что они представляют. Это легко достигается правильным редактированием выходного формата данных. Сравните два приведенных ниже списка, обратив внимание на то, что второй список имеет более наглядный характер, в то время как в первом данные представлены в общем виде.

#	Длина	Ширина	Вес	$
0039658	368	36	36—5	95.00
0078621	52	22	3—11	7.00
0113965	89	48	12	25.00

Номер товара	Длина (футы)	Ширина (дюймы)	Вес на погонный фут	Стоимость на единицу измерения
39658	368′	36″	36 фунтов 5 унций	$95.00 за погонный фут
78621	52′	22″	3 фунта 11 унций	$7.00 за фут
113965	89′	48″	12 фунтов	$25.00 за 1 ед.

Во втором списке описание данных в ряде случаев приведено как в заголовках столбцов, так и в данных под заголовками; заголовок «длина» указывает на то, что представленные величины выражают количество футов, а само выражение этой величины сопровождается стандартным символом (апострофом), служащим для указания, что предшествующее число означает

количество футов. Предполагается, что, когда данные будут распечатаны какой-либо программой, ими будет пользоваться многочисленный персонал, работающий с системой, поэтому выходные данные должны иметь по возможности наиболее логичную и четкую форму.

2. Структура операторов редактирования

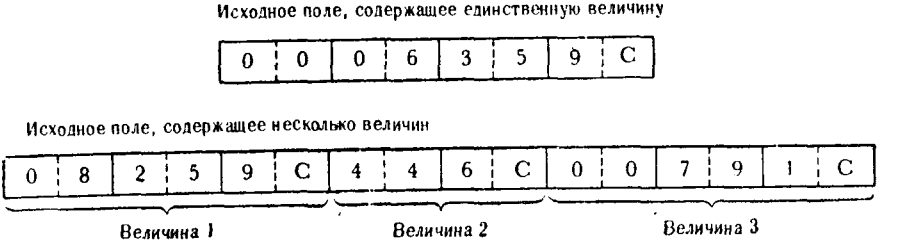
Функция редактирования выполняется командой ED или командой EDMK. Так как обе эти команды выполняют относительно сходные функции, прежде чем перейти к разбору самих команд, опишем функции редактирования. Их удобно описывать в терминах компонентов, находящихся в активном состоянии в процессе выполнения программы редактирования. Эти компоненты можно с успехом рассматривать сами по себе как отдельные маленькие команды; используя различные комбинации этих необязательных компонентов, можно получать разнообразные варианты операции редактирования. В выполнении команд редактирования участвуют следующие объекты:

1. Исходное поле.
2. Образец для редактирования:
   а) символы-заполнители,
   б) символы выбора цифры,
   в) символы начала значимости,
   г) символы разделения полей.
3. Индикатор значимости (S-индикатор).

С каждым из этих объектов связан некоторый набор правил, определяющих способ выполнения команды редактирования.

Исходное поле. Исходное поле является областью хранения исходных данных, содержащей числовую величину, которую необходимо отредактировать. Оно может содержать данные для заполнения 255 байтов выходного отредактированного формата, при условии что в процессе всего редактирования будет обработано поле с максимальной длиной 256 байтов, включая длину образца. Содержащиеся в исходном поле данные должны иметь упакованный десятичный формат. Упакованная величина в исходном поле не обязательно должна быть величиной со знаком, но если она таковой является, то в этом случае знак должен занимать самый правый полубайт поля. Все остальные полубайты поля должны содержать правильные десятичные цифры — шестнадцатеричные цифры от 0 до 9. Если нужно отредактировать исходное поле, содержащее несколько упакован-

ных десятичных величин, каждая величина может содержать код знака в самом правом полубайте.



Результат редактирования исходного поля замещает образец, который обычно находится в области памяти, предназначенной для выходной информации.

Образец для редактирования. В образце указывается, какие виды функций редактирования должны быть выполнены. В отличие от исходного поля каждый байт образца должен быть в распакованном зонном формате. Предполагается, что образец состоит из

а) символов-заполнителей;
б) символов выбора цифры;
в) символов начала значимости;
г) символов разделения полей — только при редактировании нескольких полей;
д) произвольных символов, которые должны быть без изменений помещены в отредактированное выходное поле.

Некоторые из этих символов невозможно вывести на печать, однако все они участвуют в управлении редактированием. Ниже дается общее описание каждого из них. Выполняемые ими функции в целом описываются в следующем разделе.

1. Символом-заполнителем может быть любой правильный символ, включая пробел, $, *, (,), # и т. п. Символ-заполнитель заменяет незначащие цифры в начальной части исходного поля.

2. Символ выбора цифры. Этот символ представляет собой конфигурацию X'20' и не может быть выведен на печать. Символ используется для указания в образце позиции, которая должна быть заполнена цифрой из исходного поля, если таковая в нем имеется. Символ выбора цифры всегда заменяется или символом-заполнителем образца, или цифрой из исходного поля.

3. Символ начала значимости имеет нераспечатываемую конфигурацию X'21'. Символ указывает командам редактирования

на то, что все оставшиеся (младшие) цифры исходного поля должны рассматриваться как значащие и что подавление старших незначащих нулей должно прекратиться в этой точке.

4. Символ разделения полей — нераспечатываемая конфигурация X'22'. Этот символ используется только в случае, если редактируется исходное поле, содержащее несколько числовых величин. Разделение нескольких величин происходит благодаря появлению пропуска между отдельными величинами.

5. Кроме того, в образец может быть внесен любой правильный символ, необходимый для окончательной редакции выходных данных.

Индикатор значимости (S-индикатор). Индикатор значимости является по сути переключателем состояния, который проверяется в процессе редактирования с целью определения режима обработки данных. Индикатор значимости может находиться в двух состояниях: в состоянии «выключено», соответствующем нулю, и «включено», соответствующем единице. В зависимости от содержимого образца и исходного поля индикатор значимости может несколько раз менять свое состояние. Ниже приводятся условия, переводящие индикатор в состояние «включено».

1. Наличие символа начала значимости в образце.
2. Занесение в образец значащей цифры из исходного поля прежде, чем в образце будет обнаружен символ начала значимости.

Условия, устанавливающие индикатор в состояние «выключено»:

1. Наличие в образце символа разделения полей.
2. Обнаружение в исходном поле в самом младшем полубайте знака плюс.
3. Начало каждой операции редактирования.

Кроме того, в зависимости от состояния индикатора значимости в конце выполнения каждой функции редактирования устанавливается признак результата. Если редактировалось поле с несколькими величинами, как положительными, так и отрицательными, то окончательный признак результата укажет знак последней из этих величин.

До настоящего момента функции операций редактирования и компоненты, принимающие участие в этих операциях, описывались в значительной степени разобщенно. Данное выше описание само по себе не претендует на глубокую интерпретацию свойств операций редактирования. Такое описание нужно было лишь для знакомства с терминами и названиями, которые бу-

дут использованы при детальном разборе команд редактирования. При прочтении оставшейся части главы рекомендуется иногда обращаться к предыдущим разделам. Это поможет всесторонне разобраться во всех возможных вариантах процесса редактирования.

3. Функциональные характеристики редактирования

Теперь, когда для команд редактирования уже описаны основные компоненты и выполняемые ими функции, пора рассмотреть их совместно, с тем чтобы выявить взаимодействие между ними при различных обстоятельствах. Краткое описание обычных функций, выполняемых этими компонентами, явится предисловием к такому рассмотрению. Это описание можно рассматривать как упорядоченный набор основных правил выполнения оператора редактирования.

1. По мере того как каждая цифра редактируется в соответствии с образцом, формат исходного (посылающего) поля преобразуется из упакованного десятичного в зонный десятичный (EBCDIC).
2. Редактирование и просмотр образца выполняются слева направо (от старших байтов к младшим); число обрабатываемых байтов указывается явно или неявно образцом.
3. Отредактированное исходное поле с включением всех дополнительных символов, внесенных согласно образцу, полностью замещает образец.
4. Для того чтобы впоследствии отредактировать еще одно исходное поле, используя ту же самую область памяти, нужно заново построить образец.

Так как некоторые символы образца не могут быть распечатаны, для их графического представления будут использованы псевдосимволы:

1. Ь правильный пробел; X'40'
2. d символ выбора цифры; X'20'
3. S символ начала значимости; X'21'
4. f символ разделения полей; X'22'

Подготовка к редактированию заключается в определении образца как константы. Затем эта константа пересылается в выходное поле перед каждой командой редактирования для восстановления образца. Чтобы избежать необходимости подставлять для нераспечатываемых компонентов псевдосимволы, рекомендуется константу образца записывать в шестнадцатеричной форме.

Такая константа может выглядеть следующим образом:

С использованием псевдосимволов образец будет закодирован следующим образом:

EDPATRN DC C'ъддддЅд'

Когда будет составлена и распечатана исходная программа, эта константа будет выглядеть следующим образом:

EDPATRN DC C' '

Так как константа выражена в символьной форме, сам образец представлен в виде пробелов. Причина этого — отсутствие распечатываемых символов в составе образца. Поэтому гораздо удобнее представить константу образца в шестнадцатеричной форме, с тем чтобы в листинге исходной программы шаблон мог быть отражен в виде распечатываемых символов.

После того как образец засылается в выходное поле, кодируется команда редактирования и происходит следующее.

Самый левый (старший) байт образца принимается за символ-заполнитель. Затем цифры исходного поля (полубайты) по одной слева направо пересылаются в образец. Символ-заполнитель заменяет незначащие цифры (шестнадцатеричные нули) исходного поля до тех пор, пока в образец из исходного поля не перейдет первая значащая цифра, или до тех пор, пока в образце не встретится символ начала значимости. Если значащая цифра исходного поля поступает на редактирование до того, как в образце встречается символ начала значимости, эта цифра, а также все оставшиеся цифры именно этой величины исходного поля замещают в этой части образца оставшиеся символы выбора цифры. Поскольку переход значащей цифры в образец устанавливает индикатор значимости (S-индикатор) в состояние «включено», следующие далее в этой части образца символы начала значимости игнорируются. Если символ начала значимости встречается в образце раньше значащей цифры в исходном поле, индикатор значимости устанавливается в состояние «включено» и все оставшиеся цифры обрабатываемой величины замещают находящиеся в оставшейся части образца символы выбора цифры. Несколько последних предложений были

довольно длинными и могут показаться не вполне ясными, поэтому далее следует более подробное изложение тех же вопросов.

Допустим, что образец представлен константой

```
EDITPAT     DC      X'402020212020'
```

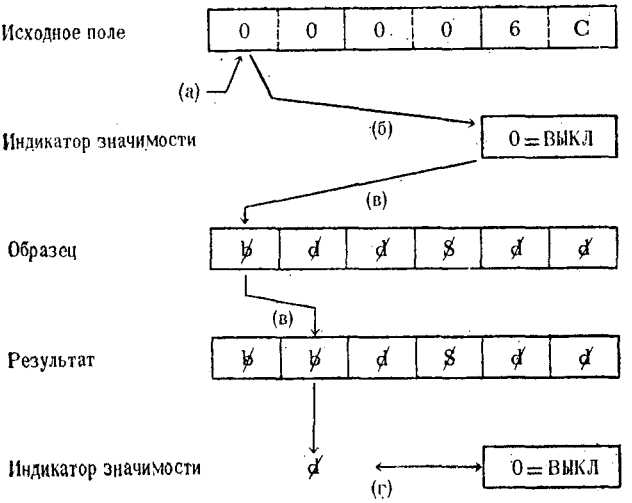
или

```
EDITPAT     DC      C'ЪддД$Дд'
```

Ниже демонстрируется редактирование двух величин с помощью данного образца; при редактировании одной величины символ начала значимости встречается раньше, чем значащая цифра; при редактировании второй значащая цифра передается из исходного поля прежде, чем в образце обнаруживается символ начала значимости.

Пример 1.

Шаг 1.



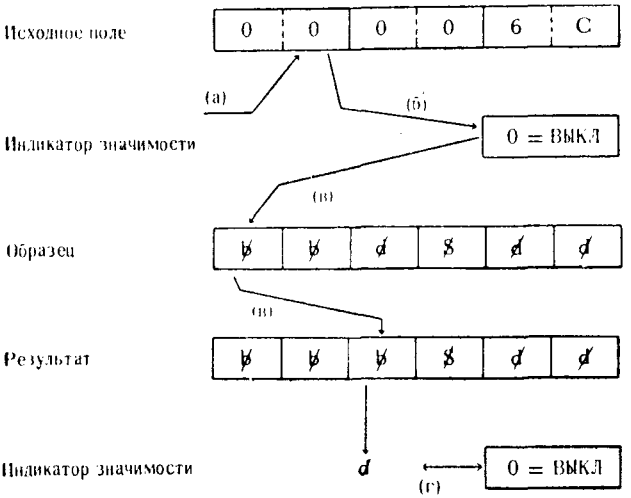
а. Проверяется первая цифра исходного поля: значащая она или нет.

б. Так как она незначащая, проверяется, включен ли индикатор значимости.

в. Индикатор значимости выключен, в соответствии с этим символ-заполнитель замещает в образце первый символ выбора цифры.

г. Проверяется, был ли замещенный в образце символ символом начала значимости. Если он таковым не был, переключатель значимости остается в выключенном состоянии.

Шаг 2.



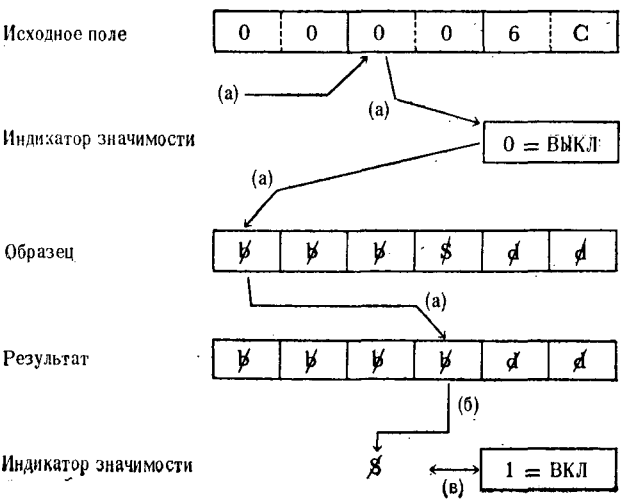
а. Проверяется вторая цифра исходного поля.

б. Она не является значащей, поэтому проверяется индикатор значимости.

в. Индикатор значимости выключен, в результате этого второй символ выбора цифры замещается символом-заполнителем.

г. Символ, замещенный символом-заполнителем, не является символом начала значимости, поэтому индикатор значимости остается в выключенном состоянии.

Шаг 3.

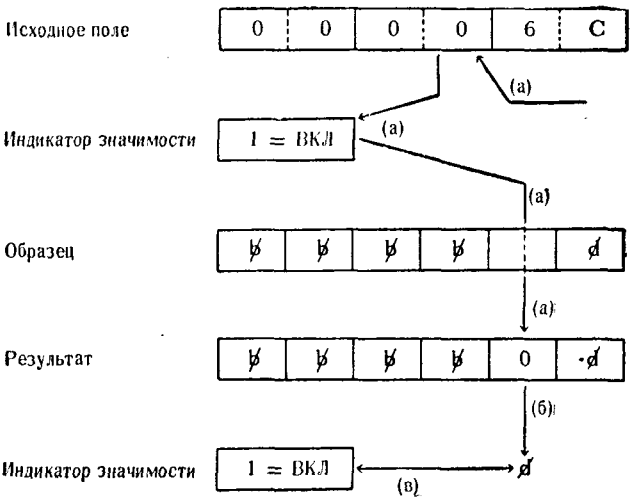


а. Третья цифра исходного поля не является значащей и индикатор значимости не включен, поэтому следующий символ образца замещается символом-заполнителем.

б. Проверяется, является ли символ, замещенный символом-заполнителем, символом начала значимости.

в. Замещенный символ является символом начала значимости, поэтому индикатор значимости устанавливается во включенное состояние. Это означает, что оставшиеся цифры независимо от того, значащие они или нет, должны быть занесены в образец.

Шаг 4.

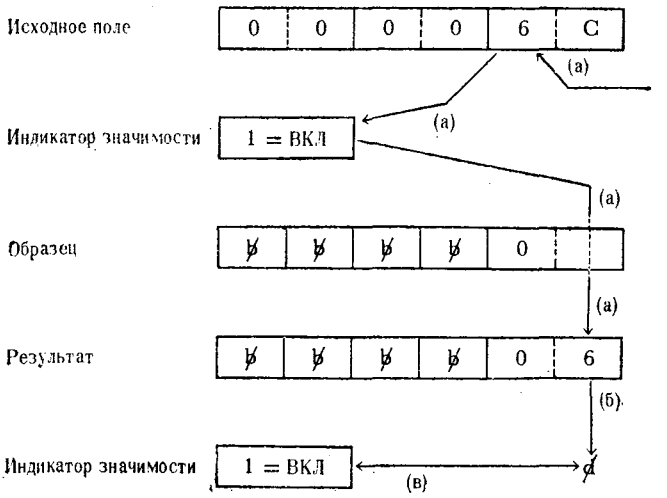


а. Четвертая цифра исходного поля оказалась незначащей, но так как переключатель значимости включен, цифра пересылается в образец, замещая следующий символ выбора цифры.

б. Проверяется символ образца, замещенный цифрой из исходного поля, и определяется, был ли этот символ символом разделения полей.

в. Замещенный символ таковым не был, поэтому переключатель значимости остается во включенном состоянии.

Шаг 5.



а. Пятая цифра исходного поля замещает следующий имеющийся в образце символ выбора цифры. Это обусловлено тем, что цифра исходного поля была значащей цифрой и/или индикатор значимости был во включенном состоянии.

б. Затем проверяется символ образца, замещенный цифрой исходного поля, и определяется, был ли он символом разделения полей.

в. Символ таковым не был, поэтому состояние индикатора значимости не изменяется.

В этот момент редактирование заканчивается, пройдя по всей длине поля, заданного для этого оператора. Если теперь этот образец распечатать или вывести на дисплей, он будет выглядеть как 06. Если бы указывалось, что при редактировании должен быть обработан еще 1 байт исходного поля, наличие правильного знака плюс явилось бы причиной установки индикатора значимости в выключенное состояние.

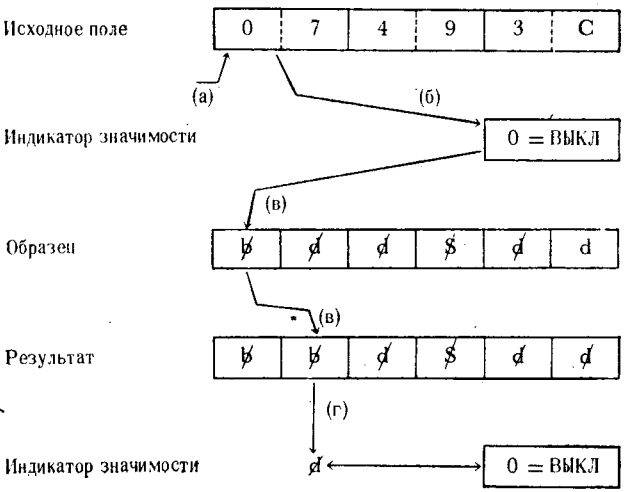
Пример 2.

Шаг 1. См. рис. на стр. 555.

а. Так же как и в предыдущем примере, проверятся, является ли первая цифра исходного поля значащей.

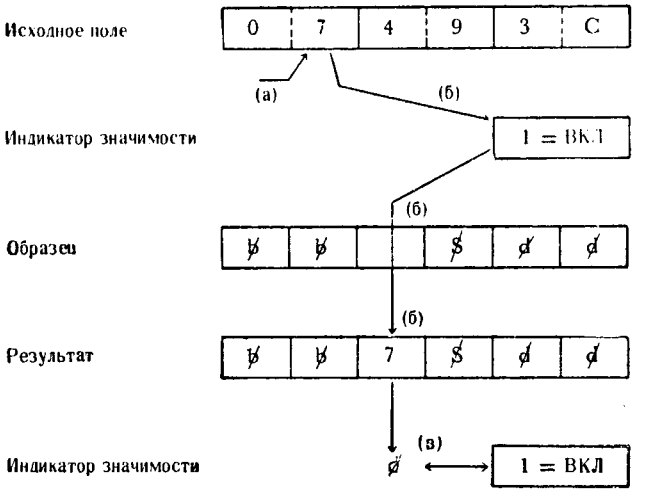
б. Так как первая цифра не является значащей, проводится проверка, находится ли индикатор значимости в состоянии «включено».

в. Так как индикатор значимости выключен, незначащая цифра исходного поля подавляется и первый символ выбора цифры в образце замещается символом-заполнителем.



г. Проверяется символ образца, замещенный символом-заполнителем, и, так как он не является символом начала значимости, состояние индикатора значимости не изменяется.

Шаг 2.

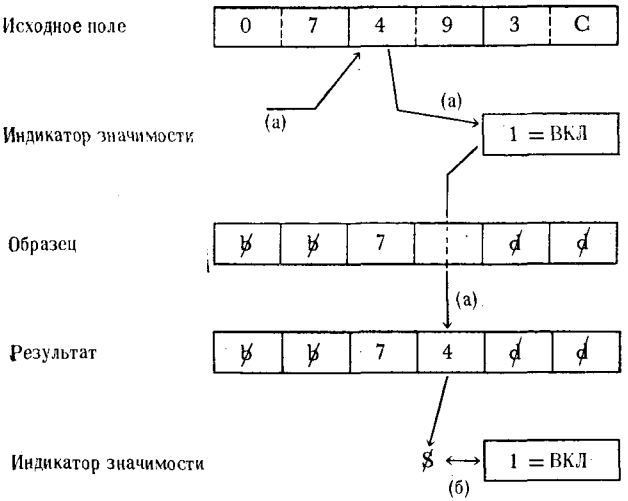


а. Проверяется второй символ исходного поля и обнаруживается, что это значащая цифра.

б. Индикатор значимости устанавливается во включенное состояние и значащая цифра исходного поля переносится в образец, замещая следующий символ выбора цифры.

в. При замещении проверяется, является ли замещаемый символ символом разделения полей. Он таковым не является, поэтому состояние индикатора значимости не изменяется.

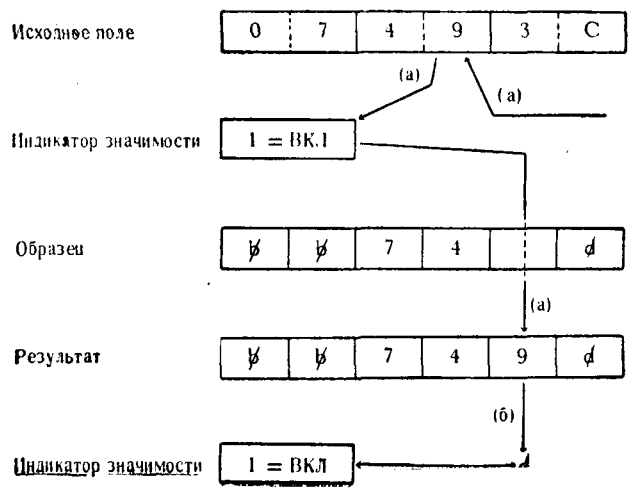
Шаг 3.



а. Третий символ исходного поля заносится в образец ввиду того, что индикатор значимости включен и заносимый символ является значащей цифрой.

б. Устраняемый из образца символ начала значимости не влияет на состояние индикатора значимости, потому что он уже был включен. Так как замененный символ не был символом разделения полей, индикатор значимости не выключается.

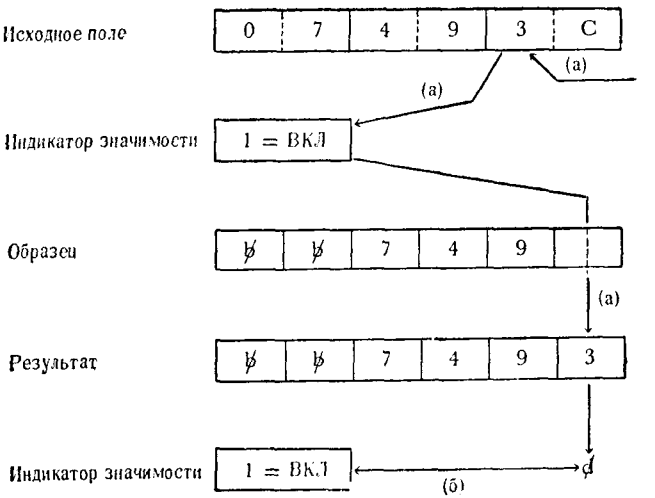
Шаг 4.



а. На этом этапе четвертая цифра исходного поля замещает в образце соответствующий символ выбора цифры; цифра эта значащая, и индикатор значимости включен.

б. Производится проверка, является ли замещенный символ символом разделения полей. Так как результат проверки отрицательный, состояние индикатора значимости не изменяется.

Шаг 5.



а. На данном этапе пятая цифра исходного поля замещает в образце последний символ выбора цифры. Эта цифра является значащей, и поэтому индикатор значимости остается включенным.

б. Проверяется, был ли замещенный символ символом разделения полей. В соответствии с отрицательным результатом проверки состояние индикатора значимости не изменяется.

Все редактируемое поле полностью просмотрено, длина его неявно определяется длиной образца, и операция редактирования заканчивается. Данные, сформированные в поле образца, после записи на устройство вывода будут иметь следующий вид:

7493

Во избежание недоразумений следует пояснить, что приведенные выше примеры и иллюстрации освещают процесс редактирования только с принципиальной стороны. Они не содержат

технических деталей, но достаточны, чтобы донести до читателя основные понятия и возможности редактирования.

До сих пор объяснялось, как взаимодействуют в процессе редактирования символ выбора цифры, символ начала значимости и индикатор значимости. Фактически на данных примерах демонстрировалась простая операция подавления незначащих нулей. Другим часто используемым вариантом редактирования является внесение в редактируемую величину знаков пунктуации или каких-либо других символов, например преобразование величины 03952867 в величину 39,528.67. Образец задается в шестнадцатеричной форме со знаками пунктуации в соответствующих местах. Образец такого вида может быть, например, определен как константа, закодированная в шестнадцатеричной форме следующим образом:

```
EDPATRN     DC      X'40206B2020214B2020'
```

или

```
EDPATRN     DC      C'bd,ddd$.dd'
```

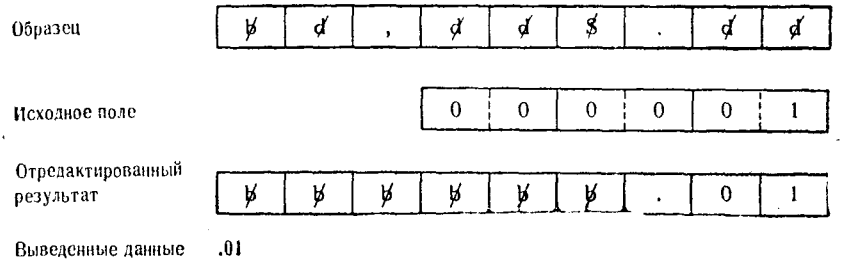
Подразумевается, что при этом образце максимальная величина исходного поля составляет +999999. Эта величина после редактирования примет вид 9,999.99. Если величина, редактируемая по этому образцу, меньше максимальной, все незначащие цифры подавляются до тех пор, пока из исходного поля не поступит первая значащая цифра или пока в шестом байте образца не встретится символ начала значимости. Если первой цифрой исходного поля является незначащая цифра, то третий байт образца будет заменен символом-заполнителем. Ниже показаны несколько исходных полей и результаты их редактирования по приведенному образцу. Было бы полезно перед тем, как приступить к изучению примеров, еще раз просмотреть правила, касающиеся состояний индикатора значимости, символа начала значимости, и правила пересылки значащей цифры.

Пример 1.

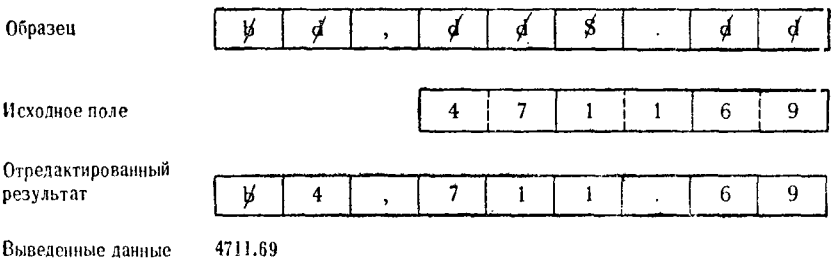
Образец	b	d		d	d	$		d	d
Исходное поле				0	0	3	6	7	5
Отредактированный результат	b	b	b	b	3	6		7	5

Выведенные данные 36.75

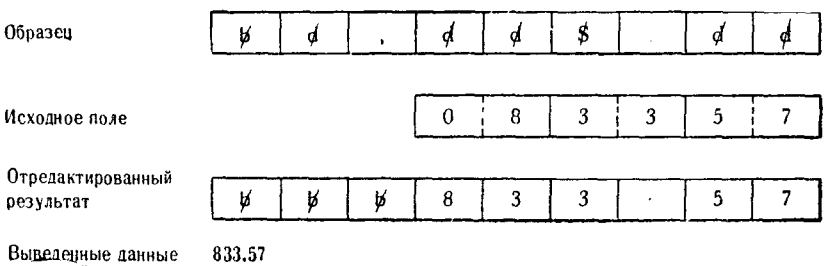
Пример 2.



Пример 3.



Пример 4.



Во многих случаях, в особенности когда распечатываются величины, связанные с денежными операциями, в программе может потребоваться, чтобы имелась видимая разница между положительными и отрицательными величинами, такими, как дебит и кредит, или приходными и расходными суммами. Как уже говорилось выше, индикатор значимости будет переведен в состояние «выключено», если в процессе редактирования в младшем полубайте упакованной десятичной величины встретится код знака плюс. Используя эту особенность, можно выделить все отрицательные величины. Для такого редактирования

образец можно задать следующим образом:

```
EDITPT    DC    X'402020214B202040C3D94B'
```

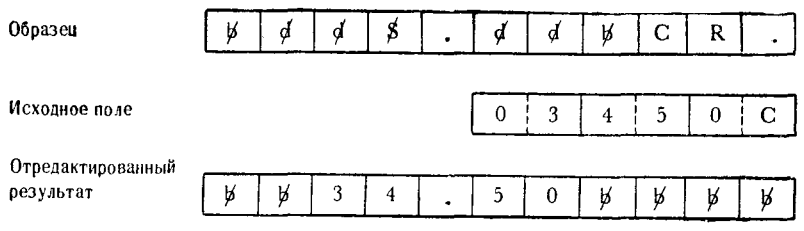
или

```
EDITPT    DC    C'bddd$.ddbCR.'
```

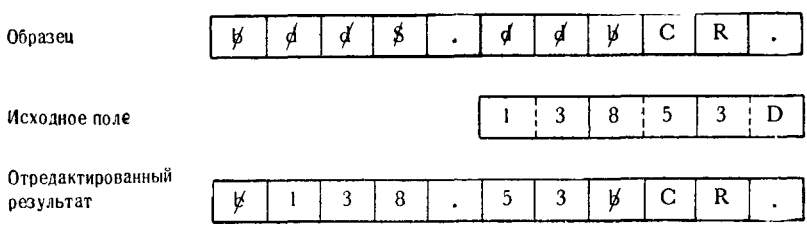
Если при редактировании с использованием этого образца в исходном поле встречается знак плюс, индикатор значимости после обработки последней цифры исходного поля выключается. Если встречается другой знак, индикатор значимости останется во включенном состоянии и непосредственно после отредактированной величины появится сочетание bCR.

Ниже показано, как в соответствии с данным образцом упакованные десятичные величины преобразуются в форму, пригодную для вывода на печать.

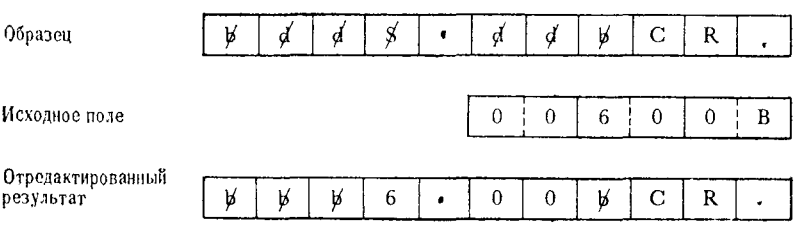
Пример 1.



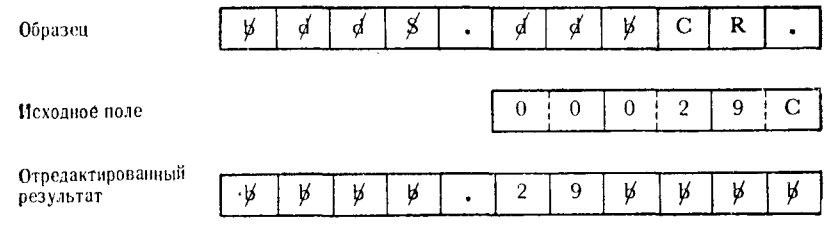
Пример 2.



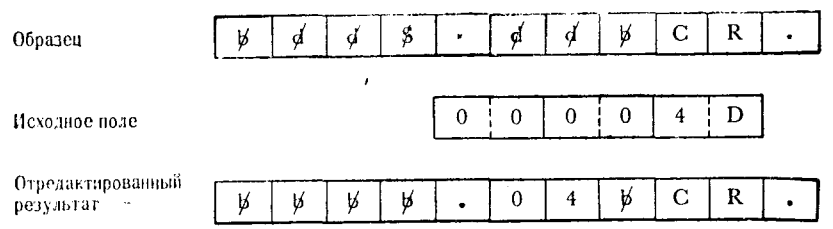
Пример 3.



Пример 4.



Пример 5.



При использовании вычислительной системы для составления торговых документов, например платежных чеков, желательно распечатывать величины, имеющие отношение к денежным операциям, в такой форме, чтобы подавленные незначащие цифры были заменены каким-либо знаком, а не пробелом. Одним из вариантов является применение в качестве символа-заполнителя звездочки (*). Образец с использованием этого знака может выглядеть следующим образом:

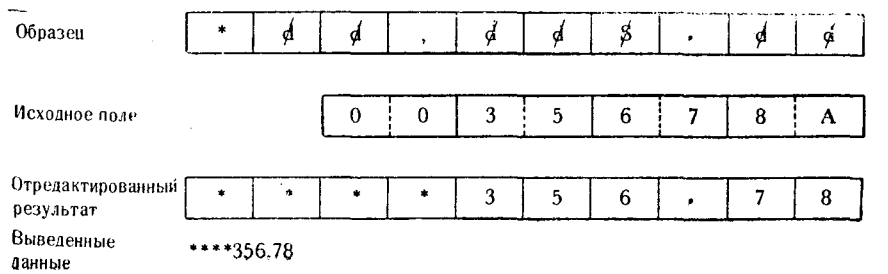
```
EDITPAY     DC      X'5C20206B2020214B2020'
```

или

```
EDITPAY     DC      C'*¢¢,¢¢$.¢¢'
```

Действие этого образца можно показать на следующих примерах:

Пример 1.



Пример 2.

Образец	*	ф	ф	,	ф	ф	$	.	ф	ф
Исходное поле	0	0	0	0	0	2	8	C		
Отредактированный результат	*	*	*	*	*	*	*	.	2	8

Выведенные данные *******.28

Пример 3.

Образец	*	ф	ф	,	ф	ф	$	.	ф	ф
Исходное поле	0	1	0	0	4	5	0	C		
Отредактированный результат	*	*	1	,	0	0	4	.	5	0

Выведенные данные **1,004.50

Пример 4.

Образец	*	ф	ф	.	ф	ф	$	.	ф	ф
Исходное поле	2	3	0	0	0	0	0	F		
Отредактированный результат	*	2	3	.	0	0	0	.	0	0

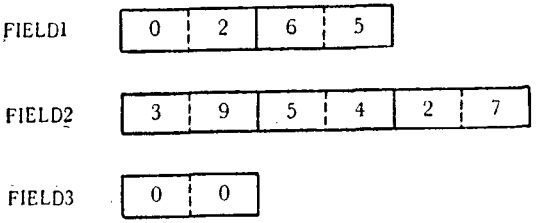
Выведенные данные *23.000.00

Во всех примерах звездочки были внесены вплоть до первой значащей цифры или до того места, где символ начала значимости установил индикатор значимости в состояние «включено». Общей целью редактирования подобного рода является предупреждение изменения денежных сумм на платежных документах путем исключения пробелов в старших позициях распечатки.

Редактирование нескольких полей осуществляется с помощью символа разделения полей. Именно этот символ обеспечивает установку индикатора значимости в состояние «выключено» после каждой величины при редактировании таких полей. Если, к примеру, образец установлен для редактирования трех примыкающих друг к другу числовых полей без знака, символ разделения полей используется для того, чтобы в каждом из трех полей были подавлены незначащие нули.

При создании образца для редактирования нескольких полей следует помнить, что символ-заполнитель необходимо указать один раз как первый (старший) символ образца.

Ниже показано несколько полей, предназначенных для редактирования, и шаблон редактирования, используемый при формировании выходных данных. Предполагается, что эти поля находятся в смежных областях памяти.




```
EDITMF    DC    X'40202021202220202021 4B2020222120'
```

или

```
EDITMF    DC    C'bdd$d F ddd$.dd F $d'
```

Пересылка цифр в образец, отредактированный образец и исходные поля показаны ниже

В распечатке отредактированные данные будут иметь вид

```
265  3954.27     0
```

Ниже поэтапно описывается ход редактирования.

1. При подготовке к редактированию индикатор значимости установлен в состояние «выключено».

2. При редактировании величины, содержащейся в поле FIELD1, индикатор значимости устанавливается в состояние «включено», когда встречается значащая цифра 2. В результате по окончании редактирования третьего байта образца индикатор значимости находится в состоянии «включено».

3. При обработке шестого байта образца встречается символ разделения полей и индикатор значимости устанавливается в состояние «выключено». Если следующая редактируемая величина содержит в своем начале незначащие цифры, то снова происходит подавление нулей. Вместо символа разделения полей в соответствующую позицию образца помещается код пробела.

4. Первая цифра поля FIELD2 оказывается значащей; поэтому она переходит непосредственно в седьмой байт образца, и в то же время индикатор значимости устанавливается в состояние «включено».

5. Так как индикатор значимости включен, оставшиеся цифры заносятся в образец, заменяя символы выбора цифры в байтах 7, 8, 9, 10, 12 и 13.

6. Следующий символ разделения полей встречается в байте 14, и индикатор значимости еще раз устанавливается в состояние «выключено». На место символа разделения полей заносится пробел.

7. Байт 15 содержит символ начала значимости, устанавливающий индикатор значимости в состояние «включено» для последующего редактирования поля FIELD3.

8. Хотя последней цифрой исходного поля является незначащая цифра, состояние «включено» индикатора значимости заставляет перенести цифру 0 исходного поля в 16-й байт образца.

9. Если длина образца не задана явно, операция прекращается после обработки 16-го байта.

После описания компонентов и функций операции редактирования наступило время описать сами эти команды и способы их кодирования в языке Ассемблера.

Отредактировать. Команда ED используется для того, чтобы вызвать выполнение какой-то одной или всех описанных ранее функций редактирования. Она выполняется в соответствии со всеми вышеуказанными правилами редактирования. И хотя различные функции редактирования уже были описаны в достаточной степени подробно, в целом средства редактирования поля данных в области памяти, из которой они должны быть выведены, детально не рассмотрены.

Во многих случаях, когда данные должны передаваться на устройство вывода, область выходной информации в программе

задается программистом. Обычно это кодируется в форме предложений DS или DC.

Для достижения общности и удобства при выполнении вспомогательных операций можно задать области выходной информации для всех устройств. Длина каждой такой области обычно устанавливается достаточно большой, чтобы вместить наибольшую возможную строку данных, которую требуется вывести на внешнее устройство. Например, считается, что стандартное АЦПУ имеет длину строки в 132 печатных символа, экран дисплея может иметь разные размеры 240, 480 или 960 знаков, в строке большинства перфокарточных устройств помещается 80 символов. Несмотря на то что каждое из этих устройств может иметь в памяти область для выходной информации, специально рассчитанную на максимальную длину данных, нет необходимости передавать на устройство вывода эту область целиком. Действительная длина данных, передаваемых на устройство, чаще всего зависит от кодирования макрокоманд метода доступа и предложений DCB, которые неодинаковы для различных видов операционных систем. Поэтому средства указания длины данных, предназначенных к передаче, здесь рассматриваться не будут. Достаточно знать, что данные, посланные из заданной области памяти на устройство вывода, могут быть различны по длине.

Допустим далее, что программист задал область с именем PRTLINE для выходных данных, а затем установил образец для редактирования как константу с именем EDITPRT. Так как программист собирается отредактировать данные в поле PRTLINE, распечатать это поле, отредактировать другую величину, введя ее в поле PRTLINE, где были отредактированы предшествующие данные, распечатать эту новую величину и т. д., то необходимо каждый раз перед выполнением команды ED пересылать образец в поле PRTLINE. При этом должен быть использован следующий образец:

```
EDITPRT    DC    X'4020206B2020214B2020'
```

или

```
EDITPRT    DC    CL10'bddd,ddd.dd'
```

Данные на редактирование берутся из области памяти, которая является упакованным десятичным полем, определяемым как

```
VALUEFLD   DC    PL4'0'
```

Алгоритм этой программы таков, что запись данных вводится с ленты в некоторое поле, затем данные этой записи упаковываются в поле VALUEFLD, содержимое поля VALUEFLD редактируется и помещается в область памяти для выходной

информации и после этого данные из этой области распечатываются на устройстве вывода. В этом примере используются стандартные макрокоманды операционной системы OS для режима пересылки (метод доступа QSAM — Последовательный метод доступа с очередями), обеспечивающие считывание с ленты входных данных и распечатку результатов. Программа имеет следующий вид:

```
BACK    GET     TAPEIN,INAREA                  0001
        PACK    VALUEFLD,INAREA+25(7)          0002
        MVC     PRTLINE+10(10),EDITPRT         0003
        ED      PRTLINE+10(10),VALUEFLD        0004
        PUT     PRTR,PRTLINE                   0005
        BC      15,BACK                        0006
```

Предложение 0001 — макрокоманда GET метода доступа QSAM операционной системы OS. Она считывает файл с именем TAPEIN и вводит логическую запись в область INAREA. Следует предупредить, что все имена (метки) операндов в этих предложениях создаются самим программистом и ни в коем случае не являются стандартными. В некотором месте программы файл TAPEIN должен быть задан макрокомандой DCB, а поле INAREA должно задаваться оператором DS или DC.

Предложение 0002 упаковывает содержимое предполагаемого семибайтового числового поля в коде EBCDIC в четырехбайтовую область VALUEFLD. Поле данных, которые должны быть упакованы, начинается с 26-го байта записи, находящейся в области INAREA. Поэтому вместо того, чтобы подразделять область INAREA на поля с новыми именами, этому полю дается имя INAREA+25.

Предложение 0003 содержит команду MVC, используемую для пересылки константы образца в область вывода на АЦПУ. Эта область PRTLINE имеет длину 132 байта и содержит дополнительные данные, помимо предназначенных для редактирования. Так как отредактированная величина должна помещаться в эту область начиная с 11-го байта поля PRTLINE, образец пересылается по адресу PRTLINE+10. Команда MVC осуществляет пересылку на длину, явно или неявно указанную первым операндом; поэтому, чтобы переслать только 10 байтов поля EDITPRT, задан указатель длины со значением 10.

Предложение 0004 содержит команду ED. Еще раз для определения длины отредактированных данных в первом операнде используется явный указатель длины. Упакованное десятичное исходное поле VALUEFLD редактируется по образцу, который теперь находится по адресу PRTLINE+10.

Предложение 0005 представляет собой обычную макрокоманду PUT метода доступа QSAM операционной системы OS. Для области выходной информации PRTLINE эта макрокоманда порождает задачу ввода-вывода, которая должна соответствующие данные вывести на устройство, задаваемое макрокомандой DCB с меткой PRTR.

Предложение 0006 — это безусловный переход к предложению с меткой BACK.

Затем весь набор предложений выполняется вторично: запись считывается с ленты, величина упаковывается, образец пересылается в область вывода, упакованная величина редактируется и помещается в область вывода, распечатывается строка, осуществляется переход к метке BACK, и т. д. Этот цикл длится до тех пор, пока на входной ленте не встретится условие конца файла (EOF), сигнализирующее о том, что ввод данных закончился.

В данном примере не предусмотрен счет строк, нумерация страниц и т. п., все это будет описано в следующей главе.

После выполнения команды ED в признаке результатов содержится информация о значении величины в только что отредактированном исходном поле. Разряды устанавливаются следующим образом:

CC	BC	Условия
00	8	Было отредактировано поле нулей
01	4	Была отредактирована отрицательная величина
10	2	Была отредактирована положительная величина

Возможно, что программист пожелает принять то или иное решение на основании анализа признака результата. Следует помнить, что окончательное состояние признака результата характеризует последнюю отредактированную величину; при редактировании нескольких полей это будет самая первая величина, при редактировании единственной величины — сама эта величина.

Как отмечалось выше, первый операнд оператора редактирования определяет длину редактируемого поля. Указатель длины для исходного поля, определенного вторым операндом, задается неявно. Следовательно, предполагается, что частью первого операнда должен быть указатель, задающий длину образца для редактирования.

Отредактировать и отметить. Команда EDMK выполняет все функции обычной команды ED. Кроме того, она указывает

адрес в образце, по которому из исходного поля была загружена первая значащая цифра, при условии что эта цифра была передана в образец до обнаружения в нем символа начала значимости. Отмеченный адрес помещается в общий регистр 1.

Эта команда используется для загрузки символов с переменным местоположением, например знака доллара, в область вывода на печать или на дисплей. Используя эту команду как стандартную подпрограмму, программист может всегда напечатать знак доллара или какой-либо другой символ в заданном диапазоне байтов редактируемых данных. Распечатка может выглядеть следующим образом:

```
  $2.56
$3,918.10
   $.06
 $715.75
   $.28
```

Так как адрес первой значащей цифры, которая должна поступить из исходного поля, не всегда загружается в общий регистр 1, как бывает в случае, когда символ начала значимости встречается первым, адрес десятичной точки можно загрузить в регистр 1 до выполнения команды EDMK. Этим гарантируется, что адрес в регистре 1 никогда не будет относиться к точке образца, расположенной правее десятичной точки. Весь процесс выполняется по следующему алгоритму:

1. Применить команду MVC для пересылки образца в область вывода на печать.
2. Загрузить общий регистр 1 адресом самой правой позиции в образце, за пределами которой нельзя помещать знак доллара. Увеличить содержимое регистра 1 на единицу.
3. Закодировать команду EDMK, указывая адрес образца, который находится в области вывода на печать, в качестве первого операнда и адрес исходного поля в качестве второго операнда.
4. Закодировать предложение, которое вычитает величину +1 из адреса, содержащегося в общем регистре 1.
5. Использовать команду MVI для пересылки знака доллара в адрес, указанный в общем регистре 1.

Допустим, что соответствующий образец переслан в область выходной информации по адресу PRTOUT+25. Если первый байт поля PRTOUT имеет адрес 23516, то первый байт образца будет иметь адрес 23541.

PRTOUT + 25

ø	ø	,	ø	ø	ø	.	ø	ø
Адрес памяти								
23541	23542	23543	23544	23545	23546	23547	23548	23549

Данные, засланные в изображенный образец, определяют, был или нет заново установлен общий регистр 1. В первую очередь при описании нашего конкретного образца нужно отметить, что программист намерен поместить знак доллара в некоторой позиции слева от десятичной точки, которая непосредственно следует за символом начала значимости. Так как впоследствии при выполнении программы из адреса в общем регистре 1 вычитается величина +1, в этот регистр загружается адрес десятичной точки.

```
LA      1,PRTOUT+31
```

Теперь регистр 1 содержит величину с фиксированной точкой +23547, представляющую адрес десятичной точки PRTOUT+31.

В соответствии с образцом, символический адрес которого PRTOUT+25, должно быть отредактировано трехбайтовое упакованное поле без знака с меткой PACKAMT. Команда EDMK для обработки этого поля выглядит следующим образом:

```
EDMK    PRTOUT+25(9),PACKAMT
```

Это предложение выполняет функции редактирования и в то же самое время может изменять адрес, содержащийся в общем регистре 1. Изменение адреса в регистре 1 (+23547) обусловливается значимостью цифр, перенесенных из исходного поля. Если из исходного поля в область, занятую образцом, между байтами 23541 и 23547 (не включая их) в результате редактирования были занесены значащие цифры, то команда EDMK загрузит в регистр 1 адрес байта образца, в который была занесена первая из этих цифр. Если в образец до байта с адресом 23547 значащая цифра занесена не была, то адрес, содержащийся в общем регистре 1, не изменяется.

После выполнения команды EDMK в регистре 1 содержится адрес, указывающий на значащую цифру или значащий символ — десятичную точку. Независимо от величины адреса в регистре 1 из него нужно вычесть величину +1 для того, чтобы узнать адрес первого имеющегося пробела, непосредственно предшествующего значащей цифре или символу. Это можно

осуществить с помощью команд SH или BCTR, как показано ниже:

```
SH      1,=H'1'
```

или

```
BCTR    1,0
```

Команда SH вычитает из общего регистра 1 величину +1, представленную в виде литерала длиной в полуслово. Команда BCTR также уменьшает величину в общем регистре 1 на +1, но перехода не происходит, так как второй операнд команды равен нулю.

Теперь когда общий регистр 1 указывает адрес памяти, по которому в редактируемое поле должен быть помещен знак доллара, записывается команда

```
MVI     0(1),C'$'
```

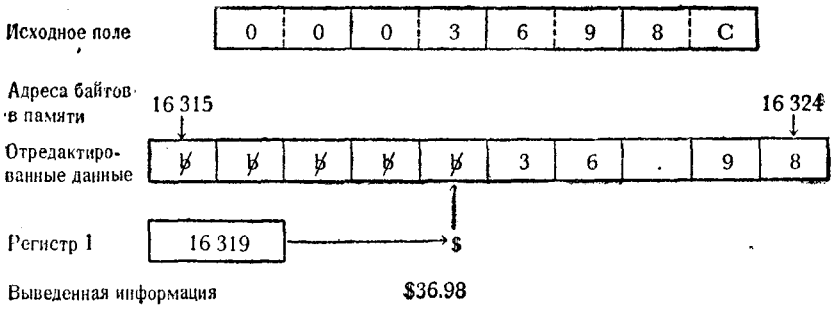
Смысл этой команды состоит в пересылке однобайтового непосредственного символа, представляющего собой знак доллара, по адресу, указанному в общем регистре 1.

Весь получившийся набор команд, используемых для выполнения этой задачи, выглядит следующим образом:

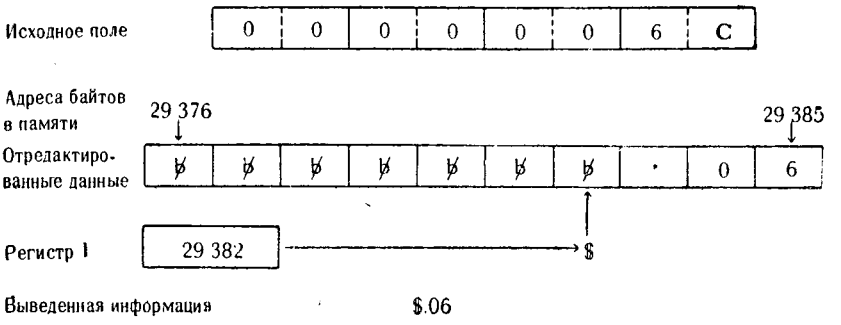
```
MVC     PRTOUT+25(9),EDITPTRN
LA      1,PRTOUT+31
EDMK    PRTOUT+25(9),PACKAMT
SH      1,=H'1'
MVI     0(1),C'$'
```

На следующих примерах демонстрируется процесс внесения знака доллара в поля редактируемых данных при обработке различных величин.

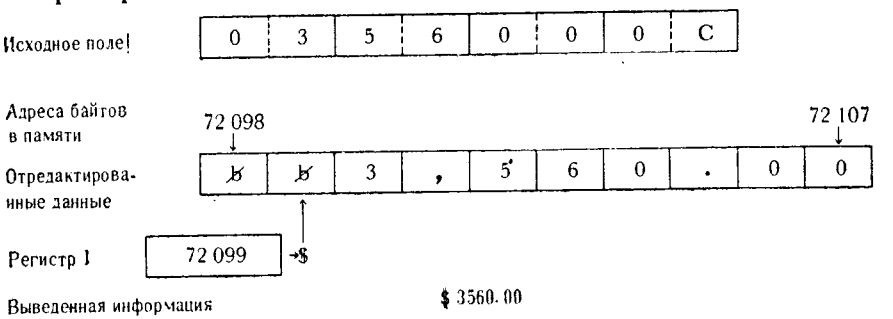
Пример 1.



Пример 2



Пример 3.



Вместо знака доллара можно поставить любой правильный символ или группу символов. Для обозначения кредита или отрицательных величин впереди суммы, а не после нее можно поставить знак минус. Это делается путем проверки признака результата, сформированного командой EDMK. Если редактируемая величина отрицательная, программа заносит код знака минус по адресу, содержащемуся в общем регистре 1; если величина положительная, команда MVI заносит по этому адресу знак плюс или вообще не заносит никакого знака. Результат, выведенный на печать, может выглядеть следующим образом:

−395.06	2,519.78	−99
+.04	−71.63	+15,921.88

По тому же принципу можно вносить такие символы, как 'DR.' и 'CR.', в старшие разряды поля редактируемых данных. Так как все поле данных может быть заполнено значащей информацией, в области памяти, непосредственно предшествующей полю редактируемых данных, должны предусматриваться несколько байтов, содержащих пробелы или устанавливаемых как пробелы после каждого редактирования. Набор команд программы такого редактирования отличается от программы,

рассмотренной ранее, только величиной, вычитаемой из общего регистра 1, пересылкой пробелов в байты, предшествующие полю редактируемых данных, и внесением символов с помощью команды MVC, а не MVI. Программа записывается так, как показано на рис. 15.1.

Предложение 0701 засылает образец в область вывода данных.

Предложение 0702 засылает пробелы в 3 байта, непосредственно предшествующие образцу.

Предложение 0703 загружает в общий регистр 1 адрес десятичной точки в образце.

Предложение 0704 содержит команду EDMK. После выполнения этой команды устанавливается признак результата.

Предложение 0705 вызывает переход к предложению 0712, если редактируемая величина была нулем; соответственно в редактируемое поле данных не будет засылаться ни сочетание символов 'DR', ни 'CR'.

Предложение 0706 вызывает переход к предложению 0710, если редактируемая величина была отрицательной.

Если величина не была отрицательной и не равна нулю, предложение 0707 вычитает +4 из адреса, содержащегося в общем регистре 1.

Предложение 0708 засылает сочетание символов DR. в 3 старших байта из 4, непосредственно предшествующих первой значащей цифре или символу в редактируемом поле данных.

Предложение 0709 — это безусловный переход к предложению 0712.

Предложение 0710 в предположении, что редактируемая величина отрицательна, вычитает из содержимого общего регистра 1 величину +4. Если редактируемая величина больше чем +0 (положительна) или меньше чем —0 (отрицательна), выполняется или предложение 0707, или предложение 0710, но не оба вместе.

Предложение 0711 засылает символы CR. в 3 старших байта из 4 непосредственно предшествующих первой значащей цифре или символу в поле редактируемых данных.

Предложение 0712, к которому осуществляется переход от предложений 0705, 0709 или которое выполняется после предложения 0711, вызывает безусловный переход к некоторой предполагаемой точке программы.

Данные, отредактированные этой программой, будут выглядеть следующим образом:

```
DR.  277.07        .00     CR.  3,984.41
  CR.  .06    DR.  75.14     DR.  23,109.88
```

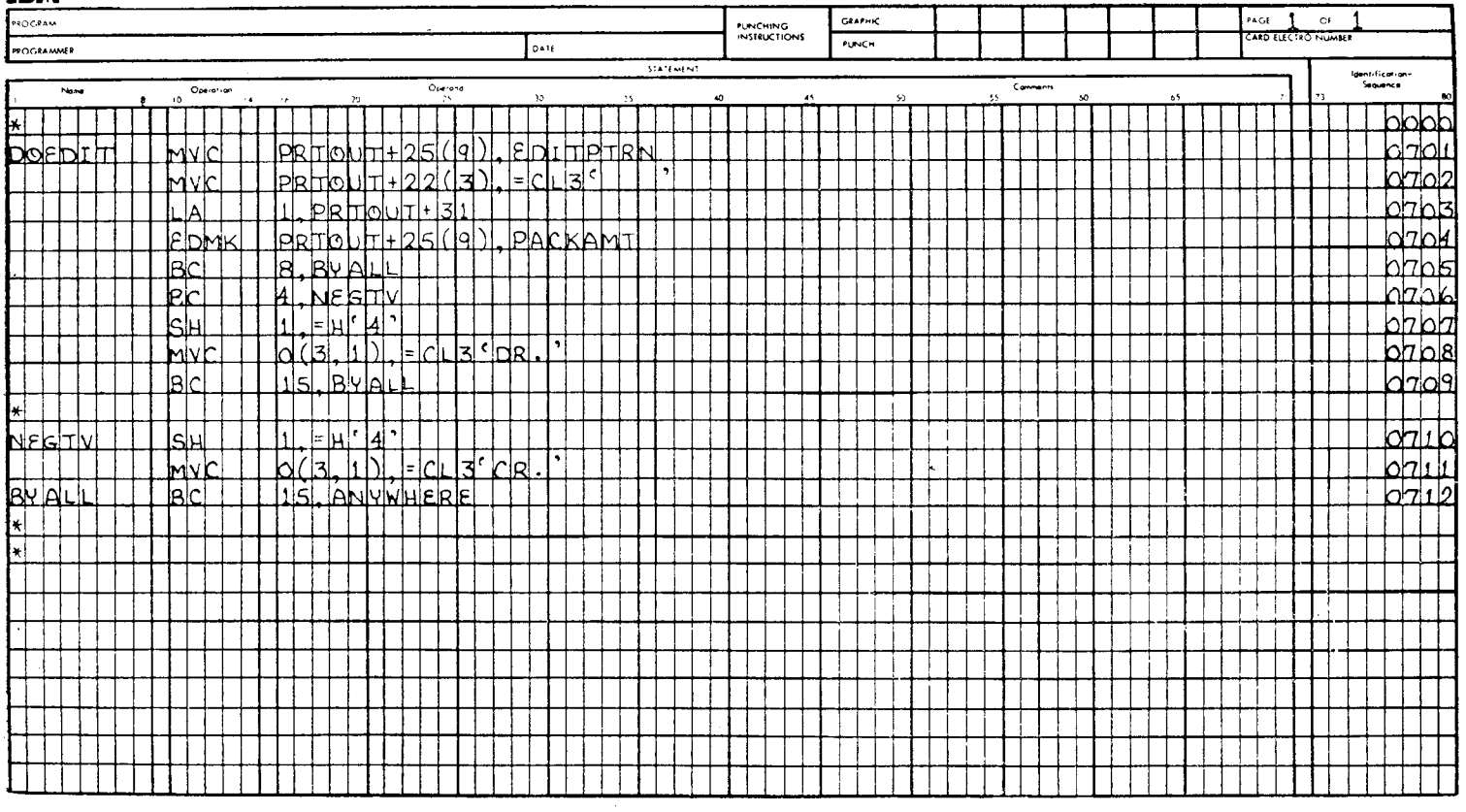
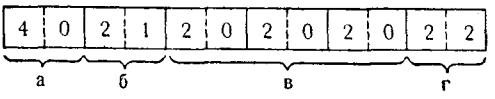

IBM
IBM System/360 Assembler Coding Form
X28-6509-3 U/M050
Printed in U.S.A.
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 1 OF 1
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
* 0000
DOEDIT MVC PRTOUT+25(9),EDITPTRN 0701
MVC PRTOUT+22(3),=CL3' ' 0702
LA 1,PRTOUT+31 0703
EDMK PRTOUT+25(9),PACKAMT 0704
BC 8,BYALL 0705
BC 4,NEGTV 0706
SH 1,=H'4' 0707
MVC 0(3,1),=CL3'DR.' 0708
BC 15,BYALL 0709
*
NEGTV SH 1,=H'4' 0710
MVC 0(3,1),=CL3'CR.' 0711
BYALL BC 15,ANYWHERE 0712
*
*

Рис. 15.1.

Команды ED и EDMK дают программисту значительные возможности в представлении числовых данных в нужном формате.

Упражнения

1. Назовите имена символов образца для редактирования, находящихся в указанных сегментах следующего поля:



а _____

б _____

в _____

г _____

2. Какой общий регистр иногда используется командой EDMK для выполнения ее функций? _____ .

3. _____ рассматривается как переключатель состояния, который проверяется в процессе выполнения редактирования и обусловливает действия команды редактирования.

4. Символ начала значимости в образце имеет следующее представление X′ _____′.

5. Если при редактировании в младшем полубайте исходного поля обнаруживается знак _____ , индикатор значимости переводится в состояние «выключено».

6. Даже если в исходном поле содержатся значащие цифры, команда EDMK не возвратит адрес первой значащей цифры, если раньше этой цифры в шаблоне редактирования встречается символ _____ .

7. К моменту выполнения команды ED предназначенные к редактированию данные должны быть в _____ формате.

8. Если исходное четырехбайтовое поле, содержащее упакованную десятичную величину +0, редактируется в соответствии со следующим образцом, то общее количество нулей, которое останется в отредактированом результате, будет равно _____ .

4	0	2	0	2	1	2	0	2	0	2	0	2	0	2	0

9. Символ выбора цифры в образце имеет следующее представление: X′ _____′.

10. Символ _____ сигнализирует о том, что подавление незначащих нулей должно окончиться в этой точке, если оно не

было прекращено ранее при пересылке значащих цифр из исходного поля.

11. При выполнении команд ED и EDMK просмотр поля производится в направлении _____.

12. Символ разделения полей имеет следующее представление: X'_____'.

13. Если при выполнении команды EDMK производится загрузка регистра 1 адресом, этот адрес указывает на байт в используемом этой командой _____.

14. Встретив в образце символ _____, команда ED установит индикатор значимости в состояние «выключено».

15. Первый (старший) символ образца рассматривается как _____.

16. Символ _____ используется для указания позиции в образце, которая должна заполниться цифрой из исходного поля или символом-заполнителем образца.

17. По окончании редактирования признак результата устанавливается в зависимости от состояния _____.

18. Символ _____ обычно используется при редактировании исходного поля, содержащего несколько величин.

19. Если требуется подавление нулей, то для замещения первых незначащих цифр исходного поля используется _____.

В каждой из следующих задач заданы образец, исходное поле, выходное поле, в которое заносятся отредактированные данные, и одна или более команд. Проанализируйте поля и команды и заполните выходные поля в символьном и шестнадцатеричном представлениях так, как они должны выглядеть после выполнения этих команд.

20.

EDPATA	4	0	2	0	2	0	2	1	2	0	2	0
SOURCEA	0	0	0	0	9	C						
RESULTA	C	1	C	2	4	0	4	0	4	0	4	0	4	0

```
MVC     RESULTA+1(6),EDPATA
ED      RESULTA+1(6),SOURCEA
```

RESULTA (Шестн.)												
(Символьн.)												

21.

EDPATB

4	0	2	0	2	1	2	0

SOURCEB

3	8	7	C

RESULTB

D	1	F	9	F	9	F	6	F	6	F	8

```
MVC     RESULTB+1(4),EDPATB
ED      RESULTB+1(4),SOURCEB
```

RESULTB (Шестн.)

(Двоичн.)

22.

EDPATC

4	0	2	1	2	0	2	0	4	B	2	0	2	0

SOURCEC

0	0	0	2	5	8

RESULTC

4	0	4	0	4	0	4	0	4	0	4	0	4	0

```
MVC     RESULTC(7),EDPATC
ED      RESULTC(7),SOURCEC
MVI     RESULTC,X'5B'
```

RESULTC (Шестн.)

(Двоичн.)

23.

EDPATD

4	0	2	0	2	0	2	0	4	0	C	3	D	9

SOURCED

0	8	0	3	6	C

RESULTD

F	0	F	0	F	0	F	0	4	0	C	3	D	9

```
MVC     RESULTD,EDPATD
ED      RESULTD,SOURCED+1
```

RESULTD (Шестн.)

(Двоичн.)

24.

EDPATE	4	0	2	1	4	B	2	0	2	0	C	3	D	9
SOURCEE	7	4	8	D										
RESULTE	4	0	4	0	4	0	4	0	4	0	C	4	D	9

```
MVC     RESULTE,EDPATE
ED      RESULTE,SOURCEE
```

RESULTE (Шестн.)

(Двоичн.)

25.

EDPATF	5	C	2	0	2	1	2	0	2	0	2	0	6	0
SOURCEF	0	0	6	2	9	D								
RESULTF	4	0	4	0	4	0	4	0	4	0	4	0	4	0

```
MVC     RESULTF,EDPATF
ED      RESULTF,SOURCEF
```

RESULTF (Шестн.)

(Двоичн.)

26.

EDPATG	4	0	2	0	2	0	2	1	4	B	2	0	2	0
SOURCEG	0	6	1	1	7	D								
RESULTG	C	1	C	2	C	1	C	2	C	1	C	2	C	1

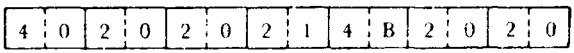
```
LA      1,RESULTG+4
MVC     RESULTG(7),EDPATG
EDMK    RESULTG(7),SOURCEG
SH      1,=H'1'
MVI     0(1),C'$'
```

RESULTG (Шестн.)

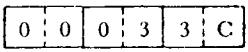
(Двоичн.)

27.

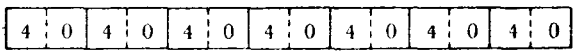
EDPATH



SOURCEH



RESULTH




```
LA      1,RESULTH+4
MVC     RESULTH(7),EDPATH
EDMK    RESULTH(7),SOURCEH
SH      1,=H'1'
MVI     0(1),X'5B'
```

RESULTH (Шестн.)

(Двоичн.)

Глава 16

Вывод данных на печать

Для практических применений нужно знать основные методы вывода данных на печать. Усвоение этих приемов позволит программисту писать соответствующие программы и получать в печатном виде данные, сформированные в процессе выполнения программы.

Некоторые программы, встречающиеся в данном разделе, включают в себя макрокоманды ввода-вывода операционной системы OS. Следует иметь в виду, что эти макрокоманды выполняют свои функции только в рамках этой операционной системы. Если используется другая операционная система, программист должен применить соответствующие операторы управления вводом-выводом.

А. ОПРЕДЕЛЕНИЕ ОБЛАСТИ ВЫВОДА

Если предполагается вывести данные на АЦПУ или другое внешнее устройство, то эти данные обычно формируются программными средствами в некоторой области основной памяти. Эта область обычно называется *областью вывода.* Хотя разные устройства способны воспринимать записи данных с различной максимальной длиной, в дальнейшем предполагается, что используется стандартное АЦПУ, способное печатать до 132 символов в строке. При определении области вывода будем считать, что она содержит 132 последовательных байта основной памяти.

Программист может задать область вывода различными способами. В частности, он может задать ее как одно поле размером в 132 байта или как группу меньших полей. Например:

```
PRTRA    DC    CL132'Б'
```

или

```
PRTRB    DS    0CL132
FIELD1   DC    CL15'Б'
FIELD2   DC    CL25'Б'
FIELD3   DC    CL10'Б'
FIELD4   DC    CL63'Б'
FIELD5   DC    CL19'Б'
```

Данные, которые следует переслать в область вывода PRTRA, можно направить в различные части этой области, изменяя соответствующим образом адрес в первом операнде команды пересылки. Это может быть записано следующим образом:

```
MVC     PRTRA+10(16),DATA1      0001
MVC     PRTRA+35(8),DATA2       0002
MVC     PRTRA+87(12),DATA3      0003
```

Предложение 0001 пересылает 16 байтов данных из поля DATA1 по адресу PRTRA+10, т. е. в часть поля PRTRA с 11-го по 26-й байт.

Предложение 0002 пересылает 8 байтов из DATA2 в поле PRTRA, начиная с 36-го байта.

Предложение 0003 пересылает 12 байтов, начиная с адреса DATA3, в часть поля PRTRA с 88-го по 99-й байт.

Данные, которые должны быть посланы в область вывода PRTRB, могут быть посланы или по подчиненным меткам, или по метке PRTRB со смещением. В каждой из следующих пар предложений обе команды указывают один и тот же адрес:

```
MVC     FIELD1,AREA1
MVC     PRTRB(15),AREA1

MVC     FIELD3,AREA2
MVC     PRTRB+40(10),AREA2

MVC     FIELD5,AREA3
MVC     PRTRB+113(19),AREA3
```

Часто программист обнаруживает, что ему выгоднее определять область вывода одной константой. При этом он может легко помещать данные в различные части каждой печатаемой строки, не путаясь с обозначениями подполей. С другой стороны, распечатку, выполненную в одном или нескольких определенных форматах, легче понять, и с ней легче работать, если область вывода разбита на определенные подполя. Область вывода может быть многократно задана так, как это показано на рис. 16.1.

При наличии заданной таким образом области вывода PRTROUT данные могут быть представлены в любом из двух форматов печатной строки. На печати они будут иметь тот формат, который был использован при пересылке данных в область вывода.

Хотя из предыдущего текста уже ясно, что представляет собой область вывода в основной памяти, следует заметить, что

IBM

IBM System/360 Assembler Coding Form

X28-6508-3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 1 OF 1

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				
PRTROUT	DS	0CL132		
AFLD1	DS	CL25		
AFLD2	DS	CL10		
AFLD3	DS	CL40		
AFLD4	DS	CL5		
AFLD5	DS	CL42		
*				
	ORG	PRTROUT		
BFLD1	DC	CL12' '		
BFLD2	DC	CL38' '		
BFLD3	DC	CL9' '		
BFLD4	DC	CL38' '		
BFLD5	DC	CL15' '		
BFLD6	DC	CL20' '		
*				
*				

Рис. 16.1.

для того, чтобы определить распечатываемые данные как набор данных в проблемной программе, необходимо кодировать макрокоманду, задающую параметры блока управления данными (DCB) (или ее эквивалент). В различных операционных системах — в Базовой операционной системе (BOS), Ленточной операционной системе (TOS), Дисковой операционной системе (DOS) и в полной Операционной системе (OS) — способы описания наборов данных различны. Для OS набор данных описывается посредством макрокоманды «Блок управления данными» (DCB). В этой макрокоманде указывается ряд числовых параметров, которые в совокупности определяют основные особенности используемого набора выходных данных. В ней указывается длина записи, формат записи, какие макрокоманды для управления вводом-выводом и дополнительные возможности должны использоваться и многие другие параметры, одно название которых без подробного анализа ничего не даст читателю. К сожалению, в рамках этого изложения невозможно описать различные макрокоманды, используемые в каждой из операционных систем. Предполагается, что программисты и студенты, изучающие эту книгу, ограничат свои интересы определенной операционной системой. Можно предположить, что школы, университеты, предприятия, где учится или работает читатель, используют на своих вычислительных системах какую-то одну или несколько из четырех операционных систем. Поэтому необходимо, чтобы любые сведения о функциональных макрокомандах или макрокомандах вывода-ввода операционных систем слушатели получили непосредственно на вычислительных центрах, с которыми связан читатель.

Для вывода на АЦПУ в OS используется макрокоманда PUT. Эта команда вызывает использование последовательного метода доступа с очередями. Ее можно записать следующим образом:

```
PUT     PRTR,PRTAREA
```

В данном случае макрокоманда PUT вызывает запись данных, содержащихся в области памяти PRTAREA, в набор данных PRTR. Это краткое описание процесса; предполагается, что полная информация о макрокоманде операционной системы, которая доступна читателю, будет получена на соответствующем вычислительном центре.

Б. ПРОПУСК СТРОК И УПРАВЛЕНИЕ СТРОКАМИ

Программист может регулировать расстояние между строками или пропуски строк при выдаче информации на печать. Расстояние между строками может составлять один, два или три интервала, его может вообще не быть или же строка может

начинаться с новой страницы. Расстояние между строками при выдаче на печать может регулироваться особыми символами управления кареткой. Для того чтобы система OS восприняла символ, необходимо при кодировании параметра RECFM макрокоманды DCB, определяющего формат записи, использовать букву A. Обычно параметр RECFM для АЦПУ записывается следующим образом:

RECFM=FSA

Буква F означает, что запись имеет фиксированную длину; буква S — что не должно быть укороченных блоков или записей; буква A — что в распечатываемой строке выходной области будет использован символ управления кареткой. Еще раз следует сказать о том, что средства, сообщающие, какой вид регулирования между строками применяется, неодинаковы для различных операционных систем.

Если для регулирования межстрочного расстояния используется символ управления кареткой, то он должен помещаться непосредственно перед первым байтом предназначенных к распечатке данных. Ниже представлены все правильные символы управления кареткой и описаны вызываемые ими действия.

Символ управления кареткой			Воздействие на каретку
␣	(пробел)	X'40'	Перед распечаткой данных пропускается одна строка (один интервал)
0	(нуль)	X'F0'	Перед распечаткой данных пропускаются две строки (два интервала)
-	(дефис)	X'60'	Перед распечаткой данных пропускаются три строки (три интервала)
+	(плюс)	X'4E'	Перед выводом на печать пропуск строк не делается (подавление интервала)
1	(единица)	X'F1'	Перед выводом на печать производится переход на новую страницу (пропуск оставшейся части страницы)

Хотя и имеется несколько других символов управления кареткой, для начинающего программиста указанных пяти более чем достаточно.

Ранее упоминалось, что стандартная длина данных в печатной строке не должна превышать 132 байта. В соответствии с этим выходное поле данных задавалось константой, имеющей длину 132 байта. Но при использовании символа управления кареткой первый байт поля вывода отводится под этот символ. Для того чтобы вместить 132 байта данных, выводимых на печать, область вывода в основной памяти должна иметь длину

133 байта: 1 байт для символа управления кареткой плюс 132 байта для данных, выводимых на печать. Здесь опять может найти применение идея о задании подполей выходных данных. Например:

```
OUTAREA   DS    0CL133
CCSYMB    DC    CL1'b'
DATA      DC    CL132'b'
```

Макрокоманду ввода-вывода для распечатки этого поля данных можно закодировать следующим образом:

```
PUT    PRTR, OUTAREA
```

Несмотря на то что OUTAREA задана как область, содержащая выходные данные, состоящие из 133 байтов, системе посредством параметра RECFM в DCB было указано, что при распечатке данных должен использоваться символ управления кареткой. Первый байт OUTAREA (то же, что и CCSYMB) поэтому будет интерпретироваться как символ управления кареткой, а остальные 132 байта (DATA) считаются байтами, которые содержат данные, предназначенные для распечатки.

Если при распечатке потребуется изменение расстояния между строками, программисту нужно будет только записать команду MVI, которая перешлет соответствующий код в байт символа управления кареткой в начале поля распечатываемой строки. Следующий пример показывает, как изменение символа управления кареткой влияет на расстояние между строками.

Область вывода задается следующим образом:

```
PRTOUT    DS    0CL133
SYMBOL    DC    CL1'b'
MESSG     DC    CL132'b'
```

Для описания выводимых на печать данных следует иметь в виду, что сообщение THIS IS THE WAY DATA LINES ARE SPACED (Это способ пропуска строк) занимает первые 37 байтов подполя MESSG. Вывод будет осуществляться посредством стандартной макрокоманды PUT операционной системы OS, в которой указана макрокоманда DCB, служащая для вывода информации и имеющая метку DEVICE. Программа имеет следующий вид:

```
PUT    DEVICE,PRTOUT     0001
PUT    DEVICE,PRTOUT     0002
MVI    SYMBOL,C'0'       0003
PUT    DEVICE,PRTOUT     0004
MVI    SYMBOL,C'—'       0005
PUT    DEVICE,PRTOUT     0006
```

Распечатанные данные будут выглядеть следующим образом:

```
Предложение 00001  THIS  IS  THE  WAY  DATA  LINES  ARE  SPACED
Предложение 00002  THIS  IS  THE  WAY  DATA  LINES  ARE  SPACED

Предложение 00004  THIS  IS  THE  WAY  DATA  LINES  ARE  SPACED

Предложение 00006  THIS  IS  THE  WAY  DATA  LINES  ARE  SPACED
```

В предложениях 00001 и 00002 в качестве символа управления кареткой был закодирован пробел, так что строки распечатались через один интервал без пропуска строки между ними.

Предложение 00003 пересылает нуль в байт символа управления кареткой.

Предложение 00004 вынуждает АЦПУ перед началом распечатки пропустить два интервала. Этот двойной интервал обеспечивает появление пустой строки между текущей распечатываемой строкой и предыдущей.

Предложение 00005 использует команду MVI для занесения дефиса в байт символа управления кареткой.

Выполнение предположения 00006 приводит к тому, что АЦПУ пропускает три интервала перед началом распечатки данных.

Чтобы программа дважды печатала на одной и той же строке, используют символ управления кареткой «подавление интервала». Хотя очевидных случаев применения этой возможности немного, все-таки эта функция может быть полезной. Средствами «подавления интервала» можно выполнять наложение двух символов, подчеркивание и формирование специальных символов. Вот некоторые образцы применения этой функции:

Ø Θ ÷ A̲B̲ ( )

Символ управления кареткой «переход на новую страницу» используется для того, чтобы напечатать следующую строку данных на новой странице. Имеется несколько способов и эквивалентных условий применения этого символа. Хорошим способом оформления печатаемого текста является помещение вверху каждой страницы заголовка или описателя ее содержимого. Если не предпринять специальных мер, строки будут печататься непрерывно без разбивки на страницы. Поэтому желательно, чтобы алгоритм программы предусматривал подсчет строк, приходящихся на одну страницу. Этот подсчет позволит определить, когда следует переходить на новую страницу, печатать необходимые заголовки или заглавные строки, после чего можно будет продолжать печать строк данных.

В. УПРАВЛЕНИЕ СТРАНИЦАМИ

Как уже отмечалось в предыдущем разделе, управление страницами важно для получения логичной и четкой распечатки. Подпрограмму для управления страницами сравнительно легко составить. Нужно лишь знать размер бумаги для печати и расстояние между строками. Ниже даются общие правила регулирования количества строк на странице.

1. Установить «счетчик», в котором накапливалось бы количество напечатанных на странице строк.
2. После распечатки каждой строки добавлять соответствующее число к счетчику строк.
3. Сравнить значение в счетчике строк с константой, равной максимальному числу строк на странице.
4. Если число в счетчике равно или больше константы, перейти на стандартную подпрограмму перехода на новую страницу.
5. В ином случае продолжать печать на этой странице.

Стандартная подпрограмма этого типа составляется из следующих предложений:

```
LISTING     PUT     PRTR1,OUTAREA      00001
            AP      LINECTR,=PL1'2'    00002
            CP      LINECTR,=PL2'56'   00003
            BC      10,NEWPAGE         00004
LISTDONE    BC      15,GO              00005
```

Предложение 00001 — это макрокоманда PUT, которая вызывает вывод на печать строки данных.

Предложение 00002 складывает упакованную десятичную величину +2, выраженную как литерал, с упакованной десятичной величиной в счетчике строк. Приращение +2 используется потому, что этот конкретный текст печатается через два интервала, так что перед каждой распечатываемой строкой данных делается одна пустая строка.

В поле LINECTR содержится счетчик строк, напечатанных на текущей странице. Предложение 00003 сравнивает содержимое LINECTR с упакованной десятичной величиной +56, заданной в форме литерала. При этом подразумевается, что при выполнении данной программы число строк на странице должно равняться 56. В результате сравнения признак результата в PSW будет указывать на соотношение между сравниваемыми величинами: «равно», «меньше» или «больше».

Предложение 00004 — это команда условного перехода. Если величина в поле LINECTR равна +56 или больше чем +56, то

происходит переход к стандартной подпрограмме NEWPAGE. Хотя это здесь и не показано, подпрограмма NEWPAGE выполняет следующие функции:

а) переход к началу новой страницы;

б) установка LINECTR в нуль;

в) вывод на печать всех нужных заголовков или заглавных строк;

г) переход к предложению LISTDONE.

Предложение 00005 представляет собой безусловный переход к другой секции программы.

Управление страницами не обязательно должно основываться только лишь на подсчете строк, распечатанных или пропущенных на одной странице. Нередко основной причиной перехода на новую страницу является логическое деление на страницы. Допустим, к примеру, что программа выводит на печать список заказов на покупки от некоторого количества потребителей. Продавец может пожелать, чтобы информация по каждому отдельному потребителю начиналась с новой страницы с тем, чтобы можно было разделить всю распечатку на страницы по идентификаторам потребителей и послать копию заказа потребителю для подтверждения. Среднее количество производимых каждым потребителем заказов равно десяти; поэтому, как правило, информация, относящаяся к одному потребителю, займет менее половины страницы. Программа, учитывающая желание продавца, должна каждый раз, когда встречается новый идентификатор «Потребитель № ...», переходить к новой странице и выводить на печать все нужные заголовки и заглавные строки. Данные, связанные с этим потребителем, будут выводиться на печать до тех пор, пока не встретится следующий идентификатор, после чего программа вновь передаст управление стандартной подпрограмме перехода на новую страницу и т. д.

Г. ОФОРМЛЕНИЕ НАЗВАНИЙ, ЗАГОЛОВКОВ И ПОДЗАГОЛОВКОВ

При выводе данных на печать всегда желательно начинать каждый титульный лист строкой названия, которое характеризует содержание печатаемого текста. На той же строке печатается номер страницы. Можно также вносить заголовки, идентифицирующие содержание данной страницы в той мере, в какой оно отличается от содержания других страниц. Если строки данных образуют столбцы, то применяются подзаголовки, которые разъясняют содержание столбца и помещаются сверху над ним.

Форма и содержание названий и заголовков могут изменяться в широких пределах. Вопросы разбивки и оформления

текста обычно решает специалист, проектирующий систему, для которой составляется данная программа. При отсутствии исходных требований ответственность за организацию распечатки выходных данных берет на себя программист. В подобных случаях он должен стремиться к тому, чтобы напечатанный материал имел логичную, хорошо продуманную структуру и для его понимания не требовалось перекрестных ссылок на другие источники информации. Следующие разделы, в которых описываются строки названий и подзаголовки, носят общий характер, но тем не менее достаточно конкретны, чтобы служить для программиста хорошим руководством. Нет никакой магической формулы, которой нужно следовать; здравый смысл и логика должны руководить вами при практическом оформлении данных, выводимых на печать.

1. Название

В любом выводимом на печать тексте название должно быть максимально содержательным и понятным. Оно должно именовать данный текст, а также систему или проект, к которым текст относится. Сюда может также входить и идентификатор программы, генерирующей текст, если такой индентификатор имеется. Имя или та часть заглавной строки, которая содержит название, всегда должны быть размещены так, чтобы их легко можно было выделить с первого взгляда. Если строка имеет 132 байта, ее середина приходится на 66-й байт. Следует подсчитать количество символов в названии, разделить это число на 2, а затем, отступив на величину частного от центра строки к ее началу, печатать само название.

Например:

Длина названия равна	38 байтам
$^1/_2$ длины названия равна	19 байтам
66 байтов минус 19 байтов равно	47 байтам

Название должно начинаться с 47-го байта выходного поля, отведенного для строки названия.

Строка названия должна содержать также два других небольших поля — одно для нумерации страниц и одно для даты. Их можно использовать так, чтобы сбалансировать название, поместив по одному с каждой стороны, или же оба можно поместить в правой части страницы. Практика использования систем показала, что лучше, если и дата и номер страницы располагаются в правом верхнем углу. Было установлено, что это как раз то место, где обычный читатель ищет эту информацию. Организовать ее можно следующим образом:

```
DATE:      PAGE#
```

Для указания номера каждой страницы программист может использовать специальный счетчик. Каждый раз, когда управление получает подпрограмма перехода к новой странице, она увеличивает содержимое счетчика на +1, заносит эту величину в строку названия непосредственно вслед за полем PAGE# и затем переходит к выводу строки названия на печать.

Информация о дате может поступать из многих источников, что в значительной степени зависит от алгоритма программы. Некоторыми средствами получения даты являются следующие:

1. Ввод управляющей карты во время выполнения данной программы. В программу должна входить подпрограмма, которая считывает эту управляющую карту, извлекает информацию о дате и пересылает ее в область вывода для строки названия.

2. Ввод с пультовой пишущей машинки оператором вычислительной системы. Алгоритм программы должен предусматривать посылку сообщения оператору, которое потребует от него ввода даты через пультовую пишущую машинку. Часть ответа, касающаяся даты, будет введена в область вывода для строки названия.

3. Получение текущей даты от самой операционной системы. В большинстве операционных систем программа может запрашивать супервизор управляющей программы системы и получать в ответ текущую дату, вычисляемую в системе. Все эти действия совершаются внутри системы без всякой связи с существом решаемой задачи. Единственным недостатком этого способа является то, что многие выводимые на печать данные в самом деле характеризуются датой, которая отличается от даты вывода на печать.

Хотя кажется, что загрузка даты в область вывода для строки названия — дело несложное, часто в целях удобства персонала, читающего распечатку, приходится использовать специальную программу преобразования даты. Иногда требуется изменить дату грегорианского календаря на соответствующую дату юлианского; иногда — сделать такое преобразование, как из формы «месяц/день/год» — в «рабочий день» и т. п. Дата может быть задана, например, следующим образом:

Месяц/день/год	Ноябрь 18, 1969
Месяц/день/год	11-18-69 или 11/18/69
Год и день года	69-322
Год и рабочий день	69-209
День/месяц/год	18 ноября 1969

Теперь, когда все компоненты строки названия рассмотрены, было бы полезным ознакомиться с соответствующими средствами введения таких данных в качестве программной

константы. Предполагается, что область вывода для строки названия задается как константа длиной 133 байта, включая символ управления кареткой, причем каждый байт константы устанавливается соответствующим образом. На рис. 16.2 показана кодировка такой константы.

Следует обратить внимание на то, что первый байт поля вывода — это 1 (символ управления кареткой), означающий, что АЦПУ перед выводом строки названия на печать должно перейти на начало новой страницы. Переход на новую страницу происходит, как только в программе выполняется команда ввода-вывода для вывода на печать этой строки названия. Еще одним преимуществом задания области вывода для строки названия посредством отдельной константы вне области вывода данных служит то, что макрокоманда PUT или ее эквивалент может непосредственно ссылаться на область TITLE1, например

```
PUT     PRTR1,TITLE1
```

Поэтому для того, чтобы распечатать строку названия, необязательно пересылать ее содержимое в область вывода данных.

Следующий набор команд представляет собой типичную подпрограмму оформления новой страницы:

```
NEWPAGE   AP     PAGECTR,=PL1'1'              0001
          MVC    TITLE1+126(4),PAGEDIT        0002
          ED     TITLE1+126(4),PAGECTR        0003
          PUT    PRTR1,TITLE1                 0004
          BCR    15,6                         0005
```

Предполагается, что данные уже были засланы в поле TITLE1 при подготовке этой программы и что она получила управление в точке NEWPAGE для того, чтобы вывести на печать строку названия на новой странице.

Предложение 0001 складывает упакованный десятичный литерал +1 со счетчиком страниц, при этом сумма в упакованном десятичном формате используется как номер новой страницы.

Предложение 0002 пересылает образец для редактирования в 4 байта поля TITLE1, непосредственно следующие за символами PAGE#. Сведения о функциях команды редактирования приведены в гл. 15.

Предложение 0003 редактирует упакованную десятичную величину из поля PAGECTR, представляющую номер страницы, помещая ее в образец, который предложением 0002 был переслан в поле TITLE1.

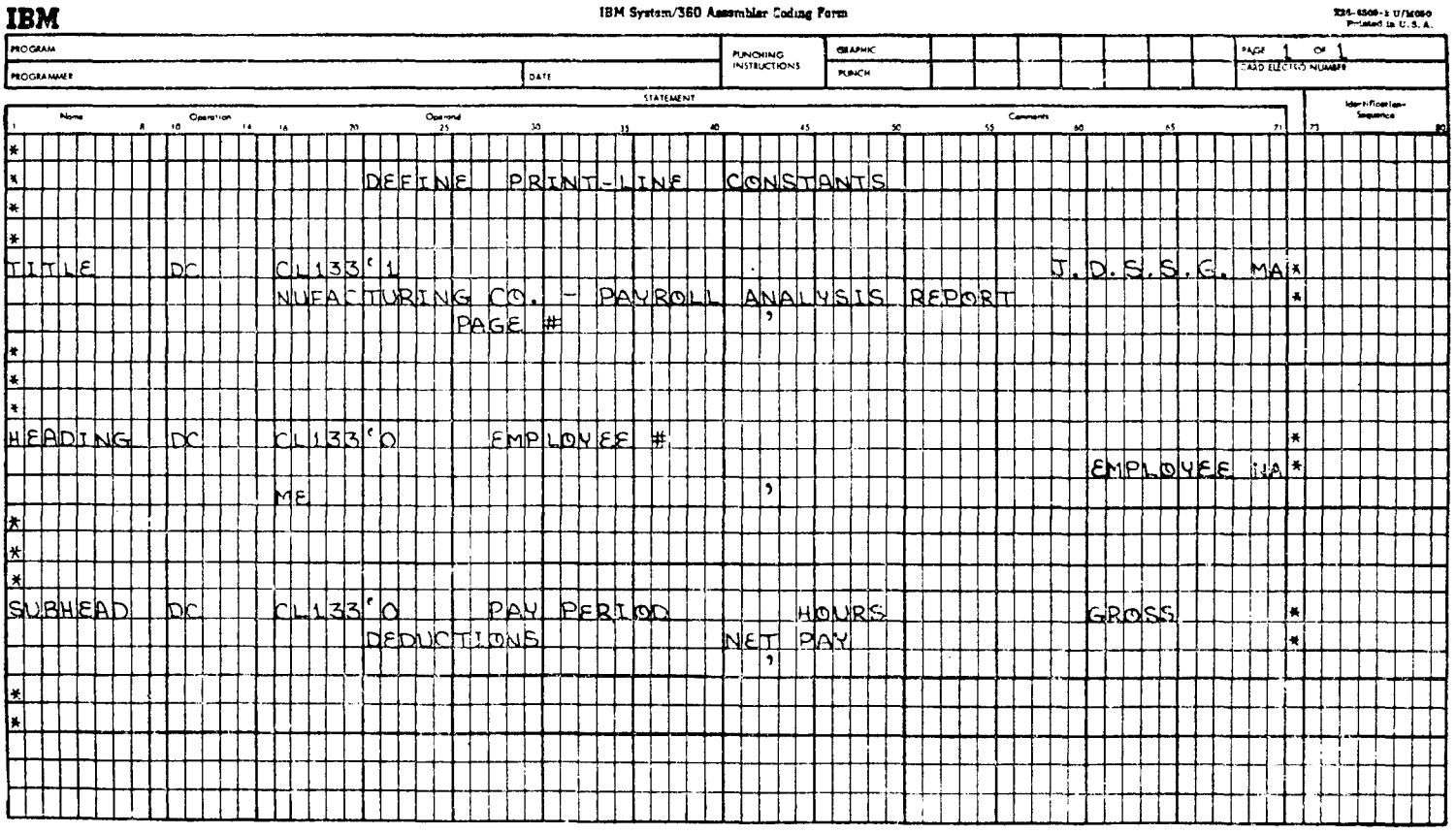

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 1 OF 1
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
DEFINE PRINT-LINE CONSTANTS
TITLE DC CL133'1
J.D.S.S.C. MA
NUFACTURING CO. - PAYROLL ANALYSIS REPORT
PAGE #
HEADING DC CL133'0
EMPLOYEE #
EMPLOYEE NA
ME
SUBHEAD DC CL133'0
PAY PERIOD
HOURS
GROSS
DEDUCTIONS
NET PAY

Рис. 16.2.

Предложение 0004 представляет собой макрокоманду PUT операционной системы OS, вызывающую вывод содержимого области TITLE1 на АЦПУ, указываемое с помощью операнда PRTR1. При выполнении этой макрокоманды АЦПУ, перед тем как вывести на печать строку названия, выполняет прогон бумаги до начала новой страницы.

Предложение 0005 содержит команду BCR, вызывающую безусловный переход по адресу, содержащемуся в общем регистре 6. Это должен быть адрес команды, следующей за командой, вызвавшей переход к подпрограмме NEWPAGE.

2. Заголовки и подзаголовки

Разница между заголовками и подзаголовками весьма условная. То, что в одном тексте можно рассматривать как заголовок, с таким же успехом может считаться подзаголовком в другом. Можно было бы сказать, что подзаголовок является вторичным по отношению к заголовку и более подробно описывает предмет, связанный с первичным заголовком. Так как это утверждение довольно нечеткое, попытаемся разъяснить значение слов заголовок и подзаголовок на примерах (см. стр. 593).

Пример 1.

Строка 1 — строка названия. В ней указывается название фирмы, для которой готовится отчет, и название отчета.

Строку 2 можно определить как заголовок. По нему можно судить, что на каждого служащего отводится раздел или страница отчета. Номер служащего, его имя и фамилия распечатываются в заголовке, чтобы указать, к кому относятся следующие за заголовком данные.

Строка 3 — это подзаголовок, состоящий из заголовков столбцов.

Строки 4, 5 и 6 — строки данных, содержащие необходимые сведения о служащем.

Пример 2. Здесь кроме названия, имеется только одна группа заголовков. Вместо того чтобы для каждого нового служащего начать новую страницу и печатать новую группу заголовков, в данном формате объединены заголовки и подзаголовки примера 1 в единую группу заголовков. Идентификатор и имя служащего встречаются только один раз, когда распечатывается первая строка сведений о служащем. Поэтому строку 2 можно рассматривать как заголовок или подзаголовок в зависимости от того, как эти термины интерпретирует программист. В общем заголовок можно определить как первую определительную строку, которая печатается за названием, а подзаголовок — как любые последующие строки, которые детализируют предшествующие заголовки или подзаголовки. Если это необходимо для про-

Пример 1.

Строка 1	КОМПАНИЯ J.D.S.S.G.	ОТЧЕТ ПО ЗАРАБОТНОЙ ПЛАТЕ			
Строка 2	СЛУЖАЩИЙ № 1049	ИМЯ СЛУЖАЩЕГО: НЭНСИ ХАНТ			
Строка 3	ОПЛАЧИВАЕМЫЙ ПЕРИОД	ЧАСЫ	ОБЩАЯ СУММА	ВЫЧЕТЫ	К ВЫПЛАТЕ
Строка 4	1	40.0	$120.00	$33.15	$86.85
Строка 5	2	36.0	$108.00	$21.62	$86.38
Строка 6	3	40.0	$120.00	$27.87	$92.13

Пример 2.

Строка 1	КОМПАНИЯ J.D.S.S.G.		ОТЧЕТ ПО ЗАРАБОТНОЙ ПЛАТЕ				
Строка 2	СЛУЖАЩИЙ №	ИМЯ	ПЕРИОД	ЧАСЫ	ОБЩАЯ СУММА	ВЫЧЕТЫ	К ВЫПЛАТЕ
Строка 3	1049	НЭНСИ ХАНТ	1	40.0	$120.00	$33.15	$ 86.85
Строка 4			2	36.0	$108.00	$21.62	$ 86.38
Строка 5			3	40.0	$120.00	$27.87	$ 92.13
Строка 6	1052	ДЖЕЙМС УОРРЕН	1	40.0	$140.00	$26.15	$113.85
Строка 7			2	40.0	$140.00	$20.02	$119.98
Строка 8			3	20.0	$ 70.00	$14.80	$ 55.20

Пример 3.

Строка 3	ПЕРИОД	ЧАСЫ	ОБЩАЯ СУММА		ВЫЧЕТЫ		К ВЫПЛАТЕ
Строка 4					УДЕРЖИВАЕМЫЕ НАЛОГИ	ПРОЧЕЕ	
Строка 5	1	40.0	$120.00	5.55	22.15	5.45	$86.85
Строка 6	2	36.0	$108.00	5.00	16.62	5.00	$86.38

ведения анализа и идентификации строк с данными отчета, можно использовать несколько строк с подзаголовками. Это показано на следующем примере.

Пример 3. В этом примере используется третья строка примера 1 и определяется связь строки подзаголовка с последующими подзаголовками.

В этом примере заголовок столбца (ВЫЧЕТЫ) разбивается далее на дополнительные подзаголовки.

Независимо от того, как программист интерпретирует или определяет заголовки и подзаголовки, они, так же как и строки названий, могут задаваться посредством неизменных или изменяемых констант. Для каждой подлежащей выводу на печать строки названия кодируется константа в 133 байта с оставлением пробелов в тех местах, куда должны вноситься изменяемые заголовки. При использовании строки названия, заголовка и подзаголовка, приведенных в примере 1 для фирмы изготовителя J.D.S.S.G, запись предложений DC иллюстрируется на рис. 16.3.

Интервалы между заголовками, представленными этими предложениями DC, не совпадают с приведенными в примере 1. Тем не менее все три типа определительных строк были сформированы с помощью этих предложений, и каждая строка может быть непосредственно определена как область вывода, связанная с таблицей DCB некоторого устройства вывода. Если для вывода на печать отводится только одна область вывода, то перед выполнением макрокоманды PUT для распечатки строки нужно засылать соответствующую группу данных в эту область вывода. При задании трех областей вывода для названия и заголовков данные можно распечатать тремя последовательными предложениями:

```
PUT     PRTDEV,TITLE        00001
PUT     PRTDEV,HEADING      00002
PUT     PRTDEV,SUBHEAD      00003
```

Предложение 00001 выводит на печать строку названия. Так как первый байт константы TITLE представляет собой 1, АЦПУ начнет печатать на новой странице.

Предложение 00002 пропускает две строки и затем печатает данные константы HEADING.

Предложение 00003 пропускает две строки и печатает данные константы SUBHEAD.

Хотя это и не показано в приведенных предложениях, программист должен также кодировать необходимые команды для засылки номера страницы в константу TITLE, а также номера, имени и фамилии служащего в константу HEADING перед выводом этих строк на печать.

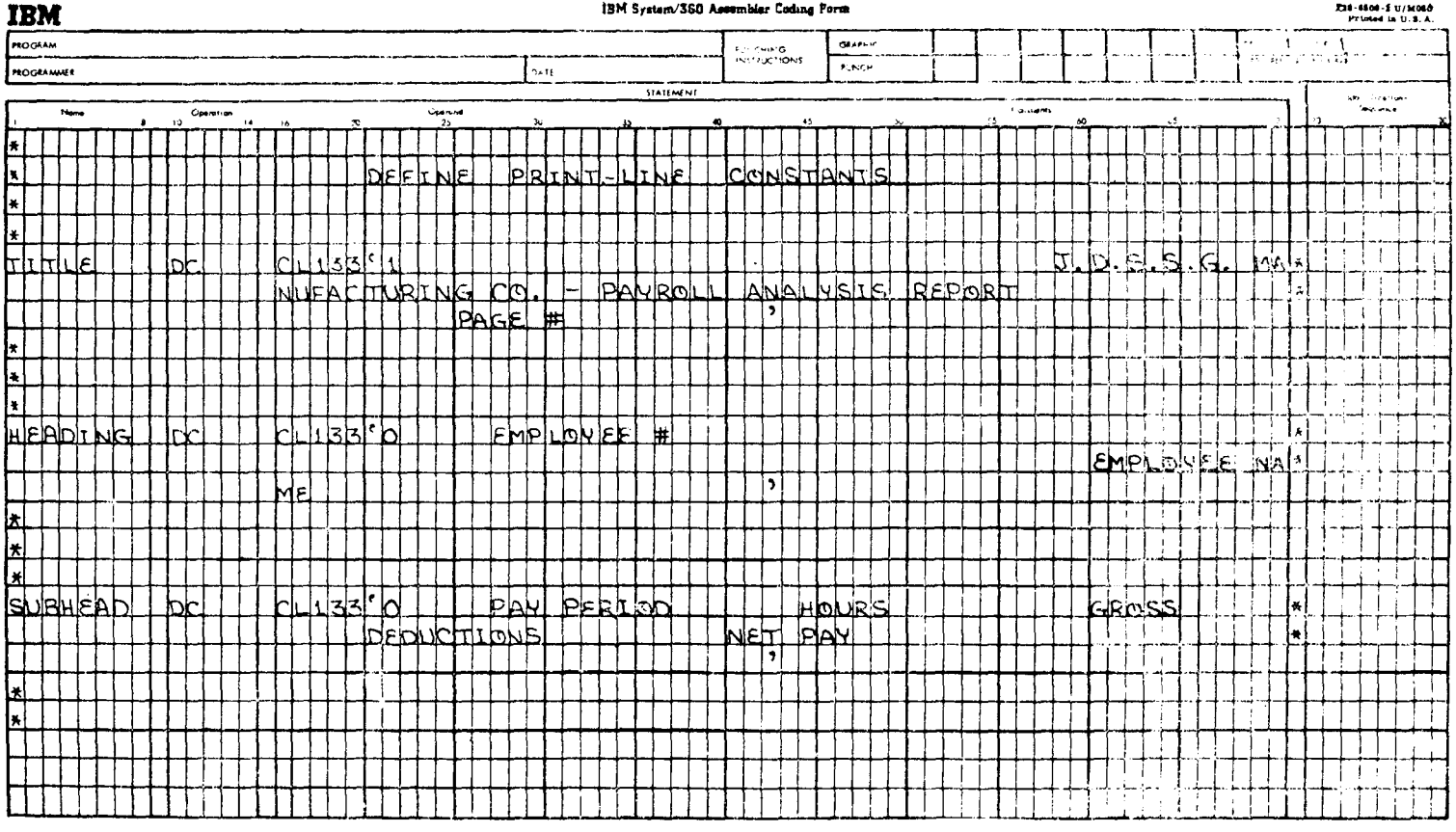

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
GRAPHIC
PUNCH
STATEMENT
DEFINE PRINT-LINE CONSTANTS
TITLE DC CL133'1
NUFACTURING CO. - PAYROLL ANALYSIS REPORT
PAGE #
J.D.S.S.G. 106
HEADING DC CL133'0
EMPLOYEE #
ME
EMPLOYEE NA
SUBHEAD DC CL133'0
PAY PERIOD
HOURS
GROSS
DEDUCTIONS
NET PAY

Рис. 16.3.

Несколько областей вывода для некоторого внешнего устройства с успехом можно использовать не только для распечатки названий и заголовков, как в приводимых здесь примерах. Разделение между логическими группами выводимых на печать данных можно усилить введением строки, состоящей из звездочек или дефисов. Большие промежутки между строками на странице можно обеспечить повторением команды PUT, которая выводит на печать строку пробелов, используя в качестве символа управления кареткой дефис, указывающий на необходимость пропустить перед печатью три строки.

Аналогичным способом можно пользоваться при выводе на дисплей данных, представленных в ряде разных форматов. При необходимости вывода на дисплей данных в определенном формате используется соответствующая этому формату область вывода.

Выбор формата листинга может служить хорошей проверкой здравого смысла и изобретательности программиста.

Глава 17

Обеспечение правильности данных

Команда Translate — TR (Перекодировать)

Мнемоника	Код операции	Формат операндов
TR	DC	$D_1(L,B_1),D_2(B_2)$

Эта команда может быть использована для преобразования конфигурации двоичных разрядов. Число двоичных разрядов при этом может быть меньше или равно восьми. Команда может использоваться для преобразования кодов ASCII, EBCDIC, кода перфоленты или в других случаях, когда пользователю требуется работать с определенной конфигурацией разрядов. Первый операнд указывает на местонахождение преобразуемых данных, а второй определяет начало таблицы перекодировки.

Обычно таблица перекодировки состоит из 256 байтов основной памяти, содержащих элементы данных, которые должны быть получены в результате преобразования. Таблица должна обеспечивать проверку всех 256 возможных двоичных восьмиразрядных конфигураций от X'00' до X'FF'. Программист должен составлять таблицу в соответствии с тем конкретным кодом, в который он желает преобразовать свои данные. Содержимое таблицы упорядочено по возрастанию шестнадцатеричных значений множества преобразуемых символов, каждый байт в таблице содержит конфигурацию разрядов символа, который в результате будет подставлен на место преобразуемого.

Команда добавляет к младшим разрядам адреса начала таблицы перекодировки шестнадцатеричное значение преобразуемого символа, обращается по увеличенному адресу, выбирает однобайтовую конфигурацию нового кода и замещает исходный символ конфигурацией, только что выбранной из таблицы перекодировки.

Пересылка перекодированных символов происходит слева направо по байту до тех пор, пока не окончится поле данных первого операнда, длина которого может быть задана неявно или явно.

В результате выполнения операции признак результата не изменяется.

Команда Translate and Test — TRT
(Перекодировать и проверить)

Мнемоника	Код операции	Формат операндов
TRT	DD	$D_1(L,B_1),D_2(B_2)$

Эта команда выполняется так же, как и команда TR, за тем исключением, что операция прекращается, когда в таблице перекодировки встречается байт, не содержащий величины X'00'. Команда используется главным образом для выявления специальных символов в процессе просмотра группы данных, причем, если такой символ встречается, информация об этом передается пользователю. Поле, содержащее предназначенные для просмотра данные, указывается первым операндом, адрес начала таблицы перекодировки указывается вторым операндом.

Принимая символы первого операнда за «проверяемые» символы, таблицу перекодировки организуют в соответствии с последовательностью этих символов как шестнадцатеричных величин. Таблица должна обеспечивать проверку всех 256 возможных двоичных восьмиразрядных конфигураций от X'00' до X'FF'. Если символ исходного поля не должен быть отмечен, то в байт таблицы, адресуемый этим символом, следует занести величину X'00'. Если символ должен быть отмечен, то в байт таблицы перекодировки, адресуемый этим символом, помещается число, отличающееся от всех других в этой таблице.

Команда TRT начинает (слева направо) проверять каждый символ поля данных, указанного первым операндом, по таблице перекодировки. Если элемент таблицы для этого символа содержит величину X'00', команда переходит к проверке следующего символа и т. д. Но если при проверке обнаружится, что величина в элементе таблицы не равна X'00', команда прерывает просмотр. Затем 24 правых разряда адреса проверяемого символа заносятся в общий регистр 1, а содержимое элемента таблицы, соответствующего этому символу, — в восемь правых разрядов (1 байт) общего регистра 2. В результате выполнения устанавливается следующий признак результата.

CC	BC	Условие
0	8	Все элементы таблицы были равны X'00'
1	4	Ненулевой элемент в таблице был обнаружен до окончания поля первого операнда
2	2	Байт в таблице для последнего проверяемого символа первого операнда был ненулевым

А. ОБЕСПЕЧЕНИЕ ПРАВИЛЬНОСТИ ДАННЫХ

Почти в каждой программе для коммерческих применений необходимо в той или иной мере обеспечить контроль правильности данных. Он осуществляется как просто путем сравнения, так и с использованием сложных подпрограмм, подтверждающих правильность числовых и символических данных. Подобный контроль над данными нужен по многим причинам; некоторые из них частично разъясняются ниже:

1. Определение категории вводимых данных, например в случае, когда таких категорий имеется несколько и для данных каждой категории имеется своя программа обработки.

2. Проверка правильности числовых величин. Она может состоять в проверке, является ли данная величина положительной, отрицательной или превышает некую минимальную величину.

3. Проверка с целью выявления специальных символов. Применяется, когда надо, например, удостовериться, что перед операцией упаковки поле чисел в коде EBCDIC не содержит пробелов.

4. Поиск определенных символов, которые, если обнаруживаются, должны преобразовываться в другой символ или группу символов или в некоторых случаях вовсе исключаться.

Имеются, вероятно, сотни причин обеспечения контроля над данными и еще больше способов осуществления этого контроля средствами языка Ассемблера. В данном разделе рассмотрено несколько методов такого контроля и изложены некоторые соображения, которые должны помочь программисту разрабатывать свои собственные приемы в этой области. Еще раз следует упомянуть о том, что каждая задача имеет много правильных с точки зрения программирования решений, а конфигурация вычислительной системы и объем имеющейся памяти нередко ограничивают возможности поиска наиболее эффективного из них.

1. Обеспечение правильности числовых величин

Проверка величин в числовом или десятичном представлении может принимать разнообразные формы. Вместо словесного описания этих форм рассмотрим в качестве примеров несколько различных задач по проверке десятичных величин. В каждом примере объясняется причина проверки, приведены необходимые команды и разъясняется их действие при выполнении.

Задача 1. Следует проверить, является ли отрицательной величина длиной в полное слово, полученная от другой части

проблемной программы. Если эта величина положительная, то она принимается; если отрицательная, то эта величина заменяется на положительный нуль. При составлении программы в первую очередь задается константа CHECKAMT длиной в полное слово, в которую до ее обработки данной контрольной подпрограммой заносится число с фиксированной точкой:

```
CHECKAMT    DC      F'0'
*
AUDIT       CLI     CHECKAMT,X'80'      0001
            BC      4,POSVALUE          0002
            MVC     CHECKAMT,=F'0'      0003
POSVALUE    — —     — — — — — — —      0004
            — —     — — — — — — —
```

Предложение 0001, первое в данной подпрограмме проверки, адресуется символической меткой AUDIT. Команда CLI определяет, в единицу или в нуль установлен самый левый знаковый разряд старшего байта. Непосредственный символ второго операнда этой команды имеет двоичное представление 1000 0000. Так как знаковый разряд отрицательной величины с фиксированной точкой равен единице, то в результате сравнения признак результата определит знак величины полного слова. Если содержимое поля CHECKAMT является величиной отрицательной, признак результата будет указывать на выполнение одного из условий: «равно» или «больше».

Предложение 0002 проверяет признак результата и, если признак результата указывает, что первый байт CHECKAMT был меньше, чем X'80', вызывает переход к POSVALUE. Если первый байт CHECKAMT не был меньше, чем X'80', что указывает на наличие в CHECKAMT отрицательной величины, перехода не произойдет и сохранится естественный порядок выполнения команд, согласно которому управление перейдет к предложению 0003. Если выполняется предложение 0003, то, согласно алгоритму программы, поле CHECKAMT содержит отрицательную величину. В соответствии с требованиями задачи отрицательная величина замещается величиной +0. Команда MVC засылает полное слово длиной в 4 байта, содержащее величину +0, в поле CHECKAMT.

Предложение 0004, адресуемое символической меткой POSVALUE, предположительно содержит команду следующей части алгоритма программы. К этому предложению совершается переход от предложения 0002, если поле CHECKAMT содержит положительную величину.

Задача 2. В соответствии с алгоритмом программы в общий регистр 6 была загружена некая величина. Данная программа проверки должна произвести обращение этой величины, если она по абсолютной величине больше чем 32768; в противном случае величина должна остаться неизменной.

```
CHEXRTE     LPR     5,6                 0001
            C       5,=F'32768'         0002
            BC      12,NOTGREAT         0003
            LCR     6,6                 0004
NOTGREAT    — —     — — — — — —         0005
            — —     — — — — — —
```

Предложение 0001 — первая команда программы. Она загружает абсолютное значение величины, находящейся в общем регистре 6, в общий регистр 5. Это позволяет выполнить проверку этой величины с помощью одной команды сравнения.

Предложение 0002 сравнивает величину, содержащуюся в общем регистре 5, с литеральной величиной с фиксированной точкой длиной в полное слово +32768. Результат сравнения («больше», «меньше» или «равно») выражается признаком результата в PSW.

Предложение 0003 проверяет результат сравнения, выполненного предложением 0002, и при этом вызывает переход к NOTGREAT, если в общем регистре 5 содержалась величина, равная или меньшая чем 32768. Если же величина в общем регистре 5 была больше чем 32768, то сохраняется естественный порядок выполнения команд и управление получает предложение 0004.

Если выполняется предложение 0004, то, согласно алгоритму, абсолютное значение величины, находящейся в общем регистре 6, независимо от того, положительная она или отрицательная, больше чем 32768. В соответствии с алгоритмом команда LCR заносит величину, содержащуюся в данный момент в общем регистре 6, снова в регистр 6, но в форме дополнения до 2. Если величина была сначала положительной, то будет загружено ее отрицательное дополнение; если величина была отрицательной, то будет загружаться положительное дополнение.

Предложение 0005 относится к следующей части алгоритма программы. Это предложение выполняется в результате перехода от предложения 0003 или после выполнения предложения 0004 при сохранении естественного порядка выполнения команд.

Задача 3. Эта подпрограмма получает ряд упакованных десятичных величин, каждая из которых может быть длиной от

2 до 4 байтов, и в самом правом полубайте может содержать правильный код знака.

Выполнение подпрограммы преследует две цели:

1. Все величины должны быть преобразованы в упакованные десятичные величины длиной 5 байтов без изменения их значений.

2. В правый полубайт каждой величины, не имеющей знака, должна быть помещена шестнадцатеричная цифра C, представляющая собой код знака плюс для чисел в упакованном десятичном формате.

Когда эта подпрограмма получает исходные упакованные десятичные числа, они уже выравнены по левой границе четырехбайтового поля с символической меткой INFIELD, а длина величины в байтах помещена в общий регистр 8 в форме числа с фиксированной точкой (рис. 17.1а и 17.1б).

Эта группа команд обеспечивает выполнение поставленной задачи, но при этом требует модификации указателя длины и величины смещения внутри команды.

В предложении 0901 (SIGNCHEX) команда LA помещает величину с фиксированной точкой +4 в общий регистр 9. Так как длина исходной упакованной десятичной величины содержится в общем регистре 8, то содержимое этого регистра вычитается из содержимого общего регистра 9; таким образом определяется величина смещения при пересылке упакованного десятичного числа в нужное поле.

Предложение 0902 вычитает содержащуюся в общем регистре 8 длину упакованной десятичной величины из максимально возможной длины в 4 байта, помещенной в общий регистр 9. Если длина упакованной десятичной величины меньше 4 байтов, результат этого вычитания даст приращение смещения.

Предложение 0903 вычитает литеральную константу с фиксированной точкой +1 длиной в полуслово из содержимого общего регистра 8. При этом подготавливается величина для внесения ее в качестве указателя длины в формат машинной команды, пересылающей исходное упакованное десятичное поле в поле результата.

Предложение 0904 записывает младший байт общего регистра 8, содержащий указатель длины упакованного десятичного поля в формате машинной команды, в однобайтовую область памяти с меткой LENGTH.

Предложение 0906 каждый раз, когда выполняется эта подпрограмма, очищает пятибайтовое упакованное десятичное поле, занося в него +0. Этим обеспечивается то, что во всех старших байтах поля результата будут содержаться нули.

IBM System/360 Assembler Coding Form

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 2 | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				08800
*				08810
*		DEFINED WORK AREAS		08820
*				08830
*				08840
LENGTH	DC	XL1'00'		08850
PACK5	DC	PL5'0'		08860
PACK1	DC	PL1'0'		08870
SIGNTEST	DC	XL1'00'		08880
OUTPACK	DC	PL5'0'		08890
*				08900
*				08910
*		AUDIT SUBROUTINE		08920
*				08930
*				08940
SIGNCHEX	LA	9,4		09000
	SR	9,8		09010
	SH	8,=H'1'		09020
	STC	8,LENGTH		09030
	ZAP	PACK5(5),=PL1'0'		09040
	LA	10,PACK5+1		09060
	MVC	LOADVAL+1(1),LENGTH		09070
	AR	10,9		09080
LOADVAL	MVC	0(1,10),INFIELD		09090

Рис. 17.1а.

IBM System/360 Assembler Coding Form

X28-6506-2 U/M050
Printed in U.S.A.

PROGRAM

PROGRAMMER

DATE

PAGE 2 OF 2

```
                MVN   SIGNTEST(1),PACK5+4
                CLI   SIGNTEST,X'0A'
                BC    12,NOTVALID
                ZAP   OUTPACK(5),PACK5
                BC    15,VALID
NOTVALID        MVI   PACK1,X'0C'
                MVO   PACK5(6),PACK5(5)
                ZAP   OUTPACK(5),PACK5+1
VALID           BC    15,COMPLETE
*
*
*
```

Рис. 17.1б.

Предложение 0907 загружает адрес второго байта упакованного десятичного поля результата в общий регистр 10. Этот адрес, сложенный с величиной приращения смещения, содержащейся в регистре 9, должен соответствующим образом выравнивать упакованную десятичную величину, пересылаемую в поле PACK5.

Предложение 0908 пересылает указатель длины исходной упакованной десятичной величины из байта в памяти, где он находится, во второй байт команды, которая осуществляет пересылку этой величины в поле PACK5.

Предложение 0909 складывает величину приращения смещения, содержащуюся в регистре 9, с содержимым общего регистра 10. Теперь общий регистр 10 содержит адрес байта поля PACK5, в который должна попасть величина из поля INFIELD.

Предложение 0910, модифицированное предложениями 0908 и 0909, пересылает соответствующее количество выравненных по левому краю упакованных десятичных байтов из поля INFIELD в соответствующие байты поля PACK5, выравнивая пересылаемое число по правому краю поля. Количество пересылаемых байтов вносится из поля LENGTH предложением 0908, а выравнивание выполняется путем сложения предложением 0909 содержимого регистра 10 с величиной приращения смещения из регистра 9.

Предложение 0911 — команда MVN — пересылает младший полубайт поля PACK5 в рабочую область памяти.

Предложение 0912 сравнивает с литеральной константой X'09' содержимое однобайтовой области, в которой старшая шестнадцатеричная цифра — 0, а младшая шестнадцатеричная цифра представляет собой полубайт, перенесенный в это поле предложением 0911. Это сравнение выполняется, чтобы определить, содержит ли упакованная десятичная величина в поле PACK5 код знака в младшем полубайте.

Предложение 0913 вызывает переход к NOTVALID, если сравнение в предложении 0912 показывает, что младший полубайт поля PACK5 содержит X'09' или меньше. Такое условие означает, что этот полубайт не содержит кода знака. Если же этот полубайт содержит правильный код знака, сохраняется естественный порядок выполнения команд, согласно которому управление получает предложение 0914.

Предложение 0914 — команда ZAP — пересылает все 5 байтов поля PACK5 в упакованное десятичное поле вывода. Независимо от того, какую длину имела величина в момент пересылки ее предложением 0910 в поле PACK5, она представляется в результате как пятибайтовое поле, содержащее в качестве старших упакованных десятичных цифр нули.

Предложение 0915 представляет собой безусловный переход к оператору выхода для этой подпрограммы.

Предложение 0916 (NOTVALID) выполняется в случае, если имеет место переход от предложения 0913. Это означает, что упакованная десятичная величина в PACK5 не содержит правильного кода знака в младшем полубайте. Предложение пересылает величину X'0C' в байт, следующий непосредственно за PACK5. Шестнадцатеричная цифра C обеспечивает стандартный код знака плюс для величины в поле PACK5.

Предложение 0917 — команда MVO — сдвигает упакованную десятичную величину, находящуюся в поле PACK5, на полбайта вправо в старший полубайт поля PACK1. В результате эта величина оказывается в непосредственном соседстве с шестнадцатеричной цифрой C, помещающейся в младшем полубайте поля PACK1, и образует вместе с ней упакованную десятичную величину с правильным кодом знака, расположенную в диапазоне адресов памяти от PACK5 + 1 по PACK1 включительно.

Предложение 0918 — команда ZAP — пересылает 5 правых байтов комбинированного шестибайтового поля с PACK5 +1 по PACK1 в пятибайтовое упакованное десятичное поле результата.

Предложение 0919 (VALID) является завершающим оператором во всей этой подпрограмме. Оно выполняет безусловный переход к предполагаемой программе COMPLETE.

Читателю предлагается взять несколько упакованных десятичных величин длиной от 1 до 4 байтов как со знаками, так и без знаков и проследить, как они обрабатываются этой подпрограммой.

Задача 4. В этой задаче программа получает шестибайтовое упакованное десятичное число с правильным кодом знака. Если обрабатываемая величина содержит младшие нули, они должны быть исключены путем сдвига всех других старших цифр полубайт за полубайтом вправо до тех пор, пока значащая цифра не поделит со знаком младший байт поля. Например, если исходное поле в формате упакованного десятичного числа выглядит следующим образом:

0	0	6	0	4	5	0	9	0	0	0	C

то после обработки оно модифицируется и приобретает вид

0	0	0	0	0	6	0	4	5	0	9	C

Поле сжимается по полубайтам до тех пор, пока первая значащая шестнадцатеричная цифра не оказывается рядом со знаком. Если в результате получается поле нулей, согласно алгоритму программы, управление получает специальная подпрограмма с меткой ZEROVALU. Эта программа состоит из следующих предложений:

```
TESTPACK  DC      PL6'0'                    04000
*
TESTRTE   ZAP     TESTPACK,INFIELD          04001
          LA      4,11                      04002
LOOPIN    TM      TESTPACK+5,X'F0'          04003
          BC      5,TESTDONE                04004
          MVO     TESTPACK(6),TESTPACK(5)   04005
          BCT     4,LOOPIN                  04006
          BC      15,ZEROVALU               04007
```

Предложение 04001 представляет собой команду ZAP, которая пересылает исходную упакованную десятичную величину в рабочую область с меткой TESTPACK.

Предложение 04002 — команда LA — помещает величину с фиксированной точкой +11 в общий регистр 4. Регистр 4 будет выполнять функцию счетчика цикла, который образуется с помощью команды BCT предложения 04006.

Предложение 04003 проверяет четыре левых разряда младшего байта поля TESTPACK. Операнд X'F0' команды TM только тогда вызовет установку разряда результата в единицу, если соответствующий бит из четырех левых разрядов самого правого байта поля TESTPACK равен единице.

Предложение 04004 — это команда условного перехода. Если результат выполнения команды TM в предложении 04003 показал, что из четырех левых разрядов поля TESTPACK+5 один или более установлены в единицу, то происходит переход от этого предложения к подпрограмме TESTDONE. Это означает, что упакованная десятичная цифра, находящаяся в настоящий момент непосредственно слева от знака, отлична от нуля.

Предложение 04005 — это команда MVO. Все полубайты поля TESTPACK, за исключением младшего байта, пересылаются вправо на один полубайт. Цифра, расположенная непосредственно слева от знака, замещается цифрой, находящейся слева от нее, и т. д. для всех цифр в поле TESTPACK. В свободный старший полубайт поля заносится нуль.

Предложение 04006 представляет собой команду BCT, которая передает управление предложению LOOPIN после завер-

шения очередного цикла. В общий регистр 4 была загружена величина +11, равная количеству десятичных цифр в шестибайтовом упакованном десятичном поле TESTPACK. Каждый раз, когда выполняется это предложение, команда BCT уменьшает содержимое общего регистра 4 на величину +1. Если величина в общем регистре 4 уменьшается до нуля, переход не происходит и сохраняется естественный порядок выполнения команд, согласно которому управление получает предложение 04007. Это будет означать, что все 11 десятичных цифр поля TESTPACK просмотрены, но значащей цифры не обнаружено.

Предложение 04007 представляет собой безусловный переход к подпрограмме с меткой ZEROVALU. Для того чтобы эта команда начала выполняться, должен сохраняться естественный порядок следования команд при выполнении команды BCT. Это означает, что исходная величина в поле TESTPACK была и остается равной нулю.

Внимательно просмотрите следующую группу команд и определите, в чем состоит отличие выполнения этой программы от программы предыдущего примера.

```
TESTRTE   ZAP   TESTPACK,INFIELD          05001
          CP    TESTPACK,=PL6'0'          05002
          BC    8,ZEROVALU                05003
LOOPIN    TM    TESTPACK+5,X'F0'          05004
          BC    5,TESTDONE                05005
          MVO   TESTPACK(6),TESTPACK(5)   05006
          BC    15,LOOPIN
```

Эта программа имеет такое же количество команд, что и предыдущая, но, вероятно, время ее выполнения будет меньше, если встречаются входные величины с нулевыми значениями. Предложения 05002 и 05003 позволяют обнаружить нулевое значение до того, как произойдет вход в цикл. Команды LA и BCT в этом случае не нужны, так как известно, что, если величина подвергается проверке в цикле, по крайней мере одна из цифр этой величины значащая.

2. Проверка на специальные символы или условия

Так как проверка этого типа производится при решении самых разнообразных задач, было бы бесполезным искать типичные примеры. Задачи на эту тему приведены просто для того, чтобы ознакомить читателя с общим принципом построения таких программ.

В большинстве случаев программа, в которой используются специальные методы проверки, имеет своей целью преобразование данных, представленных в одной форме, в другую форму. Примером может служить случай, когда файл данных содержит некоторое количество различных нераспечатываемых символов, которые необходимы для интерпретации этого файла, когда он выведен на печать. Если эти нераспечатываемые символы не будут преобразованы в печатаемые символы или в их комбинацию, каждый из них будет выведен на печать как пробел. Если, кроме того, согласно алгоритму проблемной программы для проверки данных требуется преобразование неправильных символов в правильные, то исключительно удобно применение команды TR.

Перекодировать — TR. Эта команда работает с таблицей, состоящей из 256 байтов, в которой каждый байт является представителем одного из 256 сочетаний двух шестнадцатеричных цифр, возможных в Системе/360. Каждый байт таблицы рассматривается как отдельный табличный элемент, смещение которого относительно начала таблицы определяется двоичной величиной того сочетания шестнадцатеричных цифр, которое он представляет. Например, если начало таблицы перекодировки имеет адрес 32000, то X′00′ соответствует первому элементу таблицы (32000), X′01′ — второму элементу (32001), X′02′ — третьему (32002), X′03′ — четвертому (32003) и т. д. Однако не обязательно, чтобы шестнадцатеричное содержимое байта (элемента) этой таблицы являлось тем же числом, которое выражает его смещение относительно начала таблицы.

Команда TR фактически производит следующие действия:

1. Выбирает байт данных, указанный первым операндом команды.

2. Складывает двоичную величину содержимого этого байта с адресом таблицы перекодировки, который определяется вторым операндом.

3. Переходит к элементу таблицы, используя увеличенный адрес.

4. Извлекает имеющийся в элементе символ и помещает его в поле первого операнда в байт, использованный для нахождения относительного местоположения табличного элемента.

Поле первого операнда может состоять из группы байтов; в этом случае каждый байт используется для адресации табличного элемента с тем, чтобы извлечь его содержимое.

В целях иллюстрации применения этой команды допустим, что нужно написать программу, которая исключит из полей данных все знаки пунктуации и некоторые специальные знаки, заменив их косой чертой. Это означает, что любой байт

таблицы, относительное положение которого от начала этой таблицы выражается величиной, шестнадцатеричное представление которой изображает в коде EBCDIC знак пунктуации или какой-либо специальный знак, должен содержать шестнадцатеричное представление косой черты. Стандартное представление для этого символа в коде EBCDIC будет X'61'. Соответственно следующие относительные байты таблицы содержат шестнадцатеричное число 61.

Шестнадцатеричное представление кодов EBCDIC для знаков пунктуации и специальных знаков	Представленный знак		Относительное положение в таблице соответствующего байта, содержащего X'61'
4A	Цент	¢	4A
4B	Точка	.	4B
4C	Меньше	<	4C
4D	Левая скобка	(	4D
4E	Плюс	+	4E
4F	Примыкание	|	4F
50	Коммерческое И	&	50
5A	Восклицание	!	5A
5B	Доллар	$	5B
5C	Звездочка	*	5C
5D	Правая скобка	)	5D
5E	Точка с запятой	;	5E
5F	Отрицание	¬	5F
60	Дефис	-	60
61	Косая черта	/	61
6B	Запятая	,	6B
6C	Процент	%	6C
6D	Черта снизу	_	6D
6E	Больше	>	6E
6F	Вопросительный знак	?	6F
7A	Двоеточие	:	7A
7B	Фунт	#	7B
7C	«at»	@	7C
7D	Апостроф	'	7D
7E	Равно	=	7E
7F	Кавычки	»	7F

Все другие элементы таблицы содержат стандартные представления символов в коде EBCDIC. Для пояснения сказанного приведем три таблицы.

Таблица перекодировки 1 (табл. 17.1) содержит 256 отдельных элементов. Шестнадцатеричная величина внутри каждого

элемента представляет двоичную величину, характеризующую положение элемента относительно начала таблицы. В каждом элементе представлено также соответствующее десятичное значение этих двоичных величин.

Таблица 17.1

Таблица перекодировки 1

X'00'[a] 0[b]	X'01' 1	X'02' 2	X'03' 3	X'04' 4	X'05' 5	X'06' 6	X'07' 7	X'08' 8	X'09' 9	X'0A' 10	X'0B' 11	X'0C' 12	X'0D' 13	X'0E' 14	X'0F' 15
X'10' 16	X'11' 17	X'12' 18	X'13' 19	X'14' 20	X'15' 21	X'16' 22	X'17' 23	X'18' 24	X'19' 25	X'1A' 26	X'1B' 27	X'1C' 28	X'1D' 29	X'1E' 30	X'1F' 31
X'20' 32	X'21' 33	X'22' 34	X'23' 35	X'24' 36	X'25' 37	X'26' 38	X'27' 39	X'28' 40	X'29' 41	X'2A' 42	X'2B' 43	X'2C' 44	X'2D' 45	X'2E' 46	X'2F' 47
X'30' 48	X'31' 49	X'32' 50	X'33' 51	X'34' 52	X'35' 53	X'36' 54	X'37' 55	X'38' 56	X'39' 57	X'3A' 58	X'3B' 59	X'3C' 60	X'3D' 61	X'3E' 62	X'3F' 63
X'40' 64	X'41' 65	X'42' 66	X'43' 67	X'44' 68	X'45' 69	X'46' 70	X'47' 71	X'48' 72	X'49' 73	X'4A' 74	X'4B' 75	X'4C' 76	X'4D' 77	X'4E' 78	X4F' 79
X'50' 80	X'51' 81	X'52' 82	X'53' 83	X'54' 84	X'55' 85	X'56' 86	X'57' 87	X'58' 88	X'59' 89	X'5A' 90	X'5B' 91	X'5C' 92	X'5D' 93	X'5E' 94	X'5F' 95
X'60' 96	X'61' 97	X'62' 98	X'63' 99	X'64' 100	X'65' 101	X'66' 102	X'67' 103	X'68' 104	X'69' 105	X'6A' 106	X'6B' 107	X'6C' 108	X'6D' 109	X'6E' 110	X'6F' 111
X'70' 112	X'71' 113	X'72' 114	X'73' 115	X'74' 116	X'75' 117	X'76' 118	X'77' 119	X'78' 120	X'79' 121	X'7A' 122	X'7B' 123	X'7C' 124	X'7D' 125	X'7E' 126	X'7F' 127
X'80' 128	X'81' 129	X'82' 130	X'83' 131	X'84' 132	X'85' 133	X'86' 134	X'87' 135	X'88' 136	X'89' 137	X'8A' 138	X'8B' 139	X'8C' 140	X'8D' 141	X'8E' 142	X'8F' 143
X'90' 144	X'91' 145	X'92' 146	X'93' 147	X'94' 148	X'95' 149	X'96' 150	X'97' 151	X'98' 152	X'99' 153	X'9A' 154	X'9B' 155	X'9C' 156	X'9D' 157	X'9E' 158	X'9F' 159
X'A0' 160	X'A1' 161	X'A2' 162	X'A3' 163	X'A4' 164	X'A5' 165	X'A6' 166	X'A7' 167	X'A8' 168	X'A9' 169	X'AA' 170	X'AB' 171	X'AC' 172	X'AD' 173	X'AE' 174	X'AF' 175
X'B0' 176	X'B1' 177	X'B2' 178	X'B3' 179	X'B4' 180	X'B5' 181	X'B6' 182	X'B7' 183	X'B8' 184	X'B9' 185	X'BA' 186	X'BB' 187	X'BC' 188	X'BD' 189	X'BE' 190	X'BF' 191
X'C0' 192	X'C1' 193	X'C2' 194	X'C3' 195	X'C4' 196	X'C5' 197	X'C6' 198	X'C7' 199	X'C8' 200	X'C9' 201	X'CA' 202	X'CB' 203	X'CC' 204	X'CD' 205	X'CE' 206	X'CF' 207
X'D0' 208	X'D1' 209	X'D2' 210	X'D3' 211	X'D4' 212	X'D5' 213	X'D6' 214	X'D7' 215	X'D8' 216	X'D9' 217	X'DA' 218	X'DB' 219	X'DC' 220	X'DD' 221	X'DE' 222	X'DF' 223
X'E0' 224	X'E1' 225	X'E2' 226	X'E3' 227	X'E4' 228	X'E5' 229	X'E6' 230	X'E7' 231	X'E8' 232	X'E9' 233	X'EA' 234	X'EB' 235	X'EC' 236	X'ED' 237	X'EE' 238	X'EF' 239
X'F0' 240	X'F1' 241	X'F2' 242	X'F3' 243	X'F4' 244	X'F5' 245	X'F6' 246	X'F7' 247	X'F8' 248	X'F9' 249	X'FA' 250	X'FB' 251	X'FC' 252	X'FD' 253	X'FE' 254	X'FF' 255

Примечания: [a] относительное положение элемента, выраженное шестнадцатеричным числом;
[b] десятичное представление того же числа.

Таблица перекодировки 2 (табл. 17.2), представляющая те же 256 символов, демонстрирует содержимое таблицы при условии, что каждый элемент содержит число в шестнадцатеричной форме, характеризующее местоположение элемента относительно начала таблицы. В этом случае там, где это возможно, включены представления символов в коде EBCDIC.

Таблица 17.2

Таблица перекодировки 2 (стандартное шестнадцатеричное и соответствующее ему представление в коде EBCDIC)

X'00'[a]	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'	X'08'	X'09'	X'0A'	X'0B'	X'0C'	X'0D'	X'0E'	X'0F'
X'10'	X'11'	X'12'	X'13'	X'14'	X'15'	X'16'	X'17'	X'18'	X'19'	X'1A'	X'1B'	X'1C'	X'1D'	X'1E'	X'1F'
X'20'	X'21'	X'22'	X'23'	X'24'	X'25'	X'26'	X'27'	X'28'	X'29'	X'2A'	X'2B'	X'2C'	X'2D'	X'2E'	X'2F'
X'30'	X'31'	X'32'	X'33'	X'34'	X'35'	X'36'	X'37'	X'38'	X'39'	X'3A'	X'3B'	X'3C'	X'3D'	X'3E'	X'3F'
X'40' b"	X'41'	X'42'	X'43'	X'44'	X'45'	X'46'	X'47'	X'48'	X'49'	X'4A' ¢	X'4B'	X'4C' <	X'4D' (	X'4E' +	X'4F' |
X'50' &	X'51'	X'52'	X'53'	X'54'	X'55'	X'56'	X'57'	X'58'	X'59'	X'5A' !	X'5B' $	X'5C' *	X'5D')	X'5E' ;	X'5F' ¬
X'60' -	X'61' /	X'62'	X'63'	X'64'	X'65'	X'66'	X'67'	X'68'	X'69'	X'6A' ¦	X'6B' ,	X'6C' %	X'6D' _	X'6E' >	X'6F' ?
X'70'	X'71'	X'72'	X'73'	X'74'	X'75'	X'76'	X'77'	X'78'	X'79'	X'7A' :	X'7B' #	X'7C' @	X'7D' '	X'7E' =	X'7F' "
X'80'	X'81' a	X'82' b	X'83' c	X'84' d	X'85' e	X'86' f	X'87' g	X'88' h	X'89' i	X'8A'	X'8B'	X'8C'	X'8D'	X'8E'	X'8F'
X'90'	X'91' j	X'92' k	X'93' l	X'94' m	X'95' n	X'96' o	X'97' p	X'98' q	X'99' r	X'9A'	X'9B'	X'9C'	X'9D'	X'9E'	X'9F'
X'A0'	X'A1'	X'A2' s	X'A3' t	X'A4' u	X'A5' v	X'A6' w	X'A7' x	X'A8' y	X'A9' z	X'AA'	X'AB'	X'AC'	X'AD'	X'AE'	X'AF'
X'B0'	X'B1'	X'B2'	X'B3'	X'B4'	X'B5'	X'B6'	X'B7'	X'B8'	X'B9'	X'BA'	X'BB'	X'BC'	X'BD'	X'BE'	X'BF'
X'C0'	X'C1' A	X'C2' B	X'C3' C	X'C4' D	X'C5' E	X'C6' F	X'C7' G	X'C8' H	X'C9' I	X'CA'	X'CB'	X'CC'	X'CD'	X'CE'	X'CF'
X'D0'	X'D1' J	X'D2' K	X'D3' L	X'D4' M	X'D5' N	X'D6' O	X'D7' P	X'D8' Q	X'D9' R	X'DA'	X'DB'	X'DC'	X'DD'	X'DE'	X'DF'
X'E0'	X'E1'	X'E2' S	X'E3' T	X'E4' U	X'E5' V	X'E6' W	X'E7' X	X'E8' Y	X'E9' Z	X'EA'	X'EB'	X'EC'	X'ED'	X'EE'	X'EF'
X'F0' 0	X'F1' 1	X'F2' 2	X'F3' 3	X'F4' 4	X'F5' 5	X'F6' 6	X'F7' 7	X'F8' 8	X'F9' 9	X'FA'	X'FB'	X'FC'	X'FD'	X'FE'	X'FF'

Примечания: [a] содержимое элемента в шестнадцатеричной форме.
[b] представление элемента в коде EBCDIC.

Таблица перекодировки 3 (табл. 17.3) — это модифицированная программистом таблица для конкретного применения. Следует обратить внимание на то, что элементы для знаков пунктуации в коде EBCDIC и для специальных знаков теперь содержат X'61' — код косой черты.

Предложения, приведенные на рис. 17.2а и 17.2б, задают четыре поля данных, обработанных командой TR с использованием таблицы перекодировки 3, — это поля FIELDA, FIELDB, FIELDC и FIELDD. В данном примере поля определены как константы. В действительности же данные, подвергающиеся проверке, пополняют рабочую область непрерывно.

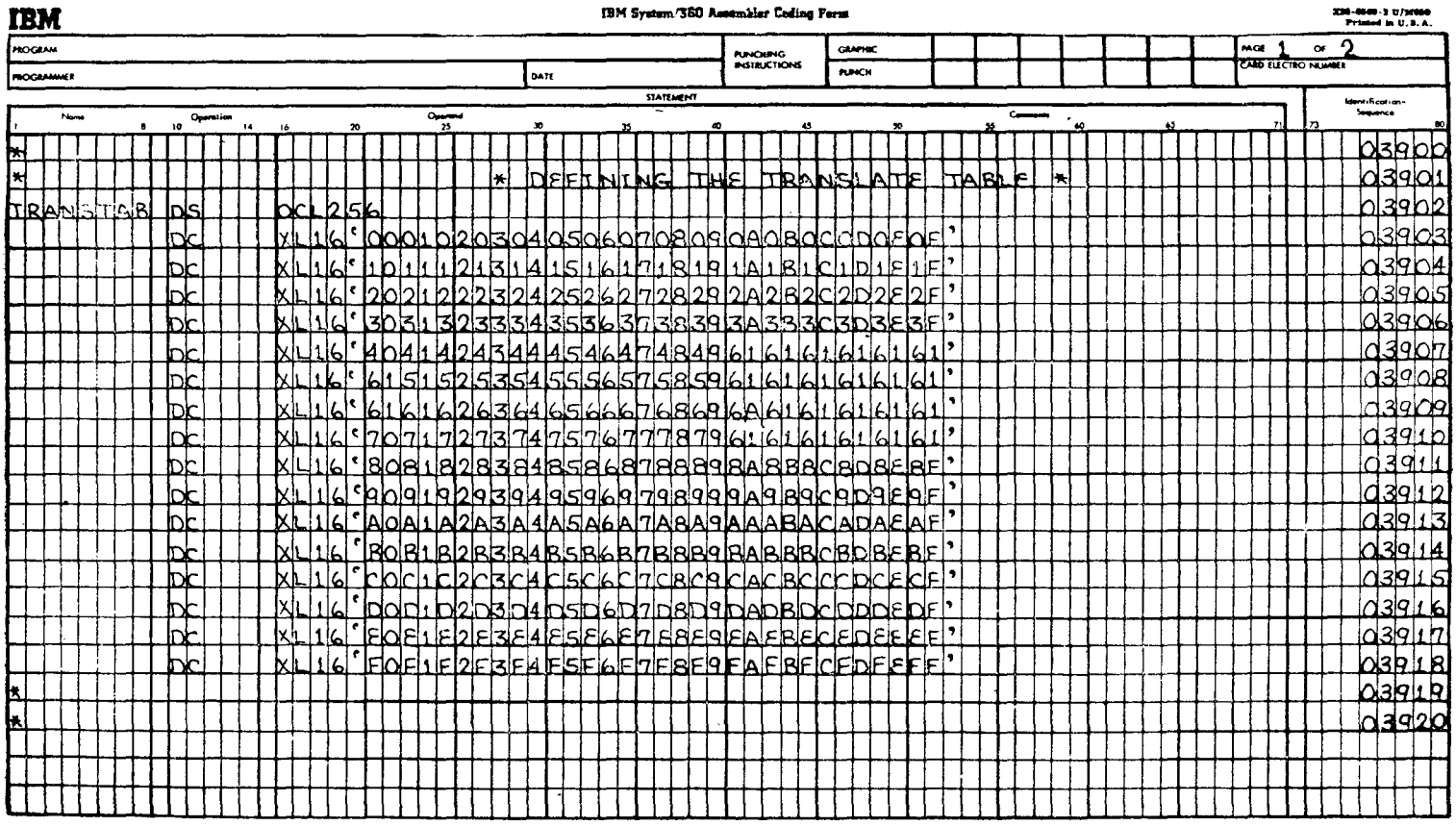

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 1 OF 2
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
* DEFINING THE TRANSLATE TABLE *
TRANSTAB DS DCL256
DC XL16'000102030405060708090A0B0C0D0E0F'
DC XL16'101112131415161718191A1B1C1D1E1F'
DC XL16'202122232425262728292A2B2C2D2E2F'
DC XL16'303132333435363738393A3B3C3D3E3F'
DC XL16'404142434445464748494B4B4C4D4E4F'
DC XL16'505152535455565758595B5B5C5D5E5F'
DC XL16'606162636465666768696B6B6C6D6E6F'
DC XL16'707172737475767778797B7B7C7D7E7F'
DC XL16'808182838485868788898B8B8C8D8E8F'
DC XL16'909192939495969798999B9B9C9D9E9F'
DC XL16'A0A1A2A3A4A5A6A7A8A9ABABACADAEAF'
DC XL16'B0B1B2B3B4B5B6B7B8B9BBBBBCBDBEBF'
DC XL16'C0C1C2C3C4C5C6C7C8C9CBCBCCCDCECF'
DC XL16'D0D1D2D3D4D5D6D7D8D9DBDBDCDDDEDF'
DC XL16'E0E1E2E3E4E5E6E7E8E9EBEBECEDEEEF'
DC XL16'F0F1F2F3F4F5F6F7F8F9FBFBFCFDFEFF'
0039000
0039001
0039002
0039003
0039004
0039005
0039006
0039007
0039008
0039009
0039010
0039011
0039012
0039013
0039014
0039015
0039016
0039017
0039018
0039019
0039020

Рис. 17.2а.

IBM

IBM System/360 Assembler Coding Form

X28 6508 2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 2

PROGRAMMER | DATE

Name	Operation	Operand	Comments	Sequence
*				03921
*				03922
FIELDA	DC	CL15'AMT. 415 PER'		03923
FIELDB	DC	CL15'ID IS 23-A65-776'		03924
FIELDC	DC	CL15'RATIO 15%PCT EA'		03925
FIELDD	DC	CL15'DATE: 7-15-69'		03926
*				03927
OUTFLDA	DC	CL15' '		03928
OUTFLDB	DC	CL15' '		03929
OUTFLDC	DC	CL15' '		03930
OUTFLDD	DC	CL15' '		03931
*				03932
*				
TRANSOUT	LA	6,4		04001
	LA	7,TRANSTAB		04002
	LA	8,FIELDA		04003
	LA	9,OUTFLDA		04004
NEXTRAN	IR	0(15,8),0(7)		04005
	MVC	0(15,9),0(8)		04006
	AH	8,=H'15'		04007
	AH	9,=H'15'		04008
	BCT	6,NEXTRAN		04009
	BC	15,DONE		04010
*				04011

Рис. 17.26.

Таблица 17.3

Таблица перекодировки 3 (элементы, представляющие знаки пунктуации и специальные знаки в коде EBCDIC, содержат код косой черты)

X'00'[a]	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'	X'08'	X'09'	X'0A'	X'0B'	X'0C'	X'0D'	X'0E'	X'0F'
X'10'	X'11'	X'12'	X'13'	X'14'	X'15'	X'16'	X'17'	X'18'	X'19'	X'1A'	X'1B'	X'1C'	X'1D'	X'1E'	X'1F'
X'20'	X'21'	X'22'	X'23'	X'24'	X'25'	X'26'	X'27'	X'28'	X'29'	X'2A'	X'2B'	X'2C'	X'2D'	X'2E'	X'2F'
X'30'	X'31'	X'32'	X'33'	X'34'	X'35'	X'36'	X'37'	X'38'	X'39'	X'3A'	X'3B'	X'3C'	X'3D'	X'3E'	X'3F'
X'40' SP[b]	X'41'	X'42'	X'43'	X'44'	X'45'	X'46'	X'47'	X'48'	X'49'	X'4A' ¢	X'4B'	X'4C' <	X'4D' (	X'4E' +	X'4F' |
X'50' &	X'51'	X'52'	X'53'	X'54'	X'55'	X'56'	X'57'	X'58'	X'59'	X'5A' !	X'5B' $	X'5C' *	X'5D')	X'5E' ;	X'5F' ¬
X'60' -	X'61' /	X'62'	X'63'	X'64'	X'65'	X'66'	X'67'	X'68'	X'69'	X'6A' ¦	X'6B' ,	X'6C' %	X'6D' —	X'6E' >	X'6F' ?
X'70'	X'71'	X'72'	X'73'	X'74'	X'75'	X'76'	X'77'	X'78'	X'79'	X'7A' :	X'7B' #	X'7C' @	X'7D' '	X'7E' =	X'7F' "
X'80'	X'81' a	X'82' b	X'83' c	X'84' d	X'85' e	X'86' f	X'87' g	X'88' h	X'89' i	X'8A'	X'8B'	X'8C'	X'8D'	X'8E'	X'8F'
X'90'	X'91' j	X'92' k	X'93' l	X'94' m	X'95' n	X'96' o	X'97' p	X'98' q	X'99' r	X'9A'	X'9B'	X'9C'	X'9D'	X'9E'	X'9F'
X'A0'	X'A1'	X'A2' s	X'A3' t	X'A4' u	X'A5' v	X'A6' w	X'A7' x	X'A8' y	X'A9' z	X'AA'	X'AB'	X'AC'	X'AD'	X'AE'	X'AF'
X'B0'	X'B1'	X'B2'	X'B3'	X'B4'	X'B5'	X'B6'	X'B7'	X'B8'	X'B9'	X'BA'	X'BB'	X'BC'	X'BD'	X'BE'	X'BF'
X'C0'	X'C1' A	X'C2' B	X'C3' C	X'C4' D	X'C5' E	X'C6' F	X'C7' G	X'C8' H	X'C9' I	X'CA'	X'CB'	X'CC'	X'CD'	X'CE'	X'CF'
X'D0'	X'D1' J	X'D2' K	X'D3' L	X'D4' M	X'D5' N	X'D6' O	X'D7' P	X'D8' Q	X'D9' R	X'DA'	X'DB'	X'DC'	X'DD'	X'DE'	X'DF'
X'E0'	X'E1'	X'E2' S	X'E3' T	X'E4' U	X'E5' V	X'E6' W	X'E7' X	X'E8' Y	X'E9' Z	X'EA'	X'EB'	X'EC'	X'ED'	X'EE'	X'EF'
X'F0' 0	X'F1' 1	X'F2' 2	X'F3' 3	X'F4' 4	X'F5' 5	X'F6' 6	X'F7' 7	X'F8' 8	X'F9' 9	X'FA'	X'FB'	X'FC'	X'FD'	X'FE'	X'FF'

Примечания: [a] содержимое элемента в шестнадцатеричной форме;
[b] представление элемента в коде EBCDIC.

Предложение 04001 (TRANSOUT) загружает в общий регистр 6 величину +4. Эта величина в регистре будет использоваться как счетчик при управлении циклом, который вызывается командой BCT (предложение 04009).

Предложение 04002 загружает адрес таблицы перекодировки (TRANSTAB) в общий регистр 7. В данном примере предполагается, что во время выполнения программы таблица TRANSTAB находится в области памяти по адресу 53200. Теперь общий регистр 7 содержит в трех младших байтах величину с фиксированной точкой +53200.

Предложение 04003 загружает адрес первого из четырех полей данных в общий регистр 8.

Предложение 04004 загружает адрес поля OUTFLDA в общий регистр 9. По мере проверки каждого входного поля перекодированные данные посылаются в соответствующее выходное поле. Поле OUTFLDA получит проверенные данные из поля FIELDA, поле OUTFLDB — из поля FIELDB, поле OUTFLDC — из поля FIELDC и поле OUTFLDD — из поля FIELDD.

Предложение 04005 — команда TR — обрабатывает 15 байтов поля данных, адрес которого содержится в общем регистре 8. Первый раз при выполнении этой команды общий регистр 8 содержит адрес поля FIELDA, загруженный предложением 04003. Затем содержимое регистра 8 увеличивается предложением 04007 на величину с фиксированной точкой +15, так что при следующем выполнении данной команды регистр 8 содержит адрес поля FIELDB. При следующих выполнениях данной последовательности команд содержимое регистра 8 увеличивается так, что он содержит адреса полей FIELDC и FIELDD соответственно.

Предложение 04006 пересылает обработанные данные, адресуемые общим регистром 8, в поле, адрес которого содержится в общем регистре 9. При первом прохождении через это предложение регистр 9 содержит адрес поля OUTFLDA. При каждом следующем прохождении через это предложение величина в регистре 9 увеличивается так, что этот регистр последовательно содержит адреса полей OUTFLDB, OUTFLDC и OUTFLDD.

Предложение 04007 складывает литеральную величину с фиксированной точкой +15 длиной в полуслово с содержимым общего регистра 8, увеличивая содержимое этого регистра так, чтобы оно являлось адресом начала следующего поля входных данных.

Предложение 04008 складывает литеральную величину с фиксированной точкой +15 длиной в полуслово с содержимым общего регистра 9, увеличивая содержимое этого регистра так, чтобы оно являлось адресом начала следующего поля выходных данных.

Предложение 04009 — команда BCT — управляет циклом; так как предложение 04001 загрузило в общий регистр 6 величину +4, при выполнении программы трижды будет сделан переход к NEXTRAN. При четвертом прохождении через это предложение величина в общем регистре 6 уменьшается до нуля, и далее операторы выполняются в естественном порядке, в соответствии с которым управление переходит к предложению 04010.

Предложение 04010 представляет собой безусловный переход к предполагаемой подпрограмме окончания задания. Воз-

можно, что эта подпрограмма извлечет данные из четырех выходных полей, поместит новые данные во входные поля, откуда они будут взяты для обработки, и затем передаст управление обратно к TRANSOUT.

Когда бы при обработке каждого из четырех входных полей ни выполнялась команда TR, она всегда будет замещать все знаки пунктуации и специальные знаки кодом косой черты. В соответствии с допущением, что первый байт поля TRANSTAB во время выполнения программы находится в памяти по адресу 53200, каждый последующий байт поля TRANSTAB будет иметь адрес, равный 53200 плюс число в двоичной форме, выражающее относительное положение этого элемента в таблице. Например, байт с относительным местоположением X'09' будет находиться по адресу 53209.

Действия, которые команда TR производит над полем FIELDA, показаны в табл. 17.4.

Таблица 17.4

Символ поля FIELDA			Сумма числа, являющегося представлением символа и адреса TRANSTAB (53200)	Символ, содержащийся в этом элементе поля TRANSTAB		Символ, помещаемый в FIELDA командой TR
символ	шестн. представление	двоичн. величина		шестн. представление	символ	
A	C1	193	53393	C1	A	A
M	D4	212	53412	D4	M	M
T	E3	227	53427	E3	T	T
.	4B	75	53275	61	/	/
␣	40	64	53264	40	␣	␣
␣	40	64	53264	40	␣	␣
␣	40	64	53264	40	␣	␣
$	5B	91	53291	61	/	/
1	F1	241	53441	F1	1	1
5	F5	245	53445	F5	5	5
␣	40	64	53264	40	␣	␣
␣	40	64	53264	40	␣	␣
P	D7	215	53415	D7	P	P
E	C5	197	53397	C5	E	E
R	D9	217	53417	D9	R	R

После того как содержимое поля FIELDA переслано в поле OUTFLDA, последнее примет вид

A	M	T	/	␣	␣	␣	/	1	5	␣	␣	P	E	R

Действия, которые команда TR производит над полем FIELDB, показаны в табл. 17.5.

Таблица 17.5

Символ поля FIELDB			Сумма числа, являющегося представлением символа и адреса TRANSTAB (53200)	Символ, содержащийся в этом элементе поля TRANSTAB		Символ, помещаемый в FIELDB командой TR
символ	шестн. представление	двоичн. величина		шестн. представление	символ	
I	C9	201	53401	C9	I	I
D	C4	196	53396	C4	D	D
␣	40	64	53264	40	␣	␣
I	C9	201	53401	C9	I	I
S	E2	226	53426	E2	S	S
␣	40	64	53264	40	␣	␣
2	F2	242	53442	F2	2	2
3	F3	243	53443	F3	3	3
—	60	96	53296	61	/	/
A	C1	193	53393	C1	A	A
6	F6	246	53446	F6	6	6
5	F5	245	53445	F5	5	5
—	60	96	53296	61	/	/
7	F7	247	53447	F7	7	7
6	F6	246	53446	F6	6	6

После того как содержимое поля FIELDB переслано в поле OUTFLDB, последнее примет вид

I	D	␣	I	S	␣	2	3	/	A	6	5	/	7	6

Действия, производимые командой TR над полем FIELDC, показаны в табл. 17.6.

Действия, производимые командой TR над полем FIELDD, показаны в табл. 17.7.

Программист может загрузить элементы таблицы перекодировки любыми однобайтовыми шестнадцатеричными конфигурациями. Затем содержимое каждого элемента заменит подвергающиеся перекодировке данные, которые формируют адрес этого элемента.

Перекодировать и проверить — TRT. Эта команда использует ту же схему адресации таблицы, что и команда TR. Это значит, что шестнадцатеричное представление проверяемого байта данных используется как число, на которое увеличи-

Таблица 17.6

Символ поля FIELDC			Сумма числа, являющегося представлением символа и адреса TRANSTAB (53200)	Символ, содержащийся в этом элементе поля TRANSTAB		Символ, помещаемый в FIELDC командой TR
символ	шестн. представление	двоичн. величина		шестн. представление	символ	
R	D9	217	53417	D9	R	R
A	C1	193	53393	C1	A	A
T	E3	227	53427	E3	T	T
I	C9	201	53401	C9	I	I
O	D6	214	53414	D6	O	O
␣	40	64	53264	40	␣	␣
1	F1	241	53441	F1	1	1
5	F5	245	53445	F5	5	5
%	6C	108	53308	61	/	/
P	D7	215	53415	D7	P	P
C	C3	195	53395	C3	C	C
T	E3	227	53427	E3	T	T
␣	40	64	53264	40	␣	␣
E	C5	197	53397	C5	E	E
A	C1	193	53393	C1	A	A

После того как содержимое поля FIELDC переслано в поле OUTFLDC, последнее примет вид

R	A	T	I	O	␣	1	5	/	P	C	T	␣	E	A

вается адрес первого байта таблицы перекодировки. Но в отличие от команды TR команда TRT не изменяет конфигурацию проверяемого поля данных.

Команда TRT использует каждый байт поля данных первого операнда для формирования адреса элемента таблицы перекодировки и анализирует этот элемент. Если этот элемент содержит конфигурацию X'00', команда продолжает обработку, проверяя по таблице следующий байт поля данных. Если при проверке элемента обнаруживается, что он содержит конфигурацию, отличную от X'00', дальнейший анализ данных прекращается. При этом команда TRT загружает в общий регистр 1 адрес байта данных первого операнда, который анализировался последним. Команда также загружает в общий регистр 2 содержимое байта таблицы перекодировки, на котором дальнейший анализ данных прекращается.

Таблица 17.7

Символ поля FIELDD			Сумма числа, являющегося представлением символа и адреса TRANSTAB (53200)	Символ, содержащийся в этом элементе поля TRANSTAB		Символ, помещаемый в поле FIELDC командой TR
символ	шестн. представление	двоичн. величина		шестн. представление	символ	
D	C4	196	53396	C4	D	D
A	C1	193	53393	C1	A	A
T	E3	227	53427	E3	T	T
E	C5	197	53397	C5	E	E
:	7A	122	53322	61	/	/
¤	40	64	53264	40	¤	¤
¤	40	64	53264	40	¤	¤
7	F7	247	53447	F7	7	7
—	60	96	53296	61	/	/
1	F1	241	53441	F1	1	1
5	F5	245	53445	F5	5	5
—	60	96	53296	61	/	/
6	F6	246	53446	F6	6	6
9	F9	249	53449	F9	9	9
¤	40	64	53264	40	¤	¤

После того как содержимое поля FIELDD переслано в поле OUTFLDD, последнее примет вид

D	A	T	-E	/	¤	¤	7	/	1	5	/	6	9	¤

Практическое применение эта команда находит тогда, когда нужно обнаружить ограничители или разделительные знаки в полях, записях или сообщениях переменной длины. Допустим, например, что построена группа записей, причем каждая запись содержит сообщения различной длины. По мере создания записи между отдельными сообщениями, а также непосредственно после последнего сообщения помещалась однобайтовая конфигурация двоичных разрядов X'01'. Теперь группу записей нужно передать в другое место и затем извлечь по одному сообщению из каждой записи. В этом случае при получении сообщения было бы удобно с целью отыскания промежутков между сообщениями использовать команду TRT. Так как кодом промежутка, вставляемым после каждого сообщения, является шестнадцатеричное число 01, то программист должен построить таблицу перекодировки, в которой бы все элементы содержали

X'00', за исключением элемента, относительный адрес которого был бы равен X'01'. Как уже указывалось в предыдущих примерах, им будет второй по порядку элемент таблицы. В таблице перекодировки (табл. 17.8) программист решил загрузить этот элемент числом X'01', выражающим адрес этого элемента относительно начала таблицы.

Таблица 17.8

Таблица перекодировки для команды TRT

X'00'	X'01'[a]	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'
X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'	X'00'

Примечание. [a] единственный элемент, прекращающий проверку данных и вызывающий запись своего содержимого в общий регистр 2.

Программа для этого примера показана на рис. 17.3.

Предложение 06001 — предложение DC — строит таблицу перекодировки из 256 байтов, каждый из которых содержит X'00'. В данном случае программист счел целесообразным заполнить таким образом всю таблицу и затем отдельной командой переслать конфигурацию X'01' в выделенный байт.

Предложение 06008 (INITL) — команда MVI — помещает шестнадцатеричное число 01 во второй элемент таблицы перекодировки.

Предложение 06009 представляет собой команду перехода с возвратом (BAL) к подпрограмме, которая должна передать основной программе записи, содержащие сообщения. Получившая управление подпрограмма будет использовать пробелы для очистки области с меткой RECAREA, поместит в эту область

IBM

IBM System 360 Assembler Coding Form

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 1

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				060000
TRANSTAB	DC	256XL1'00'		060001
RECAREA	DC	CL80' '		060002
DUMMY	DC	CL80' '		060003
LENGTH	DC	CL1' '		060004
MSGLENG	DC	CL1' '		060005
MSG	DC	CL80' '		060006
*				060007
INITL	MVI	TRANSTAB+1,X'01'		060008
NEWONE	BAL	6,GETRECD		060009
	LA	7,TRANSTAB		060010
	LA	8,RECAREA		060011
NEXTMSG	TRT	0(80,8),0(7)		060012
	BC	8,NEWONE		060013
	LR	10,1		060014
	SR	10,8		060015
	SH	10,=H'1'		060016
	STC	10,LENGTH		060017
	MVC	MOVER+1(1),LENGTH		060018
MOVER	MVC	MSG(1),0(8)		060019
	BAL	6,ROUTINEB		060020
	LA	8,1(1)		060021
	BC	15,NEXTMSG		060022
*				060023

Рис. 17.3.

исходную запись и затем передаст управление предложению 06010 основной программы.

Предложение 06010 загружает в общий регистр 7 адрес первого байта таблицы перекодировки TRANSTAB.

Предложение 06011 загружает общий регистр 8 адресом первого байта рабочей области, служащей для ввода записи. Этот адрес должен быть также и адресом первого байта первого сообщения внутри записи. Во время выполнения данной программы содержимое общего регистра 8 увеличивается так, чтобы содержать адрес начала следующего сообщения внутри записи. Но каждый раз, когда программе передается новая запись, выполняется это предложение, и общий регистр 8 заново загружается так, чтобы содержать адрес начала рабочей области для записи.

Предложение 06012 представляет собой команду TRT. Как указано в первом операнде, она начинает обработку по адресу, находящемуся в общем регистре 8, и заканчивает ее после проверки 80 байтов или после того, как встретится байт, которому соответствует элемент таблицы, не равный нулю. Второй операнд задает адрес таблицы перекодировки, которая содержит байты, используемые для проверки байтов данных. После выполнения этого предложения признак результата будет показывать, встретился ли ненулевой байт в таблице перекодировки. Вместо того чтобы учитывать число уже обработанных байтов области записи, программист задал дополнительную область, расположенную непосредственно за рабочей областью для записи. При этом нет необходимости проверять, было ли предыдущее обработанное сообщение последним в текущей записи.

Как только обработано последнее сообщение в записи, команда TRT выполняется снова. На этот раз не встретится ни одного ненулевого элемента, так как общий регистр 8 содержит адрес поля пробелов длиной 80 байтов. Программист предусмотрел переход к программе, которая получает новую содержащую сообщения запись, если признак результата показывает, что в процессе выполнения команды TRT не было встречено ненулевых элементов.

Предложение 06013 — это переход, о котором только что говорилось. Если при проверке командой TRT 80 байтов в таблице перекодировки не встретилось ни одного ненулевого байта, совершается переход для того, чтобы получить новую группу сообщений. Если же в результате проверки какого-то байта в таблице был найден ненулевой байт, переход не производится и сохраняется естественный порядок следования команд, согласно которому выполняется предложение 06014.

Предложение 06014 загружает содержимое общего регистра 1 в общий регистр 10. В этот момент в силу того, что

команда TRT прекратила выполняться, обнаружив ненулевой элемент в таблице перекодировки, общий регистр 1 содержит адрес байта данных, соответствующего этому ненулевому байту. Этот адрес будет теперь содержаться как в регистре 1, так и в регистре 10.

Предложение 06015 вычитает из величины, содержащейся в общем регистре 10, величину, содержащуюся в общем регистре 8, причем разность остается в регистре 10. Назначение этого предложения — определить количество байтов, находящихся между адресом, с которого начала обработку команда TRT (он представлен в регистре 8), и адресом, на котором она окончила обработку (он представлен в регистре 10). Получающаяся разность — это общее количество байтов правильных данных, составляющих сообщение, которое должно быть извлечено, не считая разделителя сообщений X'01'.

Предложение 06016 вычитает литеральную величину с фиксированной точкой +1 длиной в полуслово из содержимого общего регистра 10. Назначение этого предложения — уменьшить указатель длины сообщения, которое извлекается в данный момент. Общий регистр 10 теперь содержит указатель длины этого сообщения в форме, пригодной для использования в машинной команде, т. е. на 1 байт меньший действительной физической длины.

Предложение 06017 записывает младший байт общего регистра 10 в некоторый байт памяти. Теперь этот байт с меткой LENGTH содержит указатель длины текущего сообщения, которое нужно выделить из записи.

Предложение 06018 пересылает указатель длины текущего сообщения (содержимое поля LENGTH) в предложение 06019, изменяя количество данных, пересылаемых командой MVC.

Предложение 06019 извлекает сегмент текущего сообщения, на который указывает во втором операнде регистр 8, и пересылает его в область памяти с меткой MSG. Количество байтов, которые должны быть перемещены, зависит от указателя длины, внесенного в это предложение предложением 06018.

Предложение 06020 — это переход с возвратом к другой подпрограмме. Подразумевается, что эта подпрограмма обработает сообщение, извлеченное предложением 06019, и затем через предложение 06021 вернет управление основной программе.

Предложение 06021 загружает увеличенное на единицу содержимое общего регистра 1 в общий регистр 8. Допустим, что содержимое регистра 1 не изменилось и не было разрушено подпрограммой ROUTINEB, тогда этот регистр указывает на байт-разделитель сообщений, отмеченный предшествующей командой TRT. Для того чтобы получить адрес следующего за

разделителем байта сообщения, необходимо увеличить содержимое общего регистра 1 на +1. Эта функция выполняется данным предложением, в результате чего в регистре 8 содержится адрес байта памяти, непосредственно следующего за последним из встретившихся разделителей сообщений.

Предложение 06022 представляет собой безусловный переход к предложению с меткой NEXTMSG. Обрабатывающая программа выполняет команду TRT, в результате чего:

1) встречает другой разделитель сообщений и продолжает обработку до предложения 06022 или

2) на протяжении всех 80 байтов, адресованных регистром 8, не встречает разделителя сообщений, так что управление переходит к предложению 06009 для получения новой записи.

Команда TRT установит особое значение признака результата, если для последнего байта поля первого операнда в таблице перекодировки обнаружится ненулевой байт. В предыдущем примере, однако, длина данных, содержащихся в записях, была переменной, поэтому программист составил программу так, чтобы обойти эту проблему, как было показано при описании предложения 06012.

Другой актуальный пример на метод проверки, помимо команд перекодировки, использует стандартную программу, которая проверяет довольно длинное поле, состоящее почти из всех возможных алфавитно-цифровых и специальных символов. От этой программы требуется, чтобы после окончания проверки данного поля оно содержало только алфавитно-цифровые символы и дефисы, но дефисы могут встречаться только между цифровыми символами.

Сначала подпрограмма должна просматривать каждый байт отдельно. Если в нем содержится символ, отличный от алфавитно-цифрового, дефиса или знака @, то этот байт должен быть заменен на дефис. На следующем этапе программа опять просматривает все поле байт за байтом и ликвидирует все те дефисы, которые не находятся между двумя цифрами. Затем поле данных уплотняется путем сдвига байтов влево на место устраняемых дефисов. После того как дефисы останутся только между цифрами и поле уплотнится влево за счет устраненных дефисов, обрабатываются символы @. На этом этапе необходимо учитывать максимальное число байтов данных, которые могут поместиться в данном поле, а также число байтов, размещенных в данном поле в текущий момент. Каждый раз, когда символ @ обнаруживается в поле, текущее число байтов данных сравнивается с максимально допустимым числом, чтобы определить, заполнено ли поле. Если поле не заполнено, все байты справа от @ сдвигаются на 1 байт вправо. Символ @ и примыкающий к нему справа байт заменяются на сочетание

букв AT. Текущее число байтов в поле увеличивается на единицу, и поиск следующих символов @ продолжается. Если обнаружен символ @, а в поле невозможно добавить еще байт данных, то не производится никаких изменений.

Несмотря на то что только что описанная стандартная программа могла показаться несколько искусственной, ее действительно составили и применили в конкретном случае. Общая программа проверки содержала в действительности несколько больше этапов, не включенных в настоящее описание из-за своей сложности. Примеры задач, приводимые ниже, не столь сложны, но они отражают некоторые методы преобразования и обеспечения правильности данных, отличающиеся от методов, использующих команды перекодировки.

Задача 1. Подпрограмма в этой задаче обрабатывает поле смешанных данных длиной 200 байтов. Предполагается, что эта подпрограмма проверяет, является ли каждый восьмой байт данных, начиная с первого байта, правильной буквой в коде EBCDIC в диапазоне от A до I включительно. При обнаружении неправильного символа он должен заменяться на букву Z в коде EBCDIC (рис. 17.4).

Предложение 02406 пересылает 200 байтов данных из поля INFIELD в 200-байтовую рабочую область с меткой AUDITFLD.

Предложение 02407 — команда LA — помещает величину с фиксированной точкой +25 в общий регистр 5. Регистр 5 управляет циклом при продвижении по полю AUDITFLD; поле имеет длину 200 байтов и содержит 25 восьмибайтовых элементов.

Предложение 02408 загружает в общий регистр 6 адрес первого байта поля AUDITFLD. Регистр 6 будет использован как базовый регистр для продвижения по полю AUDITFLD.

Предложение 02409 — первая команда в цикле BIGLOOP — загружает величину с фиксированной точкой +9 в общий регистр 7. Регистр 7 управляет циклом при прохождении через 9 байтов поля ALPHAS. Цикл BIGLOOP состоит из всех предложений с 02409 по 02417 включительно.

Предложение 02410 загружает в общий регистр 8 адрес первого байта поля ALPHAS. Регистр 8 используется как базовый регистр для продвижения по 9 байтам поля ALPHAS по байту за один раз.

Предложение 02411 сравнивает 1 байт поля ALPHAS с 1 байтом поля AUDITFLD. Каждый раз, когда эта команда получает управление вслед за выполнением предложения 02410, код 0(1,6) порождает адрес первого символа в таблице ALPHAS. Всякий раз, когда выполнение этой команды является

IBM System/360 Assembler Coding Form

X28-6509 3 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 1 OF 1

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification Sequence
*				00200
AUDITFLD	DC	CL200' '		00201
ALPHAS	DC	CL9'ABCDEFGHI'		00202
*				00203
*				00204
*				00205
AUDIT8	MVC	AUDITFLD,INFIELD		00206
	LA	5,25		00207
	LA	6,AUDITFLD		00208
BIGLOOP	LA	7,9		00209
	LA	8,ALPHAS		00210
LITTLOOP	CLC	0(1,6),0(8)		00211
	BC	8,OKALPHA		00212
	AH	8,=H'1'		00213
	BCT	7,LITTLOOP		00214
	MVI	0(6),C'Z'		00215
OKALPHA	AH	6,=H'8'		00216
	BCT	5,BIGLOOP		00217
	BC	15,GETDATA		00218
*				00219
*				00220

Рис. 17.4.

результатом перехода от предложения 02414, код 0(1,6) порождает адрес записанного в коде EBCDIC символа поля ALPHAS, расположенного справа от только что проверенного. Цикл LITTLOOP состоит из всех предложений с 02411 по 02414 включительно.

Предложение 02412 представляет собой команду условного перехода. Если байт поля AUDITFLD, адрес которого находится в регистре 6, содержит тот же самый символ, что и байт поля ALPHAS, адрес которого находится в регистре 8, производится переход к предложению OKALPHA. Если эти 2 байта не содержат идентичных символов, то сохраняется естественный порядок выполнения команд, согласно которому выполняется предложение 02413.

Предложение 02413 увеличивает адрес в общем регистре 8, складывая его с литеральной константой с фиксированной точкой +1 длиной в полуслово. Теперь общий регистр 8 содержит адрес байта поля ALPHAS, следующего за только что проверенным предложением 02411. Если этот цикл выполняется все девять раз, то при прохождении через цикл последний раз общий регистр 8 содержит адрес байта, примыкающего к полю ALPHAS справа. Это не имеет никакого значения, так как при выполнении предложения 02414 в этом случае сохраняется естественный порядок следования команд и сравнение, в процессе которого использовался бы этот адрес, не произойдет.

Предложение 02414 — это команда BCT, управляющая циклом LITTLOOP. Переход к LITTLOOP производится до тех пор, пока содержимое регистра 7 не будет уменьшено до нуля или пока не произойдет условный переход при выполнении предложения 02412.

Предложение 02415, используя команду MVI, пересылает букву Z в коде EBCDIC в поле AUDITFLD по адресу, который содержится в общем регистре 6. То обстоятельство, что это предложение выполняется, означает, что байт поля AUDITFLD, проверяемый в настоящий момент, не содержит ни одной буквы в коде EBCDIC, содержащейся в константе ALPHAS.

Предложение 02416 складывает литеральную константу с фиксированной точкой +8 длиной в полуслово с текущим адресом, содержащимся в общем регистре 6. Теперь регистр 6 содержит адрес следующего байта поля AUDITFLD, который должен быть проверен Это предложение выполняется или вслед за предложением 02415, или после перехода от предложения 02412.

Предложение 02417 — это команда BCT, управляющая циклом BIGLOOP. Переход к BIGLOOP будет иметь место до тех пор, пока содержимое общего регистра 5 не станет равным нулю, после чего управление перейдет к предложению 02418.

Предложение 02418 — безусловный переход к программе с меткой GETDATA, которая не приведена в примере, но которая, как предполагается, состоит из набора команд, необходимых для получения новой группы данных и вторичного входа в подпрограмму AUDIT8.

Когда к проверке готова новая группа из 200 байтов данных, управление получает подпрограмма AUDIT8. Цикл BIGLOOP проверяет каждый восьмой байт поля AUDITFLD, состоящего из 200 байтов. Для каждого проверяемого байта поля AUDITFLD должен осуществляться переход к LITTLOOP. Всякий раз, когда происходит продвижение по циклу LITTLOOP, текущий байт поля AUDITFLD сравнивается с таблицей правильных алфавитных символов в поисках совпадения. Обработка выходит из цикла LITTLOOP, когда в поле ALPHAS обнаруживается символ, соответствующий содержимому текущего байта поля AUDITFLD, или когда просмотр таблицы ALPHAS заканчивается без обнаружения совпадающей пары байтов. Последнее означает, что текущий символ в поле AUDITFLD неправильный и программа заменит его на символ Z.

Задача 2. В этой задаче требуется, чтобы программа в непрерывно входящем потоке данных находила некоторые специальные символы. Входные данные передаются программе от удаленного терминала по запросам. Терминал передает данные 80-байтовыми сегментами с запоминающего устройства. В этих данных встречаются два специальных однобайтовых символа — X'37' и X'26'. Байт с шестнадцатеричным представлением 26 — это символ ОКОНЧАНИЕ БЛОКА; байт с шестнадцатеричным представлением 37 — это символ КОНЕЦ ПЕРЕДАЧИ. Хотя данные передаются проблемной программе 80-байтовыми сегментами, они накапливаются до тех пор, пока не встретится символ X'26'. В этот момент программа передает накопленные данные программе вывода, которая помещает записи переменной длины в запоминающее устройство прямого доступа. Программа должна подсчитать количество байтов в каждой выходной записи и передать это число в двух старших байтах четырехбайтового поля, которое непосредственно предшествует самой записи. Если встречается байт X'37', это означает, что теперь в программу переданы все имеющиеся данные, и выполнение подпрограммы вывода может быть закончено. Эта подпрограмма передает записи данных переменной длины, хранимые в какой-то другой ячейке 80-байтовыми сегментами, причем любой сегмент может содержать логическое окончание одной записи и начало следующей.

Прежде чем анализировать эту программу, следует отметить, что вовсе необязательно она представляет оптимальный способ выполнения требуемой задачи. Но все же она в какой-то мере помогает объяснению логики процесса редактирования этого типа.

На рис. 17.5а—17.5г приведены предложения, задающие поля, и подпрограммы, которые предназначены для выполнения этой задачи.

Предложение 03004 (PRESET) встречается только один раз при каждом выполнении программы. Функции предложений 03004 и 03005 берут на себя затем предложения 03204 и 03205 в подпрограмме BLOCKEND. Это предложение загружает в общий регистр 3 адрес первого байта данных области вывода записи. Следует обратить внимание, что общий регистр 3 назван R3; это возможно потому, что предложения с 05003 по 05007 определили для некоторых из общих регистров символические имена.

Предложение 03005 — команда LA — устанавливает регистр 7 в нуль. То же самое можно сделать с помощью команды SR 7,7.

Предложение 03006 (NEWSEGMT) — это переход с возвратом к подпрограмме, которая выполняет операцию ввода-вывода, необходимую для занесения в поле SEGFIELD 80 байтов входных данных. После получения данных управление возвращается к предложению 03007, потому что этот адрес загружен в общий регистр 6 до перехода к подпрограмме GETSEGMT. Это предложение получает управление от предложения 03020 всякий раз, когда исчерпывается очередной сегмент длиной 80 байтов.

Предложение 03007 загружает в общий регистр 4 величину с фиксированной точкой +80. Регистр 4 служит для управления циклом при побайтном продвижении по сегменту длиной 80 байтов.

Предложение 03008 загружает адрес первого байта поля SEGFIELD в общий регистр 5. Каждый раз, когда поступает новый сегмент, этот регистр устанавливается заново так, чтобы содержать адрес первого байта.

Предложение 03009 сравнивает 1 байт в поле SEGFIELD, адрес которого содержится в общем регистре 5, с шестнадцатеричным значением 26. Этим проверяется наличие символа КОНЕЦ БЛОКА (End of Block — EOB).

Предложение 03010 — это условный переход. Если при выполнении предложения 03009 обнаружено условие равенства, произойдет переход к программе BLOCKEND.

Предложение 03011 сравнивает 1 байт поля SEGFIELD, адрес которого находится в общем регистре 5, с шестнадцате-

IBM

IBM System/360 Assembler Coding Form

X28-6508-2 U/M068
Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 4 | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
*				03001
*		* * MAIN-LINE AUDITING LOGIC * *		03002
*				03003
PRESET	LA	R3,OUTRECRD		03004
	LA	R7,0		03005
NEWSEGMT	BAL	R6,GETSEGMT		03006
	LA	R4,80		03007
	LA	R5,SEGFIELD		03008
SEGLOOP	CLI	0(5),X'26'		03009
	BC	8,BLOCKEND		03010
	CLI	0(5),X'37'		03011
	BC	8,ALLDONE		03012
	MVC	0(1,3),0(5)		03013
	AH	R5,=H'1'		03014
	AH	R3,=H'1'		03015
	AH	R7,=H'1'		03016
	C	R7,=F'500'		03017
	BC	2,ERROR		03018
	BCT	R4,SEGLOOP		03019
	BC	15,NEWSEGMT		03020
*				03021
*				03022

Рис. 17.5а

IBM

IBM System/360 Assembler Coding Form

X28-6509-2 U/M050
Printed in U.S.A.

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 4

PROGRAMMER | DATE | CARD FORM #

STATEMENT

Name	Operation	Operand	Comments	Sequence
*			ROUTINE FOR END-OF-OUTPUT-BLOCK CONDITION	03200
*				03201
BLOCKEND	STH	R7,PREREC		03202
	BAL	R6,PUTRECRD		03203
	LA	R3,OUTRECRD		03204
	LA	R7,0		03205
	AH	R5,=H'1'		03206
	BCT	R4,SEGLOOP		03207
	BC	15,NEWSEGMT		03208
*				03209
*				03400
ALLDONE			(ASSUMED TO CONTAIN THE	03401
			CLOSING ROUTINES OF THE	03402
			PROBLEM PROGRAM.)	03403
*				03404
*				03405
*				03600
GETSEGMT			(ASSUMED TO CONTAIN THE	03601
			STATEMENTS NECESSARY TO	03602
			BRING IN A NEW 80-BYTE	03603
			DATA SEGMENT)	03604
				03605
	BCR	15,R6		03606
*				03607

Рис. 17.5б;

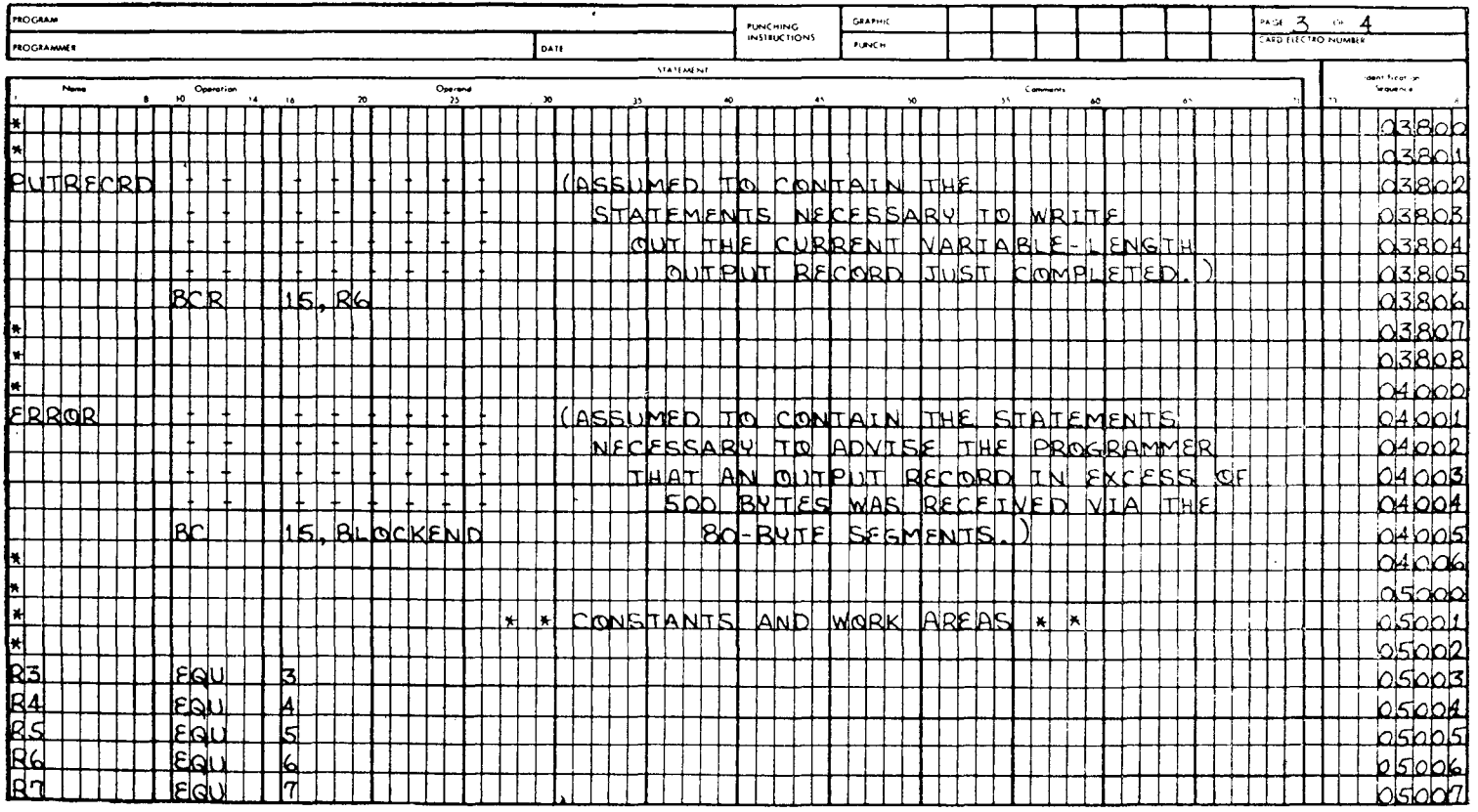

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 3 OF 4
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification Sequence
* (ASSUMED TO CONTAIN THE 03800
* STATEMENTS NECESSARY TO WRITE 03801
OUTRECRD OUT THE CURRENT VARIABLE-LENGTH 03802
OUTPUT RECORD JUST COMPLETED.) 03803
BCR 15,R6 03804
* 03805
* 03806
* 04000
ERROR (ASSUMED TO CONTAIN THE STATEMENTS 04001
NECESSARY TO ADVISE THE PROGRAMMER 04002
THAT AN OUTPUT RECORD IN EXCESS OF 04003
500 BYTES WAS RECEIVED VIA THE 04004
BC 15,BLOCKEND 80-BYTE SEGMENTS.) 04005
* 04006
* 05000
* * CONSTANTS AND WORK AREAS * * 05001
* 05002
R3 EQU 3 05003
R4 EQU 4 05004
R5 EQU 5 05005
R6 EQU 6 05006
R7 EQU 7 05007

Рис. 17.5в.

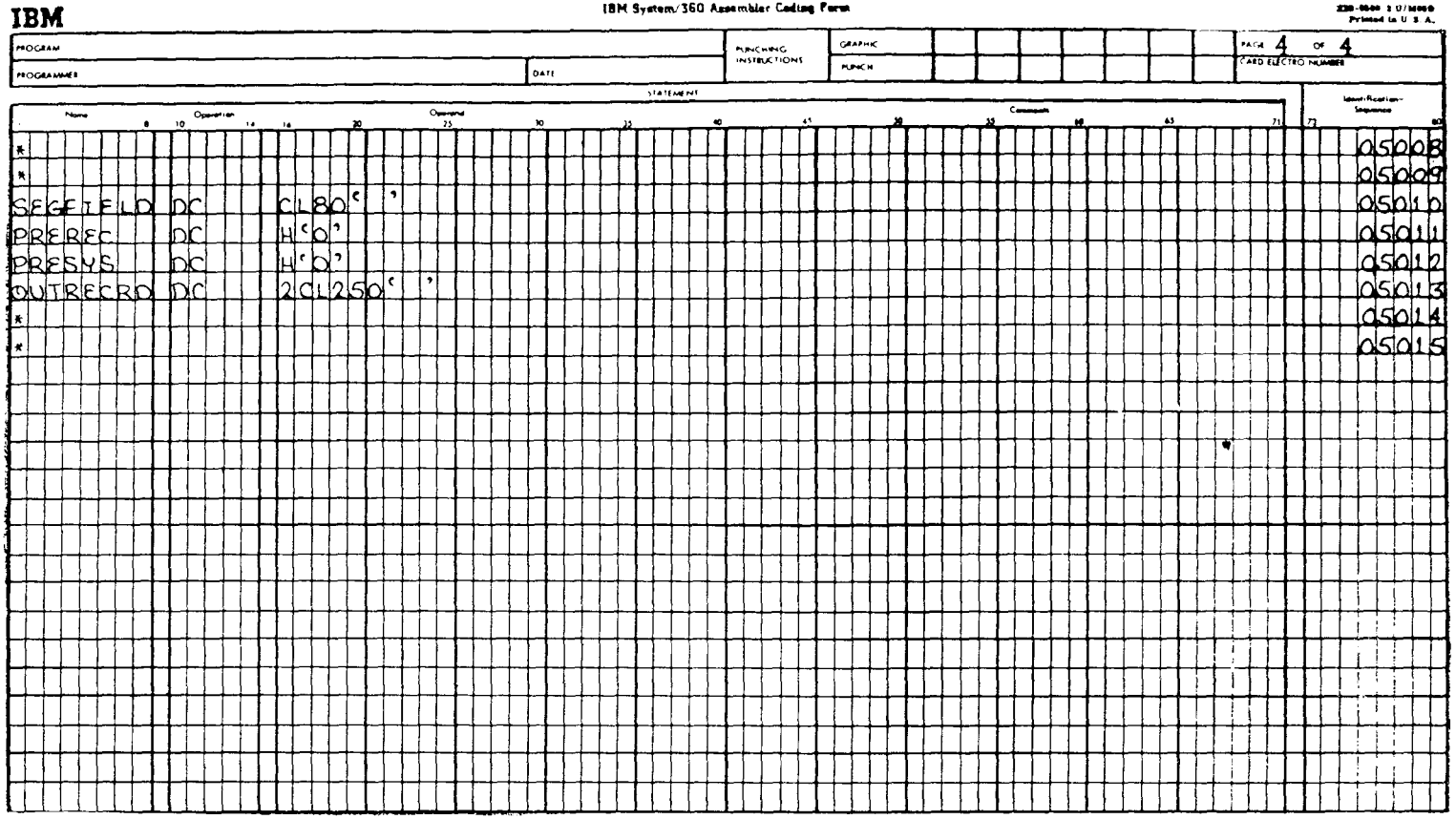

IBM
IBM System/360 Assembler Coding Form
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
PAGE 4 OF 4
CARD ELECTRO NUMBER
STATEMENT
Name
Operation
Operand
Comments
Identification-Sequence
* 05008
* 05009
SEGFIELD DC CL80' ' 05010
PREREC DC H'0' 05011
PRESYS DC H'0' 05012
OUTRECRD DC 2CL250' ' 05013
* 05014
* 05015

Рис. 17.5г.

ричным значением 37 — символом ОКОНЧАНИЕ ПЕРЕДАЧИ (End of Task — EOT).

Предложение 03012 вызывает переход к ALLDONE, если в результате выполнения предложения 03011 обнаружено условие равенства, указывающее, что переданы все доступные данные.

Предложение 03013 пересылает 1 байт данных из исходного сегмента в область, в которой формируется выходная запись. Этот байт пересылается, так как было установлено, что он не является индикатором окончания блока (EOB) или окончания передачи (EOT). Общий регистр 5 содержит соответствующий адрес в поле SEGFIELD, а общий регистр 3 — адрес в поле OUTRECRD.

Предложение 03014 складывает величину с фиксированной точкой +1 с адресом, содержащимся в общем регистре 5. Теперь этот регистр содержит адрес следующего байта в поле SEGFIELD.

Предложение 03015 складывает величину с фиксированной точкой +1 с адресом в общем регистре 3. Теперь этот регистр содержит адрес следующего байта в поле OUTRECRD, который должен получить байт данных из поля SEGFIELD.

Предложение 03016 складывает величину с фиксированной точкой +1 с содержимым общего регистра 7. Эта величина указывает количество байтов данных, которые уже помещены в выходную запись, формируемую в поле OUTRECRD.

Предложение 03017 сравнивает величину с фиксированной точкой в общем регистре 7 с литералом с фиксированной точкой +500 длиной в полное слово. Это делается для того, чтобы определить, передано ли в поле OUTRECRD 500 байтов данных (максимально допустимое количество).

Предложение 03018 выполняет переход к программе с меткой ERROR, если в результате предыдущего сравнения установлено, что общий регистр 7 содержит величину, большую +500. Это условие указывает на то, что поле OUTRECRD длиной 500 байтов уже заполнено.

Предложение 03019 — команда BCT — осуществляет управление циклом. Каждый раз, когда выполняется команда BCT, величина в общем регистре 4 уменьшается на единицу, и каждый раз совершается переход к SEGLOOP до тех пор, пока величина в общем регистре 4 не уменьшится до нуля. В этот момент перехода не произойдет, и в соответствии с естественным порядком следования команд будет выполняться предложение 03020. Эта команда обеспечивает вход в цикл SEGLOOP максимальное количество раз — 80 (по числу байтов в каждом сегменте данных).

Предложение 03020 — безусловный переход к NEWSEGMT. Это предложение выполняется в том случае, если предыдущая команда BCT установила, что обработан весь текущий сегмент и необходимо запросить следующий сегмент данных.

Предложение 03202 (BLOCKEND) — первая команда в подпрограмме, которая должна обеспечить вывод сформированной записи переменной длины. Предложение заносит число, равное длине записи в байтах, находящееся в двух младших байтах общего регистра 7, в первое из двух полуслов, непосредственно предшествующих полю OUTRECRD.

Предложение 03203 — команда перехода с возвратом к подпрограмме, которая фактически выполняет операцию ввода-вывода для записи выходных данных OUTRECRD. Она помещает адрес предложения 03204 в общий регистр 6 и затем переходит к PUTRECRD. После завершения подпрограммы ввода-вывода происходит возврат к предложению 03204.

Предложение 03204 вторично загружает общий регистр 3 с тем, чтобы он снова содержал адрес первого байта поля OUTRECRD, подготавливая формирование новой выходной записи.

Предложение 03205 вновь загружает нули в регистр 7. Регистр 7 используется для того, чтобы подсчитывать общее количество байтов, переданных в запись, формируемую в поле OUTRECRD.

Предложение 03206 складывает величину +1 с адресом в общем регистре 5. Когда управление получил этот набор команд, регистр 5 в соответствии с алгоритмом указывал на байт в поле SEGFIELD, содержащий X'26'. Так как эта подпрограмма передает управление непосредственно или предложению NEWSEGMT, или предложению SEGLOOP, минуя предложение 03014, то необходимо модифицировать регистр 5 так, чтобы он содержал адрес байта, непосредственно следующего за байтом, который содержит X'26' и указывает на окончание блока (EOB).

Предложение 03207 — команда BCT — использует тот же регистр и выполняет ту же функцию, что и предложение 03019. Оно закодировано здесь потому, что после передачи управления от этой подпрограммы предложение 03019 не выполняется, и необходимо уменьшить величину в общем регистре 4, чтобы определить, обработаны ли все 80 байтов текущего сегмента. Если байт, содержащий X'26', который был причиной перехода к этой подпрограмме, является 80-м байтом текущего сегмента, то в результате выполнения команды происходит уменьшение до нуля содержимого общего регистра 4 и выполнение команд продолжается в их естественной последовательности, т. е. переход не происходит, а выполняется предложение 03208.

Предложение 03208 — безусловный переход к NEWSEGMT. Если данное предложение получает управление, это означает, что текущий сегмент данных окончился и управление должно перейти к предложению, которое обеспечит пересылку в память нового сегмента.

Предложение 03401 (ALLDONE) — первое из набора предложений, которые закрывают файлы и выполняют все другие функции, необходимые для нормального завершения программы.

Предложение 03601 (GETSEGMT) — первая команда набора команд, организующих операцию ввода-вывода для фактического получения записи данных с терминала.

Предложение 03606 — безусловный переход по адресу, содержащемуся в общем регистре 6. На данной фазе программы этот регистр должен содержать адрес предложения 03007 — команды, непосредственно следующей за командой BAL, передающей управление подпрограмме GETSEGMT.

Предложение 03802 — первая команда стандартной подпрограммы, которая управляет операцией ввода-вывода для вывода записи, сформированной в поле OUTRECRD. Так как считается, что записи имеют переменную длину, для операции ввода-вывода будет указано, что выходная запись имеет длину 504 байта: 500 байтов — длина записи данных и 4 байта — два предшествующих полуслова.

Предложение 03806 — безусловный переход по адресу, содержащемуся в общем регистре 6. Этот регистр должен содержать адрес предложения 03204, непосредственно следующего за командой BAL, передающей управление предложению 03802.

Предложение 04001 — первое в подпрограмме, задача которой принять решение о дальнейших действиях в случае, если в области выходной записи не встречается индикатора окончания блока (EOB). Алгоритм программы может предусматривать команды, необходимые для вывода сообщения на пультовую пишущую машинку для уведомления программиста или оператора. С другой стороны, может быть поставлена задача только подсчитывать количество записей приведенного здесь вида, получаемых программой. Алгоритм данной программы таков, что, если область вывода записи заполняется, законченная запись выводится в результате перехода к программе BLOCKEND, а затем продолжается обработка любых оставшихся байтов как новой записи.

Предложение 04005 — это безусловный переход к только что описанной программе BLOCKEND.

Предложения от 05003 до 05007 сопоставляют некоторые общие регистры с символическими именами. Поэтому любые из этих пяти общих регистров обозначаются или символическими именами (как, например, R4), или просто числом (4).

Предложение 05010 — это предложение DC, формирующее 80-байтовое поле пробелов. Каждый 80-байтовый сегмент данных по мере запроса его программой помещается операцией ввода-вывода в эту область.

Предложения с 05011 по 05013 задают максимальную область (500 байтов, перед которыми находится четырехбайтовый счетчик) для выходной записи переменной длины.

Предложение 05011 определяет константу длиной в полуслово, содержащую нуль в формате с фиксированной точкой. Длина каждой сформированной записи извлекается из общего регистра 7 и помещается в это полуслово, как показано в предложении 03202.

Предложение 05012 определяет вторую константу длиной в полуслово, содержащую нули. Эти 2 байта используются системой при формировании выходных записей переменной длины.

Предложение 05013 формирует 500-байтовую цепочку пробелов, указывая для 250-байтовой константы коэффициент кратности, равный 2. Эта константа образует поле выходной записи.

Этот набор программ в большей степени, чем программы в большинстве предшествующих примеров, выполняет вспомогательные действия. Эти программы показаны неполностью, многие предложения опущены, в частности предложения для задания базовых регистров, открытия наборов данных, макрокоманды ввода и вывода соответственно входных и выходных записей, закрытия наборов данных, начала и завершения выполнения служебных функций и многие другие. Эти предложения были опущены сознательно в силу большого разнообразия средств выражения этих функций в различных операционных системах. Необходимые знания программист получит на вычислительном центре, где он работает. Для операционной системы OS примеры использования этих функций приведены в гл. 5.

Хотя в двух последних задачах этого раздела не использовались команды перекодировки, следует помнить о том, что использование этих команд в подобных задачах было бы очень удобным и полезным. Программист сам выберет предпочтительный способ кодировки. Можно надеяться, что он независимо от того, как много усилий ему понадобится на кодирование, изберет способ, обеспечивающий наиболее эффективную обработку данных.

Упражнения

1. Выбор байта в таблице перекодировки осуществляется путем прибавления двоичной величины шестнадцатеричного представления байта исходного поля к адресу начала _____.

2. Таблица перекодировки состоит обычно из _____ байтов.

3. Когда команда TRT встречает в таблице перекодировки ненулевой байт, она загружает в общий регистр _____ адрес байта исходных данных, который соответствует этому ненулевому байту.

4. Когда встречается условие, описанное в предыдущем пункте, команда TRT загружает шестнадцатеричное содержимое ненулевого байта таблицы перекодировки в младший байт общего регистра _____.

5. Содержимое отдельного элемента таблицы перекодировки, указанное исходным байтом, замещает содержимое _____.

6. Программист может формировать содержимое таблицы перекодировки любым способом, который отвечает требованиям алгоритма конкретной программы. (Верно) (Неверно)

7. В команде TRT двоичная величина шестнадцатеричной конфигурации байта исходных данных используется как _____ к адресу _____ байта таблицы перекодировки.

8. Второй операнд команды TR представляет собой адрес _____.

9. Команда TRT не изменяет содержимого поля данных, подвергающихся проверке. (Верно) (Неверно)

10. _____ операнд команды TR указывает поле данных, которые должны быть перекодированы.

11. Выполнение команды TRT прекращается, если в таблице перекодировки встречается элемент, содержимое которого отлично от X'_____'.

Все остальные упражнения относятся к перекодировке данных с использованием команды TR. Пользуясь следующей таблицей перекодировки, закодируйте команду для преобразования исходного поля. Затем приведите символьное и шестнадцатеричное представление каждого байта исходного поля, перекодированного в результате выполнения команды TR. Следует помнить, что в приведенной здесь таблице перекодировки определено содержимое каждого элемента; шестнадцатеричное представление каждого из этих элементов не обязательно совпадает с шестнадцатеричным представлением адреса этих байтов.

Ссылка на таблицу перекодировки делается с помощью метки TRANTABL.

Таблица перекодировки

(Содержимое каждого элемента указывается в шестнадцатеричной форме.)

Адреса	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1_	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2_	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3_	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4_	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5_	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6_	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7_	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8_	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9_	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A_	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B_	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C_	C0	C7	D6	C2	E9	D4	E3	C9	E4	D7	CA	CB	CC	CD	CE	CF
D_	D0	D9	F6	C4	F0	F5	C8	F2	C3	C5	DA	DB	DC	DD	DE	DF
E_	E0	E1	D3	F3	F1	D1	E2	E5	D8	D2	EA	EB	EC	ED	EE	EF
F_	F4	D5	C6	F7	E6	F8	C1	E8	F9	E7	FA	FB	FC	FD	FE	FF

12.

DATAFLDA (Символьн.)

F		H		E		C		S		R	
C	6	C	8	C	5	C	3	E	2	D	9

(Шестн.)

TRANSA TR DATAFLDA(6), TRANTABL

DATAFLDA (Шестн.)

(Символьн.)

13.

DATAFLDB (Символьн.)	P		J		B		C		S		R		M
(Шестн.)	D	7	D	1	C	2	C	3	E	2	D	9	D	4

```
TRANSB      TR      DATAFLDB + 1(5), TRANTABL
```

DATAFLDB (Шестн.)													
(Символьн.)													

14.

DATAFLDC (Символьн.)	S		B		1		A		V		B		C
(Шестн.)	E	2	C	2	F	1	C	1	E	5	C	2	C	3

```
TRANSC      LA      8,TRANTABL
            TR      DATAFLDC(7),0(8)
```

DATAFLDC (Шестн.)													
(Символьн.)													

15.

DATAFLDD (Символьн.)	J		G		A		0		F	
(Шестн.)	D	1	C	7	C	1	D	6	C	6

```
            LA      11,TRANTABL
            TR      DATAFLDD(5),0(11)
```

DATAFLDD (Шестн.)										
(Символьн.)										

16.

DATAFLDE (Символьн.)	U		K		(пробел)		4		R		W		F
(Шестн.)	E	4	D	2	4	0	F	4	D	9	E	6	C	6

```
TRANSE      TR      DATAFLDE(7), TRANTABL
```

DATAFLDE (Шестн.)													
(Символьн.)													

17.

DATAFLDF												
(Символьн.)	A		P		A		U		A		4	
(Шестн.)	C	1	D	7	C	1	E	4	C	1	F	4

DATAFLDG										
(Символьн.)	6		F		9		Z		R	
(Шестн.)	F	6	C	6	F	9	E	9	D	9

```
TRANSRTE    LA      5,TRANTABL
            TR      DATAFLDF(6),0(5)
            MVC     DATAFLDG(5),DATAFLDF+1
            TR      DATAFLDG(5),0(5)
```

DATAFLDF												
(Шестн.)												
(Символьн.)												

DATAFLDG										
(Шестн.)										
(Символьн.)										

Глава 18

Истолкование шестнадцатеричного дампа основной памяти

Каждая операционная система и каждая разновидность управляющей программы распечатывают содержимое основной памяти не совсем так, как это делается в другой операционной системе или в той же операционной системе другим вариантом управляющей программы. Поэтому в этой книге невозможно исчерпывающе описать процесс истолкования дампа основной памяти и нахождения по нему ошибок в проблемной программе. Чтобы распознать признаки конкретных ошибок, программисту следует обратиться к пособиям по соответствующей операционной системе. В настоящей главе программист не получит такой детальной информации, но она может помочь ему при анализе полей данных, содержащихся в той части распечатки дампа основной памяти, которая относится к проблемной программе.

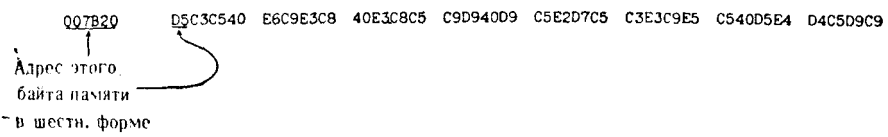
Во-первых, шестнадцатеричные данные в распечатке дампа не описывают сами себя в той мере, как представление этих данных в адекватном их смыслу формате. Шестнадцатеричное представление — это наиболее общее представление любых разновидностей данных; двоичное представление имеет ту же общность, но оно более громоздко. Поэтому необходимо, чтобы при анализе дампа программист понимал значение анализируемых данных, используя их шестнадцатеричное представление. Чтобы получить это адекватное представление, он должен соотнести указанные в распечатке дампа адреса в основной памяти с относительными адресами в распечатке исходного модуля.

В большинстве случаев дамп печатается в виде строк данных, например так:

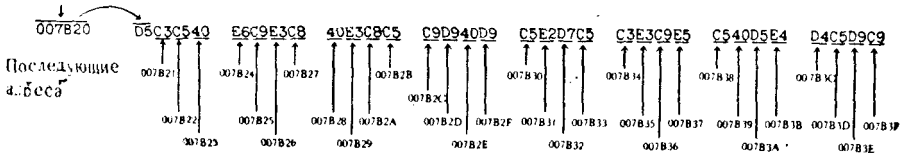
```
007B20   D5C3C540 E6C9E3C8 40E3C8C5 C9D940D9 C5E2D7C5 C3E3C9E5 C540D5E4 D4C5D9C9
```

Шесть левых шестнадцатеричных цифр представляют собой начальный адрес области памяти, содержимое которой напечатано

в данной строке:



Так как система не может определить, какой смысл имеет каждый байт, она печатает его в виде пары шестнадцатеричных цифр. Адрес каждой такой пары шестнадцатеричных цифр на 1 байт (+1) больше, чем адрес предыдущей пары.



В шестнадцатеричной распечатке дампа основной памяти, используемой в данной главе в качестве иллюстраций, на каждой строке представлены 32 однобайтовые области памяти, что в шестнадцатеричной форме представляется 64 цифрами. Шестнадцатеричные цифры сгруппированы по восемь, что составляет 4 байта на группу. Адрес любой пары шестнадцатеричных цифр, представляющих байт памяти, легко можно вычислить, увеличивая (в шестнадцатеричной форме) адрес первой пары шестнадцатеричных цифр в распечатке памяти.

Чтобы ознакомиться с шестнадцатеричным представлением различных типов полей данных, следует изучить рис. 18.1, на котором изображена часть распечатки дампа основной памяти, содержащая строки с адресами от 007A20 до 007CA0. Сначала распечатка интерпретирована путем символьного представления шестнадцатеричных цифр. Следует отметить, что шестнадцатеричное сочетание X'40' — это пробел, и, следовательно, ему не сопоставляется никакой символ.

Начиная с адреса 007B80, имеются девять арифметических полей, которые будут определены позже. По адресам с 007BA0 по 007C11 содержатся дополнительные символьные данные. По адресам с 007C12 по 007C51 содержатся команды программы на машинном языке. Остальная часть распечатки снова содержит символьные данные.

Пусть в данном случае программист задал арифметические поля, как показано следующими предложениями. В момент выполнения данного шестнадцатеричного дампа основной памяти эти поля еще не были изменены.

```
          T H I S    I S    T H E    W A Y    T H A T    A    C O R E    D U M P
007A20    E3C8C9E2 40C9E240 E3C8C540 E6C1E840 E3C8C1E3 40C140C3 D6D9C540 C4E4D4D7

            M I G H T    A P P E A R    I N      H E X    A D E C I M A L    F O R
007A40    40D4C9C7 C8E340C1 D7D7C5C1 D940C9D5 40C8C5E7 C1C4C5C3 C9D4C1D3 40C6D6D9

            M A T          D A T A    A P P E    A R I N G    I N    S T R A I G H
007A60    D4C1E34B 404040C4 C1E3C140 C1D7D7C5 C1D9C9D5 C740C9D5 40E2E3D9 C1C9C7C8

            T    E B C D I C    C H A R A C T    E R    F O R M    C O U L D    B E
007A80    E340C5C2 C3C4C9C3 40C3C8C1 D9C1C3E3 C5D940C6 D6D9D440 C3D6E4D3 C440C2C5

              I N T E R P R E T E D    I N    T H E    S A M E    M A N N E R
007AA0    40C9D5E3 C5D9D7D9 C5E3C5C4 40C9D540 E3C8C540 E2C1D4C5 40D4C1D5 D5C5D940

            A S    T H I S    L I N E    Y O U    A R E    R E A D I N G .
007AC0    C1E240E3 C8C9E240 E3C9D5C5 40E8D6E4 40C1D9C5 40D9C5C1 C4C9D5C7 4B404040

            F I X E D -  P O I N T    A N D    P A C K E D    D E C I M A L    V
007AE0    C6C9E7C5 C460D7D6 C9D5E340 C1D5C440 D7C1C3D2 C5C440C4 C5C3C9D4 C1D340E5

            A L U E S    A R E    I N T E R P    R E T E D    I N    A C C O R D A
007B00    C1D3E4C5 E240C1D9 C540C9D5 E3C5D9D7 D9C5E3C5 C440C9D5 40C1C3C3 D6D9C4C1

            N C E    W I T H    T H E I R    R E S P E C T I V E    N U M E R I
007B20    D5C3C540 E6C9E3C8 40E3C8C5 C9D940D9 C5E2D7C5 C3E3C9E5 C540D5E4 D4C5D9C9

            C    R E P R E S E N T A T I O N    A S    I N D I C A T E D    W I
007B40    C340D9C5 D7D9C5E2 C5D5E3C1 E3C9D6D5 40C1E240 C9D5C4C9 C3C1E3C5 C440E6C9

            T H I N    T H E    F O L L O W I N G    F I E L D S .
007B60    E3C8C9D5 40E3C8C5 40C6D6D3 D3D6E6C9 D5C740C6 C9C5D3C4 E24B4040 40404040

                  #1       #2       #3       #4                 #5          #6  #7   #8  #9
007B80    00000014 539C0013 0003B7A5 0000025D 00000021 3116478C 5C00000C 0555FFFF

            P R O G R A M    I N S T R U C T    I O N    S T A T E M E N T S    A
007BA0    D7D9D6C7 D9C1D440 C9D5E2E3 D9E4C3E3 C9D6D540 E3E3C1E3 C5D4C5D5 E3E240C1

            R E    S H O W N    I N    M A C H    I N E    L A N G U A G E    F O R
007BC0    D9C540E2 C8D6E6D5 40C9D540 D4C1C3C8 C9D5C540 D3C1D5C7 E4C1C7C5 40C6D6D9

            M A T    A S    R E P R E S E N T    E D    B Y    T H E    F O L L O W
007BE0    D4C1E340 C1E240D9 C5D7D9C5 E2C5D5E3 C5C440C2 E840E3C8 C540C6D6 D3D3D6E6

            I N G    E X A M P L E S ,                          #1       #2       #3
007C00    C9D5C740 C5E7C1D4 D7D3C5E2 4B404040 4040D20D A0056E4B 4AA0568A 58806C98

              #4       #5                 #6       #7       #8       #9       #10
007C20    58906C9C FA106CB0 56BFF911 6CB0568C 4770449C 95E76CB3 47804462 92E76CB3

              #11      #12      #13               #14
007C40    41A07B01 4AA0568E F8106CB0 56BA47F0 449C4040 40404040 40404040 40404040

            E A C H    2    H E X    D I G I T S    A P P E A R I N G    I N    T
007C60    C5C1C3C8 40F240C8 C5E740C4 C9C7C9E3 E240C1D7 D7C5C1D9 C9D5C740 C9D540E3

            H E    C O R E    D U M P    R E P    R E S E N T S    1    B Y T E    O
007C80    C8C540C3 D6D9C540 C4E4D4D7 40D9C5D7 D9C5E2C5 D5E3E240 F140C2E8 E3C540D6

            F    C O R E    S T O R A G E .
007CA0    C640C3D6 D9C540E2 E3D6D9C1 C7C54B40 40404040 40404040 40404040 40404040
```

Рис. 18.1.

```
FIELD1    DC    PL6'14539'
FIELD2    DC    H'19'
FIELD3    DC    F'243621'
FIELD4    DC    PL4'-25'
FIELD5    DC    PL8'213116478'
FIELD6    DC    PL1'5'
FIELD7    DC    PL3'0'
FIELD8    DC    H'1365'
FIELD9    DC    H'-1'
```

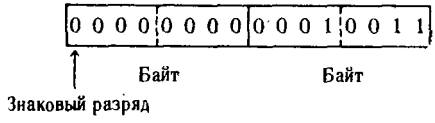
Эти арифметические поля интерпретируются следующим образом:

FIELD1 — адреса с 007B80 по 007B85. Это поле содержит шестибайтовую упакованную десятичную величину +14539. Шестнадцатеричное С в правом полубайте поля свидетельствует, что данная величина положительна, а остальные старшие шестнадцатеричные цифры выражают саму величину.

FIELD2 — адреса с 007B86 по 007B87. Это двухбайтовая величина с фиксированной точкой +19 (длиной в полуслово). В случае двоичных величин и величин с фиксированной точкой шестнадцатеричные цифры не имеют простой связи с цифрами десятичного представления той же величины. С помощью таблицы преобразования чисел из шестнадцатеричной формы в десятичную можно убедиться, что шестнадцатеричная цифра 3 выражает величину +3, а шестнадцатеричная цифра 1, будучи левой цифрой пары, — величину +16, при этом (+16)+(+3)= =(+19).

Так как левый бит поля нулевой, величина в целом будет считаться положительной. Этот бит является старшим битом самой левой шестнадцатеричной цифры 0 в этом поле.

Двоичная конфигурация этого поля выглядит следующим образом:

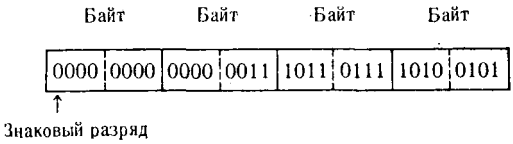


Это поле можно рассматривать как представление двоичной величины без знака. Так как оно расположено на границе полуслова (адрес кратен 2), его также можно рассматривать как представление двоичной величины со знаком длиной в полуслово.

FIELD3 — адреса с 007B88 по 007B8B. Предполагается, что это поле содержит величину с фиксированной точкой со знаком +243621 длиной в полное слово. Оно выравнено по границе

полного слова (его адрес кратен 4). Его также можно ривать как двоичную величину без знака.

Двоичная конфигурация этого поля имеет вид



Если интерпретировать это поле как величину с фиксированной точкой длиной в полное слово, то эта величина была бы положительной — знаковый разряд содержит нуль.

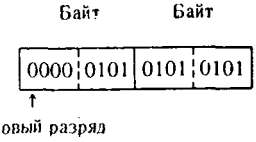
FIELD4 — адреса с 007B8C по 007B8F. Хотя это поле и можно рассматривать как полное слово с фиксированной точкой (так как оно выравнено по границе полного слова) с величиной +605 предположим, что программист задал его как четырехбайтовое упакованное десятичное поле. Поэтому содержимое этого поля можно интерпретировать как упакованную десятичную величину —25. Знак величины выражен шестнадцатеричной цифрой D в младшем полубайте поля.

FIELD5 — адреса 007B90 по 007B97. Это поле содержит восьмибайтовую упакованную десятичную величину +213116478. Младшая шестнадцатеричная цифра C этого поля указывает, что величина является положительной.

FIELD6 — адрес 007B98. Это однобайтовая упакованная десятичная величина +5. Если бы программист не задал этот единственный байт как упакованное десятичное поле, эту конфигурацию можно было бы интерпретировать как символ. В этом случае шестнадцатеричная величина 5C является представлением символа «звездочка» в коде EBCDIC.

FIELD7 — адреса с 007B99 по 007B9B. Программист задал это поле как упакованное десятичное. Его можно интерпретировать как упакованную десятичную величину +0.

FIELD8 — адреса с 007B9C по 007B9D. Так как программист задал это поле как величину с фиксированной точкой длиной в полуслово, здесь оно интерпретируется как содержащее целое число +1365. Битовая конфигурация поля выглядит следующим образом:



Знаковый бит, установленный в нуль, свидетельствует о том, что данная величина является положительной. То, что поле содержит величину +1365, можно проверить, суммируя значе-

ния (степени числа 2) двоичных разрядов, содержащих единицу, или используя таблицу преобразования шестнадцатеричных чисел в десятичные.

FIELD9 — адреса с 007B9E по 007B9F. Это поле программист также задал как величину с фиксированной точкой длиной в полуслово. В данном примере содержимое полуслова равно —1. Битовая конфигурация выглядит следующим образом:

Установленный в единицу знаковый разряд указывает на то, что эта величина с фиксированной точкой является отрицательной величиной в форме дополнения до 2. Если программист по ошибке решит, что это поле содержит двоичную величину без знака, а не величину с фиксированной точкой, то при такой интерпретации это поле содержит целое число 65535.

Команды, представленные на этой распечатке дампа основной памяти, начиная с адреса 007C12, были закодированы в проблемной программе следующим образом:

```
MVC   0+5(14,10),BOLE     001
AH    R10,=H'133'         002
L     R8,HOLDP8           003
L     R9,HOLDP9           004
AP    PECON,=PL1'1'       005
CP    PECON,=PL2'030'     006
BC    7,NOTWANTA          007
CLI   MINT2,C'X'          008
BC    8,TAP60             009
MVI   MINT2,C'X'          010
LA    R10,PAVE1           011
AH    R10,=H'66'          012
ZAP   PECON,=PL1'0'       013
BC    15,NOTWANTA         014
```

Просматривая эти предложения, следует иметь в виду, что где-то в программе были закодированы предложения, приравнивающие некоторые регистры символическим меткам: регистр 8 приравнивается к R8, регистр 9 — к R9 и регистр 10 — к R10.

Поля данных и метки, к которым обращаются эти предложения, не показаны в данной части распечатки дампа основной

памяти. Однако представление этих предложений на машинном языке может быть проанализировано по их шестнадцатеричному представлению следующим образом:

Предложение 001 — адреса с 007C12 по 007C17.

Код команды D2 или MVC (Пересылка символов)

Длина пересылки 0D или 13 (запись в машинном коде для длины 14 байтов)

Адрес первого операнда A005

A означает, что регистр 10 является базовым
005 — смещение относительно этого адреса

Адрес второго операнда 6E4B

6 указывает, что регистр 6 является базовым
E4B — смещение относительно этого адреса

Это предложение пересылает 14 байтов из поля с меткой BOLE по адресу, на 5 байтов большему, чем адрес, содержащийся в регистре 10. Адрес поля BOLE определяется как сумма адреса, указанного общим регистром 6, и десятичного значения величины X'E4B', т. е. смещения.

Предложение 002 — адреса с 007C18 по 007C1B.

Код команды 4A или AH (Сложение полуслова)

Первый операнд A или общий регистр 10

Адрес второго операнда 0568A

0 — индексный регистр; составляющая адреса равна нулю
5 — базовый регистр
68A — смещение

Предложение прибавляет литеральную константу с фиксированной точкой (находящуюся по адресу, образованному суммой содержимого общего регистра 5 и смещения X'68A') к содержимому общего регистра 10.

Предложение 003 — адреса в 007C1C по 007C1F.

Код команды 58 или L (Загрузка)

Первый операнд 8 или общий регистр 8

Адрес второго операнда 06C98

0 — индексный регистр; составляющая адреса равна нулю
6 — базовый регистр
C98 — смещение

Предложение загружает содержимое полного слова HOLDP8 (находящегося по адресу, получающемуся в результате сложе-

ния содержимого общего регистра 6 со смещением X'C98') в общий регистр 8.

Предложение 004 — адреса с 007C20 по 007C23.

Код команды	58 или L (Загрузка)
Первый операнд	9 или общий регистр 9
Адрес второго операнда	06C9C
0 — индексный регистр; составляющая адреса равна нулю	
6 — базовый регистр	
C9C — смещение	

Предложение загружает содержимое полного слова HOLDP9 (находящегося по адресу, получающемуся в результате сложения содержимого общего регистра 6 со смещением X'C9C') в общий регистр 9.

Предложение 005 — адреса с 007C24 по 007C29.

Код команды	FA или AP (Сложение десятичное)
Длина первого операнда	1 (на машинном языке — 2 байта)
Длина второго операнда	0 (на машинном языке — 1 байт)
Адрес первого операнда	6CB0
6 — базовый регистр	
CB0 — смещение	
Адрес второго операнда	56BF
5 — базовый регистр	
6BF — смещение	

Предложение складывает упакованную десятичную однобайтовую литеральную константу (находящуюся по адресу, получающемуся в результате сложения содержимого общего регистра 5 со смещением X'6BF') с содержимым двухбайтового упакованного десятичного поля PECON (находящегося по адресу, получающемуся в результате сложения содержимого общего регистра 6 со смещением X'CB0').

Предложение 006 — адреса с 007C2A по 007C2F.

Код команды	F9 или CP (Сравнение десятичное)
Длина первого операнда	1 (на машинном языке — 2 байта)
Длина второго операнда	1 (на машинном языке — 2 байта)
Адрес первого операнда	6CB0
6 — базовый регистр	
CB0 — смещение	
Адрес второго операнда	568C
5 — базовый регистр	
68C — смещение	

Предложение сравнивает упакованную десятичную двухбайтовую литеральную константу (находящуюся по адресу, получающемуся в результате сложения содержимого общего регистра 5 и смещения X'68C') с содержимым двухбайтового упакованного десятичного поля PECON (находящегося по адресу, получающемуся в результате сложения содержимого общего регистра 6 со смещением X'CB0').

Предложение 007 — адреса с 007C30 по 007C33.

Код команды	47 или BC (Условный переход)
Значение маски для признака результата	7
Адрес второго операнда	0449C

0 — индексный регистр; составляющая адреса равна нулю
4 — базовый регистр
49C — смещение

Это предложение по значению маски в первом операнде проверяет конфигурацию разрядов признака результата в Слове состояния программы на условие неравенства. Если признак результата показывает, что в результате выполнения предложения 006 выявлено условие неравенства, то происходит переход к предложению NOTWANTA (находящемуся по адресу, полученному в результате сложения содержимого общего регистра 4 со смещением X'49C').

Предложение 008 — адреса с 007C34 по 007C37.

Код команды	95 или CLI (Сравнение непосредственное)
Непосредственный символ	E7 или буква X
Адрес первого операнда	6CB3

6 — базовый регистр
CB3 — смещение

Это предложение сравнивает один восьмибитовый символ из второго байта машинной команды с 1 байтом данных, находящихся в поле MINT2 (по адресу, полученному в результате сложения содержимого общего регистра 6 и смещения X'CB3').

Предложение 009 — адреса с 007C38 по 007C3B.

Код команды	47 или BC (Условный переход)
Значение маски для признака результата	8
Адрес второго операнда	04462

0 — индексный регистр; составляющая адреса равна нулю
4 — базовый регистр
462 — смещение

Это предложение по значению маски первого операнда проверяет конфигурацию разрядов признака результата в Слове состояния программы на условие равенства. Если признак результата указывает, что в результате выполнения предложения 009 установилось условие равенства, то совершается переход к предложению TAP60 (находящемуся по адресу, получаемому путем сложения содержимого общего регистра 4 со смещением X'462').

Предложение 010 — адреса с 007C3C по 007C3F.

Код команды	92 или MVI (Пересылка)
Непосредственный символ	E7 или буква X
Адрес первого операнда	6CB3
6 — базовый регистр	
CB3 — смещение	

Это предложение пересылает один восьмибитовый символ, содержащийся во втором байте машинной команды, в 1 байт памяти, адресуемый меткой MINT2 (адрес, получаемый в результате сложения содержимого общего регистра 6 и величины смещения X'CB3').

Предложение 011 — адреса с 007C40 по 007C43.

Код команды	41 или LA (Загрузка адреса)
Первый операнд	A или общий регистр 10
Адрес второго операнда	07B01
0 — индексный регистр; составляющая адреса равна нулю	
7 — базовый регистр	
B01 — смещение	

Предложение загружает адрес поля PAVE1 в общий регистр 10. (Адрес PAVE1 — это сумма, полученная в результате сложения величины смещения X'B01' с адресом, содержащимся в общем регистре 7.)

Предложение 012 — адреса с 007C44 по 007C47.

Код команды	4A или AH (Сложение полуслова)
Первый операнд	A или общий регистр 10
Адрес второго операнда	0568E
0 — индексный регистр; составляющая адреса равна нулю	
5 — базовый регистр	
68E — смещение	

Предложение складывает литеральную константу с фиксированной точкой длиной в полуслово (находящуюся по адресу, получающемуся в результате сложения содержимого общего регистра 5 с величиной смещения X'68E') с содержимым общего регистра 10.

Предложение 013 — адреса с 007C48 по 007C4D.

Код команды	F8 или ZAP (Сложение с очисткой)
Длина первого операнда	1 (на машинном языке — 2 байта)
Длина второго операнда	0 (на машинном языке — 1 байт)
Адрес первого операнда	6CB0
6 — базовый регистр	
CB0 — смещение	
Адрес второго операнда	56BA
5 — базовый регистр	
6BA — смещение	

Предложение устанавливает поле первого операнда (PECON) в нуль и затем помещает в него второй операнд — упакованную десятичную константу, которая находится по адресу, получающемуся в результате сложения смещения X'6BA' с содержимым общего регистра 5. Адрес поля PECON определяется как сумма смещения X'CB0' и содержимого общего регистра 6.

Предложение 014 — адреса с 007C4E по 007C51.

Код команды	47 или BC (Условный переход)
Значение маски для признака результата	F или 15
Адрес второго операнда	0449C
0 — индексный регистр; составляющая адреса равна нулю	
4 — базовый регистр	
49C — смещение	

Это предложение вызывает переход в проблемной программе. Оно по значению маски первого операнда проверяет конфигурацию разрядов признака результата. Так как эта маска соответствует всем возможным значениям признака результата, программа осуществляет переход к предложению NOTWANTA (находящемуся по адресу, получающемуся в результате сложения смещения X'49C' с содержимым общего регистра 4).

Если представить в качестве иллюстрации всю распечатку дампа основной памяти, то читатель сможет найти все адреса, указанные этими предложениями.

Несмотря на то что распечатка дампа основной памяти содержит полный набор кодов команд проблемной программы, адреса этих команд, констант и областей не совпадают с адресами, указанными в распечатке программы в исходном коде. Проблемная программа, находясь в основной памяти вычислительной машины, смещена относительно начального нулевого адреса на определенное количество байтов. Смещение это может составлять 10 000, 45 000, 300 000 байтов или любое количество байтов в пределах доступного объема основной памяти вычислительной машины. Количество байтов в смещении изменяется не только в зависимости от видов операционных систем, но различно также и для сходных операционных систем. Последнее различие возникает из-за выбора различных возможностей при генерации данной операционной системы.

Так как, вообще говоря, нет общей для всех систем точки начала проблемной программы в памяти, то распечатка дампа памяти содержит указатель адреса первого байта программы. К сожалению, это еще один момент, который в данном случае нельзя рассмотреть во всей полноте: дампы состояния основной памяти, генерируемые различными вариантами различных операционных систем, отличаются по формату.

Смещение проблемной программы относительно начального адреса основной памяти называется *коэффициентом перемещения*. Допустим, к примеру, что некоторая проблемная программа начинается в основной памяти с адреса X'0035A6'. Если программист пожелает проверить содержимое поля данных с начальным адресом в исходной распечатке X'124', он должен прибавить этот адрес к коэффициенту перемещения X'0035A6' (X'124'+X'0035A6'=X'0036CA'). Искомое поле находится в распечатке дампа основной памяти по адресу X'0036CA'.

Из предыдущего примера распечатки дампа основной памяти видно, что команды представлены на машинном языке и обычно используют адреса памяти в форме сочетания базового регистра и смещения. В большинстве случаев в распечатке дампа основной памяти приводится содержимое общих регистров в момент, непосредственно предшествующий выполнению дампа. Программист может прибавить к содержимому базового регистра желаемое смещение и по сумме, получившейся в результате сложения, найти в проблемной программе точку, на которую имеется ссылка в рассматриваемом предложении. Допустим, к примеру, что команда BC 8,GOTO представлена в распечатке дампа основной памяти на машинном языке в следующем формате:

47806390

В этом примере

47 — код команды BC на машинном языке;
8 — значение маски для проверки признака результата;
0 — значение индекса;
6 — базовый регистр;
390 — смещение.

Если в этот момент общий регистр 6 содержал величину X'0000380D', то действительный адрес поля (GOTO) в распечатке дампа основной памяти можно определить сложением смещения X'390'с содержимым базового регистра X'0000380D'; сумма будет равна X'00003B9D'.

```
                                        WORKAREA
010F60  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040

                                                                        TABLE 4
                                        WORKAREA                  BYTESWIT  S1       S2       S3
010F80  40404040 40404040 40404040 40404040  4040404C 404040E7  00000C00 000C0000

                                        TABLE 4
            S4       S5       S6       S7       S8       S9       S10      S11      S12      S13
010FA0  0C00073C 00000C00 568C0000 0C07394D  00000C00 000C0512  4C00006C 00001C00

                                        TABLE 4
            S14      S15      S16      S17      S18      S19      S20      S21      S22      S23      S24
010FC0  000C0700 0C00009C 00220C00 000C6003  2C00000C 00000C00  000C0113 0C00000C

                                        TABLE 4                                 HAFWD1 BITSWIT
            S25      S26      S27      S28      S29      S30      FILL     FULLWD   HAFWD2   FILL
010FE0  04034C00 003D0000 0C00819C 00007D00  004D0000 0000000A  00000014 08000000

                                        TABLE 5
            F1       F2       F3       F4       F5       F6       F7       F8
011000  00000000 00000009 00000000 0000002A  00000000 00000052  00000000 00000000

                                        TABLE 5
            F9       F10      F11      F12      F13      F14      F15      F16
011020  00000080 00000000 00000021 00000000  11111111 0000000C  00000000 0000003D

                                                LETTERS                         SWITCHNO
            COUNTER1 S1       S2       S3       S4       S5       S6       S7       S8       S9       S10      S11      S12      S13      S14
011040  00008C00 0C000C00 0C000C00 0C000C00  0C000C00 0C000C00  0C000C00 0C000C00 0C000C40
```

Рис. 18.2.

На рис. 18.2 дана еще одна иллюстрация распечатки дампа основной памяти с константами различной длины и конфигурации. Представлены таблицы, рабочие области, переключатели и константы. Содержимое некоторых из этих областей изменилось по сравнению с конфигурацией, которую они имели при загрузке проблемной программы; другие области сохранили свое

первоначальное содержимое. Предложения, представленные распечаткой дампа основной памяти, кодировались в проблемной программе следующим образом:

```
WORKAREA    DC    CL55'Ь'
BYTESWIT    DC    CL1'Ь'
TABLE4      DC    30PL3'0'
FULLWD      DC    F'10'
HAFWD1      DC    H'0'
HAFWD2      DC    H'20'
BITSWIT     DC    X'00'
TABLE5      DC    16F'0'
COUNTER1    DC    PL3'0'
LETTERS     DC    14XL2'0C'
SWITCHNO    DC    CL1'Ь'
```

Содержимое этой части распечатки дампа основной памяти организовано в той же последовательности, что и представленные ранее предложения DC.

WORKAREA — адреса памяти с 010F60 по 010F96. Это поле содержит 55 байтов, каждый байт — шестнадцатеричное представление кода пробела (X'40'). Несмотря на то что константа указывала только один пробел (который должен появиться в первой позиции поля), оставшимся байтам поля вправо от первого байта также была придана конфигурация пробела. Так получается в результате того, что если символьной константе задана длина, превышающая количество символов, закодированных в константе, то все оставшиеся байты справа от заданных константой заполняются кодом пробела X'40'.

BYTESWIT — адрес памяти 010F97. Это однобайтовая символьная константа, содержащая теперь шестнадцатеричную конфигурацию буквы X. Подразумевается, что алгоритм программы ввел букву X в этот байт где-то в промежутке времени между началом выполнения программы и распечаткой дампа основной памяти.

TABLE4 — адреса памяти с 010F98 по 010FF1. Первоначально эта область была определена как таблица, состоящая из 30 трехбайтовых упакованных десятичных сегментов, причем каждый сегмент содержал упакованную десятичную величину +0. После начала выполнения программы в некоторые из них были помещены упакованные десятичные величины. Теперь содержимое сегментов таково:

Номер сегмента	Адрес	Значение
S1	010F98	+0
S2	010F9B	+0
S3	010F9E	+0
S4	010FA1	+73
S5	010FA4	+0
S6	010FA7	+568
S7	010FAA	+0
S8	010FAD	—7394
S9	010FB0	+0
S10	010FB3	+0
S11	010FB6	+5124
S12	010FB9	+6
S13	010FBC	+1
S14	010FBF	+0
S15	010FC2	+7000
S16	010FC5	+9
S17	010FC8	+220
S18	010FCB	+0
S19	010FCE	+60032
S20	010FD1	+0
S21	010FD4	+0
S22	010FD7	+0
S23	010FDA	+1130
S24	010FDD	+0
S25	010FE0	+4034
S26	010FE3	—3
S27	010FE6	+0
S28	010FE9	+819
S29	010FEC	—7
S30	010FEF	—4

FULLWD — адреса памяти с 010FF4 по 010FF7. Это константа с фиксированной точкой длиной в полное слово, содержащая величину +10. Так как в предложении DC, задающем это поле, указана длина в полное слово, первый байт был выравнен по границе следующего полного слова. В результате этого выравнивания между последним байтом предыдущей константы и первым байтом этой константы появилось два неиспользованных байта памяти.

HAFWD1 — адреса памяти с 010FF8 по 010FF9. Это константа с фиксированной точкой длиной в полуслово со значением

+0. Пропускать байты памяти не было необходимости, так как следующий байт находился на границе полуслова.

HAFWD2 — адреса с 010FFA по 010FFB. Это константа с фиксированной точкой длиной в полуслово со значением +20. Она выравнена соответствующим образом, так как следует непосредственно за предшествующим полусловом.

BITSWIT — адрес 010FFC. Предполагается, что этот байт памяти содержит восемь двоичных переключателей. Хотя при инициализации все разряды были установлены в нуль, сейчас он содержит единицу в пятом разряде слева. На основе положения этих переключателей программист может сделать заключение о выполнении некоторых подразумеваемых логических условий.

TABLE5 — адреса памяти с 011000 по 01103F. Эта таблица состоит из 16 четырехбайтовых сегментов с фиксированной точкой. Как указывается предложением DC, формирующим эту таблицу, каждый сегмент выравнен по границе полного слова. Для того чтобы выравнять первое полное слово сегмента таблицы, компилятор должен пропустить 3 байта, непосредственно следующие за предыдущей константой. Первоначально каждый сегмент таблицы содержал величину +0. В процессе выполнения программы в несколько сегментов были помещены величины с фиксированной точкой. Текущее содержимое сегментов представлено ниже.

Номер сегмента длиной в полное слово	Адрес	Значение
F1	011000	+0
F2	011004	+9
F3	011008	+0
F4	01100C	+42
F5	011010	+0
F6	011014	+82
F7	011018	+0
F8	01101C	+0
F9	011020	+128
F10	011024	+0
F11	011028	+33
F12	01102C	+0
F13	011030	−1
F14	011034	+12
F15	011038	+0
F16	01103C	+61

COUNTER1 — адреса с 011040 по 011042. Как можно судить по метке, это трехбайтовое упакованное десятичное поле программист использует как счетчик. Хотя первоначально в него была помещена величина +0, теперь оно содержит величину +8. Независимо от того, что считает счетчик, события или, например, сообщения, их было восемь к моменту распечатки дампа.

LETTERS — адреса памяти с 011043 по 01105E. Эту область задали как состоящую из 14 двухбайтовых сегментов. Но в предложении DC задали только действительное содержимое 1 байта (две шестнадцатеричные цифры) для каждого сегмента. Так как константы были заданы как шестнадцатеричные, компилятор с языка Ассемблера поместил 1 байт данной константы в младшую позицию каждого сегмента, а оставшиеся части каждого сегмента заполнил шестнадцатеричными нулями. Нет точного указания на предполагаемое использование этих сегментов, но общее количество отведенных в памяти байтов, равное 28, разделено на 14 двухбайтовых полей, каждое из которых содержит упакованную десятичную величину +0.

SWITCHNO — адрес памяти 01105F. Это однобайтовая константа, содержащая код правильного пробела; может быть использована для повторной установки байтового переключателя BYTESWIT.

Представленный в главе материал является только частью того, что необходимо знать для полного понимания информации, содержащейся в распечатке дампа. Он лишь создает основу для интерпретации данных. Кроме того программист должен знать формат распечатки дампов состояния основной памяти, предоставляемых конкретной системой, а также должен изучить руководства по данной системе.

ЧАСТЬ IV

Методы доступа и наборы данных

Глава 19

Наборы данных и форматы записей

Набор данных — это совокупность логических записей данных. Формат и содержимое записей образуют единство, необходимое для обеспечения одной или нескольких задач входными данными. В данной главе подразумевается, что наборы данных размещаются на магнитных лентах, томах прямого доступа или на других запоминающих устройствах подобного типа. Записи наборов данных могут быть похожи одна на другую или могут отличаться одна от другой по формату, размеру и содержащимся данным. В общем случае считается, что запись содержит одно или несколько полей данных, которые отличают ее от всех других записей. Например, в отделе кадров на каждого из сотрудников может быть заведен набор данных, состоящий из нескольких записей. В каждой записи зафиксированы имя и фамилия, адрес, возраст, иждивенцы, род работы, размер зарплаты и множество других сведений. Допустим, что на каждого сотрудника ведется одна запись. В этом случае каждая запись уникальна, так как имеет единственный в своем роде номер. На больших предприятиях имеется много людей с похожими или одинаковыми именами, и поэтому сотрудники регистрируются не по именам, а по номерам.

Может существовать несколько наборов данных, в которых используются записи, сходные по внешним признакам, но при этом записи в одном наборе данных содержат информацию, отличную от информации в других наборах. Картотеку на сотрудников некоторой организации — еще раз возвращаемся к этому примеру — могут образовывать несколько наборов данных, в каждом из которых для различения записей используется номер сотрудника. Один набор данных может содержать записи с биографическими сведениями о каждом из сотрудников; другой набор данных — записи с информацией о текущей годовой заработной плате сотрудника, например общую сумму заработ-

ка, удержанные налоги, вычеты в фонд социального обеспечения, чистый заработок и т. п.; третий набор данных может содержать записи, в которых дается оценка производительности сотрудника в течение определенного периода времени. Каждый из этих наборов данных может содержать записи с одним и тем же номером сотрудника, но в то же время каждый набор данных является единственным в своем роде в силу того, что состоит из записей, содержащих различную информацию.

Различают несколько способов *организации наборов данных*. Например:

1. Набор данных, состоящий из записей логически или физически последовательных, без внешних или внутренних индексов, позволяющих производить выборочное обращение к записям, называют последовательным набором данных.

2. Набор данных, формируемый программами индексно-последовательного метода доступа, называется индексно-последовательным набором данных. Этот вид организации набора имеет несколько уровней внутренней индексации, которая строится данным методом доступа. Структура индекса используется для нахождения ключей, содержащихся в записях набора данных.

3. Набор данных, имеющий прямую организацию, может иметь несколько различных физических признаков. Сами записи могут содержать в наборе данные в логически последовательном порядке, в логически последовательном, но физически случайном порядке или в логически последовательном порядке с пробелами, когда между записями, у которых значения ключей не следуют непосредственно одно за другим, помещаются фиктивные записи.

Последовательность записей для этого последнего вида может выглядеть, например, следующим образом: 1, 2, 3, D, D,6, D,8, 9, 10, D,12 и т. д., где D обозначает фиктивную запись. Фиктивная запись помещается там, где должна быть запись с логическими ключами 4, 5, 7 и 11, так как записи данных с этими индексами отсутствуют.

4. Библиотечный набор данных состоит из групп логически последовательных записей, причем каждая группа помещается в наборе данных там, где имеется достаточно места для размещения входящих в эту группу записей. При такой организации каждая группа записей называется *разделом* набора данных, для каждого из которых имеется элемент в *справочнике* библиотечного набора данных. Справочник состоит из группы элементов по одному для каждого раздела набора данных. Элемент содержит имя и *указатель* (адрес) первой записи раздела.

Все эти различные виды организации наборов данных можно применять при размещении наборов данных на устройствах

прямого доступа. Для магнитных лент имеется ограничение — на них можно размещать наборы данных только с последовательной организацией.

А. ФОРМАТЫ ЗАПИСЕЙ

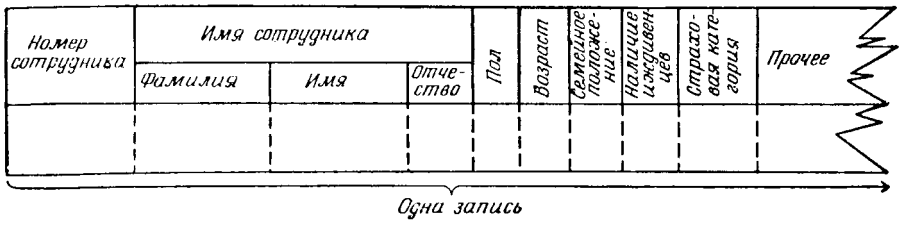
Записью считается логический набор полей, содержащих данные, непосредственно связанные с единственным в своем роде идентификатором, который позволяет отличить данный набор полей данных от других наборов полей данных или от других записей. В зависимости от типа данных в записях или от алгоритма проблемной программы для записей не обязательно требуется идентификатор, но в большинстве наборов данных идентификатор записи, или ключ, присутствует.

При занесении записей в набор данных их можно записать поодиночке или группой в рамках одной операции ввода-вывода. Последний способ называется *блокированием* или образованием сблокированных записей. Для устройства, содержащего данные, нет никакой разницы между одиночной записью и блоком записей — одна запись обрабатывается аппаратурой так же, как и блок данных. Программы некоторых методов доступа определяют, как производится выполнение чтения или записи — по одной записи или группами записей.

Для выделения начала и конца логических блоков данных независимо от того, состоят ли они из одной или нескольких записей, после каждого блока данных запоминающее устройство пропускает некоторый интервал на носителе. Хотя его обычно называют промежутком между записями (Inter-Record Gap — IRG), было бы более подходящим называть этот интервал промежутком между блоками, так как таковой имеется физически именно между блоками. Следует обратить внимание на некоторое различие в терминологии, которая может иногда вводить в заблуждение: запоминающее устройство воспринимает каждую порцию поступающих в него при операции ввода-вывода данных как блок; программа или метод доступа, которые передают данные на запоминающее устройство, могут интерпретировать их как блок записей или как одну запись. Другими словами, запоминающее устройство обрабатывает одинаково как одну запись, так и блок записей, а именно как логический блок данных.

Чтобы проиллюстрировать понятие записи и составляющих ее элементов, нужно мысленно представить себе, как выглядела бы эта информация при распечатке ее на бумажной ленте.

Ниже представлена часть обычной записи со сведениями о сотрудниках, входящей в соответствующий набор данных:

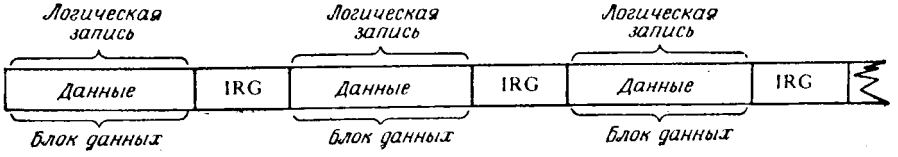


Поля в этой записи не могут быть самостоятельно распознаны стандартными программами управления данными операционной системы. Поля эти имеют значение только для проблемной программы, которая использует эти записи. На рисунке показаны следующие основные поля:

Номер сотрудника
Имя сотрудника
Пол
Возраст
Семейное положение
Наличие иждивенцев
Страховая категория

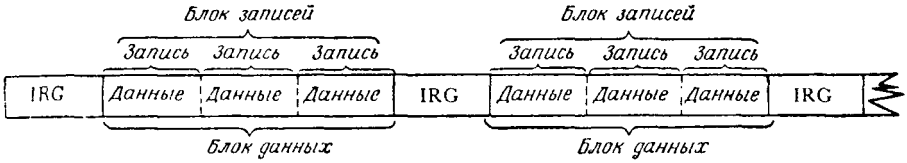
и другие поля, которые могут появиться на оставшейся части записи. В поле «имя сотрудника» имеются подполя, ограничивающие длину помещаемых в них данных: фамилии, имени и отчества. Эти подполя, так же как и более крупные поля, известны только проблемной программе. Небольшие блоки под каждым полем и подполем изображают область с переменным количеством байтов, отведенную под данные, которые должны находиться в соответствующем поле. Схема этой записи сделана лишь для наглядности; когда эта запись считывается или записывается, существуют только байты данных. Описания и заголовки полей отсутствуют в записи, хранящейся на запоминающем устройстве.

Если смысл терминов «запись», «блок записей» и «блок данных» теперь понятен, перейдем к рассмотрению физической структуры этих объектов, которую они будут, по нашим представлениям, иметь на запоминающем устройстве. Одиночные несблокированные записи располагаются на поверхности носителя информации запоминающего устройства следующим образом:



Для запоминающего устройства данные, содержащиеся между двумя промежутками (IRG), представляют собой блок данных; для проблемной программы те же самые данные являются логической записью.

Когда данные состоят из сблокированных записей, на запоминающем устройстве они могут быть представлены следующим образом:



Следует еще раз обратить внимание на то, что запоминающее устройство интерпретирует данные между двумя промежутками как блок данных. Но или проблемная программа, или часть стандартных программ управления данными (если им была передана соответствующая информация) интерпретируют те же самые данные как блоки, каждый из которых состоит из трех отдельных записей. Эти рисунки дают общее представление о расположении данных на запоминающем устройстве; имеются некоторые различия в способе представления этих данных на магнитной ленте и запоминающем устройстве с прямым доступом.

Когда применяется метод доступа, в котором предусмотрены средства объединения записей в блоки и разблокирования блоков данных, в программе нет необходимости учитывать, когда новый блок заполнен и готов к физической операции записи в набор данных. Программа просто выдает для каждой записи макрокоманду ввода-вывода и продолжает выполняться обычным способом. Эти запросы на ввод-вывод воспринимаются программами метода доступа, и записи помещаются по одной в блок. Когда обработано достаточно записей, чтобы заполнить блок, метод доступа выдает команду ввода-вывода, чтобы записать готовый блок в набор данных. Затем метод доступа начинает заново заполнять блок следующим набором записей по мере их поступления от программы. Если записи считываются с запоминающего устройства, программы метода доступа работают почти противоположным образом. Весь блок записей сначала заносится в память, а затем, по мере того как проблемная программа выдает запрос на чтение, метод доступа по одной передает записи из этого блока программе. После освобождения этого блока программам метода доступа предоставляется другой блок, записи с которого передаются в проблемную программу.

Отдельные записи могут иметь один из трех форматов: с фиксированной длиной, с переменной длиной и с неопределенной длиной. Каждый из этих трех форматов описывается в следующих разделах.

1. Записи фиксированной длины

Если указывается, что записи в наборе данных имеют фиксированную длину, то это означает, что каждая запись имеет точно такое же количество байтов, что и другие записи. Например, если указывается, что набор данных состоит из записей с фиксированной длиной 300 байтов, то каждая запись в этом наборе будет содержать точно 300 байтов. Хотя длина записи фиксирована, анализ полей данных внутри записи не производится. Стандартные программы метода доступа главным образом управляют и формируют сами записи, проблемная же программа управляет данными внутри записи. Следовательно, набор данных, содержащий записи фиксированной длины, может на самом деле состоять из записей нескольких различных видов или из записей, содержащих данные различных типов. Проблемная программа несет ответственность за определение типа обрабатываемой записи. Это можно показать на наборе данных, содержащем записи, относящиеся к системе отчетности за торговые операции по заказам. Набор данных может содержать записи четырех типов, а именно:

тип 1 — запись заявки;
тип 2 — запись накладной;
тип 3 — запись отчета о получении;
тип 4 — запись оплачиваемого счета.

Внутренний формат каждого из типов записей может выглядеть следующим образом:

Тип 1 — запись заявки

№ накладной	Код записи	№ заявки	Дата заявки			Отдел	Цена	Кол-во	Прочее
			М-ц	День	Год				
	1								

Тип 2 — запись накладной

№ накладной	Код записи	№ заявки	Дата заявки			Код продавца	Код покупателя	№ позиции	Прочее
			М-ц	День	Год				
	2								

Тип 3 — запись отчета о получении

№ накладной	Код записи	Код продавца	Дата записи			Кол-во ордеров	Полученное кол-во	Описание
			М-ц	День	Год			
	3							

Тип 4 — запись оплачиваемого счета

№ накладной	Код записи	Код продавца	Сроки	Стоимость за штуку		Кол-во	Сумма по накладной	
				Долл	Цент		Долл.	Цент
	4							

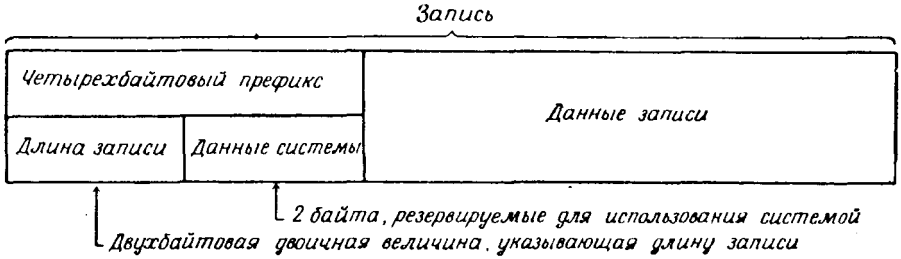
В этих примерах приведены начальные части записей различных типов внутри этого набора данных. Использованный для формирования набора данных или выборки записей метод доступа не отмечает различия во внутренней структуре записей; признак того, что записи имеют фиксированную длину, требует только, чтобы стандартные программы управления данными системы удостоверились в правильности длины каждой записи. Вероятно, в функции проблемной программы будет входить задача просмотра байта кода записи с тем, чтобы определить тип записи, входящей в этот набор данных, и затем выполнить соответствующую этому типу данных обработку.

Если алгоритм проблемной программы требует последовательной обработки записей набора данных, то блокировка записей имеет значительные преимущества. Размеры блоков должны быть настолько крупны, насколько это удобно с учетом конфигурации системы и доступной для проблемной программы памяти. Каждый раз при выполнении физической операции чтения или записи обрабатывается весь блок данных независимо от того, состоит ли он из одной-единственной или из 15 записей. Понятно, что гораздо меньше времени требуется на считывание или запись блока данных, состоящего из 15 записей, чем на выполнение 15 операций чтения или записи. Когда для формирования набора данных, состоящего из записей фиксированной длины, используются сблокированные данные, все блоки имеют одну и ту же длину, за исключением, возможно, последнего блока набора. Если количество записей, приходящихся на последний блок, недостаточно для формирования полного блока, записывается *укороченный блок*. Операционная система автоматически проверяет наличие укороченных блоков и разрешает их запись.

2. Записи переменной длины

Записи переменной длины могут содержать разное количество данных и состоять из различных типов полей. Каждая запись может иметь как минимальную длину, допускаемую запоминающим устройством, на котором она хранится, так и максимальную длину, указанную для данного набора данных. Для того чтобы операционная система могла определить длину любой одиночной записи, как сблокированной, так и несблокированной, каждая запись должна иметь в своем составе четырехбайтовый префикс, в котором указана длина этой записи. Два младших байта этого префикса используются операционной системой, два старших байта содержат выраженную в двоичной форме длину записи. При формировании набора данных проблемная программа должна занести эту длину в каждую запись. При выборке записи из набора данных система использует существующую информацию о длине записи при проверке длины и разблокировании, если оно производится.

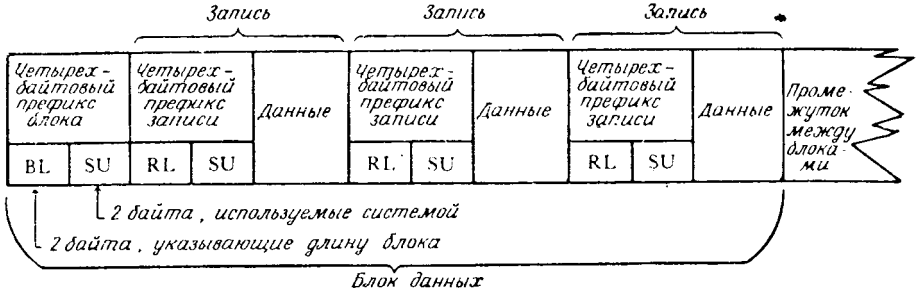
Запись переменной длины, извлеченная из набора данных, может иметь следующий вид:



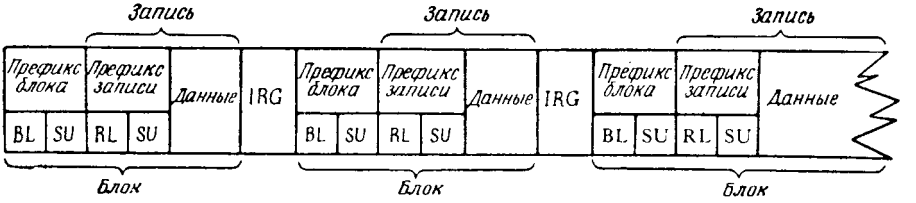
Каждая запись переменной длины должна иметь четырехбайтовый префикс, в котором зафиксирована ее длина.

Кроме того, каждый блок данных, который должен быть считан или записан с использованием записей переменной длины, должен иметь подобный четырехбайтовый префикс. Префикс блока должен присутствовать независимо от того, состоит ли он из одной записи или из нескольких записей. Так же как и в случае с префиксом записи, старшие 2 байта префикса блока содержат двоичную величину, а младшие 2 байта образуют поле, резервируемое для использования операционной системой. Если метод доступа, используемый для формирования набора данных, обеспечивает автоматическую блокировку записей, система подсчитывает длину блока записи и помещает эту величину в двоичной форме в 2 старших байта префикса блока. Если проблемная программа, которая формирует файл, берет на себя функции блокирования, то она должна вычислить длину блока и поместить это значение в префикс блока.

Сблокированные записи переменной длины выглядят следующим образом:



Несблокированные записи переменной длины требуют четырехбайтового поля для подсчета длины, даже если блок данных состоит только из одной записи. Это будет выглядеть следующим образом:



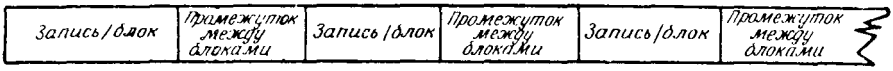
В последнем примере предполагается, что каждая запись является также и блоком данных, поэтому, помимо четырехбайтового префикса записи, используется четырехбайтовый префикс блока.

3. Записи неопределенной длины

В некоторых руководствах записи неопределенной длины описываются как записи, которые не относятся ни к записям фиксированной длины, ни к записям переменной длины. Но такой подход не дает достаточного представления о записях неопределенной длины. Если программа формирует записи данных, то программист может предпочесть использовать все записи одной и той же длины или записи переменной длины, причем в первом случае будет считаться, что он использует записи фиксированной длины, а во втором случае — переменной длины. Тогда, что же такое запись неопределенной длины? Можно дать такое определение записи неопределенной длины: любой блок данных, считываемый с существующего набора данных, формат и длина записи которого неизвестны. Система обрабатывает каждый логический блок данных как одиночную запись неза-

висимо от того, состоит ли в действительности блок из записей фиксированной или переменной длины, сблокированных или несблокированных. Если программист задает длину блока меньшую, чем длина полной записи, система считывает такое количество данных, какое может поместиться в отведенное для них место.

Запись неопределенной длины независимо от ее действительной организации интерпретируется операционной системой следующим образом:



Данные продолжают считываться в основную память до тех пор, пока не обнаружится промежуток между записями или не заполнится отведенная для данных область памяти. Записи данных обрабатываются во многом так же, как и несблокированные записи переменной длины, за тем исключением, что в записях неопределенной длины необязательно явным образом представлен префикс блока и префикс записи. Если в проблемную программу были в действительности считаны записи переменной длины, то два четырехбайтовых префикса обрабатываются как часть записи данных.

Б. РАСПЕЧАТАННЫЕ, ОТПЕРФОРИРОВАННЫЕ ИЛИ ВЫВЕДЕННЫЕ НА ДИСПЛЕЙ ЗАПИСИ

В некоторых случаях общий термин «набор данных» используется для обозначения набора записей, выводимых на устройства ввода-вывода, такие, например, как АЦПУ, перфоратор или дисплей. Корректность такой интерпретации термина «набор данных» зависит от установки, но, как правило, под набором данных понимают записи, занесенные на такое устройство, которое позволяет хранить их там неопределенный период времени, извлекать их оттуда, изменять и пополнять.

Данные, передаваемые на устройства вывода, которые обычно не рассматриваются как запоминающие, совершенно справедливо можно считать состоящими из записей. Записи эти могут быть как сблокированными, так и несблокированными в зависимости от того, что допускают система и программы управления данными, обрабатывающие их. В настоящий момент об этом виде данных говорится только с целью определения функции системы, позволяющей включить в запись *управляющий символ*. Хотя введение упомянутого символа является второстепенной функцией проблемной программы, она должна сформировать его и поместить в выходную запись.

Если используется управляющий символ, программист должен сообщить об этом системе посредством специального параметра в соответствующем блоке управления данными (DCB). Однобайтовый символ становится первым байтом записи; поэтому в качестве длины логической записи должна задаваться увеличенная на единицу длина данных.

Управляющий символ применяется в следующих целях:

1. Если выходные данные перфорируются, управляющий символ указывает нужный приемный карман для перфокарт.

2. Если данные выводятся на печать, управляющий символ определяет расстояние между распечатываемой строкой и предыдущей.

В случае, если программист укажет системе, что в выходную запись должен быть включен управляющий символ, но самого символа не задаст, система в качестве управляющего символа воспримет первый байт записи. И наоборот, если программист формирует в первом байте каждой записи управляющий символ и не сообщает системе об этом, этот байт считается частью поля данных и входит в длину записи.

Если содержащая управляющий символ запись передается в устройство, которое или не использует управляющие символы, или не опознает этот символ в качестве управляющего, этот байт считается частью полей данных, входящих в запись.

В. НАБОРЫ ДАННЫХ НА МАГНИТНОЙ ЛЕНТЕ

Наборы данных, записанные на магнитной ленте, хранятся на катушках, называемых *томами*. Каждому тому, или катушке, присваивается свой серийный номер; этот номер наносится на внешнюю поверхность катушки. Если производится внутренняя маркировка магнитной ленты, то формируется внутренняя метка, содержащая серийный номер тома. После этого оператор вычислительной машины или операционная система легко размещает наборы данных: оператор для опознавания использует внешний серийный номер, система — внутреннюю метку с серийным номером. Один том магнитной ленты может содержать один или несколько наборов данных; один набор данных может занимать несколько последовательных томов.

В силу различий между вычислительными центрами и операционными системами в рамках данного изложения не будет проводиться анализ структуры внутренних меток на магнитных лентах. В общем случае метка тома на магнитной ленте содержит идентификатор метки, номер типа метки, серийный номер тома и другие системные данные. Помимо метки тома, каждый набор данных на магнитной ленте может иметь свой собственный набор меток — головные метки, метки пользовате-

ля и хвостовые метки. Головные и хвостовые метки содержат информацию о наборе данных, такую, например, как формат записей, длина блока, длина записи, плотность записи и другую информацию, которую система использует для обработки данных, содержащихся в наборе. Стандартные метки пользователя обычно формируются в соответствии с правилами, установленными на конкретном вычислительном центре, и сообщают программе дополнительную информацию об этих наборах данных.

Если набор данных на магнитной ленте занимает более одной катушки, стандартные хвостовые метки извещают операционную систему о том, имеются ли еще тома, относящиеся к этому набору данных. Для этого в хвостовой метке указывается условие конца тома EOV (End of Volume). Встретившееся условие конца тома EOV извещает систему о том, что достигнут физический конец той части данных, которые записаны на ленте, но что сам набор данных продолжается на следующем томе или нескольких томах. Указатель EOF (End of File — Конец файла) в хвостовой метке извещает систему, когда заканчиваются все данные, содержащиеся в наборе данных.

Все, что выше разъяснено в нескольких словах, можно было бы развернуть в обширное изложение. Каждому программисту рекомендуется узнать, какой тип и способ обработки меток используется на его вычислительном центре, а затем ознакомиться с этим типом меток по специальным руководствам.

Г. НАБОРЫ ДАННЫХ С ПРЯМЫМ ДОСТУПОМ

Понятие «том» применимо также и к устройствам прямого доступа, в частности к дискам и барабанам. Каждый съемный пакет дисков идентифицируется внутренне и внешне серийным номером тома. Эта идентификация осуществляется внутренней меткой, называемой меткой тома и находящейся обычно на первой дорожке первого цилиндра. Но в отличие от магнитных лент на томах прямого доступа содержится специальный небольшой набор данных, который служит для индексации в операционной системе. Этот специальный набор данных называется оглавлением тома (Volume Table of Contents — VTOC). Он состоит из группы элементов, называемых блоками управления набором данных (Data Set Control Block — DSCB), которые формируются по одному при добавлении нового набора данных. Блоки DSCB содержат сведения о каждом из существующих наборов данных, такие, например, как адрес, имя, общие описания, а также сведения, касающиеся самого тома, — оставшееся в томе свободное место, место, занятое оглавлением, и другую информацию подобного рода.

Инициализация томов с прямым доступом и формирование оглавления не выполняются операционной системой автоматически. Например, новый пакет дисков для устройства 2314 нельзя распаковать, установить и сразу же использовать для запоминания данных. Он должен быть инициализирован вспомогательной программой; в полной операционной системе она называется DASDI (Direct Access Storage Device Initialization — Инициализация запоминающего устройства с прямым доступом). Вспомогательные программы и действия, которые они производят над томами, различаются в разных системах. Ниже приводятся общие функции программы типа DASDI.

1. Формирование метки тома и запись его идентификатора.

2. Формирование оглавления на основе информации, поступающей от управляющих операторов на перфокартах. Эта информация определяет количество дорожек, отводимых для оглавления, и начальный адрес оглавления.

3. Анализ записывающей поверхности всех стандартных дорожек, которые могут быть адресованы программистом. При обнаружении поврежденной поверхности программа DASDI должна указать другую, альтернативную дорожку из числа имеющихся на томе запасных дорожек. Это делается так, что программисту нет необходимости знать, что произошла замена дорожки. На томе устанавливается указатель так, что любое обращение к дефектной дорожке переадресовывается к альтернативной дорожке, заменяющей дефектную.

Одна или более дорожек на устройствах прямого доступа могут быть повреждены в процессе эксплуатации вследствие износа записывающей поверхности. В таких случаях существующие наборы данных можно полностью скопировать на другое устройство, а этот том заново инициализировать программой DASDI. После анализа дефектных и указания альтернативных дорожек том можно снова загрузить данными. Тома устройств с прямым доступом, такие, как диски и барабаны, имеют физические ограничения, которые не распространяются на тома магнитных лент. Для некоторых запоминающих устройств прямого доступа задается максимальное допустимое количество байтов данных на дорожке. Реальная эффективность использования имеющегося в памяти свободного пространства почти полностью зависит от указанных программистом параметров набора данных. Например, для 100 30-байтовых несблокированных записей потребуется значительно больше места в памяти, чем для одного блока данных, состоящего из 100 30-байтовых записей. Причиной этого является наличие промежутка между каждой парой логических блоков данных. В то время как промежутки между блоками на магнитной ленте всегда имеют одну и ту же длину независимо от длины записи,

промежутки между записями на устройстве прямого доступа имеют переменную длину, хотя эту длину можно определить и заранее. Имеется формула для расчета длины промежутка, следующего за каждым блоком данных на устройстве прямого доступа, но достаточно сказать, что длина промежутка зависит от длины предыдущего блока данных. Чем больше длина записанного блока данных, тем больше длина промежутка. Исключением из этого правила является случай, когда блок данных записывается до или почти до физического конца дорожки и промежуток вообще не нужен. Например, если есть запись, требующая промежутка в 160 байтов, записанная так близко к физическому концу дорожки, что для промежутка остается только 5 байтов, то этого для системы достаточно.

Некоторые устройства прямого доступа имеют специальные средства, позволяющие записывать начало блока данных на одной дорожке, а продолжение на одной или более последующих дорожках. Обычно они называются в зависимости от конфигурации аппаратуры и используемой операционной системы, средствами переполнения дорожки и переполнения цилиндра. Для программиста это означает, что теперь он не связан длиной записи или длиной блока, которые должны быть равны или меньше максимальной длины в байтах одной дорожки. Стандартная емкость в байтах одной дорожки на пакете дисков для устройства IBM2314—7294 байта, но с применением средств переполнения дорожки, имеющихся на этом устройстве, проблемная программа может записать блок данных длиной 20 000 байтов, невзирая на присущие дорожке физические ограничения. Условившись, что блок данных начинается с первого байта некоторой дорожки, система запишет 7294 байта на этой дорожке, следующие 7294 байта — на следующей дорожке и, наконец, 5412 байтов — на следующей, третьей дорожке. Следует подчеркнуть, что средства переполнения дорожки функционируют лишь тогда, когда набор данных занимает сплошной массив дорожек. Попытка сделать такую запись, которая потребовала бы перехода на дорожку, не являющуюся соседней, или на другой том, рассматривается как ошибка.

Распределение памяти на устройствах прямого доступа осуществляется во время формирования набора данных. В полной операционной системе OS оно выполняется средствами языка управления заданиями с помощью соответствующих управляющих карт. Запрос на определенный объем памяти, выраженный в количестве дорожек, цилиндров или блоков, кодируется параметром 'SPACE=' на языке управления заданиями на карте DD (Data Definition — определения данных). Заказывая память для набора данных, программист может указать исходное количество памяти и приращение, которое будет добавлено

к исходному количеству, если его окажется недостаточно для размещения всего набора данных. Когда при таком типе распределения исходное количество памяти израсходовано или при формировании набора данных, или при последующих добавлениях к набору, система дополнительно распределяет память набору данных в количестве, равном указанному в запросе вторичного приращения. Распределенные в соответствии с этими условиями области называются вторичными экстентами; исходное количество памяти называется нулевым экстентом. Если в языке управления заданиями это специально не оговаривается, то вторичные экстенты не обязательно примыкают к нулевым экстентам или даже друг к другу. Адрес каждого из экстентов набора данных содержится в элементах DSCB оглавления тома.

Представленная в данном разделе информация носит довольно общий характер в силу различий между операционными системами. Программисту необходимо получить копии руководств по операционной системе, на которой он работает, и ознакомиться с ними.

Глава 20

Методы доступа операционной системы

Эта часть книги предназначена для читателей, желающих глубже понять принципы организации наборов данных и программ, создающих наборы данных и осуществляющих доступ к ним. Она написана с ориентацией на полную Операционную систему (OS), так как последняя является наиболее мощной и гибкой из всех известных в настоящее время операционных систем. Изложение построено в форме обзора различных методов доступа и их вариантов. Цель обзора — дать общее представление об отношении методов доступа к системам организации файлов и устройствам ввода-вывода без детального разбора логической структуры отдельных модулей и макрокоманд операционной системы. Предполагается, что этот материал позволит программисту усвоить основные особенности методов доступа и систем организации файлов, хотя и не будет достаточным для применения рассматриваемых понятий.

В данной главе подразумевается, что при выборе метода доступа учитываются следующие факторы:

1. Устройства ввода-вывода, которые предстоит использовать.
2. Логический формат наборов данных, подлежащих обработке.
3. Требования проблемной программы.

Проблемная программа не обязательно использует один стандартный метод доступа. Для правильного и эффективного функционирования она может пользоваться сочетаниями методов доступа или разновидностями одного метода.

А. ОБЩЕЕ ОПИСАНИЕ И ТЕРМИНОЛОГИЯ

Метод доступа может быть определен как группа макрокоманд, предназначенных для связи между центральным процессором (ЦП) вычислительной машины и конкретным устройством (или устройствами) с целью выполнения задач ввода-вывода для проблемной программы. Во многих случаях сюда же входит проверка выполнения операций ввода-вывода, обработка ошибок, преобразование данных в нужный формат,

контроль правильности, поиск и указание записей данных и т. д.

Ввиду широкого диапазона частных особенностей устройств, а также неограниченного разнообразия проблемных программ программисту предоставлен целый ряд различных методов доступа. Это позволяет ему выбрать метод доступа, который наилучшим образом отвечает его задаче, и выполнить эту задачу, кодируя минимум параметров.

Во время генерации системы системный программист может задать, будет ли ему позволено каждый раз, когда эта система загружается в ЦП, по своему усмотрению включать в ядро системы загрузочные модули резидентных методов доступа. Если он задает возможность включения резидентных методов доступа в свою систему, то оператор вычислительной машины имеет во время начальной загрузки (IPL — Initial Program Loading) три возможности.

1. Оператор вычислительной машины может через консольную машинку сообщить операционной системе, что не следует делать резидентными никакие модули методов доступа. Это обычно позволяет предоставить дополнительную основную память для работы проблемных программ.

2. Он может позволить системе найти и сделать резидентными модули методов доступа в соответствии с заданным списком. Список должен находиться в *библиотеке параметров* операционной системы. Системный программист, ответственный за данную операционную систему вычислительной машины, может в любое время создать или изменить такой список. Так как для резидентных модулей приходится отводить основную память, необходимы осторожность и опыт при определении, какие модули должны находиться в списке резидентных модулей, а какие нет. Системный программист, ответственный за создание списка резидентных загрузочных модулей методов доступа, обязан провести их анализ для определения приоритетов, в соответствии с которыми эти модули должны становиться резидентными. В основе анализа должны лежать учет потребности в основной памяти, частота обращений к модулям, время, которое экономится, если эти модули резидентны, и другие подобные факторы.

3. Оператор может ввести имя альтернативного списка (или списков) тех загрузочных модулей методов доступа, которые должны быть сделаны резидентными на время отладки или специального прогона системы. Примером этого может служить практика отладки телекоммуникационной аппаратуры. Во время обычной работы системы наличие резидентных модулей телекоммуникационного метода доступа может не дать никаких преимуществ. Однако если нужно провести длительный теле-

коммуникационный тест, то оператор поступит правильно, если он заново инициализирует систему и сделает резидентными только модули телекоммуникационного метода доступа вместе со всеми другими модулями, которыми пользуется этот тест.

Вот те методы доступа, которые главным образом будут здесь обсуждаться:

BDAM — базисный прямой метод доступа;
BISAM — базисный индексно-последовательный метод доступа;
BPAM — базисный библиотечный метод доступа;
BSAM — базисный последовательный метод доступа;
BTAM — базисный телекоммуникационный метод доступа;
QISAM — индексно-последовательный метод доступа с очередями;
QSAM — последовательный метод доступа с очередями;
QTAM — телекоммуникационный метод доступа с очередями;
GAM — графический метод доступа;
EXCP — выполнить программу канала.

Последний метод доступа в этом перечне (EXCP) здесь не рассматривается. Это полноправный метод доступа, но рядовые программисты обычно им не пользуются. Как правило, метод доступа EXCP используется только системными программистами, досконально знающими внутреннюю структуру операционной системы и решающими особую задачу, требующую перехода на опрос устройств ввода-вывода и управления ими «вручную».

Решение, какую конкретную разновидность того или иного метода доступа следует использовать, часто является компромиссом. Во многих случаях принятая стандартная или существующая система данных предопределяет выбор методов доступа. В противном случае приходят к компромиссу отчасти на основе знаний и способностей программиста. Вполне вероятно, что опытный программист может использовать базисный метод доступа более эффективно, чем разновидность метода доступа с очередями. Дополнительные усилия, затрачиваемые на детальное кодирование команд и макрокоманд для реализации управления, вознаграждаются повышением эффективности. Однако тот, кто недостатотчно разбирается в тонкостях базисного метода доступа, может найти для себя удобным при каждой возможности использовать метод доступа с очередями. Неэффективное использование метода доступа или применение метода, плохо согласованного с решаемой задачей, может привести к значительному снижению производительности проблемной программы.

Б. БАЗИСНЫЕ МЕТОДЫ ДОСТУПА В СРАВНЕНИИ С МЕТОДАМИ ДОСТУПА С ОЧЕРЕДЯМИ

Базисные методы доступа можно сравнить с приготовлением чашки кофе, сваренного по вашему вкусу, с точным добавлением ингредиентов, в то время как методы доступа с очередями — с нажатием кнопки автоматической кофеварки и получением налитой для вас чашки готового кофе. Результаты могут случайно оказаться одинаковыми, но это маловероятно.

Для достижения желаемого результата при базисном методе программист обязан задать все необходимые параметры и выполняемые функции, при методе доступа с очередями многие частные функции выполняются без участия программиста. Его обязанности ограничиваются только управлением наиболее важными функциональными параметрами.

Пять функциональных различий в уровне автоматизации между базисными методами доступа и методами доступа с очередями заключаются в синхронизации совмещенных операций ввода-вывода, превентивной буферизации, блокировании и разблокировании записей, проверке выполнения операций ввода-вывода, анализе ошибок и организации выходов.

1. Синхронизация совмещенных операций ввода-вывода

Базисные методы. Каждый раз, когда в проблемной программе встречается запрос на ввод-вывод, этот запрос передается системе для предстоящего выполнения. Однако управление возвращается проблемной программе немедленно. Если в соответствии с алгоритмом необходимо, чтобы задача физического ввода-вывода была выполнена до того, как возобновится обработка, программа должна ждать завершения этой операции ввода-вывода и проверить успешность ее завершения.

Предполагается, что программист может предвидеть необходимость запроса на ввод-вывод и, следовательно, поставить такой запрос до той точки в своей программе, где ход выполнения программы зависит от результата ввода-вывода. Этот прием не пригоден, если от хода выполнения программы зависит содержание запроса на ввод-вывод: какая следующая запись должна быть считана или создана или же к какому следующему устройству нужно обратиться.

Методы с очередями. При обсуждении превентивной буферизации будет показано, что выдача запроса на ввод-вывод необязательно вызывает задержку обработки с момента, когда запрос был выдан проблемной программой. Однако, если запрос на ввод-вывод привел к немедленному выполнению операций физического ввода-вывода, программа не получит управления до тех пор, пока все они не будут выполнены.

2. Превентивная буферизация

Базисные методы. Так как OS не указывает последовательности, в которой записи обрабатываются базисным методом доступа, автоматическая буферизация не производится. Все же программисту даны макрокоманды для управления буфером. Буферы целиком заполняются или освобождаются каждый раз, когда проблемная программа инициирует запрос на ввод-вывод.

Методы с очередями. Операционная система так использует буферы, что запрос проблемной программы на ввод-вывод не всегда вызывает физическое выполнение операции ввода-вывода в тот же самый момент. Если это так, весь буфер освобождается или заполняется при выполнении одной операции физического ввода-вывода, избавляя от необходимости каждый раз, когда проблемная программа требует еще одну отдельную единицу данных, проводить отдельную операцию физического ввода-вывода. Состояние и местоположение следующего выбираемого буфера для предназначенных к обработке данных регулируется автоматически.

3. Блокирование и разблокирование

Базисные методы. Макрокоманды базисных методов доступа обрабатывают данные только блоками. Поэтому блокирование и разблокирование данных, состоящих из нескольких логических записей, должно выполняться проблемной программой.

Методы с очередями. Макрокоманды последовательных методов доступа автоматически осуществляют блокирование и разблокирование записей. Это действие определяется параметрами, указанными в блоке управления данными (DCB) проблемной программы, в операторах управления заданиями или в метках наборов данных. Проблемная программа только запрашивает запись, после чего метод доступа и другие компоненты операционной системы предоставляют ей следующую запись, которая должна быть обработана. Программисту нет необходимости знать, из какого блока или через какой буфер передана ему запись, а также выполнилась ли в этот момент физическая операция ввода-вывода. Когда проблемная программа формирует записи для помещения их в выходной набор данных, действия выполняются в обратном порядке. Сформированные записи передаются программам метода доступа, и уже эти программы определяют, когда следует выполнять физическую операцию ввода-вывода для занесения записей в набор данных.

4. Завершение операции ввода-вывода и выходы после ошибки

Базисные методы. Для того чтобы удостовериться, что операция ввода-вывода завершилась правильно, проблемная программа может выдать макрокоманду CHECK, которая выполняет следующие действия:

1) вызывает переход проблемной программы в состояние ожидания до тех пор, пока не закончится физическая операция ввода-вывода,

2) проверяет по блоку управления событием (ECB — Event Control Block), успешно ли завершена физическая операция ввода-вывода,

3) возвращает управление проблемной программе, если физическая операция ввода-вывода завершилась успешно, или

4) передает управление программе анализа ошибок, если при выполнении операции ввода-вывода обнаружена ошибка или особая ситуация.

Вместо макрокоманды CHECK проблемная программа может выдать свою собственную макрокоманду WAIT, а затем проверить состояние блока управления событием с тем, чтобы определить состояние кода завершения, вырабатываемого системой при выполнении операции ввода-вывода.

Методы с очередями. Программы методов доступа автоматически выполняют функции проверки кода завершения и условия ошибки, планирования и выполнения программ обработки ошибок, проверки условий конца файла и конца тома. Управление проблемной программе не возвращается до тех пор, пока эти функции не будут выполнены.

Следует отметить, что не для каждого базисного метода доступа существует аналогичный метод доступа с очередями. Базисный метод доступа может быть прямым, индексно-последовательным, библиотечным, последовательным и телекоммуникационным, в то время как метод доступа с очередями может быть только индексно-последовательным, последовательным и телекоммуникационным. Прямой и библиотечной организациям данных присущи внутренние характеристики, не допускающие использования метода доступа с очередями. Например, превентивная буферизация, как уже говорилось ранее, обеспечиваемая методами доступа с очередями, зависит от возможности предсказать последовательность обращения к записям. Эта последовательность обычно не предсказуема при прямой или библиотечной организациях данных, в связи с чем в настоящее время нет методов доступа с очередями для этих типов организации данных

В. МЕТОДЫ ПРЯМОГО ДОСТУПА

Как ясно из названия, метод прямого доступа используется для формирования наборов данных, размещенных на запоминающих устройствах прямого доступа, или для обращения к ним. В эту категорию попадают запоминающие устройства на магнитных дисках IBM 2311 и IBM 2314, магнитные барабаны IBM 2301 и IBM 2303 и запоминающее устройство на магнитных картах IBM 2321.

Цель и смысл прямой адресации с помощью соответствующих методов заключается в том, чтобы найти случайным образом расположенные записи в наборе данных, не прибегая к считыванию записей, которые могут предшествовать искомой или следовать за ней. Ценой приемлемых затрат эту цель можно достичь, используя один из трех методов адресации: метод прямой адресации, метод прямой адресации с использованием таблицы перекрестных ссылок и метод косвенной адресации на основе случайных или числовых преобразований. При использовании этих методов определение конкретной записи может выполняться несколькими способами:

по относительному номеру записи;
по относительному номеру записи и абсолютному ключу;
по относительному номеру дорожки и абсолютному номеру записи;
по относительному номеру дорожки и абсолютному ключу;
по абсолютному адресу, состоящему из номеров устройства, цилиндра, дорожки и записи.

Кроме того, имеется возможность расширенного поиска, при котором просматривается заданное количество записей или дорожек с целью определения местоположения желаемой записи. Три из этих методов задания адреса записи определяются ниже.

1. Относительный номер записи. Этот тип адресации указывает местоположение записи внутри всего набора данных относительно точки, непосредственно предшествующей первой записи внутри набора данных. Затем метод доступа на основе физического количества записей, приходящихся на одну дорожку, вычислит действительное местоположение дорожки и записи. Этот прием возможен только тогда, когда используются несблокированные записи фиксированной длины.

2. Относительная дорожка и абсолютный ключ. В данном случае указывается относительное местоположение дорожки, содержащей запись (относительно начала набора данных), и адрес области в основной памяти, содержащей ключ,

индентифицирующий искомую запись. Система вычисляет абсолютный адрес дорожки, а затем на этой дорожке ищет запись, ключ которой совпадает с заданным ключом.

3. Относительный номер дорожки и абсолютный номер записи. При этом типе адресации указывается относительное положение дорожки, содержащей запись (относительно начала набора данных), и абсолютный номер записи на этой дорожке. Система вычисляет абсолютный адрес дорожки, а затем переходит к абсолютному номеру записи на ней.

Метод прямой адресации использует тот принцип, что каждая запись, идентификатор записи или ключ отражают последовательную систему нумерации, выражающую относительное положение записи в наборе данных. Этот метод используется при относительной адресации записи. Например, если набор записей содержит встроенные ключи, величины которых последовательно изменяются от 1 до 500, запись с ключом 236 рассматривается как 236-я запись в наборе данных.

Когда программы метода доступа получают запрос на доступ к записи через относительный номер записи, программы вычисляют абсолютные номера дорожки и записи и направляют операции ввода-вывода по этим адресам. Этот способ применим только для записей типа F (записи фиксированной длины).

Метод прямой адресации с использованием таблицы перекрестных ссылок можно интерпретировать как форму индексации. При использовании этого метода проблемная программа при записи данных в запоминающее устройство строит таблицу, содержащую действительный адрес (или относительные номера дорожки и записи), которые помещаются в эту таблицу. Построение этой таблицы должно быть предусмотрено в проблемной программе. При попытке найти запись данных проблемная программа должна отыскать в таблице идентификатор этой записи, найти связанный с ним абсолютный или относительный адрес, а затем использовать эту информацию для направления запроса на ввод-вывод к нужной записи данных. В таком виде эта структура непосредственной адресации может оказаться не подходящей для многотомных файлов. Но вполне возможно использовать подтаблицы для записей, содержащих идентификаторы, которые группируются по тем или иным признакам, например подобно организации уровней индексов в словарях. Например, если индентификаторы записей состоят из букв, можно организовать следующие наборы подтаблиц: от буквы A до буквы F, от G до K, от L до Q, от R до U и от V до Z. Следует учитывать возможность группировки по частоте встречаемости некоторых символов или величин, так как может случиться, что большинство записей попадет в небольшое количество таблиц,

в результате чего получается крайне несбалансированные распределения записей по подтаблицам.

Косвенная адресация с использованием случайных или числовых преобразований может быть довольно сложной. В общем случае она выполняется путем математического преобразования значения ключа записи таким образом, чтобы сформировать индентификатор, определяющий адрес дорожки, содержащей запись. Имеется много алгоритмов операций рандомизации преобразования, которые могут обладать значительными достоинствами. Из-за большого разнообразия имеющихся методов в большинстве руководств даже не делается попытки дать полное определение или рекомендовать конкретную операцию подобного рода. Пользователь по своему усмотрению может выбрать наиболее подходящий способ.

По существу, метод прямого доступа используется для адресации и получения доступа к любой позиции в наборе данных, состоящем из записей, которые могут располагаться произвольным образом.

Г. МЕТОДЫ ПОСЛЕДОВАТЕЛЬНОГО ДОСТУПА

Методы последовательного доступа обеспечивают максимальную независимость от устройств по сравнению со всеми остальными методами доступа. Они могут быть использованы при работе с запоминающими устройствами на магнитной ленте, перфоленточными устройствами, АЦПУ, устройствами чтения и перфорации карт и устройствами прямого доступа. Именно широта применения этих методов позволяет существенно расширить возможности вычислительной машины, обеспечивая реализацию принципа независимости от устройств.

Последовательная организация файла представляет собой мост между наборами данных и программой, в которой реализован принцип независимости от устройств. Проблемная программа может не содержать никакой информации о специфических характеристиках устройств. В момент выполнения такие характеристики устройств могут вноситься или оператором, или через входной поток заданий. Если проблемная программа запрашивает 80-символьные записи, то она может быть составлена таким образом, чтобы принимать подобные записи от любого устройства, которое может передавать их программе, — от устройства чтения карт, от магнитной ленты или от устройства прямого доступа. Это может оказаться полезным для эффективного использования вычислительной машины.

Хотя естественно было бы предположить, что последовательный метод доступа используется только для обработки последовательно организованных наборов данных, на самом деле это

не так. Набор данных, размещенный на устройстве прямого доступа, может обрабатываться последовательно, независимо от организации данных. В общем, однако, суть последовательного метода доступа заключается в организации последовательного обращения к потоку данных (следующим друг за другом блокам или записям) независимо от того, расположены или нет данные в логической последовательности величин или идентификаторов.

С другой стороны, считается, что последовательный набор данных должен быть упорядочен по признаку возрастания или уменьшения конкретных идентификаторов или величин. Кроме того, последовательный метод доступа может сформировать наборы данных, не организованные последовательно и впоследствии обрабатываемые совсем другим методом доступа в соответствии с форматом, который он имеет.

Если последовательный метод доступа с очередями (QSAM) используется в сочетании с групповой буферизацией, достигается наиболее эффективное совмещение работы центрального процессора и устройств ввода-вывода. Если проблемная программа задает три буферные области, каждая из которых будет содержать блок данных, состоящий из 20 записей, физическая операция ввода-вывода будет совершаться для каждых 20 обрабатываемых записей — каждый раз используются все данные одного буфера. Это будет означать, что к моменту завершения физической операции ввода-вывода проблемная программа все еще располагает записями для обработки и последний освобожденный к этому времени буфер (буферы) может быть снова заполнен данными и поставлен в очередь на использование. Следует заметить, однако, что каждый буфер требует выделения памяти в разделе, в котором находится программа, — три 2000-байтовых буфера займут 6000 байтов памяти. Следует решить, что важнее: время обработки или ресурсы основной памяти.

Д. ИНДЕКСНО-ПОСЛЕДОВАТЕЛЬНЫЕ МЕТОДЫ ДОСТУПА

Индексно-последовательные наборы данных организованы по признаку возрастания или убывания конкретных величин или идентификаторов, связанных с каждой записью. При занесении каждого блока записей или записи система снабжает блок данных ключом-идентификатором. Ключ содержит максимальный по величине идентификатор для записи в этом блоке. В случае логически последовательного расположения записей ключ будет содержать идентификатор последней записи в блоке.

Для того чтобы можно было легко отыскать конкретную запись, программы управления данными формируют во время

построения набора данных до трех уровней индекса. Это индекс дорожек, индекс цилиндров и главный индекс. Уровень главного индекса оказывается полезным только тогда, когда набор данных особенно велик. Путем использования параметров языка управления заданиями любой индекс, отличный от индекса дорожек, можно поместить в середину набора данных, в начало или конец набора данных или на отдельном томе.

Для конкретности предположим, что речь идет о некоем наборе данных, содержащем главный файл деталей. Файл был построен в порядке последовательного возрастания от записи с наименьшим по величине до записи с наибольшим по величине номером детали. При формировании набора данных, когда записывается блок записей с номерами деталей, система создает ключ, содержащий наибольший по величине номер детали для этого блока данных, а затем записывает этот ключ непосредственно перед этим блоком данных.

Для каждого формируемого цилиндра данных метод доступа системы генерирует на первой дорожке цилиндра *индекс дорожек*. В нем имеется элемент для каждой дорожки этого цилиндра. В нашем примере такой элемент наряду с адресом дорожки содержит ключ с наибольшим по значению номером детали на этой дорожке. Если набор данных сформирован на устройстве IBM 2314, содержащем 20 дорожек на цилиндр, то 19 дорожек будет выделено для записей номеров деталей и одна дорожка — для индекса дорожек.

Для каждого индекса дорожек (охватывающего записи данных на одном полном цилиндре) система формирует элемент для *индекса цилиндров*. Индекс цилиндров содержит один элемент для каждого цилиндра, занимаемого набором данных. Такой элемент содержит адрес индекса дорожек и наибольший по величине номер детали, связанный с этим индексом.

Если набор данных так велик, что индекс цилиндров занимает много дорожек, программист имеет возможность сформировать уровень главного индекса. В соответствии со ступенчатой структурой индексов низшего уровня главный индекс может содержать элемент для каждой дорожки, на которой записаны элементы индекса цилиндров. Этот элемент дает адрес каждой дорожки индекса цилиндров и наибольший по величине номер детали, содержащийся на этой дорожке. Уровень главного индекса может также состоять из трех уровней индексов, если набор данных достаточно велик и программист полагает, что это необходимо. Каждый уровень главного индекса образует элементы для следующего, более высокого уровня этого индекса: число уровней не должно превышать трех.

При описании процедуры выборки записи допустим, что искомая запись идентифицируется номером детали 32967. Поиск

этой записи по индексам, начиная с главного, состоит из следующих этапов:

1. Нахождение в главном индексе адреса той дорожки индекса цилиндров, на которой наибольший номер детали превышает 32967 или равен этому числу; поиск в главном индексе ведется от меньших значений к большим.

2. Переход к найденной дорожке индекса цилиндров и поиск в ней такого элемента индекса дорожек, в котором наибольший по величине номер детали превышает 32967 или равен этому числу; поиск ведется от меньших значений к большим.

3. Переход к найденной дорожке в индексе дорожек и поиск в ней адреса дорожки с данными, на которой наибольший по величине номер детали превышает 32967 или равен этому числу; поиск ведется от меньших значений к большим.

4. Поиск по ключу на дорожке блока данных, в котором наибольший номер детали превышает 32967 или равен этому числу; поиск ведется от меньших значений к большим.

5. Извлечение этого блока данных и нахождение записи данных с номером детали 32967.

Хотя эта последовательность действий, направленная на нахождение одной записи, может показаться довольно длинной, она вполне целесообразна. Этот вид обращения к файлу имеет особое преимущество, когда нужно найти «головную» запись, а затем группу непосредственно следующих за ней «хвостовых».

Индексно-последовательный метод доступа обладает следующими возможностями:

1. Набор данных можно считывать и записывать как последовательно, так и произвольно.

2. Отдельные записи можно обрабатывать в любом порядке.

3. Записи могут быть ликвидированы.

4. Можно добавить новые записи.

5. Новые записи можно автоматически размещать по логическому признаку внутри набора данных.

6. Если при внесении новой записи в соответствующую область происходит переполнение дорожки, то предусмотрена автоматическая обработка этой ситуации.

Варианты индексно-последовательного метода доступа — базисный (BISAM) и с очередями (QISAM) — используются в сочетании друг с другом с целью повышения общей производительности.

Эти варианты метода доступа обладают следующими специфическими возможностями:

1. Создание набора данных — только QISAM.

2. Обработка и обновление записей данных — QISAM и BISAM.

3. Внесение новых записей — только BISAM.

Частое внесение новых записей, как правило, приводит к использованию многочисленных областей переполнения, что вызывает резкое замедление выборки записей. Поэтому следует периодически реорганизовывать индексно-последовательные наборы данных, учитывая размер файла и частоту обращений к нему.

Е. БИБЛИОТЕЧНЫЙ МЕТОД ДОСТУПА

Этот метод доступа используется для создания библиотечных наборов данных и обращения к ним; библиотечные наборы данных могут быть построены только на устройствах прямого доступа.

Так как библиотечный набор данных логически последователен, а физически случаен, он имеет свойства как индексно-последовательных наборов данных, так и последовательно организованных наборов данных. Его часто используют для хранения групп программ, часто называемых библиотеками. Библиотечный набор данных состоит из двух основных частей: справочника и области, содержащей разделы набора данных.

Разделы набора данных

Раздел набора данных представляет собой набор записей, к которому внутри набора данных производится обращение по символической метке или имени. Один или несколько разделов вместе со справочником составляют библиотечный набор данных. Записи внутри раздела расположены последовательно, а сами разделы (по всему набору данных) могут быть расположены случайно. Для того чтобы по данному конкретному имени раздела сразу найти его местоположение, система строит справочник, что очень похоже на простую индексацию. Ссылаясь на справочник, помещаемый в самом начале набора данных, система может найти местоположение раздела в наборе данных. Раздел может быть исключен, обновлен или добавлен.

Однако когда раздел исключается из набора данных, то ликвидируется, по существу, только элемент справочника. Пространство, которое занимал этот раздел, нельзя вновь использовать до тех пор, пока набор данных не будет полностью перестроен.

Справочник

Сам справочник состоит из ряда 256-байтовых блоков. Каждый блок состоит из записей переменной длины, которые можно рассматривать как элементы, подобные элементам индекса. Из 256 байтов, отведенных для одного блока справочника, до 254

байтов могут использоваться для записей. Каждому блоку непосредственно предшествует восьмибайтовое поле, содержащее имя раздела последней записи блока. Каждая запись (элемент) 256-байтового блока справочника содержит:

1. Поле с именем соответствующего раздела.

2. Трехбайтовое поле, содержащее адрес первой записи этого раздела. Этот адрес представляет собой относительный номер дорожки (относительно начала набора данных) и относительный номер блока на этой дорожке.

3. Однобайтовое поле в двоичном формате, содержащее:

а) код, показывающий, являются ли данные в поле имени «псевдонимом» для этого раздела;

б) число полуслов данных пользователя в этом элементе;

в) индикатор того, имеется ли в данных пользователя один или несколько указателей на этот раздел.

4. До 62 байтов данных пользователя, занимающих целое число полуслов, относящихся к указателям на дополнительные входы в этот раздел.

Ввиду специфических особенностей библиотечного набора данных операционная система предоставляет для обработки элементов справочника и разделов следующие специальные макрокоманды:

BLDL — используется для составления списка элементов справочника;

FIND — находит начальный адрес заданного раздела и помещает этот адрес в блок управления данными с тем, чтобы последующая макрокоманда READ обращалась к этому разделу;

STOW — добавляет в существующий справочник элемент для некоторого раздела или же исключает, заменяет или изменяет имя раздела в элементе справочника.

Выборка раздела может осуществляться методами доступа QSAM, BSAM или BPAM.

Ж. ГРАФИЧЕСКИЙ МЕТОД ДОСТУПА

Графический метод доступа — это набор макрокоманд и управляющих программ, которые обеспечивают взаимодействие операционной системы, устройств управления и дисплеев.

К числу этих устройств относится графический дисплей IBM 2250 и алфавитно-цифровой дисплей IBM 2260.

Фактически эти дисплеи являются просто устройствами ввода-вывода другой категории, чем те, которые должны использоваться как часть вычислительной системы. Их специфика заключается в том, что как входные, так и выходные данные могут обрабатываться под визуальным контролем с подключением центрального процессора на короткие промежутки вре-

мени. Данные вводятся, выводятся и обрабатываются в мультипрограммном режиме с минимальным ущербом для обычных программ, выполняющихся в этот момент в системе.

Для того чтобы без участия центрального процессора поддержать изображение на экране электронно-лучевой трубки, применяется память на линиях задержки. Данные, обрабатываемые с помощью клавиатуры дисплея, или данные, передаваемые от центрального процессора на устройство управления, содержатся в определенной последовательности буферов устройства управления или в управляющем блоке дисплея. Содержимое этих буферов непрерывно записывается на экран электронно-лучевой трубки с интервалами, достаточно частыми для предотвращения угасания или мигания изображения. Высвечиваемые данные остаются неизменными до тех пор, пока не будут стерты или изменены с помощью клавиатуры устройства или под управлением программы.

Эти действия требуют единого метода управления, а именно графического метода доступа. Но в этом смысле графический метод доступа не просто метод доступа для опроса и создания наборов данных способом, который для него характерен. Правильнее считать его средством доступа к устройству, которое в свою очередь может производить доступ к различным наборам данных под управлением программы.

К числу трех главных функций графического метода доступа относятся: управление графическими данными, управление вводом-выводом и отработка сигнала внимания.

Управление графическими данными включает управление буфером и запись графических приказов и данных в указанные пользователем области вывода. Оно несет ответственность за распространение и освобождение буферов основной памяти, содержимого памяти на линиях задержки и размещение данных в буферах.

Функции управления вводом-выводом состоят в выполнении макрокоманд OPEN, CLOSE и инициализации передачи данных по запросам различных макрокоманд чтения и записи, относящихся к графическим дисплеям. Эти действия включают подготовку управления блоков полей данных и областей памяти, необходимых для передачи данных между центральным процессором и дисплеями. Подготовка слов управления каналом и составление канальных программ также являются функциями управления вводом-выводом.

Программы обработки сигнала внимания бывают двух видов — базисные и срочные. Основное назначение сигнала внимания — это указать, на какой дисплей должен обратить внимание центральный процессор для передачи данных или ввода запроса для ответа.

Работу двух типов программ обработки сигнала внимания можно сравнить с базисным методом доступа и методом доступа с очередями. При срочной обработке сигнала внимания системой осуществляется большая часть обработки ошибок, анализа данных и функций управления, аналогично тому, как эти функции выполняются методами доступа с очередями. Базисная обработка сигналов внимания требует больше операций и непосредственного контроля за этими функциями со стороны проблемной программы, подобно усилиям, затрачиваемым при базисном методе доступа. Однако из этого не следует делать вывод, что для упрощения написания проблемной программы всегда следует пользоваться срочной обработкой сигнала внимания. Проблемная программа может реализовывать специальный алгоритм, который требует прямой обработки сигнала внимания; в этом случае используются программы базисной обработки этого сигнала.

Возможности макрокоманд графического метода доступа и управляющих программ предлагают широкую, гибкую основу для построения проблемных программ. Многие варианты чтения и вывода на экран данных возможны благодаря заданию соответствующих параметров при кодировании макрокоманд. Например, макрокоманда GREAD позволяет делать выбор из шести различных операций чтения. Использование этого многообразия макрокоманд и управляющих программ полностью зависит от требований проблемной программы, а также конфигурации и режима работы дисплеев и вычислительной машины.

Следует отметить, что дисплеи не всегда управляются проблемной программой посредством макрокоманд. Некоторые устройства, такие, как алфавитно-цифровой дисплей IBM 2260, можно с успехом использовать как удаленный терминал. Применение в качестве удаленного терминала заключается в передаче данных между дисплеем и вычислительной машиной по линиям связи (например, по телефонным линиям) в противоположность локальному режиму, при котором дисплей и устройство управления связаны с центральным процессором через канал (по кабелю). Хотя дисплей, в конечном счете, функционирует в локальном режиме, так же как в дистанционном, в последнем случае передачей промежуточных данных управляет телекоммуникационный метод доступа.

3. ТЕЛЕКОММУНИКАЦИОННЫЕ МЕТОДЫ ДОСТУПА

В силу исключительно большого числа характеристик, логических модулей, макрокоманд и управляющих модулей, относящихся к телекоммуникационному методу доступа, анализ этого

метода проведен только в самых общих чертах. Сделана попытка рассмотреть обычные возможности методов, не вдаваясь детально в варианты выполняемых ими функций.

Под телекоммуникацией в применении ее в вычислительных машинах можно понимать обмен и передачу данных по линиям связи между удаленным терминалом и центральным процессором вычислительной системы. Телекоммуникационные методы доступа операционной системы OS используются для управления передачей данных в этих условиях. Эти методы доступа обеспечивают работу, например, следующих терминалов: системы сбора данных IBM 1030, группового алфавитно-цифрового дисплея IBM 2848/2260 и терминала связи IBM 2740. Эти устройства должны связываться с центральным процессором вычислительной машины посредством специальных адаптеров и мультиплексоров передачи данных. Так же как и другие методы доступа, телекоммуникационный метод доступа имеет варианты: базисный и с очередями. Эти варианты различаются так же, как соответствующие варианты других методов доступа: вариант метода доступа с очередями выполняет многие функции автоматически, в то время как базисный метод требует, чтобы соответствующие функции выполняла сама проблемная программа.

Основная функция телекоммуникационного метода доступа — это управление передачей, приемом, анализом единиц работы, которые называются сообщениями. При этом необходимо анализировать ситуации, возникающие при взаимодействии линий связи, проверять занятость и доступность устройства и правильность передачи данных вообще. И все это — в дополнение к обычным функциям метода доступа: управлению вводом-выводом, управлению буферами, инициализацией программ чтения и записи, опросу, адресации и обработке сообщений, которые предоставляются программисту в виде обширного списка макрокоманд.

Одной из специфичных задач, выполняемых телекоммуникационным методом доступа, является опрос. Он представляет собой последовательность периодических действий, предпринимаемых центральным процессором или управляющим терминалом, в ответ на которые удаленные терминалы сообщают, находится ли какой-то терминал в состоянии ожидания возможности передать данные. Опрос заключается в том, что устройство управления периодически передает определенные сочетания символов, и в зависимости от того, каковы эти символы, они передаются определенному терминалу или компонентам терминала. Если этот терминал или компонент хранит сообщение для передачи на устройство управления, он отвечает закодированным «да», в противном случае он отвечает закодированным

«нет». В этом режиме работы устройство управления организует по определенной периодической системе выбор терминалов, которые могут передавать сообщение. Противоположным режимом передачи данных является тот, при котором всегда, когда терминал имеет данные для передачи, он пытается получить линию связи в монопольное пользование. Последний режим работы, известный под названием режим *соперничества*, может неявным образом создавать неблагоприятные задержки в передаче от некоторого терминала, если его запрос на управление линией связи постоянно «оттесняется» другими терминалами.

Устройство управления и терминал могут производить другой вариант опроса, известный как коммутация сообщений. В этом случае, когда осуществляется связь устройство управления — терминал или терминал — терминал, первое устройство передает несколько определенных сочетаний символов, указывающих на вызываемый терминал и извещающих этот терминал, что посылающее устройство имеет сообщение для передачи. Получающий терминал в свою очередь дает ответ, свободен ли он в этот момент для приема сообщения.

Очевидно, что телекоммуникационные методы доступа, конечно же, более сложные, чем обычные методы доступа, используемые в вычислительных системах. Однако правильное использование этих методов позволяет обслуживать сеть «сателлитов», обеспечивающих дистанционный ввод данных, диалог, планирование и передачу сообщений, а также возможность дистанционной обработки заданий, известную как дистанционный ввод заданий.

ЧАСТЬ V

Таблицы и глоссарий

Таблицы

А. ТАБЛИЦА ЗНАЧЕНИЙ ДВОИЧНЫХ РАЗРЯДОВ ИЛИ СТЕПЕНЕЙ ЧИСЛА 2

Двоичный разряд	Степень числа 2	Десятичное значение двоичного разряда	Двоичный разряд	Степень числа 2	Десятичное значение двоичного разряда
1	0	1	17	16	65536
2	1	2	18	17	131072
3	2	4	19	18	262144
4	3	8	20	19	524288
5	4	16	21	20	1048576
6	5	32	22	21	2097152
7	6	64	23	22	4194304
8	7	128	24	23	8388608
9	8	256	25	24	16777216
10	9	512	26	25	33554432
11	10	1024	27	26	67108864
12	11	2048	28	27	134217728
13	12	4096	29	28	268435456
14	13	8192	30	29	536870912
15	14	16384	31	30	1073741824
16	15	32768	32	31	2147483648

Считается, что двоичные разряды перенумерованы справа налево, от младших к старшим.

Величина степени числа 2 для заданного двоичного разряда считается фиктивной, если он не содержит единицы.

Б. ТАБЛИЦА ПРЕОБРАЗОВАНИЯ ШЕСТНАДЦАТЕРИЧНЫХ ЧИСЕЛ В ДЕСЯТИЧНЫЕ

Четырехбайтовое полное слово

Полуслово № 2								Полуслово № 1							
Байт 4				Байт 3				Байт 2				Байт 1			
Полубайт 8		Полубайт 7		Полубайт 6		Полубайт 5		Полубайт 4		Полубайт 3		Полубайт 2		Полубайт 1	
шестн.	десятичное	шестн.	десятичное	шестн.	десятичное	шестн.	десятичное	шестн.	десятичное	шестн.	десятичное	шестн.	десятичное	шестн.	десятичное
1	268 435 456	1	16 777 216	1	1 048 576	1	65 536	1	4 096	1	256	1	16	1	1
2	536 870 912	2	33 554 432	2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	805 306 368	3	50 331 648	3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	1 073 741 824	4	67 108 864	4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	1 342 177 280	5	83 886 080	5	5 242 880	5	327 680	5	20 480	5	1 280	5	80	5	5
6	1 610 612 736	6	100 663 296	6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	1 897 048 192	7	117 440 512	7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	2 147 483 648	8	134 217 728	8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	2 415 919 104	9	150 994 944	9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	2 684 354 560	A	167 772 160	A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	2 952 790 016	B	184 549 376	B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	3 221 225 472	C	201 326 592	C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	3 489 660 928	D	218 103 808	D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	3 758 096 384	E	234 881 024	E	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	4 026 531 840	F	251 658 240	F	15 728 640	F	983 040	F	61 440	F	3 840	F	240	F	15

В. ТАБЛИЦА ПРИЗНАКОВ РЕЗУЛЬТАТА

Данная таблица содержит значения признака результата, которые устанавливаются в результате выполнения команд языка Ассемблера. Заглавие каждого столбца результата представляет собой значение маски, которая соответствует конфигурации битов поля признака результата в PSW. Эти значения маски используются в операторе команды, например BC 8, LABEL. Указанный признак относится к результату в первом операнде или к результату сравнения первого операнда со вторым

Команда	Мнемоника	Маска=8	Маска=4	Маска=2	Маска=1
Вычитание	S	нуль	отриц.	полож.	перен.
Вычитание	SR	нуль	отриц.	полож.	перен.
Вычитание десятичное	SP	нуль	отриц.	полож.	перен.
Вычитание кодов	SL	—	не нуль, нет переноса	нуль, перенос	не нуль, перенос
Вычитание кодов	SLR	—	не нуль, нет переноса	нуль, перенос	не нуль, перенос
Вычитание полуслова	SH	нуль	отриц.	полож.	перен.
Загрузка дополнения	LCR	нуль	отриц.	полож.	перен.
Загрузка и проверка	LTR	нуль	отриц.	полож.	—
Загрузка отрицательная	LNR	нуль	отриц.	—	—
Загрузка положительная	LPR	нуль	—	полож.	перен.
И	N	нуль	не нуль	—	—
И	NC	нуль	не нуль	—	—
И	NR	нуль	не нуль	—	—
И непосредственное	NI	нуль	не нуль	—	—
ИЛИ	O	нуль	не нуль	—	—
ИЛИ	OC	нуль	не нуль	—	—
ИЛИ	OR	нуль	не нуль	—	—
ИЛИ непосредственное	OI	нуль	не нуль	—	—
Исключающее ИЛИ	X	нуль	не нуль	—	—
Исключающее ИЛИ	XC	нуль	не нуль	—	—

Продолжение

Команда	Мнемоника	Маска=8	Маска=4	Маска=2	Маска=1
Исключающее ИЛИ	XR	нуль	не нуль	—	—
Исключающее ИЛИ непосредственное	XI	нуль	не нуль	—	—
Отредактировать	ED	нуль	отриц.	полож.	—
Отредактировать и отметить	EDMK	нуль	отриц.	полож.	—
Перекодировать и проверить	TRT	нуль	неполное	полн.	—
Проверить по маске	TM	все биты нулевые	смешанные биты	—	все биты единичные
Сдвиг влево арифметический	SLA	нуль	отриц.	полож.	переп.
Сдвиг влево двойной арифметический	SLDA	нуль	отриц.	полож.	переп.
Сдвиг вправо арифметический	SRA	нуль	отриц.	полож.	—
Сдвиг вправо двойной арифметический	SRDA	нуль	отриц.	полож.	—
Сложение	A	нуль	отриц.	полож.	переп.
Сложение	AR	нуль	отриц.	полож.	переп.
Сложение десятичное	AP	нуль	отриц.	полож.	переп.
Сложение кодов	AL, ALR	нуль, нет переноса	не нуль, нет переноса	нуль, перенос	не нуль, перенос
Сложение полуслова	AH	нуль	отриц.	полож.	переп.
Сложение с очисткой	ZAP	нуль	отриц.	полож.	переп.
Сравнение	C	равно	меньше	больше	—
Сравнение	CR	равно	меньше	больше	—
Сравнение десятичное	CP	равно	меньше	больше	—
Сравнение кодов	CL	равно	меньше	больше	—
Сравнение кодов	CLC	равно	меньше	больше	—
Сравнение кодов	CLR	равно	меньше	больше	—
Сравнение непосредственное	CLI	равно	меньше	больше	—
Сравнение полуслова	CH	равно	меньше	больше	—

Г. ТАБЛИЦА УСЛОВИЙ ПРОГРАММНОГО ПРЕРЫВАНИЯ

Команда	Мнемоника	Возможные программные прерывания					
		адр.	спец.	переп.	защ.	опер.	прочие
Вычитание	S	X	X	F	X		
Вычитание	SR			F			
Вычитание десятичное	SP	X		D	X	X	Данные
Вычитание кодов	SL	X	X		X		
Вычитание полуслова	SH	X	X	F	X		
Деление	D	X	X		X		F
Деление	DR		X				F
Деление десятичное	DP	X	X		X	X	Данные, D
Загрузка	L	X	X		X		
Загрузка групповая	LM	X	X		X		
Загрузка дополнения	LCR			F			
Загрузка положительная	LPR			F			
Загрузка полуслова	LH	X	X		X		
Запись в память	ST	X	X		X		
Запись в память групповая	STM	X	X		X		
Запись в память полуслова	STH	X	X		X		
Запись в память символа	STC	X			X		
И	N	X	X		X		
И	NC	X			X		
И непосредственное	NI	X			X		
ИЛИ	O	X	X		X		
ИЛИ	OC	X			X		
ИЛИ непосредственное	OI	X			X		
Исключающее ИЛИ	X	X	X		X		
Исключающее ИЛИ	XC	X			X		
Исключающее ИЛИ непосредственное	XI	X			X		
Отредактировать	ED	X			X	X	Данные
Отредактировать и отметить	EDMK	X			X	X	Данные
Перекодировать	TR	X			X		
Перекодировать и проверить	TRT	X			X		
Пересылка зон	MVZ	X			X		
Пересылка непосредственная	MVI	X			X		
Пересылка символов	MVC	X			X		
Пересылка со сдвигом	MVO	X			X		

Продолжение

Команда	Мнемоника	Возможные программные прерывания					
		адр.	спец.	перен.	защ.	опер.	прочие
Пересылка цифр	MVN	X			X		
Преобразование в двоичную	CVB	X	X		X		Данные, F
Преобразование в десятичную	CVD	X	X		X		
Проверить по маске	TM	X			X		
Прочитать символ	IC	X			X		
Распаковать	UNPK	X					
Сдвиг влево арифметический	SLA			F	X		
Сдвиг влево двойной арифметический	SLDA		X	F	X		
Сдвиг влево двойной кода	SLDL		X				
Сдвиг вправо двойной арифметический	SRDA		X				
Сдвиг вправо двойной кода	SRDL		X				
Сложение	A	X	X	F	X		
Сложение	AR			F			
Сложение десятичное	AP	X		D			
Сложение кодов	AL	X	X		X	X	Данные
Сложение полуслова	AH	X	X	F	X		
Сложение с очисткой	ZAP	X		D	X	X	Данные
Сравнение	C	X	X		X		
Сравнение десятичное	CP	X			X		
Сравнение кодов	CL	X	X		X	X	Данные
Сравнение кодов	CLC	X			X		
Сравнение кодов	CLR						
Сравнение кодов непосредственное	CLI	X			X		
Сравнение полуслова	CH	X	X		X		
Умножение	M	X	X		X		
Умножение	MR		X				
Умножение десятичное	MP	X	X		X	X	Данные
Умножение полуслова	MH	X	X		X		
Упаковать	PACK	X			X		

Пояснения:

адр. — адресация
спец. — спецификация
перен. — переполнение
D — десятичное
E — порядка
F — с фиксированной точкой
защ. — защита памяти
опер. — операция
прочие: D — десятичное деление
F — деление с фиксированной точкой

Д. ТАБЛИЦА СИМВОЛОВ КОДА EBCDIC

Эта таблица содержит графическое представление букв, цифр и прочих символов, входящих в набор символов кода EBCDIC. Малые (строчные) буквы не являются стандартными, но их можно применять в качестве стандартных на АЦПУ и графических устройствах

Шестн. представление	Начертание	Битовая конфиг.	Шестн. представление	Начертание	Битовая конфиг.	Шестн. представление	Начертание	Битовая конфиг
40	(пробел)	0100 0000	84	d	1000 0100	C8	H	1100 1000
4A	¢	0100 1010	85	e	1000 0101	C9	I	1100 1001
4B	.	0100 1011	86	f	1000 0110	D1	J	1101 0001
4C	<	0100 1100	87	g	1000 0111	D2	K	1101 0010
4D	(	0100 1101	88	h	1000 1000	D3	L	1101 0011
4E	+	0100 1110	89	i	1000 1001	D4	M	1101 0100
4F	|	0100 1111	91	j	1001 0001	D5	N	1101 0101
50	&	0101 0000	92	k	1001 0010	D6	O	1101 0110
5A	!	0101 1010	93	l	1001 0011	D7	P	1101 0111
5B	$	0101 1011	94	m	1001 0100	D8	Q	1101 1000
5C	*	0101 1100	95	n	1001 0101	D9	R	1101 1001
5D	)	0101 1101	96	o	1001 0110	E2	S	1110 0010
5E	;	0101 1110	97	p	1001 0111	E3	T	1110 0011
5F	¬	0101 1111	98	q	1001 1000	E4	U	1110 0100
60	-	0110 0000	99	r	1001 1001	E5	V	1110 0101
61	/	0110 0001	A2	s	1010 0010	E6	W	1110 0110
6B	,	0110 1011	A3	t	1010 0011	E7	X	1110 0111
6C	%	0110 1100	A4	u	1010 0100	E8	Y	1110 1000
6D	_	0110 1101	A5	v	1010 0101	E9	Z	1110 1001
6E	>	0110 1110	A6	w	1010 0110	F0	0	1111 0000
6F	?	0110 1111	A7	x	1010 0111	E1	1	1111 0001
7A	:	0111 1010	A8	y	1010 1000	F2	2	1111 0010
7B	#	0111 1011	A9	z	1010 1001	F3	3	1111 0011
7C	@	0111 1100	C1	A	1100 0001	F4	4	1111 0100
7D	'	0111 1101	C2	B	1100 0010	F5	5	1111 0101
7E	=	0111 1110	C3	C	1100 0011	F6	6	1111 0110
7F	"	0111 1111	C4	D	1100 0100	F7	7	1111 0111
81	a	1000 0001	C5	E	1100 0101	F8	8	1111 1000
82	b	1000 0010	C6	F	1100 0110	F9	9	1111 1001
83	c	1000 0011	C7	G	1100 0111			

Е. ТАБЛИЦА ШЕСТНАДЦАТЕРИЧНЫХ КОДОВ КОМАНД И ИХ МНЕМОНИКИ НА ЯЗЫКЕ АССЕМБЛЕРА

Шестн. код	Мнемоника	Команда
05	BALR	Переход с возвратом
06	BCTR	Переход по счетчику
07	BCR	Условный переход
10	LPR	Загрузка положительная
11	LNR	Загрузка отрицательная
12	LTR	Загрузка и проверка
13	LCR	Загрузка дополнения
14	NR	И (формат RR)
15	CLR	Сравнение кодов
16	OR	ИЛИ (формат RR)
17	XR	Исключающее ИЛИ (формат RR)
18	LR	Загрузка
19	CR	Сравнение
1A	AR	Сложение
1B	SR	Вычитание
1C	MR	Умножение
1D	DR	Деление
40	STH	Запись в память полуслова
41	LA	Загрузка адреса
42	STC	Запись в память символа
43	IC	Прочитать символ
45	BAL	Переход с возвратом
46	BCT	Переход по счетчику
47	BC	Условный переход
48	LH	Загрузка полуслова
49	CH	Сравнение полуслова
4A	AH	Сложение полуслова
4B	SH	Вычитание полуслова
4C	MH	Умножение полуслова
4E	CVD	Преобразование в десятичную
4F	CVB	Преобразование в двоичную
50	ST	Запись в память
54	N	И (формат RX)
55	CL	Сравнение кодов (формат RX)
56	O	ИЛИ (формат RX)
57	X	Исключающее ИЛИ (формат RX)
58	L	Загрузка
59	C	Сравнение
5A	A	Сложение
5B	S	Вычитание
5C	M	Умножение

Продолжение

Шестн. код	Мнемоника	Команда
5D	D	Деление
5E	AL	Сложение кодов
5F	SL	Вычитание кодов
86	BXH	Переход по индексу больше
87	BXLE	Переход по индексу меньше или равно
88	SRL	Сдвиг вправо кода
89	SLL	Сдвиг влево кода
8A	SRA	Сдвиг вправо арифметический
8B	SLA	Сдвиг влево арифметический
8C	SRDL	Сдвиг вправо двойной кода
8D	SLDL	Сдвиг влево двойной кода
8E	SRDA	Сдвиг вправо двойной арифметический
8F	SLDA	Сдвиг влево двойной арифметический
90	STM	Запись в память групповая
91	TM	Проверить по маске
92	MVI	Пересылка непосредственная
94	NI	И непосредственное (формат SI)
95	CLI	Сравнение кодов непосредственное
96	OI	ИЛИ непосредственное (формат SI)
97	XI	Исключающее ИЛИ непосредственное (формат SI)
98	LM	Загрузка групповая
D1	MVN	Пересылка цифр
D2	MVC	Пересылка символов
D3	MVZ	Пересылка зон
D4	NC	И (формат SS)
D5	CLC	Сравнение кодов
D6	OC	ИЛИ (формат SS)
D7	XC	Исключающее ИЛИ (формат SS)
DC	TR	Перекодировать
DD	TRT	Перекодировать и проверить
DE	ED	Отредактировать
DF	EDMK	Отредактировать и отметить
F1	MVO	Пересылка со сдвигом
F2	PACK	Упаковать
F3	UNPK	Распаковать
F8	ZAP	Сложение с очисткой
F9	CP	Сравнение десятичное
FA	AP	Сложение десятичное
FB	SP	Вычитание десятичное
FC	MP	Умножение десятичное
FD	DP	Деление десятичное

Глоссарий

Адрес — абсолютный идентификатор места в памяти.

Алфавитно-цифровое печатающее устройство (АЦПУ) — любое из группы устройств, выводящих информацию в печатной форме. Эта группа включает и пишущую машинку, однако термин «АЦПУ» чаще применяется к устройствам, способным печатать со скоростью несколько слов в секунду, в конструкции которых используется рейка или цепь.

Алфавитно-цифровой — состоящий из смеси букв и цифр, обычно представленных в форме 8-разрядных символов.

Аппаратные средства — блоки и устройства, составляющие вычислительную систему. К ним относятся, например, блок центрального процессора, запоминающие устройства, АЦПУ и соответствующие устройства управления.

Базисный библиотечный метод доступа (Basic Partitioned Access Method — BPAM) — один из методов доступа Системы/360.

Базисный индексно-последовательный метод доступа (Basic Indexed Sequential Access Method — BISAM) — базисная версия метода доступа, который обеспечивает непрерывно-последовательную, случайно-последовательную или случайную обработку записей данных. Метод доступа строит структуру индексов, используя ключи, сопоставленные с каждой записью.

Базисный последовательный метод доступа (Basic Sequential Access Method — BSAM) — один из методов доступа Системы/360.

Базисный прямой метод доступа (Basic Direct Access Method — BDAM) — метод, используемый для прямой или случайной выборки записей данных из набора данных, который расположен на запоминающем устройстве с прямым доступом.

Базовая операционная система (Basic Operating System — BOS) — одна из операционных систем Системы/360.

Байт — мера памяти в Системе/360. Байт состоит из 8 двоичных разрядов, что соответствует длине символа в коде EBCDIC. Байт может содержать любую из 256 битовых конфигураций. Он является наименьшим адресуемым элементом памяти.

Барабан — устройство прямого доступа, такое, например, как IBM 2301 или IBM 2303. Данные записываются на внешней поверхности цилиндрического барабана, для каждой дорожки имеется неподвижная головка для чтения или записи.

Безусловный переход — команда перехода, которая всегда вызывает при своем выполнении переход. Естественный поря-

док выполнения команд никогда не сохраняется, за исключением случая, когда предусмотрен переход к следующей команде.

Библиотечный набор данных — набор данных, построенный посредством программы базисного библиотечного метода доступа (BPAM). Библиотечный набор данных состоит из справочника и разделов (групп записей). Элементы справочника указывают местоположение разделов в наборе данных.

Бит — наименьший компонент данных любого типа. Сочетания битов образуют двоичную систему представления данных. Один бит может быть представлен единицей или нулем в двоичном разряде.

Бит единичный — бит, установленный в единицу. Двоичная позиция, содержащая цифру 1.

Бит нулевой — бит, установленный в нуль. Двоичная позиция, содержащая цифру 0.

Блок записей — группа записей, которые при занесении их на запоминающее устройство выступают как один физический сегмент данных. Определенные программы управления данными объединяют записи в блоки и извлекают из блоков отдельные записи.

Блок укороченный — блок данных, который содержит меньше логических записей или байтов, чем задано для блоков данного набора данных.

Блок управления — совокупность специальных параметров, формируемая и используемая операционной системой.

Блок управления данными (Data Control Block — DCB) — блок управления в операционной системе, содержащий идентификационные признаки набора данных. В качестве параметров блока управления данными программист может задать длину и формат записей, фактор блокировки, методы буферизации, использование некоторых системных средств и т. п.

Блок управления набором данных (Data Set Control Block — DSCB) — элемент оглавления тома на запоминающем устройстве прямого доступа. Содержит информацию о местоположении и организации набора данных. Каждый набор данных имеет свой блок управления в оглавлении тома.

Вызов — действие программы, которое состоит в запросе, извлечении или активизации внешнего модуля или подпрограммы. Независимая программа или подпрограмма может быть доступна любой программе и вызываться ею.

Выравнивание по границе — размещение полей фиксированной длины по границе полуслова, полного слова или двойного слова. Адрес начала поля, выравненного по границе, кратен соответственно 2, 4 или 8.

Генерация системы — процесс построения операционной системы или ее варианта; формирование программ управления

данными, программ супервизора и других системных программ в соответствии с конкретным составом технических средств, для обслуживания которых предназначена генерируемая операционная система.

Головная метка — метка набора данных, помещаемая в начале набора данных или в начале продолжения набора данных, расположенного на одном или нескольких томах запоминающего устройства.

Граница двойного слова — адрес памяти, кратный числу 8.

Граница полного слова — адрес памяти, кратный числу 4.

Граница полуслова — адрес памяти, кратный 2.

Графический метод доступа (Graphic Access Method—GAM)— совокупность макрокоманд, используемых в операционной системе OS/360 для программирования терминала с электронно-лучевой трубкой, такого, как IBM 2250.

Дамп основной памяти — распечатка данных, содержащихся в основной памяти в данный момент времени. Обычно выполняется каждый раз, когда происходит аварийное завершение программы, и используется затем программистом для анализа области проблемной программы с целью нахождения причины аварийного завершения.

Двоичный — относящийся к позиционной системе счисления по основанию 2, в которой для представления чисел используются только нули и единицы.

— относящийся к представлению данных, при котором данные выражаются с помощью нулей и единиц. Двоичное представление могут иметь символьные, числовые и специальные данные.

Двойное слово — восьмибайтовое поле, адрес которого в памяти кратен 8. Хотя двойные слова могут содержать данные любого типа, они в первую очередь предназначены для арифметики с плавающей точкой.

Дисковая операционная система (Disk Operating System — DOS) — одна из операционных систем Системы/360.

Дисплей — устройство, применяемое для ввода и вывода данных в визуальной форме. Хотя под это определение подходит и пишущая машинка, обычно имеется в виду представление данных на электронно-лучевой трубке, подобной телевизионной.

Дополнение до 2 — величина, выраженная в форме двоичного отрицательного числа с фиксированной точкой.

Записи неопределенной длины — записи данных, имеющие неизвестную длину и формат.

Записи фиксированной длины — содержимое набора данных или группы записей данных, в которых все записи имеют одну и ту же фиксированную длину.

Запись фиктивная — форматная запись, псевдозапись. Искусственные записи, образуемые для придания набору данных определенного формата или заполнения его. Обычно используется при построении наборов данных или выборке посредством методов прямого доступа.

Знак — часть арифметического поля, содержимое которого определяет положительность или отрицательность величины, находящейся в этом поле.

Зонный формат — формат представления десятичных чисел в виде последовательности символов кода EBCDIC; младший разряд числа и знак представлены кодом буквы.

Индексно-последовательный метод доступа с очередями (Queued Indexed Sequential Access Method — QISAM) — метод доступа, или набор программ управления данными, которые обеспечивают создание наборов данных, имеющих индексно-последовательную организацию, и доступ к ним.

Индикатор значимости — логический переключатель, функционирующий в процессе редактирования. Может находиться в состояниях «включено» и «выключено». Если индикатор значимости выключен, происходит подавление нулей и символ-заполнитель записывается вместо незначащих цифр из исходного поля на место символа выбора цифры в образце для редактирования. Если индикатор значимости включен, цифры из исходного поля замещают символы выбора цифры в образце.

Инициализация программы — действия, обычно выполняемые в начале программы, такие, как инициализация базовых регистров, сохранение исходного содержимого регистров, инициализация областей данных и другие, которые программист считает целесообразными перед входом в основную часть программы.

Исходная колода — колода перфокарт, содержащая предложения некомпилированной программы, представленные в коде EBCDIC.

Исходное поле — поле данных, которое содержит информацию, символы или величины для выполнения операции или некоторой команды. Если данные пересылаются из поля B в поле A, то исходным полем является поле B.

K — символ десятичной величины 1024; выражение «4K» интерпретируется как десятичная величина 4096. Иногда используется для представления величины 1000.

Канал — канал передачи данных; электрический кабель для передачи импульсов, представляющих данные, между устройствами вычислительной системы.

Карта-ограничитель — карта, которая используется для указания о конце некоторого действия или конце входного набора данных на картах.

Ключ — поле идентификатора или сам идентификатор записи. Имя или величина, сопоставленная конкретной записи для ее идентификации или отождествления ее с другой записью.

Колода объектная — колода перфокарт, на которой программа записана на машинном языке: выходная колода компилятора.

Команда — ключевое слово или логический оператор языка программирования: идентифицирует, какие действия следует произвести.

Компилятор — программа, которая воспринимает код на исходном языке программирования, интерпретирует закодированные предложения и формирует модуль или колоду программы, содержащие программу на машинном языке, эквивалентную исходной программе.

Константа — имеющее метку поле данных, к которому при выполнении программы производятся многократные обращения. Если содержимое этого поля не изменяется программой явно, данные, входящие в это поле, постоянны.

Конфигурация битов — представление данных в виде совокупности отдельных битов.

Коэффициент кратности — составная часть предложения определения константы (DC-предложения) или предложения определения памяти (DS-предложения). Коэффициент кратности указывает, сколько копий (дубликатов) константы или области памяти должен построить компилятор.

Коэффициент перемещения — величина, равная разности между первым адресуемым байтом основной памяти и адресом, по которому расположена программа в основной памяти; величина, выражающая количество байтов между двумя определенными адресами.

Лента — обиходное название магнитной ленты; носитель данных.

Лента магнитная — носитель данных. Катушка магнитной ленты состоит из полимерной пленки, которая с одной стороны покрыта оксидным соединением. Данные магнитным способом записываются на оксидной поверхности группами параллельных битов. Биты представляются путем перестройки магнитной структуры вещества покрытия.

Ленточная операционная система (Tape Operating System — TOS) — одна из операционных систем Системы/360.

Литерал — константа, заданная как второй операнд команды, а не с помощью предложения DC. Формат константы этого типа такой же, как и для обычной константы, за исключением того, что перед константой ставится знак равенства (=).

Макро — макрокоманда; разновидность языка, применяемая при написании макрокоманд.

Макрокоманда — ключевое слово, метка или имя, которые активизируют программу, написанную на макроязыке. Эта программа содержит группу предложений для выполнения специальных функций. Обычно программы на макроязыке хранятся в макробиблиотеке; некоторая программа может использовать программы из этой библиотеки, если содержит обращение к ним по имени.

Маска — поле данных (1 байт или более), которые используются, чтобы проанализировать или изменить другое поле данных. Битовое или байтовое представление маски обусловливает определенные действия над соответствующими битами или байтами анализируемого или изменяемого поля.

Математическое обеспечение — любая программа или набор программ, предназначенных для выполнения вспомогательных функций при работе проблемных программ; управляющая программа операционной системы.

Метка — символическая ссылка на адрес в программе.

Метка набора данных — имя, однозначно определяющее конкретный набор данных.

Метка символическая — имя, или идентификатор, которое используется в программе для обращения по определенному адресу. Символическая метка вызывает формирование адреса в форме смещения и базы, значения которых зависят от части программы, которая имеет эту метку.

Метка тома — блок управления, идентифицирующий том для хранения данных. Метка тома обычно записывается в начале тома. Она содержит серийный номер тома.

Младший (байт, бит, разряд) — самая правая позиция поля; разряд, находящийся справа от самого левого разряда или разрядов данной конфигурации; разряд единиц десятичной величины.

Модификатор длины — указатель длины в предложениях DC и DS. Указатель длины, присвоенный константе, литералу или области памяти.

Модуль — замкнутый набор команд программы в исходной объектной или загрузочной форме; программа или подпрограмма.

Модуль объектный — программа на машинном языке, представленная в виде колоды перфокарт или записанная в памяти как данные; является результатом компиляции.

Набор данных — одна или несколько записей данных, составляющих логически связанный комплекс информации, к которому можно обращаться по его имени. Обычно предполагается, что наборы данных расположены на дисках, барабанах или магнитных лентах.

Настройка адреса — модификация адреса, определенного меткой, путем сочетания метки с приращением, например: FIELDA + 6 — это адрес области памяти, начинающейся на 6 байтов дальше от начала памяти, чем поле FIELDA.

Непосредственный символ — самоопределенный байт данных, применяемый в качестве операнда некоторых предложениях языка Ассемблера. Непосредственный символ может выражаться как символ, как пара шестнадцатеричных цифр или как 8 битов.

Область памяти — область основной памяти, используемая выполняющейся программой.

Образец для редактирования — поле, содержащее подборку управляющих символов, которые используются командами ED и EDMK. Упакованные десятичные данные редактируются, полностью замещая образец.

Оглавление тома (Volume Table of Contents — VTOC) — метки наборов данных на устройстве прямого доступа. Оглавление тома служит для обращения к наборам данных на томе. Элементами оглавления являются блоки управления наборами данных.

Операнд — метка, величина или символическая ссылка, используемая в операторе команды для идентификации величины или поля.

Предложение — оператор команды; оператор языка, представленный вместе со всеми своими операндами или ключевыми словами, необходимыми для его интерпретации.

Операционная система (OS) — самая большая из четырех операционных систем Системы/360. Способна обслуживать любые технические средства Системы/360. Требует не менее 65 536 байтов основной памяти.

Относительное положение — положение любого объекта относительно некоторой другой точки. Если рассматривать представление данных ABCDE в символьной форме, то относительное положение B по отношению к A равно 1, а относительное положение E к A—4 и т. п. Используется для характеристики положения или адреса записи в наборе данных, находящемся на устройстве прямого доступа, в отличие от использования абсолютного физического адреса. Например, если говорится, что запись XYZ — 3296-я запись в наборе данных, это означает, что эта запись является 3295-й относительно начала набора данных. Первая физическая запись в наборе данных является записью 0 (нуль) относительно начала набора данных.

Параметр — поле, величина или набор символов, которые должны быть приведены явно с целью указать ряд характеристик в предложении какого-либо языка, например операнды команд в языке Ассемблера, поля операндов макрокоманды,

управляющая информация на картах управления заданиями.

Переключатель — бит, байт или поле данных в основной памяти, которые свидетельствуют о наличии определенного условия или определенной ситуации в процессе обработки. Формируется или изменяется операционной системой или программой в соответствии с ходом обработки.

Переключатель байтовый — логический переключатель, занимающий байт памяти в проблемной программе. Его содержимое может представлять не только одно из состояний типа да/нет, включено/выключено и т. п., но и все 256 различных состояний по числу возможных битовых конфигураций в 1 байте.

Переключатель битовый — логический переключатель, состоящий из одного двоичного разряда. Он представляет два состояния, такие, как да/нет, включено/выключено и т. п.

Переключатель символьный — байтовый переключатель.

Переполнение — арифметическое переполнение; возникает при арифметической операции, когда результат не вмещается в отведенное для него поле.

— средство переполнения дорожки; обеспечивается как техническими средствами, так и средствами математического обеспечения, допускает автоматический перенос продолжения записи данных на следующую дорожку на устройстве прямого доступа. Это средство имеется в устройстве IBM 2314 и в некоторых других устройствах прямого доступа.

Переход — изменение порядка выполнения команд программы так, чтобы следующей выполнялась заданная команда, в противоположность естественному последовательному порядку выполнения команд.

Переход условный — переход в программе; выполняющийся только в том случае, если в процессе обработки сформировалось определенное условие.

Подавление нулей — средство или метод формирования числовых величин в коде EBCDIC. Заключается в замене всех незначащих нулей пробелами или особыми символами.

Подпрограмма — одна команда или более, составляющие логически законченную часть алгоритма программы. Обычно составляется и хранится отдельно от основной части программы. Второстепенная программа по отношению к основной программе.

Полное слово — четырехбайтовое поле, адрес которого кратен числу 4. Считается, что содержимое полного слова имеет формат числа с фиксированной точкой, хотя оно может содержать данные любого типа.

Полубайт, тетрада — любая половина байта; или младшие или старшие 4 двоичных разряда в байте.

Полуслово — 2 байта данных, адрес первого из которых кратен числу 2. Считается, что полуслово имеет формат числа с фиксированной точкой, хотя оно может содержать любые данные.

Последовательная обработка — режим обработки записей данных, при котором записи анализируются и обрабатываются в соответствии с последовательностью их физического расположения в памяти или во входном потоке.

Последовательный метод доступа с очередями (Queued Sequential Access Method — QSAM) — метод доступа, который используется для последовательной обработки последовательно организованных наборов данных.

Прерывание программное — прерывание программы, возникшее в результате предусмотренных или непредусмотренных обстоятельств, препятствующих выполнению алгоритма программы без дополнительного вмешательства программных средств или оператора.

Признак неявный — признаки или характеристики в предложении DC определения константы, которые, будучи опущены, определяются содержимым константы.

Признак результата — внутренний индикатор вычислительной системы, состоящий из двух двоичных разрядов Слова состояния программы. В поле признака результата в результате выполнения команды может быть помещена одна из четырех возможных битовых конфигураций: 00, 01, 10, 11.

Признак явный — признаки или характеристики константы, явно указанные в предложении DC определения константы.

Приравнять (Equate — EQU) — оператор языка Ассемблера, указывающий, что метка (символ) представляет десятичную величину или выражение, которые могут быть как абсолютными (самоопределенными), так и перемещаемыми. Оператор приравнивания может записываться, например, в виде следующего предложения: REG2 EQU 2. В этом случае метка REG2 может теперь использоваться вместо цифры 2 для обозначения регистра. Метка REG2 может также использоваться в любом месте, где программист хочет задать величину 2, например

```
MVC FIELD(REG2,REG2),INPUT
```

В этом примере первая запись REG2 обозначает длину пересылаемых данных, а вторая запись REG2 — регистр базы, приписываемый метке FIELD.

Программа управления данными — компоненты операционной системы, обеспечивающие организацию файлов и выполнение задач ввода-вывода.

Промежуток между блоками — область, в которую не производится запись и которая служит для разделения групп физических записей на запоминающем устройстве.

Прямая обработка — режим обработки записей данных. Записи посылаются на устройство с прямым доступом или считываются с него посредством указания их относительного местоположения или ключа. Одна или более записей могут записываться на файл независимо от их логической или физической последовательности.

Раздел — логически связанная группа записей в библиотечном наборе данных; имя раздела идентифицирует раздел в библиотечном наборе данных.

Разряд двоичный — положение данного бита в двоичном представлении данных.

Разряд знаковый — старший разряд в поле для величины с фиксированной точкой, который используется, чтобы определить, является ли эта величина положительной или отрицательной.

Расширенный двоично-кодированный десятичный код для обмена информацией (Extended Binary Coded Decimal Interchange Code — EBCDIC) — система двоичного представления стандартных алфавитно-цифровых и специальных символов и знаков пунктуации, в которой для представления каждого символа используются 8 двоичных разрядов.

Регистр — фиксированное поле, имеющее длину 32 двоичных разряда (одно полное слово). В Системе/360 используется для хранения адресов и адресных ссылок. Имеется 16 общих регистров, пронумерованных от 0 до 15. Программист на языке Ассемблера может использовать эти регистры для временного хранения данных, в качестве рабочих областей при арифметических операциях с фиксированной точной и как регистры базы для адресации при обеспечении связи между проблемными программами.

Регистр базы — общий регистр, выполняющий в программе функции базы адресации. Эта база в сочетании со смещением используется для адресации любого из последующих 4096 байтов памяти.

Регистр нечетный — любой общий регистр, имеющий нечетный номер, например регистр 1, регистр 3, регистр 5, регистр 7 и т. д.

Регистр четный — любой общий регистр, имеющий четный номер, например регистр 2, регистр 4, регистр 6, регистр 8 и т. д.

Самоопределенный — величина или символ, представление которых полностью выражает их смысл. Данные, подразумеваемое символьное представление которых понятно без дальнейшей интерпретации.

Сегмент — часть таблицы, содержащая группу данных, имеющих относительно самостоятельное значение. Любой элемент или любая величина в таблице. Обычно содержит величину или

набор символов, отличающих данный сегмент от остальных сегментов таблицы.

Сегмент-ограничитель — сегмент таблицы в проблемной программе, свидетельствующий об исчерпании данных или области.

Символ — любой из элементов стандартного кода EBCDIC; буквы, арабские цифры, а также другие знаки и символы, такие, как $, #, *, %, &, ? и т. п.

Символ выбора цифры — один из нераспечатываемых символов, выполняющий вспомогательную функцию в процессе редактирования по командам ED и EDMK. Этот символ имеет шестнадцатеричное представление 20. В образце редактирования он указывает те однобайтовые области, в которые должна быть помещена цифра из исходного поля или символ-заполнитель образца.

Символ-заполнитель — первый (самый левый) символ в образце для редактирования. При редактировании символ-заполнитель используется для замены тех символов выбора цифры в образце, которые не замещаются цифрами из исходного поля. Символ-заполнитель пересылается в образец, если только индикатор значимости выключен.

Символ начала значимости — один из нераспечатываемых символов, выполняющий вспомогательную функцию в процессе редактирования. Этот символ имеет шестнадцатеричное представление 21. В образце для редактирования он указывает конец подавления нулей. Если подавление нулей не прекращено до обнаружения символа начала значимости в образце, индикатор значимости устанавливается в состояние «включено».

Символ разделения полей — один из управляющих символов, используемых операторами редактирования языка Ассемблера. Символ разделения полей используется при редактировании группы полей для установки индикатора значимости в состояние «выключено» между соседними полями.

Символ управления кареткой — однобайтовый символ в коде EBCDIC, который помещается в начале поля данных, предназначенных для вывода на печать. Этот символ позволяет управлять пропуском строк и расстоянием между строками.

Система операционная — любая система, которая управляет работой технических средств вычислительной системы. В Системе/360 предусмотрены Базовая операционная система (BOS), Дисковая операционная система (DOS), Ленточная операционная система (TOS) и полная Операционная система (OS).

Слово — единица данных или памяти, состоящая из 4 байтов; сокращенное название полного слова.

Слово состояния программы (Program Status Word — PSW) — поле размером в двойное слово (64 двоичных разряда) в процессоре. Содержит адреса, индикаторы состояния, индикаторы ошибок и другие данные о состоянии выполняемой программы. Слово состояния программы модифицируется каждый раз, когда выполняется следующая машинная команда.

Смещение — интервал в байтах между двумя точками памяти.

Сохранение естественного порядка следования команд — переход к следующей команде в основной памяти при выполнении команды условного перехода в отличие от перехода.

Справочник — внутренний указатель разделов библиотечного набора данных.

Сравнение кодов — сравнение двух символов или полей, при котором слева направо сравниваются соответствующие двоичные разряды обоих полей. Сравнение прекращается, как только обнаруживаются два неравных бита. Знаки арифметических величин или символьное представление полей во внимание не принимаются.

Сравнение побитное — разновидность сравнения, при котором данные сравниваются на основе содержимого соответствующих двоичных разрядов независимо от смысла самих полей данных. Так, числовая величина в любой форме может сравниваться с полем, которое рассматривается как содержащее алфавитно-цифровые символы.

Старший — самая левая позиция поля, разряд, находящийся слева от самого правого разряда или разрядов данной конфигурации.

Степени 2 — возрастающая последовательность степеней по основанию 2. Используется для определения возможных значений последовательных двоичных разрядов справа налево. Каждый двоичный разряд потенциально имеет величину, в два раза большую, чем соседний справа разряд.

Таблица — данные, организованные определенным образом, имеющие заданное содержание или ссылки к другим данным. Таблица обычно состоит из сегментов одинаковой длины.

Таблица перекодировки — таблица длиной 256 байтов, которая применяется в командах TR и TRT.

Том — модуль технических средств, служащий носителем данных, например пакет дисков, барабан, ячейка данных в запоминающем устройстве на магнитных картах, катушка магнитной ленты.

Том на ленте — одна катушка магнитной ленты, обычно имеющая собственный серийный номер. Хотя катушка ленты считается отдельным томом, набор данных, определенный

одним именем, может, сохраняя свою непрерывность, переходить с одной катушки на другую.

Том прямого доступа — сменное хранилище данных, часть устройства прямого доступа. Если имеется в виду пакет дисков запоминающего устройства на магнитных дисках, то считается, что том прямого доступа имеет свой серийный номер, или идентификатор, по которому к нему можно обращаться.

Указатель — адрес, символ или метка, которые используются для доступа к определенной точке основной памяти или запоминающего устройства.

Упакованная десятичная — одна из арифметических систем Системы/360. Числовая величина представляется как группа шестнадцатеричных цифр, каждая из которых имеет величину от 0 до 9. В этом случае шестнадцатеричные цифры изображают величину в десятичном представлении. Величине может быть присвоен знак путем помещения одной из шестнадцатеричных цифр от A до F в правом полубайте поля.

Упаковать — преобразовать представление числовой величины в коде EBCDIC в упакованную десятичную форму.

Условие конца тома (End of Volume — EOV) — это условие возникает каждый раз, когда обнаруживается конец тома на запоминающем устройстве. Условие формируется программами управления данными, когда задача ввода-вывода обнаруживает на запоминающем устройстве марку конца тома или хвостовую метку.

Условие конца файла (End of File — EOF) — это условие возникает каждый раз, когда обнаруживается конец набора данных (файла) на томе. Условие формируется программами управления данными, когда задача ввода-вывода обнаруживает на запоминающем устройстве марку конца тома или хвостовую метку.

Устройство запоминающее на магнитных картах — в частности устройство IBM 2321, обладающее большой емкостью. Данные на этом устройстве записываются магнитным способом на гибкие полоски магнитной ленты, хранящиеся пачками в ячейках. Когда должна быть записана или считана информация, полоски извлекаются из пачки и перемещаются мимо читающих или записывающих головок.

Устройство запоминающее с прямым доступом (Direct Access Storage Device — DASD) — общее название устройств с прямым доступом. Сокращение DASD иногда используется применительно к запоминающему устройству на магнитных дисках IBM 2311.

Устройство с прямым доступом — любое из групп запоминающих устройств, допускающих возможность прямого или случай-

ного доступа к данным, и, в частности, прямой или случайной выборки данных.

Файлов организация — организация, или структура, наборов данных (файлов). В большинстве случаев организация файлов непосредственно связана с методом доступа, использованным при построении этого набора данных. Например, набор данных, имеющий индексно-последовательную организацию, создается с помощью индексно-последовательного метода доступа (ISAM).

Фиксированная точка — относится к арифметике с фиксированной точкой. Числовые величины с фиксированной точкой выражаются двоичной битовой конфигурацией, в которой старший бит указывает, является ли величина положительной или отрицательной.

Формат двоичный — представление поля «разряд за разрядом» (каждый разряд двоичный) безотносительно к смыслу или интерпретации этого поля.

Хвостовая метка — метка набора данных, или блок управления, который записывается в конце набора данных. Содержит информацию о наборе данных, включая число блоков данных, содержащихся в данном файле.

Целое число — десятичная величина, выражаемая числовым полем с фиксированной точкой; в формировании целого числа участвуют все разряды поля, за исключением старшего, знакового разряда.

Центральный процессор (Central Processing Unit — CPU) — компонент оборудования, содержащий основные блоки памяти, логические и арифметические блоки.

Цикл — метод непрерывного повторения одной и той же группы команд определенное количество раз. Количество повторений в цикле определяется счетчиком или одним или несколькими условиями, которые проверяются в процессе повторения. Этот метод часто применяется при поиске в таблице или просмотре полей данных.

Шестнадцатеричная цифра — цифра системы счисления по основанию 16. Четыре двоичных разряда (полубайт) могут служить для выражения одной шестнадцатеричной цифры, обозначаемой как арабские цифры от 0 до 9 или буквы от A до F включительно.

Язык — в вычислительной технике язык — это коды, слова, предложения, применяемые, чтобы запрограммировать действие вычислительной машины. Запись желаемых действий на данном языке вводится в вычислительную машину на перфокартах и переводится компилятором (транслятором) для данного языка в выполняемую программу в машинных кодах. В качестве

выполняемой программы используется объектный модуль или объектная колода.

Язык машинный — уровень кодирования, допускающий непосредственное выполнение вычислительной машиной без дальнейшей трансляции; кодирование этого уровня порождается языковым компилятором. Код машинного языка Системы/360 имеет шестнадцатеричную форму и может непосредственно восприниматься операционной системой.

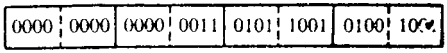
Ответы к упражнениям

Глава 3.

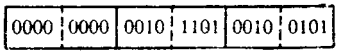
1. Нуль и единица (0 и 1)
2. Знаковый разряд; целое
3. Тридцати одного (31)
4. Шестнадцатеричному
5. Восьми (8)
6. Двенадцати (12)
7. Шестнадцатеричного
8. Зонный десятичный

9 а.

9 б.

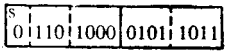


9 в.

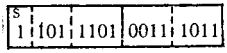


10 а. +28781
10 б. +2216797
10 в. +2686
10 г. +10438659
10 д. −7659
10 е. −277522

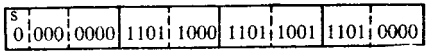
11 а.



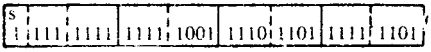
11 б.



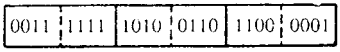
11 в.

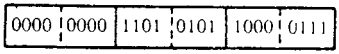


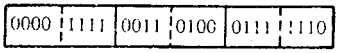
11 г.

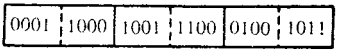


12 а. 9552023
12 б. 65535
12 в. 22764
12 г. 6784601
13 а. Шестн. 5CCA Десятичн. 23754
13 б. Шестн. 3621 Десятичн. 13857
13 в. Шестн. DAAC Десятичн. 55980
13 г. Шестн. D1F4 Десятичн. 53748
13 д. Шестн. 65F2 Десятичн. 26098
13 е. Шестн. BB8A Десятичн. 48010
13 ж. Шестн. 76D6 Десятичн. 30422
13 з. Шестн. 5010 Десятичн. 20496

14 а. 

14 б. 

14 в. 

14 г. 

15 а. Шестн. F25F Десятичн. 62047
15 б. Шестн. 2EF3 Десятичн. 12019
15 в. Шестн. 70EA Десятичн. 28906
15 г. Шестн. 38C6E Десятичн. 232558
16 а. +400
16 б. —30970
16 в. +65002
16 г. —8700
16 д. 10786039 (Считается, что это положительная величина)
17 а. —3022570649
17 б. —7034
17 в. +5581
17 г. —40067997

Глава 4.

1. Четыре; полное слово
2. Оператор
3. Меткой
4. Шестнадцать (16)

5. Буквой
6. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 и 15
7. Восьми
8. Кодом операции
9. Шестнадцатеричные
10. Положительные
11. Четыре (4)
12. Операндов
13. Положительный; отрицательный
14. Два (2)
15. Первой
16. +32767 —32768
17. Четыре (4)
18. Звездочка (*)
19. Явный
20. Регистрах
21. DC, DS
22. Не очищает
23. Верно
24. Неверно (Метка адресует лишь первое 50-байтовое поле. Остальные поля доступны посредством модификации адреса)
25. Шестьдесят (60) байтов
26. Полное слово

27 а. Восемнадцать (18)
27 б. Шестьдесят (60)
27 в. Нуль (0)

28.
```
SETDATA1 FLDA1 R6SET Z32MSG
   DC        G     Y     ORG
```

29. 256
30. Три (3)
31. X'40'
32. Тринадцать (13)

33.

S	T	A	R	T	Ø	Ø	Ø

34. Нуль (метка содержит 9 байтов и потому неправильна)

35.

0	0	5	C	0	0	5	C	0	0	5	C

36.

0001	0001	0001	0001	0001	0001

37.

3	E	7	F

38. Шестнадцати (16)

39.

0	0	0	1	5	C	0	0	0	1	5	C

Глава 6.

1.

Символьн.	A		B		C		X		Y		Z	
Шестн.	C	1	C	2	C	3	E	7	E	8	E	9

2.

Символьн.	7		8		9	
Шестн.	F	7	F	8	F	9

3.

Символьн

7	8	I	7	8	I	7	8	I

Шестн.

F	7	F	8	C	9	F	7	F	8	C	9	F	7	F	8	C	9

4.

Символьн.	$						/		/	
Шестн.	5	B	4	B	4	B	6	1	6	1

5.

Символьн.	Ь		Ь	
Шестн.	4	0	4	0

6.

Символьн.	$		6		1		0		$		6		1		0	
Шестн.	5	B	F	6	F	1	F	0	5	B	F	6	F	1	F	0

7.

Символьн.	A		B		C	
Шестн.	C	1	C	2	C	3

8.

Символьн.	9		9		9		9		9	
Шестн.	F	9	F	9	F	9	F	9	F	9

9.

Символьн.	A		ъ		B		ъ		A		ъ		B		ъ	
Шестн.	C	1	4	0	C	2	4	0	C	1	4	0	C	2	4	0

10.

Символьн.	$		
Шестн.	5	B	6	0

11.

Символьн.	A		D		H		L		P		T	
Шестн.	C	1	C	4	C	8	D	3	D	7	E	3

12.

Символьн.	1		3		1		3		1		3		1		3	
Шестн.	F	1	F	3	F	1	F	3	F	1	F	3	F	1	F	3

13.

Символьн.	ъ	ъ	ъ	ъ	ъ
	ъ	ъ	ъ	ъ	ъ
	ъ	ъ	ъ	ъ	ъ
Шестн.	4 0	4 0	4 0	4 0	4 0
	4 0	4 0	4 0	4 0	4 0
	4 0	4 0	4 0	4 0	4 0

14.

Шестн.	2	2	5	C	2	2	5	C	2	2	5	C	2	2	3	C

15.

Шестн.	0	0	0	0	0	0	5	1	9	8	3	D

16.

Шестн.	0	0	6	1	0	C

17.

Шестн.	1	5	9	8	7	6	2	C	1	5	9	8	7	6	2	C

18.

Шестн.	9	D	9	D	9	D	9	D	9	D

19.

Шестн.	5	9	8	7	7	C	5	9	8	7	7	C

20.

Символьн.	0		0		2		1		5		Q	
Шестн.	F	0	F	0	F	2	F	1	F	5	D	8

21.

Символьн.	7		8		I		7		8		I	
Шестн.	F	7	F	8	C	9	F	7	F	8	C	9

22.

Символьн.	0		0		3		3		C	
Шестн.	F	0	F	0	F	3	F	3	C	3

23.

Символьн.	0		N		0		N		0		N	
Шестн.	F	0	D	5	F	0	D	5	F	0	D	5

24.

Символьн.	4		1		7		B	
Шестн.	F	4	F	1	F	7	C	2

25.

Символьн.	0		8		5		9		1		G	
Шестн.	F	0	F	8	F	5	F	9	F	1	C	7

26.

Шестн.	4	0	4	0	4	0	4	0
Двоичн.	0100	0000	0100	0000	0100	0000	0100	0000

27.

Шестн.	0	0	D	1	D	2
Двоичн.	0000	0000	1101	0001	1101	0010

28.

Шестн.	F	0
Двоичн.	1111	0000

29.

Шестн.	0	0
Двоичн.	0000	0000

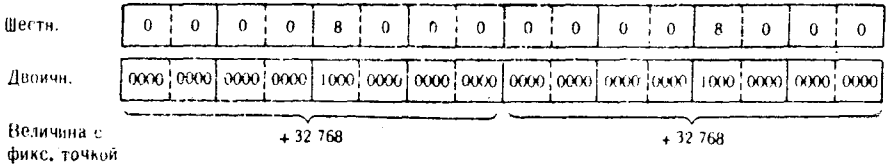
30.

Шестн.	0	0	F	F	F	F
Двоичн.	0000	0000	1111	1111	1111	1111

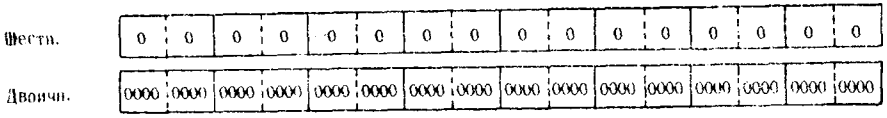
31.

Шестн.	7	F	F	F	7	F	F	F	7	F	F	F
Двоичн.	0111	1111	1111	1111	0111	1111	1111	1111	0111	1111	1111	1111
Величина с фикс. точкой	+32 767				+32 767				+32 767			

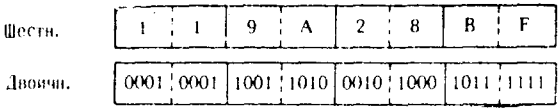
32.



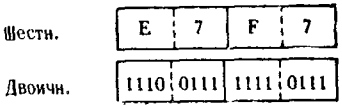
33.



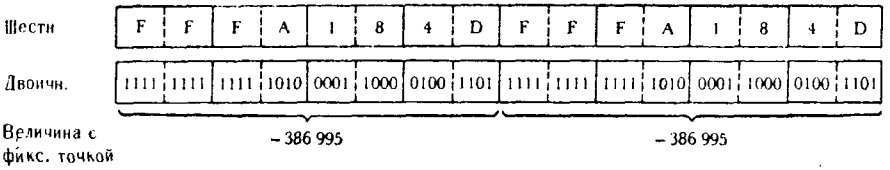
34.



35.



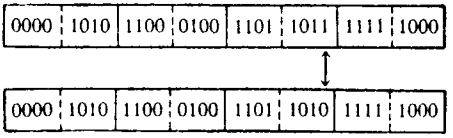
36.



Глава 7.

1. Самой старшей; самой младшей; двоичными
2. Непосредственный
3. С фиксированной точкой
4. B; D
5. 16 байтов
6. Равна
7. Больше
8. 256

9.



10. Равен
11. Меньше
12. Нет
13. Правильный
14. Правильный
15. Правильный
16. Правильный
17. Неправильный
18. Правильный
19. Неправильный
20. Неправильный
21. Правильный
22. Правильный
23. Неправильный
24. Правильный
25. Неправильный
26. Неправильный
27. Правильный
28. Правильный
29. Неправильный
30. Правильный
31. Неправильный
32. Неправильный
33. Правильный
34. Неправильный
35. Правильный
36. Неправильный
37. Правильный
38. Правильный
39. Неправильный
40. Неправильный
41. Неправильный
42. Неправильный
43. Правильный
44. Правильный
45. Неправильный
46. Неправильный
47. Правильный
48. Неправильный
49. Правильный
50. Неправильный
51. Правильный
52. Правильный
53. Правильный

Глава 8.

1. Безусловные
2. BC и BCR
3. F; Безусловного
4. 8 4 2 1
5. Первом; +1

6а. Первый операнд меньше второго или равен ему
6б. Первый операнд не равен второму или первый операнд больше второго или меньше его
6в. Первый операнд больше второго или равен ему
6г. Первый операнд больше второго
6д. Первый операнд равен второму
6е. Первый операнд меньше второго, равен второму или больше, чем второй
6ж. Первый операнд больше второго

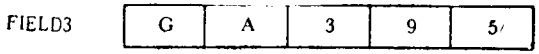
6 з. Первый операнд меньше второго
7. +2147483647
8. Регистре
9. Три
10. Нуль
11. Адрес
12. Регистры
13. 0 (нуль)
14. 74 (семьдесят четыре)
15. +3250; —25; +50
16. Адрес предложения с меткой BACK
17. CONTIN
18 а. BC 4,SUBRT1
18 б. BC 2,SUBRT5
18 в. BC 8,SUBRT7
18 г. BC 4,SUBRT10
18 д. BC 6,SUBRT12A
18 е. BC 15,SUBRT14
18 ж. BC 2,SUBRT17

Глава 9.

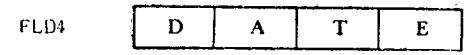
Команды пересылки

1. SI
2. 4
3. Пересылка — MVI
4. Пересылка зон — MVZ
5. Скользящая; Первому
6. MVZ
7. Пересылка цифр — MVN
8. Первым
9. Пересылка символов — MVC
10. 7
11. Цепочки (последовательности)

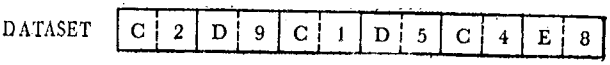
12 а.



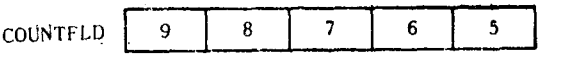
12 б.



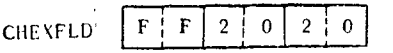
12 в.



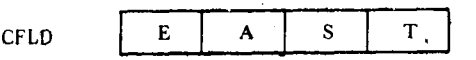
12 г.



12 д.



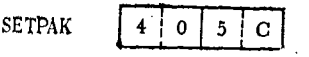
12 е.



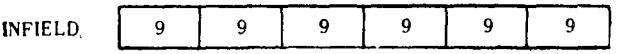
12 ж.

12 з.

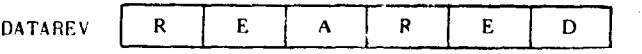
12 и.



12 к.



12 л.



Команды записи в память

13. Нет
14. Запись в память полуслова — STH
15. Запись в память групповая — STM
16. Запись в память — ST
17. Нет
18. Запись в память символа — STC
19. Два; старших
20. Девять
21. Десять
22. —319771
23. Восемь

24 а.

FULLWD6	2	5	3	1	0	9	7	D

24 б.

SETWD1	0	0	0	3	9	5	6	C
SETWD2	0	0	0	0	0	F	C	3
SETWD3	0	0	6	9	2	1	C	6
SETWD4	0	0	0	C	0	7	4	2

24 в.

DAFLD	4	C	0	0	0	0	0	0

24 г.

DATA8	0	0	0	0	0	5	C	6
	F	F	2	9	0	0	6	C

Команды загрузки

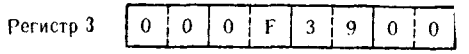
25. Загрузка полуслова — LH
26. Загрузка адреса — LA
27. Загрузка положительная — LPR
28. Одного; шестидесяти четырех
29. Положительность; отрицательность
30. Загрузка — L
31. Прочитать символ — IC
32. Загрузка отрицательная — LNR
33. +32767
34. Тридцать два (32)
35. Загрузка — LR
36. Загрузка и проверка — LTR
37. X'00'
38. X'712400FF'
39. Загрузка дополнения — LCR
40. Загрузка групповая — LM
41.

Регистр 8	F	F	F	F	C	1	C	2

42. X'FCFBFAFA'

43. X'003D0E46'

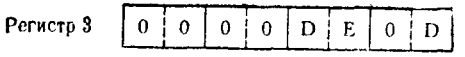
44 а.



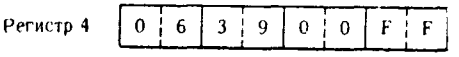
44 б.

44 в.

44 г.



44 д.



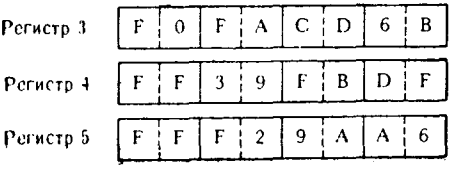
44 е.

44 ж.

44 з.

44 и.

44 к.



44 л.

Регистр 12	0	0	0	0	7	6	5	4

44 м.

Регистр 12	0	0	C	6	F	9	D	9

Команды сдвига

45. Второй
46. Восемь
47. Сдвиг влево кода — SLL
48. Знакового бита
49. Сдвиг вправо арифметический — SRA
50. Четного
51. Сдвиг влево двойной кода — SLDL
52. Четный
53. Сдвиг вправо кода — SRL
54. Отлично
55. Сдвиг влево двойной арифметический — SLDA
56. Нечетным
57. Сдвиг вправо арифметический — SRA
58. Нули
59. Сдвиг вправо двойной кода — SRDL; нулями
60. Тридцать два (32)
61. Сдвиг вправо двойной арифметический — SRDA; шестьдесят три
62. Нуль

63 а.

Регистр 5 (Двоичн.)	1111	1111	0000	0001	1101	1111	0000	1110
(Шестн.)	F	F	0	1	D	F	0	E

63 б.

Регистры 2 и 3 (Двоичн. и шестн.)

1111	1111	1000	0000	0011	1111	1111	1001	0110	0110	0001	0000	0000	0000	0000	0000
F	F	8	0	3	F	F	9	6	6	1	0	0	0	0	0

63 в.

Регистр 10 (Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0000
(Шестн.)	0	0	0	0	0	0	0	0

63 г.

Регистр 9 (Двоичн.)

0000	0000	0000	1111	0100	1110	0101	1101
0	0	0	F	4	E	5	D

(Шестн.)

63 д.

Регистры 6 и 7 (Двоичн. и шестн.)

1000	0111	1011	1010	0000	1110	0101	0000	0000	0100	1101	1001	1111	0101	1011	0000
8	7	B	A	0	E	5	0	0	4	D	9	F	5	B	0

63 е.

Регистры 10 и 11 (Двоичн. и шестн.)

1111	1111	1111	1111	1111	1111	1111	1111	0000	0000	0000	0000	0000	0001	0000	0010
F	F	F	F	F	F	F	F	0	0	0	0	0	1	0	2

63 ж.

Регистр 2 (Двоичн.)

0100	0110	0000	1110	1010	0001	0000	0000
4	6	0	E	A	1	0	0

(Шестн.)

63 з.

Регистры 8 и 9 (Двоичн. и шестн.)

0000	0000	0000	0000	0010	1000	0000	1100	1111	1101	0001	1000	1111	1101	0001	1000
0	0	0	0	2	8	0	C	F	D	1	8	F	D	1	8

Глава 10.

1. PACK
2. Преобразование в десятичную — CVD
3. Длины
4.

PAKFLD

0	0	0	0	2	5	9	F

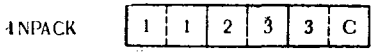
5.

OUTFLD

F	0	F	0	F	0	F	0	F	3	F	9	C	6

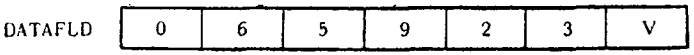
6.

7.

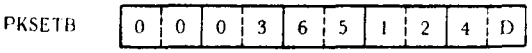


8.

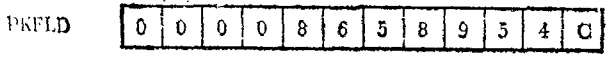
9.



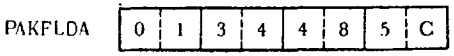
10.



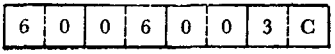
11.



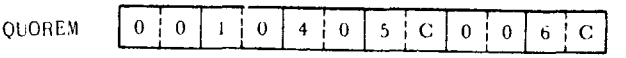
12. Восьми (8)
13. Десяти (10)
14. Восьми; правой
15. 123456
16. Делителя
17. Второй
18. Преобразование в десятичную — CVD; Распаковать — UNPK
19. Старшие
20. Влево
21. X'F0'
22.



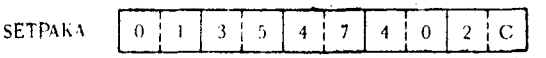
23.



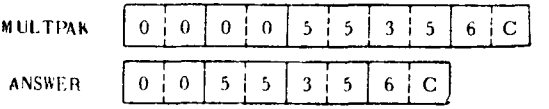
24.



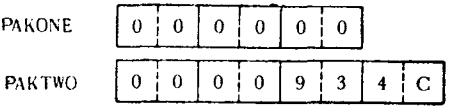
25.



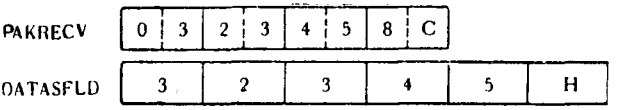
26.



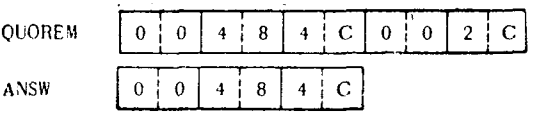
27.



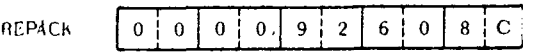
28.



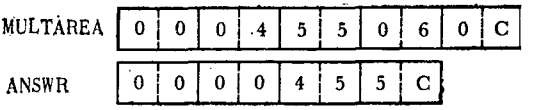
29.



30.



31.



Глава 11.

1. Неправильно
2. Вычитание кодов — SLR
3. Умножение; деление

4. Единице
5. Дополняется; складывается
6. Четным
7. +4096
8. Правильно
9. Первого
10.

F	F	F	F	8	0	0	0

11. Правильно
12. SL
13. Неправильно
14.

Регистр 3									
(Шестн.)	0	0	0	0	2	6	2	5
(Двоичн.)	0000	0000	0000	0000	0010	0110	0010	0101

15.

Регистр 11								
(Шестн.)	0	0	0	6	7	B	E	8
(Двоичн.)	0000	0000	0000	0110	0111	1011	1110	1000

16.

Регистр 7								
(Шестн.)	0	1	E	E	3	7	0	C
(Двоичн.)	0000	0001	1110	1110	0011	0111	0000	1100

17.

Регистр 9								
(Шестн.)	0	2	5	B	9	9	8	6
(Двоичн.)	0000	0010	0101	1011	1001	1001	1000	0110

18.

Регистр 10								
(Шестн.)	0	0	0	2	6	3	3	C
(Двоичн.)	0000	0000	0000	0010	0110	0011	0011	1100

19.

Регистр 4								
(Шестн.)	0	0	0	0	0	0	0	0
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0000

20.

Регистр 12 (Шестн.)	0	0	0	0	9	1	F	7
(Двоичн.)	0000	0000	0000	0000	1001	0001	1111	0111

21.

Регистр 7 (Шестн.)	0	0	0	4	F	F	0	F
(Двоичн.)	0000	0000	0000	0100	1111	1111	0000	1111

22.

Регистр 10 (Шестн.)	0	1	4	8	E	2	E	D
(Двоичн.)	0000	0001	0100	1000	1110	0010	1110	1101

23.

Регистр 3 (Шестн.)	0	0	2	F	3	4	8	D
(Двоичн.)	0000	0000	0010	1111	0011	0100	1000	1101

24.

Регистр 2 (Шестн.)	0	0	0	0	0	0	0	0
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0000
Регистр 3 (Шестн.)	0	7	3	C	D	A	4	8
(Двоичн.)	0000	0111	0011	1100	1101	1010	0100	1000

25.

Регистр 4 (Шестн.)	0	0	0	0	0	0	0	0
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0000
Регистр 5 (Шестн.)	0	3	F	5	7	5	0	8
(Двоичн.)	0000	0011	1111	0101	0111	0101	0000	1000

26.

Регистр 6 (Шестн.)	0	0	0	0	0	0	0	0
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	0000
Регистр 7 (Шестн.)	0	0	0	5	E	3	7	6
(Двоичн.)	0000	0000	0000	0101	1110	0011	0111	0110

27.

Регистр 8 (Шестн.)	0	0	0	0	0	0	3	9
(Двоичн.)	0000	0000	0000	0000	0000	0000	0011	1001
Регистр 9 (Шестн.)	0	0	0	0	0	0	8	4
(Двоичн.)	0000	0000	0000	0000	0000	0000	1000	0100

28.

Регистр 4 (Шестн.)	0	0	0	0	0	0	5	7
(Двоичн.)	0000	0000	0000	0000	0000	0000	0101	0111
Регистр 5 (Шестн.)	0	0	0	0	0	3	5	8
(Двоичн.)	0000	0000	0000	0000	0000	0011	0101	1000

Главы 3—11 — дополнительные упражнения

1. Правильно
2. Правильно
3. Правильно
4. Правильно
5. Правильно
6. Правильно
7. Правильно
8. Правильно
9. Правильно
10. Правильно
11. Неправильно
12. Правильно
13. Неправильно
14. Неправильно
15. Правильно
16. Правильно
17. Правильно
18. Неправильно
19. Неправильно
20. Неправильно
21. Неправильно
22. Неправильно
23. Правильно
24. Неправильно
25. Правильно
26. Неправильно
27. Правильно
28. Правильно
29. Неправильно
30. Правильно
31. Правильно
32. Неправильно
33. Правильно
34. Правильно
35. Неправильно
36. Правильно
37. Неправильно
38. Неправильно
39. Неправильно
40. Неправильно
41. Неправильно
42. Правильно
43. Правильно
44. Правильно
45. Неправильно
46. Неправильно
47. Правильно
48. Правильно
49. Неправильно
50. Правильно
51. Правильно
52. Правильно
53. Правильно
54. Правильно

55. Правильно
56. Правильно
57. Правильно
58. Правильно
59. Правильно
60. Неправильно
61. Правильно
62. Правильно
63. Неправильно
64. Правильно
65. Неправильно
66. Правильно
67. Неправильно
68. Неправильно
69. Неправильно
70. Неправильно
71. Правильно
72. Правильно
73. Неправильно
74. Правильно
75. Правильно
76. Правильно
77. Неправильно
78. Неправильно
79. Правильно
80. Правильно
81. Правильно
82. Неправильно
83. Правильно
84. Правильно
85. Правильно
86. Неправильно
87. Правильно
88. Правильно
89. Правильно
90. Правильно
91. Неправильно
92. Неправильно
93. Неправильно
94. Неправильно
95. Правильно
96. Правильно
97. Правильно
98. Правильно
99. Неправильно
100. Неправильно
101. Неправильно
102. Неправильно
103. Неправильно
104. Правильно
105. Правильно
106. Правильно
107. Правильно
108. Неправильно
109. Правильно
110. Неправильно
111. Неправильно
112. Правильно
113. Правильно
114. Правильно
115. Правильно
116. Неправильно
117. Неправильно
118. Неправильно
119. Правильно
120. Неправильно
121. Неправильно
122. Неправильно
123. Неправильно
124. Правильно
125. Неправильно

Глава 12.

1. Битами; битов
2. И
3. Сложение
4. Нулевые; единичные
5. ХС
6. Маска
7. Единице

8. 32
9. Единичный; нулевой
10. Регистр; словом
11. Непосредственный
12. Умножения
13. И; 256
14. Регистр
15. Регистр; регистр
16. Включено
17. Нулевой
18. Единичные
19.

Регистр 3 (Двоичн.)	0010	0000	1111	0011	0100	0101	0101	1111
(Шестн.)	2	0	F	3	4	5	5	F

20.

Регистр 8 (Двоичн.)	0001	0001	0111	0011	0111	0111	1111	1111
(Шестн.)	1	1	7	3	7	7	F	F

21.

DATAFLD (Двоичн.)	1101	0001	1100	0010	1110	0011
(Шестн.)	D	1	C	2	E	3

22.

DATATEST (Двоичн.)	1100	0100	1111	0011	1100	0100	0011	1100	1110	1001
(Шестн.)	C	4	F	3	C	4	3	C	E	9

23.

Регистр 12 (Двоичн.)	0000	0000	1100	0111	0101	0110	1001	1100
(Шестн.)	0	0	C	7	5	6	9	C

24.

DATASET (Двоичн.)	1100	0111	1100	1000	0000	0010	0000	0000	0000	0000
(Шестн.)	C	7	C	8	0	2	0	0	0	0

25.

CHKFLD (Двоичн.)	0010	1101	1011	1111	1111	0010	1101	0101	1111	0011
(Шестн.)	2	D	B	F	F	2	D	5	F	3

26.

FLDSET									
(Двоичн.)	1110	0011	0110	1011	0101	1011	1001	1100
(Шестн.)	E	3	6	B	5	B	9	C

27.

Регистр 4									
(Двоичн.)	0000	0000	0011	1001	0010	0110	0111	1000
(Шестн.)	0	0	3	9	2	6	7	8

28.

SWITCHES									
(Двоичн.)	0000	0000	0010	1010	0101	1111	0000	0001
(Шестн.)	0	0	2	A	5	F	0	1

29.

Регистр 8									
(Двоичн.)	0000	0000	0000	0000	0011	0010	0110	0000
(Шестн.)	0	0	0	0	3	2	6	0

30.

DATASW										
(Двоичн.)	1000	0110	1110	0011	1000	0010	1010	0011	1100	0110
(Шестн.)	8	6	E	3	8	2	A	3	C	6

31.

WORDBASE										
(Двоичн.)	1111	0011	1111	0101	1100	0100	1100	0001	1101	0111
(Шестн.)	F	3	F	5	C	4	C	1	D	7

32.

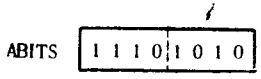
Регистр 6								
(Двоичн.)	1100	0010	1100	0111	1100	0101	1100	1000
(Шестн.)	C	2	C	7	C	5	C	8

33.

Регистр 5								
(Двоичн.)	0000	0000	0000	0000	0000	0000	0000	1100
(Шестн.)	0	0	0	0	0	0	0	C

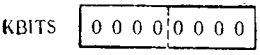
Глава 13.

1. Восьми
2. Сравнение непосредственное — CLI
3. Проверить по маске — TM
4. Символьным
5. Единицы
6. Пересылка непосредственная — MVI
7. 256
8.

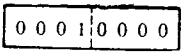



```
OI  ABITS,X'A8'
```

9.

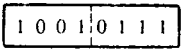


10.



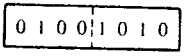

```
NI  HBITS,X'10'
```

11.



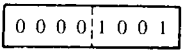

```
OI  BBITS,X'97'
```

12.



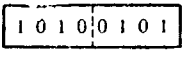

```
NI  EBITS,X'5E'
OI  EBITS,X'48'
```

13.



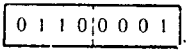

```
NI  CBITS,X'AB'
```

14.



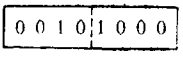

```
XI  FBITS,X'3A'
```

15.




```
TM  GBITS,X'4B'
```

16.




```
NI  DBITS,X'38'
```

Глава 15.

1. а) Символ-заполнитель
   б) Символ начала значимости
   в) Символ выбора цифры
   г) Символ разделения полей
2. Регистр 1
3. Индикатор значимости
4. X'21'
5. Плюс
6. Начала значимости
7. Упакованном десятичном
8. Пяти
9. X'20'
10. Начала значимости
11. Слева направо
12. X'22'
13. Образце
14. Разделения полей
15. Символ-заполнитель
16. Выбора цифры
17. Индикатора значимости
18. Разделения полей
19. Символ-заполнитель
20.

RESULTA (Шестн.)	C	1	4	0	4	0	4	0	4	0	F	0	F	9			
(Символьн.)	A										0		9				

21.

RESULTB (Шестн.)	D	1	4	0	F	3	F	8	F	7	F	8					
(Символьн.)	J				3		8		7		8						

22.

RESULTC (Шестн.)	5	B	4	0	F	0	F	0	4	B	F	2	F	5			
(Символьн.)	$				0		0		.		2		5				

23.

RESULTD (Шестн.)	4	0	4	0	F	3	F	6	4	0	4	0	4	0			
(Символьн.)					3		6										

24.

RESULTE (Шестн.)	4 0	F 7	4 B	F 4	F 8	C 3	D 9
(Символьн.)		7	-	4	8	C	R

25.

RESULTF (Шестн.)	5 C	5 C	5 C	F 6	F 2	F 9	6 0
(Символьн.)	•	•	•	6	2	9	—

26.

RESULTG (Шестн.)	4 0	5 B	F 6	F 1	4 B	F 1	F 7
(Символьн.)		$	6	1	.	1	7

27.

RESULTH (Шестн.)	4 0	4 0	4 0	5 B	4 B	F 3	F 3
(Символьн.)				$	.	3	3

Глава 17.

1. Таблицы перекодировки
2. 256
3. 1
4. 2
5. Исходного байта
6. Верно
7. Приращение; первого
8. Таблицы перекодировки
9. Верно
10. Первый
11. X'00'
12.

DATAFLDA (Шестн.)	E 3	E 4	D 4	C 2	D 3	C 5
(Символьн.)	T	U	M	B	L	E

13.

DATAFLDB (Шестн.)	D 7	D 9	D 6	C 2	D 3	C 5	D 4
(Символьн.)	P	R	O	B	L	E	M

14.

DATAFLDC (Шестн.)	D	3	D	6	D	5	C	7	D	1	D	6	C	2
(Символьн.)	L		O		N		G		J		O		B	

15.

DATAFLDD (Шестн.)	D	9	C	9	C	7	C	8	E	3
(Символьн.)	R		I		G		H		T	

16.

DATAFLDE (Шестн.)	F	1	F	6	4	0	E	6	C	5	E	2	E	3
(Символьн.)	1		6				W		E		S		T	

17.

DATAFLDF (Шестн.)	C	7	F	2	C	7	F	1	C	7	E	6
(Символьн.)	G		2		G		1		G		W	

DATAFLDG (Шестн.)	C	6	C	9	D	5	C	9	E	2
(Символьн.)	F		I		N		I		S	

Предметный указатель

Оглавление

Часть III

Логические применения языка Ассемблера

Часть IV

Методы доступа и наборы данных

Часть V

Таблицы и глоссарий

Д. Стэблн

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ В СИСТЕМЕ/360

Редактор *Л. Н. Бабынина*

Художник *Г. И. Мануйлов.* Художественный редактор *В. И. Шаповалов.*
Технический редактор *Е. С. Потапенкова.* Корректор *Н. А. Гиря.*

Сдано в набор 20/VI 1973 г. Подписано к печати 6/II 1974 г.
Бумага тип. № 3 $60\times90^{1}/_{16}$=23,5 бум. л. 47 печ. л. Уч.-изд. л. 43,64 Изд. № 1/6659
Цена 3 р. 22 к. Зак. 703

ИЗДАТЕЛЬСТВО «МИР» Москва, 1-й Рижский пер., 2

Ордена Трудового Красного Знамени
Ленинградская типография № 2 имени Евгении Соколовой
Союзполиграфпрома при Государственном комитете Совета Министров СССР
по делам издательств, полиграфии и книжной торговли
198052, Ленинград, Л-52, Измайловский проспект, 29