Г.В.ВИГДОРЧИК
А.Ю.ВОРОБЬЕВ
В.Д.ПРАЧЕНКО

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АССЕМБЛЕРЕ ДЛЯ СМ ЭВМ

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Г.В.ВИГДОРЧИК
А.Ю.ВОРОБЬЕВ
В.Д.ПРАЧЕНКО

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АССЕМБЛЕРЕ ДЛЯ СМ ЭВМ

Под общей редакцией В.П. Семика

2-е издание, переработанное и дополненное

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1987

ББК 32.973
В41
УДК 681.3.06

Рецензент: канд. физ.-мат. наук С. А. ХРИСТОЧЕВСКИЙ

Вигдорчик Г. В. и др.

В41 Основы программирования на ассемблере для СМ ЭВМ/ Г. В. Вигдорчик, А. Ю. Воробьев, В. Д. Праченко; Под общей ред. В. П. Семика. — 2-е изд., перераб. и доп. — М.: Финансы и статистика, 1987. — 240 с.: ил.

Рассматриваются архитектура вычислительных комплексов типа СМ-4, система команд, приводится описание языка макроассемблера в соответствии с его современной версией.

Второе издание книги (1-е издание — 1983 г.) дополнено описанием команд процессора плавающей запятой, особенностей 22-разрядной адресации памяти. Изложение сопровождается примерами программ.

Для различных категорий пользователей СМ ЭВМ. Может служить справочным пособием для начинающих программистов и студентов вузов.

В $\frac{2405000000-080}{010(01)-87}$ 105—87

ББК 32.973

ВВЕДЕНИЕ

В результате более чем десятилетнего сотрудничества стран—членов СЭВ в области вычислительной техники создан ряд моделей мини- и микроЭВМ, эффективно используемых в системах автоматизации технологических процессов и научных исследований, в сфере организационного управления и управления производством, в медицине, образовании и т. д.

Особой популярностью пользуются развивающие архитектурную линию СМ-3, СМ-4 микроЭВМ типа СМ1300, СМ1300.01, а также высокопроизводительные комплексы типа СМ1420 и СМ1600.

Близки к ним по архитектуре и назначению мини- и микроЭВМ серии «Электроника», а также программно совместимые с ними диалоговые вычислительные комплексы, школьные и бытовые компьютеры.

Для удовлетворения разнообразных запросов пользователей в составе программного обеспечения машин архитектурной линии СМ-4 разработаны имеющие различную ориентацию операционные системы РАФОС [7], ОСРВ [8], ДИАМС [9], ДОСКП [10], ИНМОС [11], а также ряд пакетов прикладных программ и системы программирования на языках высокого уровня, таких, как Фортран, Кобол, Бейсик, Паскаль, Модула, СИ. Они позволяют специалистам, имеющим различный уровень подготовки в области вычислительной техники, эффективно использовать управляющие вычислительные комплексы СМ ЭВМ в различных областях науки и народного хозяйства.

Однако, несмотря на наличие различных языков высокого уровня и мобильных операционных систем, на практике широко используется машинно-ориентированный язык ассемблера. Этот язык наиболее полно учитывает архитектурные возможности вычислительных комплексов и является практически единственным языком, позволяющим эффективно управлять внешними устройствами и устройствами межсистемных связей. Машинно-зависимые части мобильных операционных систем также написаны на языке ассемблера.

Наконец, следует упомянуть еще одну область, где необходимо глубокое знание архитектуры и машинного языка, — это проектирование встраиваемых систем и промышленных контроллеров на базе программно совместимых микропроцессорных наборов.

Предлагаемая книга посвящена основам программирования на уровне машинно-ориентированного языка ассемблера для мини- и микроЭВМ, программно совместимых с СМ-4.

В отличие от первого издания (1983 г.) второе издание книги включает ряд аспектов, связанных с появлением новых технических средств и существенной модернизацией базового программного обеспечения. Добавлены разделы, рассматривающие особенности работы диспетчера памяти на комплексах СМ1420, СМ1600 с оперативной памятью более 128 Кслов. Приведено описание команд процессора плавающей запятой.

Описание макроассемблера базируется на его современной версии, принятой в операционных системах РАФОС-2, РОСРВ и ДОСКП. Изложение материала дается вне связи с какой-либо конкретной операционной системой СМ ЭВМ.

Приводимые сведения могут быть использованы с некоторыми оговорками при работе на программно совместимых ЭВМ типа «Электроника 60», «Электроника 100/25», «Электроника 79», «Электроника 85», ДВК-2 (2М) и др.

Изложение материала построено следующим образом.

В первой главе приводятся необходимые программистам сведения об архитектуре моделей СМ-4, СМ1420, СМ1600, СМ1300 и излагаются основные системные понятия, такие, как распределение памяти, организация прерываний, способы подключения внешних устройств и др.

Во второй главе описаны способы представления данных и режимы адресации, приведено описание системы машинных инструкций и примеры их использования.

Особое внимание уделено описанию языка макроассемблера (главы 3, 4, 5 и 6).

В седьмой главе изложены специальные приемы программирования на языке макроассемблера, учитывающие архитектурные особенности моделей СМ ЭВМ с магистральной структурой.

Восьмая глава посвящена программированию операций ввода-вывода.

Справочные материалы, необходимые при программировании, приведены в приложениях к книге.

Книга ориентирована на читателей, знакомых с современными ЭВМ и основами программирования. При этом главы 1 и 2 могут быть рекомендованы и начинающим программистам. Главы 3—7 рассчитаны на системных программистов.

Освещение в книге таких вопросов, как общая архитектура и система команд, режимы адресации и способы представления данных, программирование внешних устройств, будет полезным для системных программистов, а также для проектировщиков устройств на базе программно совместимых с СМ-4 микропроцессоров.

Книга может служить справочным пособием как для опытных программистов, так и для студентов вузов.

1

ГЛАВА

АРХИТЕКТУРА МОДЕЛЕЙ ТИПА СМ-4

Основная особенность управляющих вычислительных комплексов (УВК) типа СМ-4 заключается в том, что взаимодействие между всеми устройствами, входящими в состав комплексов, включая процессор, и оперативным запоминающим устройством (ОЗУ) осуществляется при помощи единого унифицированного интерфейса, получившего название «Общая шина» (ОШ). Общая шина является каналом, через который передаются адреса, данные, управляющие сигналы на все устройства комплекса, включая процессор и память.

Процессор использует установленный набор сигналов для связи с памятью и для связи с внешними устройствами, благодаря чему в системе отсутствуют специальные команды ввода-вывода.

Все устройства комплекса подключаются в ОШ по единому принципу. Некоторым регистрам процессора, регистрам внешних устройств, которые являются источниками или приемниками при передаче информации, на ОШ отводятся адреса. В программах адреса регистров устройств рассматриваются как адреса ячеек памяти, что позволяет обращаться к ним с помощью адресных инструкций. Так, программирование операций вывода данных на внешнее устройство практически сводится к пересылке этих данных по определенному адресу.

На рис. 1.1 приведена общая структурная схема комплексов типа СМ-4.

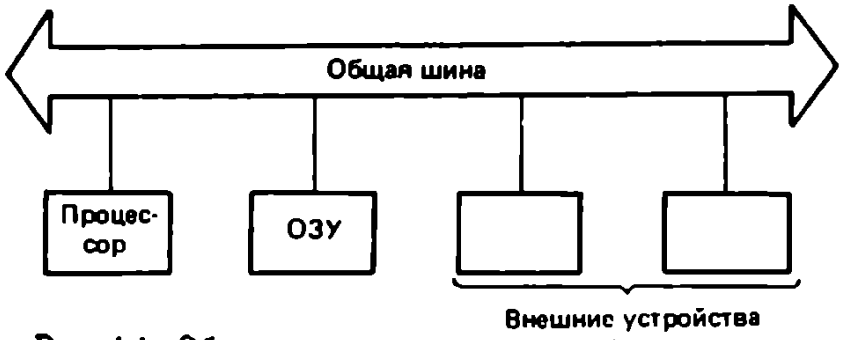


Рис. 1.1. Общая структурная схема УВК типа СМ-4

Все модели типа СМ-4 идентичны по своей архитектуре. Процессоры типа СМ-4 отличаются высокой производительностью и наличием дополнительных устройств (расширителей), обеспечивающих аппаратную реализацию операций умножения и деления, операций с плавающей запятой. Исключение составляет СМ1300, в которой указанные операции выполняются программным способом.

Все модели, кроме СМ1300, имеют диспетчер памяти, позволяющий подключать к ОШ ОЗУ емкостью до 128 Кслов (256 Кбайт), а в СМ1600 и СМ1420 — до 2048 Кслов (4096 Кбайт).

В СМ1300 объем ОЗУ составляет 32 Кслова (64 Кбайта). СМ1300 и остальные модели полностью совместимы «снизу-вверх», т. е. программы, работающие на СМ1300, могут выполняться на других моделях без каких-либо изменений. Совместимость «сверху-вниз» обеспечена частично, т. е. программы, составленные для других моделей, могут выполняться на СМ1300 только в том случае, если они не используют дополнительных аппаратных возможностей, о которых шла речь [1].

В приложении 1 приведены технические характеристики всех моделей СМ ЭВМ типа СМ-4.

1.1. ПРОЦЕССОР

Процессор выполняет машинные инструкции, обслуживает запросы на прерывание, контролирует работу ОШ.

Для связи процессора с ОШ, а через нее и с другими устройствами комплекса в процессоре выделяются: регистр слова состояния процессора, универсальные и функциональные регистры.

Регистр слова состояния процессора и универсальные регистры имеют длину 16 разрядов и доступны программисту.

Слово состояния процессора. При работе программы после выполнения каждой машинной инструкции вырабатываются специальные признаки — коды условий, информирующие о результате выполнения инструкции. Кроме того, для процессора может быть установлен приоритет, определяющий его взаимодействие с другими устройствами общей шины. Приоритет процессора и коды условий хранятся в слове состояния процессора, обозначаемом обычно в программах как PS (Processor Status). Структура PS показана на рис. 1.2.

Пояснения отдельных разрядов PS

0—3 — коды условий, вырабатываемые процессором после выполнения каждой конструкции:

разряд 0 (C) устанавливается в 1, если в результате выполнения инструкции имел место перенос разряда;

---

[1] Ограничения СМ1300 оговариваются в книге особо.

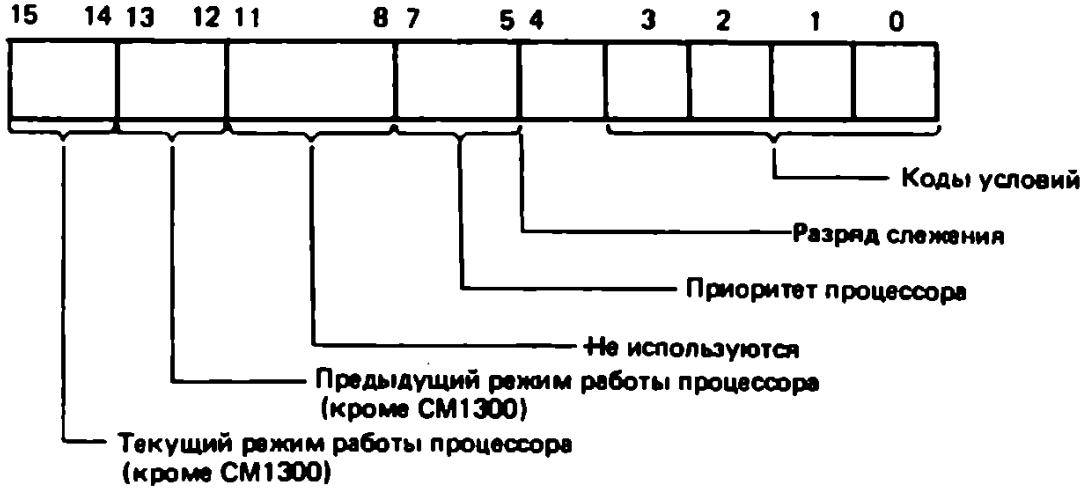


Рис. 1.2. Структура PS

разряд 1 (V) устанавливается в 1, если в результате выполнения инструкции имело место переполнение, в противном случае 0;

разряд 2 (Z) устанавливается в 1, если результат выполнения инструкции равен нулю, в противном случае 0;

разряд 3 (N) устанавливается в 1, если результат выполнения инструкции отрицательный, в противном случае 0.

4 — разряд слежения. Если установлен в 1, то после выполнения каждой инструкции происходит прерывание работы программы. Используется в программах-отладчиках. Устанавливается или сбрасывается программно.

5—7 — приоритет процессора. Может быть установлен один из восьми уровней приоритета процессора от 0 (000 — в разрядах 5—7) до 7 (111 — в разрядах 5—7). Приоритет определяет возможность прерывания работы процессора по запросу от внешнего устройства. Наивысший приоритет 7 означает, что работа процессора прервана быть не может. Приоритет процессора устанавливается или изменяется программно [1].

12—13 — предшествующий режим работы процессора; определяет режим, в котором находился процессор перед последним прерыванием,

14—15 — текущий режим работы процессора.

В каждый момент времени процессор может работать в одном из двух режимов: системном (привилегированном), т. е. в состоянии обработки системной программы, для которой доступны все ресурсы комплекса, и пользовательском (непривилегированном), т. е. в состоянии программы пользователя, работающей под управлением операционной системы. Для таких программ вводятся некоторые ограничения по использованию отдельных ресурсов

[1] Подробно приоритеты рассматриваются в п. 1.5.

комплекса, например запрещено использование некоторых инструкций.

Значение указанных разрядов 00 означает работу процессора в системном режиме, значение 11 — в пользовательском. Данные разряды используются только при работе с диспетчером памяти.

PS на общей шине имеет адрес 777776 и доступно программисту для анализа и модификации. Например, динамическое изменение приоритета процессора позволяет программе реального времени эффективно вести обработку событий в зависимости от их приоритетов.

Универсальные регистры. В состав процессора входят восемь универсальных регистров, пронумерованных от 0 до 7, которые используются для рабочих целей, для запоминания промежуточных результатов в процессе выполнения программы, в качестве индексных и т. п.

При программировании регистры задаются непосредственно в инструкциях. В зависимости от режима адресации каждый регистр содержит непосредственно операнд, либо адрес операнда, либо индекс. Мнемоническое обозначение регистров в инструкциях: R0, R1, R2, ..., R5, SP (Stack Pointer), PC (Program Counter). Регистры SP и PC используются процессором специальным образом. Регистр SP отдельными машинными инструкциями интерпретируется как указатель стека. Регистр PC используется как счетчик инструкций, т. е. всегда содержит адрес очередной инструкции, подлежащей выполнению.

Функциональные регистры[1]. Функциональные регистры используются аппаратным диспетчером памяти для преобразования виртуальных адресов в физические.

1.2. СТРУКТУРА ПАМЯТИ

Оперативная память в моделях СМ ЭВМ разделена на слова. Каждое слово имеет длину 16 разрядов и состоит из 2 байт по 8 разрядов каждый.

Нумерация разрядов в слове производится от 0 до 15, нумерация разрядов в байте — от 0 до 7. Старшинство разрядов в слове или в байте определяется в порядке возрастания их номеров. Правый байт слова называется младшим, левый байт — старшим байтом. Самый старший разряд (15 — в слове, 7 — в байте) называется знаковым. Структура слова изображена на рис. 1.3.

В моделях СМ ЭВМ предусмотрены инструкции, рассчитанные на работу со словами и с байтами, что обеспечивается их раздельной адресацией. При этом байт является наименьшей адресуемой единицей.

---

[1] Более подробно назначение функциональных регистров будет рассмотрено в гл. 2.

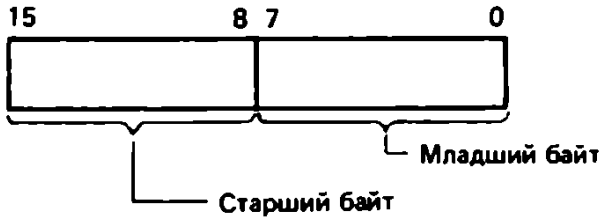


Рис. 1.3. Структура слова памяти

Каждый адрес слова или байта имеет длину 16 разрядов, адрес слова всегда четный. Допустимый диапазон 16-разрядных адресов (от 000000_8 до 177777_8) обеспечивает доступ к 32768 словам (32 Кслова) или 65536 байтам (64 Кбайта). На рис. 1.4 приведена структура памяти с учетом адресации слов и байтов.

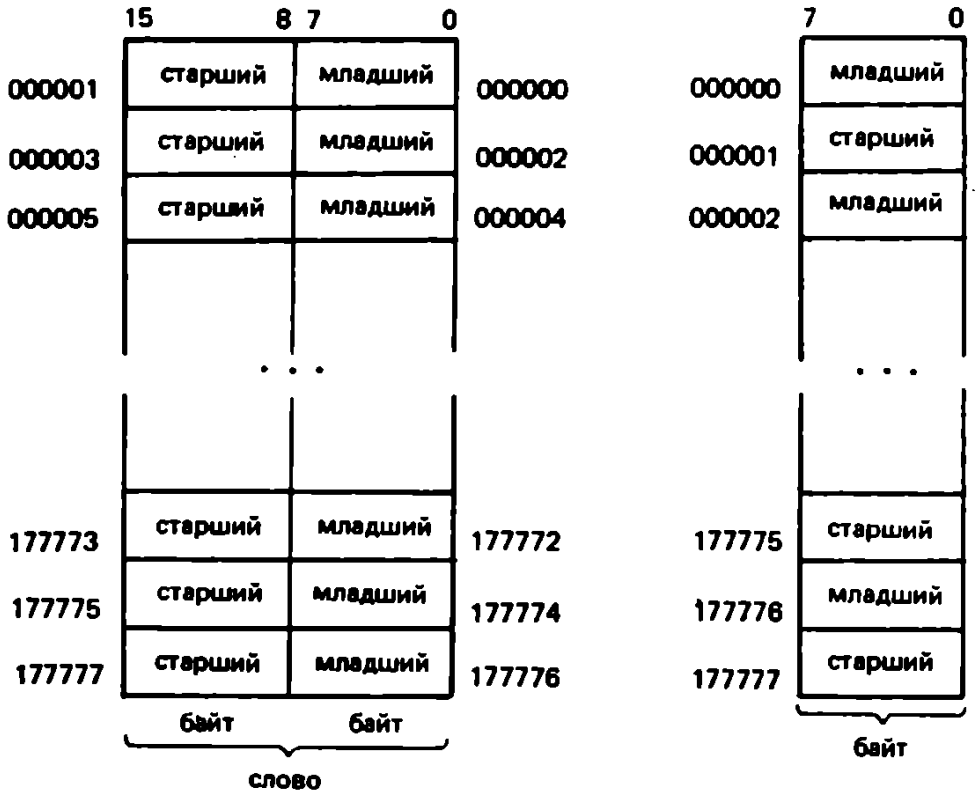


Рис. 1.4. Адресация памяти

Несмотря на то что процессор СМ1300 без диспетчера памяти обеспечивает 16-разрядную адресацию памяти, в распоряжении программиста остается только 28 Кслов памяти. Старшие 4 Кслова памяти с адресами от 160000_8 до 177777_8 в этом случае не используются, а соответствующие им старшие адреса ОШ зарезервированы для регистров внешних устройств и для отдельных регистров процессора.

При использовании на других моделях процессора с диспетчером памяти имеется возможность увеличения объема используемой для программы памяти до 124 Кслов (248 Кбайт) и до

1920 Кслов (3840 Кбайт) на СМ1420, СМ1600. Соответствие между адресами памяти и адресами ОШ рассмотрено в 1.4.

Нижний участок памяти с адресами от 00000 до 000777 обычно используется для хранения векторов прерываний. При этом первые 400_8 байт с адресами от 000000 до 000377 используются для хранения векторов прерываний стандартных системных устройств. Участок памяти с адресами от 000400 до 000777 может быть использован для хранения векторов прерываний дополнительных внешних устройств, подключенных к ЭВМ пользователями. Более подробно векторы прерываний рассматриваются в 1.5.

Общая структура памяти СМ1300 без диспетчера памяти приведена на рис. 1.5, а; структура памяти емкостью 128 Кслов — на рис. 1.5, б, емкостью 2048 Кслов для моделей СМ1420, СМ1600 — на рис. 1.5, в.

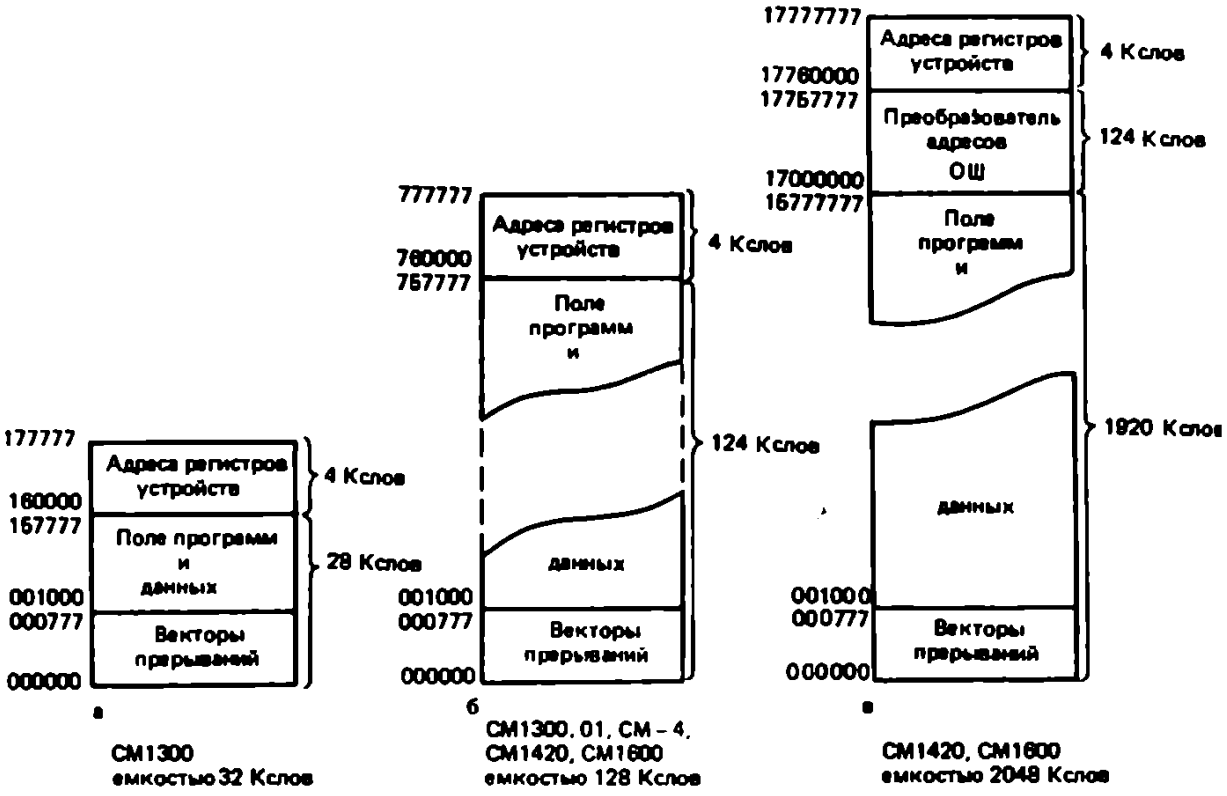


Рис. 1.5. Общая структура памяти моделей типа СМ-4

Особенности работы на СМ ЭВМ с диспетчером памяти приведены при описании диспетчера памяти в гл. 2.

Следует отметить, что на СМ1300, функционирующей без диспетчера памяти, размер программы не может превышать 28 Кслов. В моделях с диспетчером памяти, несмотря на то, что память для программ может быть увеличена до 1920 Кслов, объем одной программы не может превышать 32 Кслов. Использование остальной памяти в этом случае достигается за счет мультипрограммирования в соответствии с особенностями конкретных операционных систем СМ ЭВМ.

1.3. СТЕК

Стек — это непрерывный участок памяти, используемый для временного хранения данных. Занесение элементов в стек или извлечение элементов из стека может проводиться аппаратно или программно. Элементы пересылаются в стек и извлекаются из стека пословно или побайтно в соответствии с правилом LIFO (последний пришел — первый вышел).

Стек используется процессором при обработке прерываний и при работе с подпрограммами. В первом случае в стеке запоминается слово состояния процессора (PS) и адрес инструкции, находящийся в PC в момент прерывания. Во втором случае в стеке запоминаются параметры подпрограммы, адрес возврата и другая информация.

Место памяти, резервируемой под стек, определяется программистом; обычно это память в начале программы (область младших адресов). Стек линейно увеличивается по мере занесения в него элементов и уменьшается по мере их извлечения. На рис. 1.6 приведена схема занесения элементов в стек и их извлечения из стека.

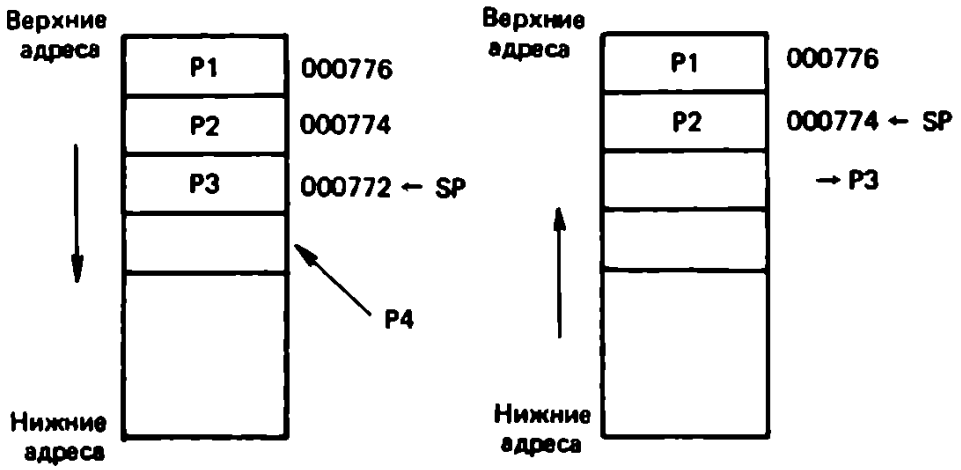


Рис. 1.6. Схема работы стека

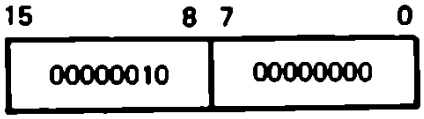
После добавления элемента P_3 к элементам P_2 и P_1, находящимся в стеке, стек увеличивается на одно слово. После извлечения P_3 стек соответственно уменьшается. Аналогичная ситуация возникает и при работе с байтами.

Адрес элемента, который был занесен в стек последним, должен находиться в регистре SP. В качестве SP может быть использован любой универсальный регистр, за исключением регистра 7 (PC). Однако инструкции процессора, работающие со стеком, например инструкции для работы с подпрограммами и инструкции прерываний, в качестве указателя стека всегда используют регистр 6, называемый указателем аппаратного стека. По этой причине при программировании вводятся системные соглашения, определяющие особые правила использования регистра SP.

Занесение и извлечение элементов и соответствующая корректировка указателя стека при занесении или извлечении элементов производятся аппаратно процессором, при работе соответствующих инструкций или программистом. В последнем случае применительно к регистру SP используется режим адресации с автоуменьшением при занесении элемента в стек и с автоувеличением при извлечении элементов из стека. Подробно режимы адресации рассмотрены в гл. 2. Начальное значение указателя стека (обычно это адрес первого слова программы) устанавливается программно.

Занесению очередного элемента в стек всегда предшествует уменьшение SP на 2 (как при работе со словами, так и при работе с байтами) Соответствующее увеличение SP производится после извлечения элемента из стека. В примере, приведенном на рис. 1.6, начальным значением SP был адрес 001000. После занесения в стек элемента P_3 в SP находился адрес 000772, после извлечения P_3 из стека SP указывает на адрес 000774.

Ограничитель стека[1]. Для обеспечения защиты участков памяти, например поля векторов прерываний, при работе со стеком введен ограничитель аппаратного стека SL (Stack Limiter), позволяющий программисту следить за его переполнением. Ограничитель стека — это функциональный регистр процессора, имеющий адрес на ОШ. Младший байт ограничителя стека содержит нули. В старший байт помещается величина, которая вместе с содержимым младшего байта образует адрес, кратный 400_8 байтам (200_8 слов). Например, содержимое ограничителя стека задает адрес 1000_8.



Адрес, задаваемый в ограничителе стека, определяет его нижнюю границу. Ниже границы стека находится так называемая «желтая зона», состоящая из 16 слов. При обращении в эту зону фиксируется ее нарушение, т. е. операции в этой зоне завершаются, и происходит прерывание работы программы по вектору с адресом 4 (см. 1.5).

Ниже «желтой зоны» находится «красная зона», нарушение которой прекращает работу со стеком и вызывает прерывание работы программы по вектору с адресом 4.

Зоны определяются следующим образом:

«красная зона» $\leqslant$ SL+337_8;

«желтая зона» = SL+(от 340_8 до 377_8),

где SL — адрес, находящийся в ограничителе стека.

Если SL равен 0, то «красная зона» занимает ячейки памяти с адресами от 0 до 337_8, «желтая зона» — от 340_8 до 377_8. Несложно заметить, что ограничитель стека защищает векторы прерываний внешних устройств. Он устанавливается и изменяется

[1] В СМ1300 отсутствует.

программным путем. На рис. 1.7 приведено изображение стека с использованием SL.

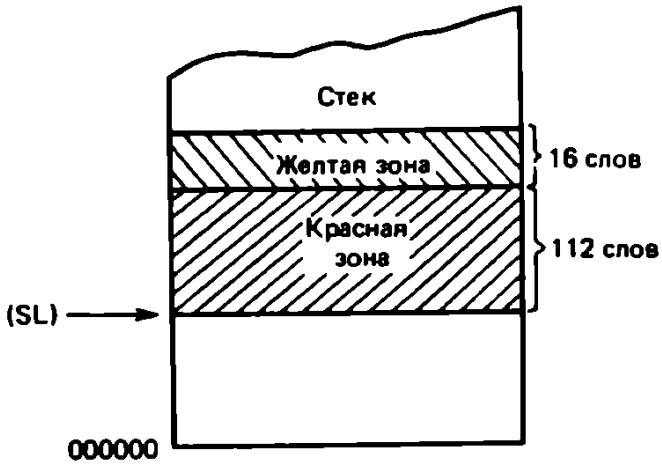


Рис. 1.7. Схематическое изображение стека с использованием SL

1.4. ОБЩАЯ ШИНА

Физически ОШ представляет собой высокочастотную магистраль передачи данных, состоящую из 56 линий, по которым передается вся информация, циркулирующая в ЭВМ; данные, адреса, управляющие сигналы.

Ячейкам памяти, отдельным регистрам процессора и регистрам внешних устройств на ОШ присваиваются 18-разрядные адреса.

При работе программ на СМ1300 без диспетчера памяти 16-разрядные адреса памяти (за исключением адресов старших 4 Кслов) отображаются в 18-разрядные адреса ОШ путем аппаратного добавления к 16-разрядным адресам памяти двух дополнительных разрядов, содержащих нули.

На рис. 1.8 приведено отображение 16-разрядного адреса памяти в 18-разрядный адрес ОШ.

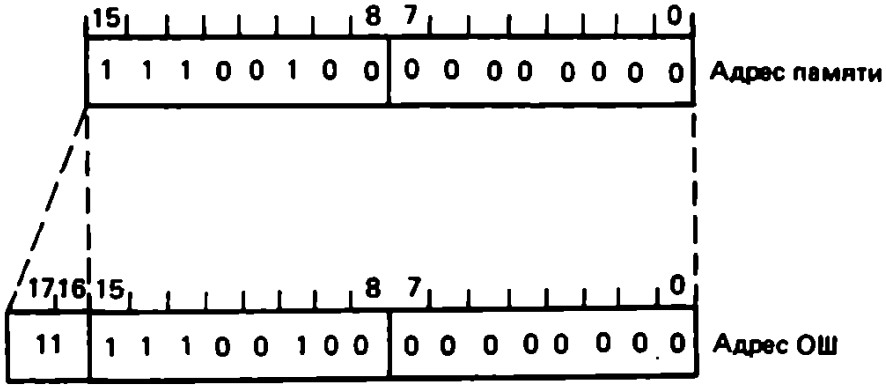


Рис. 1.8. Пример отображения адреса памяти в адрес ОШ на СМ1300

Адреса старших 4 Кслов, т. е. адреса в диапазоне от 160000_8 до 177777_8, отображаются в 18-разрядные адреса ОШ путем аппаратного добавления к 16-разрядным каждого адреса памяти из указанного диапазона двух дополнительных разрядов, содержащих единицы (рис. 1.9).

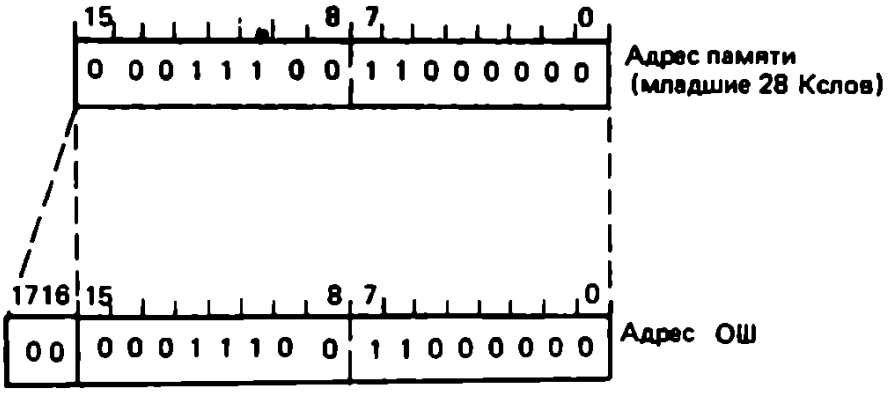


Рис. 1.9. Пример отображения адреса памяти (из старших 4 Кслов) в адрес ОШ на СМ1300

Старшие адреса ОШ в диапазоне от 760000 до 777777 отведены регистрам внешних устройств и некоторым регистрам процессора. Поэтому на СМ1300 соответствующие им адреса памяти в диапазоне от 160000 до 177777 рассматриваются как адреса регистров устройств, и обращение в программах по этим адресам обычно проводится при программировании операций ввода-вывода.

Отображение 16-разрядного адресного пространства на 18-разрядное адресное пространство ОШ в комплексах СМ1300 без диспетчера памяти приведено на рис. 1.10.

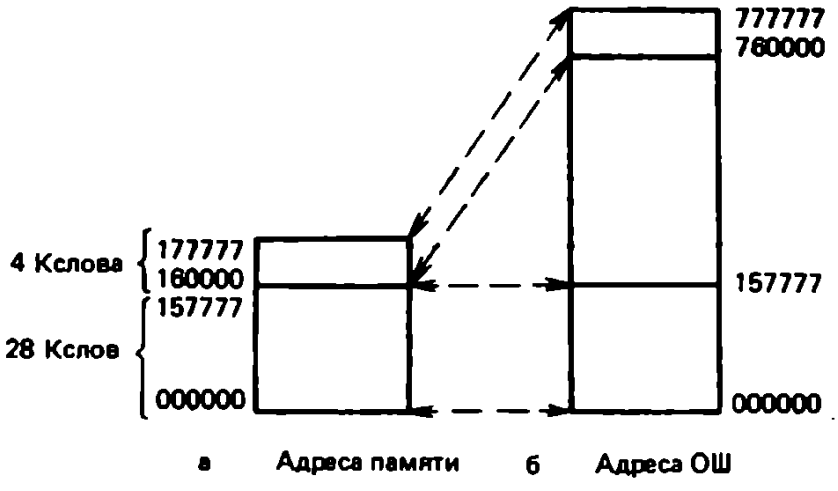


Рис. 1.10. Отображение адресного пространства процессора на адресное пространство ОШ

Аналогичное отображение адресов для комплексов с диспетчером памяти рассматривается в гл. 2 при описании диспетчера памяти.

Распределение адресов ОШ в интервале от 760000 до 777777 приведено в приложении 2.

Подключение устройств к ОШ производится в соответствии с заданными уровнями приоритета.

В моделях типа СМ-4 предусмотрены уровни приоритета с номерами 4, 5, 6, 7 и уровень внепроцессорного обмена, на каждый из которых может быть подключено несколько разных устройств. Номер уровня определяет приоритет устройств по отношению друг к другу и к процессору. Приоритет процессора, как уже указывалось ранее, устанавливается программным путем. Уровень приоритета устройства определяет возможность обслуживания поступившего от устройства запроса с целью получения доступа к ОШ, например, для передачи данных. Чем выше номер уровня, тем выше приоритет устройства. Максимальный приоритет имеют устройства, подключенные к уровню внепроцессорного обмена, т. е. обмена данными между устройством и памятью или другими устройствами без участия процессора. Приоритет остальных устройств определяется порядком номеров уровней по убыванию: 7, 6, 5, 4.

Если несколько устройств подключено к одному и тому же уровню приоритета, то устройства, подключенные физически ближе к процессору, имеют более высокий приоритет.

При программировании следует учитывать, что если приоритет процессора, задаваемый в PS, имеет тот же номер, что и номер уровня приоритетов каких-либо устройств, то приоритет процессора считается выше, чем приоритет соответствующих устройств. Это означает, что работа процессора не может быть прервана запросами от этих устройств до тех пор, пока в PS программно не будет установлен более низкий приоритет.

Исключение составляют запросы, поступающие от устройств, подключенных к линии внепроцессорного обмена, которые для получения доступа к ОШ могут прерывать работу процессора независимо от его приоритета.

Особое значение имеют случаи, когда процессор выполняет инструкцию с плавающей запятой (только для СМ-4) или сам занимает ОШ. В первом случае процессору дается не более 8 мкс для завершения операции, во втором случае ОШ остается в распоряжении процессора до конца передачи данных.

Расположение устройств на ОШ в соответствии с уровнями их приоритетов приведено на рис. 1.11.

Как видно из схемы, приведенной на рис. 1.11, наивысший приоритет имеет устройство У9, затем в порядке убывания приоритета: У7, У8, У4, У5, У6, У3, У1, У2.

Взаимодействие устройств, подключенных к общей шине, осуществляется по принципу «задатчик-исполнитель». Задатчиком является управляющее устройство, т. е. устройство — инициатор обмена, исполнителем — управляемое устройство. В качестве примера можно привести взаимодействие между процессором (задатчик) и памятью (исполнитель).

Память всегда является исполнителем. Все остальные устройства могут быть как задатчиками, так и исполнителями.

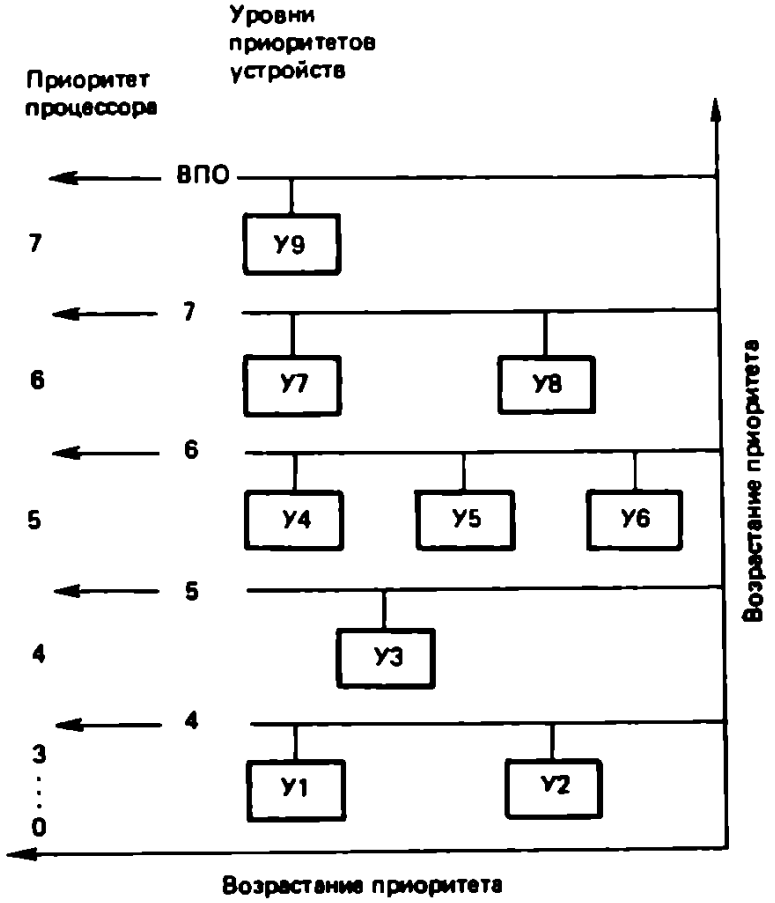


Рис. 1.11. Расположение устройств на ОШ: ВПО— внепроцессорный обмен; У1—У9 — обозначение устройств

Для инициации обмена задатчик выставляет запрос на ОШ. Специальное устройство процессора, которое управляет приоритетными прерываниями, называемое «арбитр», определяет возможность занятия задатчиком ОШ в зависимости от занятости шины и приоритета запрашиваемого устройства. Запрос удовлетворяется, если приоритет устройства выше приоритетов других устройств, занимающих ОШ или пославших запрос на ее занятие одновременно. Если разрешение на занятие ОШ получено, то синхронизируется исполнитель и начинается обмен данными между ним и задатчиком.

Обмен производится под управлением процессора для устройств, подключенных к уровням с номерами от 4 до 7 (программная передача), или, минуя процессор, для устройств, подключенных к уровню внепроцессорного обмена (внепроцессорная передача). В последнем случае во время обмена процессор может быть занят обработкой программы.

1.5. СИСТЕМА ПРЕРЫВАНИЙ

Система прерываний комплексов типа СМ-4, построенная в соответствии с многоуровневой системой приоритетов устройств и процессора, обеспечивает быструю реакцию процессора на внутренние и внешние события. Команда прерывания вызывает прекращение работы выполняющейся программы и запуск программы обслуживания прерывания. Например, если устройство-задатчик получает доступ к ОШ и выставляет процессору сигнал прерывания, то вызывается программа, обслуживающая данное устройство.

Различают три вида прерываний: внешние прерывания, инициируемые запросами внешних устройств; внутренние прерывания, инициируемые средствами диагностики процессора (например, ошибочная машинная инструкция, нечетный адрес); программные прерывания, инициируемые специальными машинными инструкциями. Обработка прерываний осуществляется в соответствии с приоритетами источников прерываний.

Обработка прерываний. Каждому источнику прерываний в оперативной памяти отводится вектор прерываний, состоящий из двух слов. Первое слово содержит адрес запуска программы обработки прерывания PC; второе слово — PS, которое устанавливается для этой программы. Для размещения векторов прерываний отводится нижний участок оперативной памяти, который начинается с нулевого адреса (см. 1.2). В этом участке памяти каждому типу внешних устройств отведен свой адрес для размещения соответствующего вектора прерывания. Аналогичные адреса отведены для векторов прерываний от внутренних источников и от специальных машинных инструкций.

Независимо от причины обработка прерываний осуществляется идентичным образом по следующему алгоритму:

1. Одновременно с сигналом прерывания процессору сообщается адрес вектора источника, который послал сигнал прерывания (например, внешнего устройства);
2. PC_c и PS_c выполняемой в данный момент программы (старые PC и PS) запоминаются во внутренних регистрах процессора.
3. Из полученного процессором вектора прерывания извлекаются новые $PC_н$ и $PS_н$, после чего PC_c и PS_c пересылаются в стек.
4. В соответствии с новыми значениями $PC_н$ и $PS_н$ производится запуск программы обслуживания прерывания.
5. Программа обслуживания прерывания должна завершаться машинной инструкцией «выход из прерывания» RTI, выполнение которой состоит в восстановлении из стека старых значений PC_c и PS_c и возобновлении работы ранее прерванной программы.

На рис. 1.12 приведена диаграмма, поясняющая процесс прерывания.

В интервале времени $[t_0—t_1]$ выполняется программа А. По сигналу прерывания, который получен процессором в момент t_1,

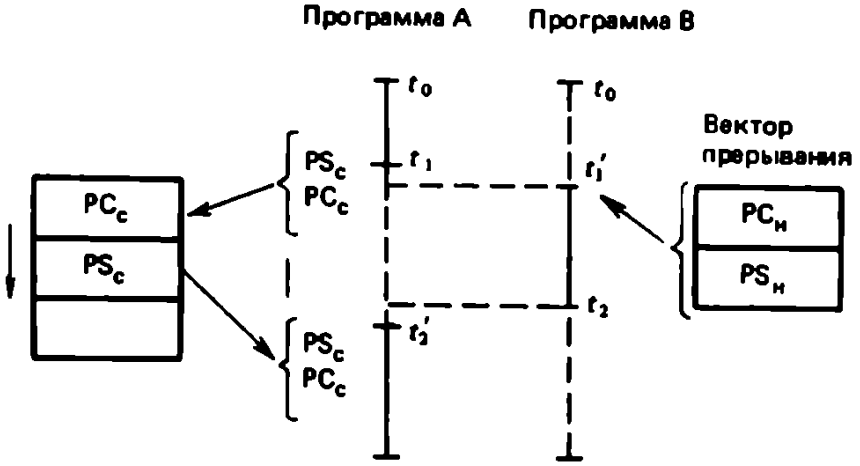


Рис. 1.12. Диаграмма процесса прерывания

PC_c и PS_c программы А пересылаются в стек, а из вектора прерывания извлекаются и устанавливаются $PC_н$ и $PS_н$, соответствующие программе обработки прерывания (программа В), которая запускается в момент времени t_1.

$$t_1' = t_1 + t_{пр},$$

где $t_{пр}$ — время обработки прерывания (см. приложение 1).

После завершения работы программы В в момент времени t_2 из стека восстанавливаются PC_c и PS_c и в момент времени t_2' возобновляется работа программы А.

Здесь

$$t_2' = t_2 + t_{Впр},$$

где $t_{Впр}$ — время выхода из прерывания.

Обработка прерываний построена по иерархическому принципу. Это означает, что работа одной программы обслуживания прерывания может быть прервана по сигналу прерывания от источника с более высоким приоритетом; в свою очередь работа этой программы может быть прервана другим прерыванием и т. д. На рис. 1.13 приведен пример временной диаграммы работы нескольких программ обслуживания прерываний. Эта диаграмма иллюстрирует работу программы А (во время ее работы приоритет процессора равен 4) и программ обслуживания устройств D_1 (приоритет устройства 5) и D_2 (приоритет устройства 6). В момент времени t_1 устройство D_1 выставило запрос на получение ОШ для передачи данных. Далее работа программы А прерывается (приоритет процессора меньше приоритета устройства D_1) и в момент t_1' начинает работать программа обслуживания устройства D_1. В момент времени t_2 поступает аналогичный запрос на передачу данных от устройства D_2, работа программы D_1 прерывается, и в момент времени t_2' ($t_2' = t_2 + t_{пр}$) начинает работу программа D_2. После завершения работы этой программы в момент времени t_3' ($t_3' = t_3 + t_{Впр}$) возобновляется работа программы D_1, по оконча-

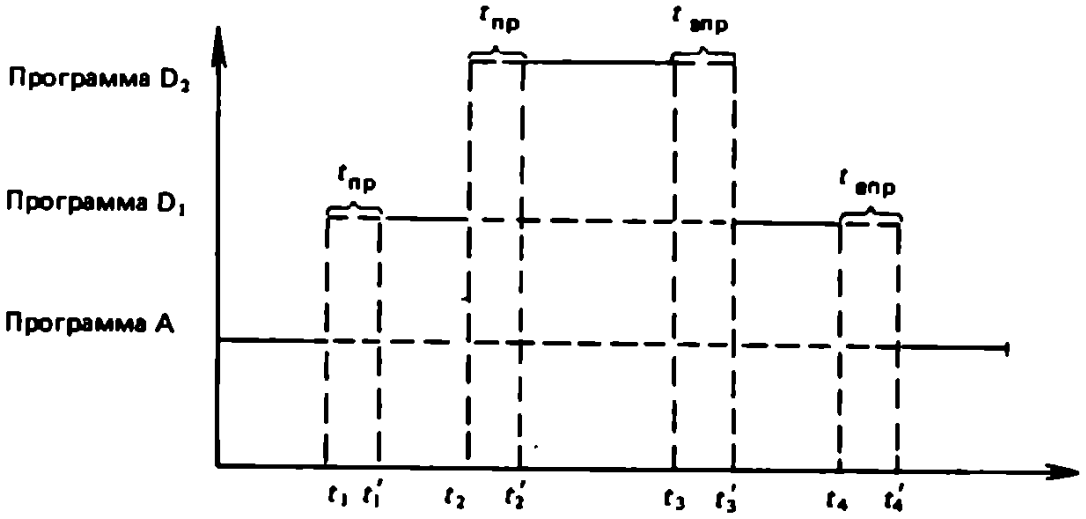


Рис. 1.13. Диаграмма работы программ обслуживания прерываний

нии которой продолжается работа программы А (в момент t_4', $t_4' = t_4 + t_{Впр}$).

Следует отметить, что в системе фиксированы только адреса векторов прерываний. Заполнение соответствующих ячеек, т. е. занесение значений PC и PS в векторы прерываний, осуществляется программным образом. Как правило, это делается операционной системой.

Адреса стандартных векторов прерываний приведены в приложении 3.

2

ГЛАВА

ПРЕДСТАВЛЕНИЕ ДАННЫХ И МАШИННЫЕ ИНСТРУКЦИИ

Машинные инструкции всех моделей типа СМ-4 идентичны и делятся на следующие группы: одноадресные, двухадресные, передачи управления и безадресные.

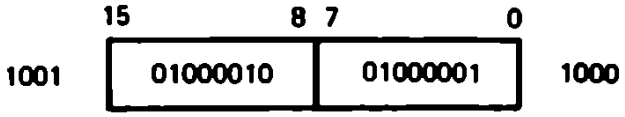
Безадресные, или служебные, инструкции состоят из одного кода операции. Одноадресные инструкции состоят из кода операции и одного операнда (приемника), двухадресные — из кода операции и двух операндов (источника и приемника). В инструкциях передачи управления указываются код операции и константа перехода либо код операции и операнд, определяющий адрес перехода. В большинстве инструкций для каждого операнда задаются режим адресации, уточняющий адрес перехода и местонахождение операнда в регистре или в памяти. В зависимости от кода операции и режимов адресации операндов инструкции могут занимать одно, два или три слова.

2.1. ФОРМАТЫ ДАННЫХ

Данные, обрабатываемые машинными инструкциями, делятся на три группы: логические коды, числа с фиксированной запятой и числа с плавающей запятой.

Логические коды могут размещаться в отдельных байтах и в словах. Для их представления используются все разряды: для байта — от 0-го до 7-го, для слова — от 0-го до 15-го. Логическими кодами могут быть представлены: символьные величины, числа без знака и битовые величины.

Символьные величины задаются в коде КОИ-7 (см. приложение 4), каждый символ занимает один байт, разряд 7 которого всегда содержит 0. Например, символы АВ размещаются в байтах с адресами 1000 и 1001 следующим образом:



где 101_8 — код символа A; 102_8 — код символа B.

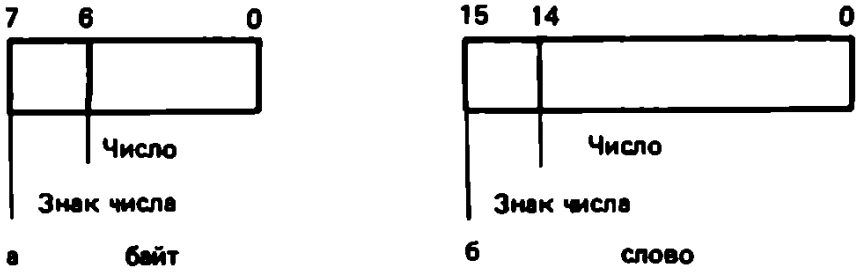
Числа без знака имеют диапазон представления: от 000 до 377_8 — для байта; от 000000 до 177777_8 — для слова.

Упорядоченность чисел без знака определяется старшинством всех составляющих его разрядов и имеет существенное значение при выполнении инструкций, связанных с передачей управления.

Битовые величины задают значения отдельных разрядов байта или слова.

Указанные разновидности логических кодов различаются только в контексте обрабатывающих их машинных инструкций и практически какого-либо самостоятельного значения не имеют.

Число с фиксированной запятой может занимать байт или слово. Если число с фиксированной запятой занимает байт, то для его представления используются разряды с 0-го по 6-й. Разряд 7 называется знаковым. При размещении числа с фиксированной запятой в слове для его представления используются разряды с 0-го по 14-й. Знак числа содержится в разряде 15. Ниже приведен формат числа с фиксированной запятой.



Значение знакового разряда: 0 — для положительных чисел; 1 — для отрицательных чисел.

Отрицательные числа представляются в дополнительном коде. Напомним, что дополнительный код образуется путем инверсии числовых разрядов (0—6 — для байта, 0—14 — для слова) и добавления 1 к самому младшему разряду. Примеры представления чисел с фиксированной запятой:

Число	Восьмеричный код — байт	Восьмеричный код — слово
+5	005	000005
—5	373	177773
0	000	000000

Диапазон представления чисел с фиксированной запятой: для байта — от -128_{10} до $+127_{10}$; для слова — от -32768_{10} до $+32767_{10}$.

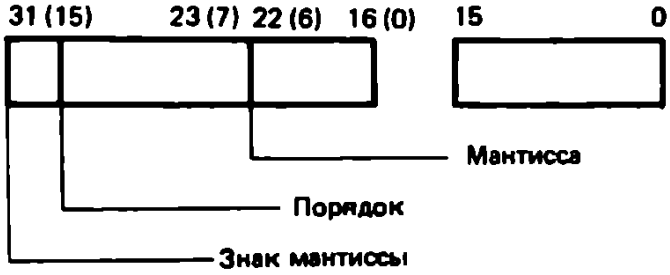
Числа с плавающей запятой могут быть одинарной или двойной точности.

Выполнение операций над числами с плавающей запятой осуществляется:

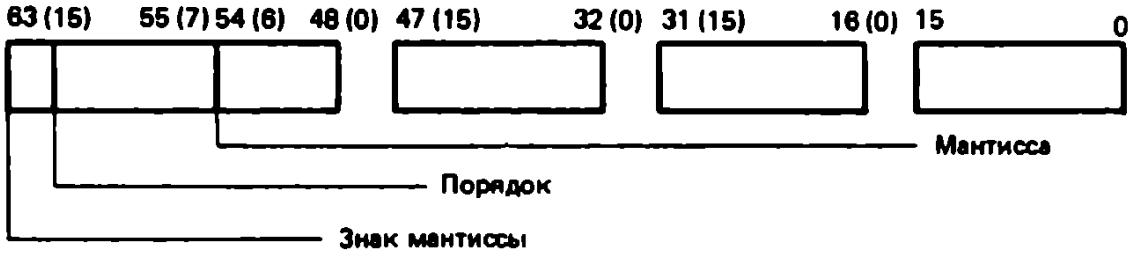
программным способом для моделей СМ ЭВМ, не имеющих аппаратных средств для работы с плавающей арифметикой.

аппаратным способом для моделей СМ ЭВМ, оснащенных расширителем (Floating Instruction Set—FIS) или процессором плавающей запятой (Floating Point Processor—FPP).

Число одинарной точности занимает два слова (32 разряда) и представляется в формате



Число двойной точности занимает 4 слова (64 разряда) и представляется в формате



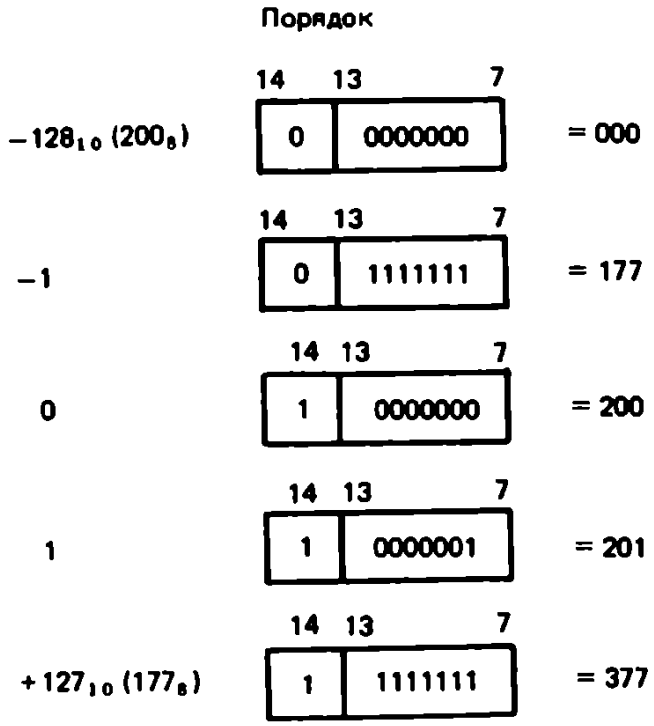
Знак мантиссы определяет знак числа и имеет значения: 0 — для положительных чисел, 1 — для отрицательных чисел.

Мантисса отрицательных чисел представляется в дополнительном коде.

Порядок числа с плавающей запятой изменяется в диапазоне от -128_{10} (200_8) до $+127_{10}$ (177_8) и в разрядах 7—14 первого слова запоминается увеличенным на 200_8 (128) (см. пример на с. 23).

Такой способ представления порядка называется смещенным. Смещенный порядок 0 (-200_8) имеет особое значение, о котором подробно будет сказано в п. 2.8.

Мантисса перед выполнением операций с плавающей запятой должна быть нормализованной, т. е. десятичная запятая должна располагаться левее самого старшего значащего разряда. При нормализации производится соответствующее изменение порядка.



Старший разряд мантиссы нормализованного числа не хранится, т. е. имеет место так называемый скрытый разряд. Однако при аппаратном выполнении операций этот разряд автоматически восстанавливается и учитывается при выполнении операций. Таким образом, хотя для мантиссы отведено 23 разряда — для чисел одинарной точности и 55 разрядов — для чисел двойной точности, в операциях участвуют 24 и 56 разрядов соответственно. Порядок числа учитывает скрытый старший разряд мантиссы.

Следует отметить, что нормализованная мантисса m всегда представляется в диапазоне

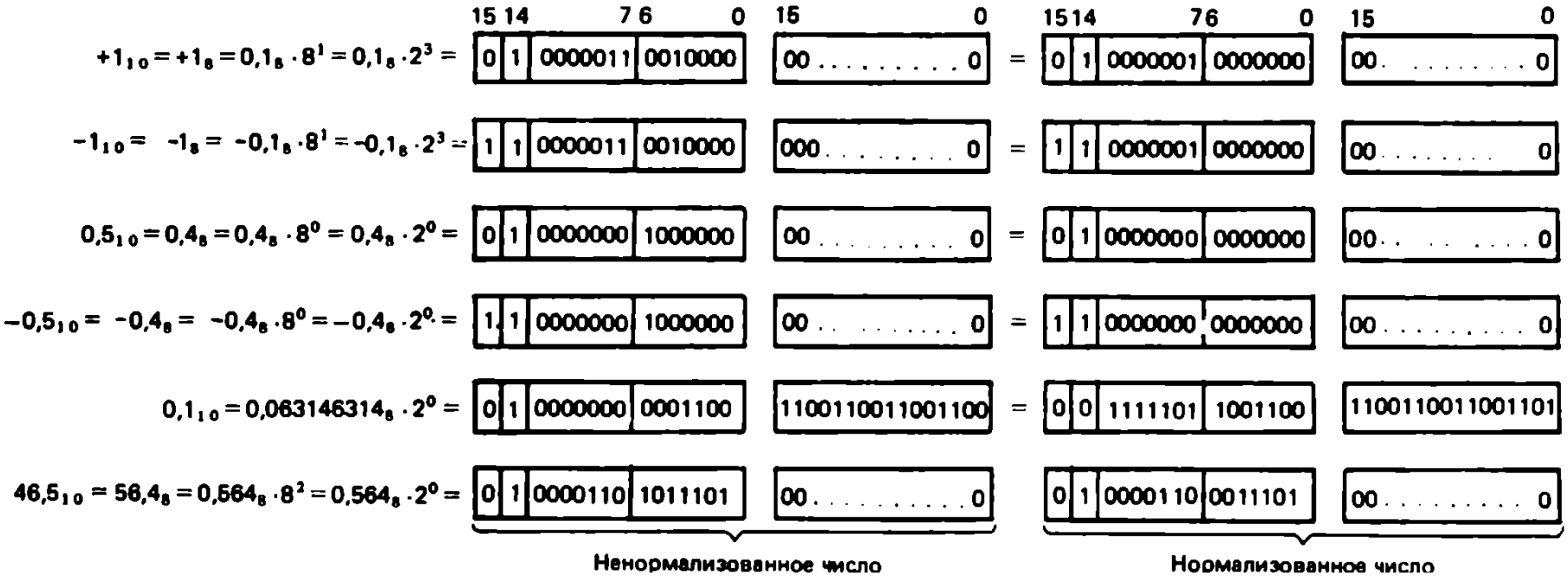
$$0{,}5 \leqslant m < 1.$$

Фактически допустимый при выполнении операций с плавающей запятой порядок от -128_{10} до $+127_{10}$ и представление мантиссы в нормализованном виде определяют диапазон представления чисел с плавающей запятой от $\pm 2{,}9 \cdot 10^{-37}$ до $1{,}7 \cdot 10^{38}$, который одинаков для одинарной и двойной точности.

Примеры представления чисел с плавающей запятой приводятся на с. 24.

2.2. РЕЖИМЫ АДРЕСАЦИИ ОПЕРАНДОВ

В машинных инструкциях режимы адресации операндов задаются специальными кодами, местоположение которых в инструкциях строго фиксировано. При программировании для обозначения режимов адресации введены специальные соглашения, которые учитываются языком макроассемблера (см. гл. 3). Для опи-


15 14 7 6 0
15 0
15 14 7 6 0
15 0
$+1_{10} = +1_8 = 0{,}1_8 \cdot 8^1 = 0{,}1_8 \cdot 2^3 =$
0 1 0000011 0010000
00 0
=
0 1 0000001 0000000
00 0
$-1_{10} = -1_8 = -0{,}1_8 \cdot 8^1 = -0{,}1_8 \cdot 2^3 =$
1 1 0000011 0010000
000 0
=
1 1 0000001 0000000
00 0
$0{,}5_{10} = 0{,}4_8 = 0{,}4_8 \cdot 8^0 = 0{,}4_8 \cdot 2^0 =$
0 1 0000000 1000000
00 0
=
0 1 0000000 0000000
00 0
$-0{,}5_{10} = -0{,}4_8 = -0{,}4_8 \cdot 8^0 = -0{,}4_8 \cdot 2^0 =$
1 1 0000000 1000000
00 0
=
1 1 0000000 0000000
00 0
$0{,}1_{10} = 0{,}063146314_8 \cdot 2^0 =$
0 1 0000000 0001100
1100110011001100
=
0 0 1111101 1001100
1100110011001101
$46{,}5_{10} = 56{,}4_8 = 0{,}564_8 \cdot 8^2 = 0{,}564_8 \cdot 2^0 =$
0 1 0000110 1011101
00 0
0 1 0000110 0011101
00 0
Ненормализованное число
Нормализованное число

сания режимов адресации введем следующие обозначения: R — выражение, определяющее универсальный регистр, E — выражение, A — режим адресации.

Различают следующие виды адресации: прямую адресацию, косвенную адресацию, адресацию с использованием PC. Каждому из указанных видов соответствует по четыре режима адресации, которые рассматриваются ниже.

2.2.1. РЕЖИМЫ ПРЯМОЙ АДРЕСАЦИИ

Прямая адресация через регистр (код адресации 0). Формат

«A» : R

Операнд задается непосредственно в регистре, определяемом выражением R. Например:

```
CLR R0; ДАННАЯ ИНСТРУКЦИЯ ОЧИЩАЕТ РЕГИСТР 0
```

Прямая адресация с автоувеличением (код адресации 2). Формат

«A» : (R)+

Регистр R содержит адрес операнда и после выполнения инструкции (за исключением особых случаев) увеличивается на 2 для операций над словами и на 1 — для операций над байтами. Например:

```
INCB  (R1)+   ; СОДЕРЖИМОЕ БАЙТА, АДРЕС КОТОРОГО
              ; НАХОДИТСЯ В R1, УВЕЛИЧИВАЕТСЯ НА 1.
              ; R1 ТАКЖЕ УВЕЛИЧИВАЕТСЯ НА 1.
CLR   (R0)+   ; КАЖДАЯ ДАННАЯ ИНСТРУКЦИЯ ОЧИЩАЕТ СЛОВО
CLR   (R4)+   ; ПО АДРЕСУ, СОДЕРЖАЩЕМУСЯ В УКАЗАННОМ РЕГИСТРЕ,
CLR   (R2)+   ; И УВЕЛИЧИВАЕТ СОДЕРЖИМОЕ ЭТОГО РЕГИСТРА НА 2.
```

Особые случаи. В инструкциях JMP в СМ-3 (безусловный переход) и JSR (переход к подпрограмме), использующих режим адресации с автоувеличением, содержимое регистра увеличивается перед его использованием. В двухадресных инструкциях с режимами адресации «R», «(R)+» или «R», «—(R)», где источником и приемником является один и тот же регистр, операнд источника извлекается как уже увеличенная или уменьшенная величина, а регистр приемника в момент его использования еще содержит ранее существовавший адрес.

Пример. Пусть в R0 первоначально содержится 100, тогда

```
MOV  R0, (R0)+   ; ВЕЛИЧИНА 102 ПЕРЕСЫЛАЕТСЯ ПО АДРЕСУ 100
MOV  R0, -(R0)   ; ВЕЛИЧИНА 76 ПЕРЕСЫЛАЕТСЯ ПО АДРЕСУ 100
```

Прямая адресация с автоуменьшением (код адресации 4). Формат

«A» : — (R)

Перед выполнением операции регистр R, содержащий адрес операнда, уменьшается на 2 для операций над словами и на 1 — для операций над байтами. Например:

```
CLR  -(R0)     ; СОДЕРЖИМОЕ РЕГИСТРОВ R0, R2, R4
CLR  -(R2)     ; УМЕНЬШАЕТСЯ НА 2, УКАЗЫВАЯ НА
CLR  -(R4)     ; АДРЕСА ОЧИЩАЕМЫХ СЛОВ
CLRB -(R1)     ; СОДЕРЖИМОЕ РЕГИСТРОВ R1 И R3
CLRB -(R3)     ; УМЕНЬШАЕТСЯ НА 1, УКАЗЫВАЯ
               ; НА АДРЕСА ОЧИЩАЕМЫХ БАЙТОВ
```

Индексация (код адресации 6). Формат

«A» : E(R)

Значение выражения E хранится в памяти как второе или третье слово инструкции. Действительный адрес вычисляется как сумма величины E и содержимого регистра R. Величина E есть смещение, содержимое регистра R — база. Например:

```
CLRB  -2(R3)   ; ДЕЙСТВИТЕЛЬНЫЙ АДРЕС
               ; ОЧИЩАЕМОГО БАЙТА РАВЕН - 2 ПЛЮС СОДЕРЖИМОЕ
               ; РЕГИСТРА 3
CLR   X+2(R1)  ; ДЕЙСТВИТЕЛЬНЫЙ АДРЕС СЛОВА,
               ; СОДЕРЖИМОЕ КОТОРОГО ОЧИЩАЕТСЯ,
               ; РАВЕН X+2 ПЛЮС СОДЕРЖИМОЕ РЕГИСТРА 1
```

2.2.2. РЕЖИМЫ КОСВЕННОЙ АДРЕСАЦИИ

Косвенная адресация через регистр (код адресации 1). Формат

«A» : @R или (R)

Регистр R содержит адрес операнда. Например:

```
CLR  @R1       ; ЭТИ ИНСТРУКЦИИ ОЧИЩАЮТ СЛОВО ПО
CLR  (R1)      ; АДРЕСУ, НАХОДЯЩЕМУСЯ В РЕГИСТРЕ R1
```

Косвенная адресация с автоувеличением (код адресации 3). Формат

«A» : @(R)+

Регистр R содержит указатель (адрес) адреса операнда. По-

сле выполнения инструкции содержимое регистра R увеличивается на 2. Например:

```
CLR  @(R3)+     ; СОДЕРЖИМОЕ R3 УКАЗЫВАЕТ НА ЯЧЕЙКУ
                ; (СЛОВО) ПАМЯТИ, СОДЕРЖАЩУЮ АДРЕС ОЧИЩАЕМОГО
                ; СЛОВА. ПОСЛЕ ВЫПОЛНЕНИЯ ОПЕРАЦИИ R3
                ; УВЕЛИЧИВАЕТСЯ НА 2
```

Косвенная адресация с автоуменьшением (код адресации 5). Формат

«A» :@— (R)

Содержимое регистра R уменьшается на 2 и рассматривается как адрес адреса операнда. Например:

```
CLR  @–(R1)     ; СОДЕРЖИМОЕ РЕГИСТРА 1 УМЕНЬШАЕТСЯ НА 2,
                ; УКАЗЫВАЯ НА ЯЧЕЙКУ ПАМЯТИ,
                ; СОДЕРЖАЩУЮ АДРЕС ОЧИЩАЕМОГО СЛОВА
```

Косвенная индексация (код адресации 7). Формат

«A» : @E(R)

Выражение E плюс содержимое регистра R дает указатель адреса операнда, как и при индексации, величина E называется смещением, содержимое регистра R — базой. Например:

```
CLR  @114(R4)   ; ЕСЛИ РЕГИСТР 4 СОДЕРЖИТ 100, ТО УКАЗАТЕЛЬ
                ; ПОЛУЧАЕТ ЗНАЧЕНИЕ 214, И ЕСЛИ ЯЧЕЙКА ПАМЯТИ
                ; 214 СОДЕРЖИТ 2000, ТО ОЧИЩАЕТСЯ СОДЕРЖИМОЕ
                ; ЯЧЕЙКИ 2000
```

2.2.3. РЕЖИМЫ АДРЕСАЦИИ С ИСПОЛЬЗОВАНИЕМ PC

Как уже указывалось, PC, в качестве которого используется регистр 7, всегда содержит адрес инструкции, подлежащей выполнению. Однако в ряде случаев этот регистр может быть задан, кроме того, в качестве операндов инструкции. В этом случае используются специальные режимы адресации, рассматриваемые ниже.

Непосредственная адресация (код адресации 2). Формат

«A» : #E

Этот режим адресации обеспечивает работу с операндами, заданными непосредственно в инструкции. При непосредственном режиме адресации операнд E запоминается во втором или третьем слове инструкции. Этот режим реализуется как автоувеличение содержимого регистра 7. Например:

```
MOV   #100, R0      ; ПЕРЕСЛАТЬ 100 (ВОСЬМЕРИЧНОЕ) В РЕГИСТР 0
MOV   #X, R0        ; ПЕРЕСЛАТЬ ЗНАЧЕНИЕ СИМВОЛА X В РЕГИСТР 0
```

Абсолютная адресация (код адресации 3). Формат

«A»:@#E

В этом режиме E указывает абсолютный адрес, который хранится во втором или третьем слове инструкции. Значение слова, следующего непосредственно за инструкцией, воспринимается как абсолютный адрес операнда. Например:

```
MOV   @#100, R0     ; ПЕРЕСЛАТЬ ЗНАЧЕНИЕ СОДЕРЖИМОГО ЯЧЕЙКИ 100
                    ; (ВОСЬМЕРИЧНОЕ) В РЕГИСТР 0
CLR   @#X           ; СОДЕРЖИМОЕ ЯЧЕЙКИ, АДРЕС КОТОРОЙ УКАЗАН
                    ; СИМВОЛОМ X, ОЧИЩАЕТСЯ
```

Абсолютная адресация интерпретируется как косвенное автоувеличение регистра 7. Например, инструкция

MOV @ #100, R0

помещенная в памяти с адресом 20, имеет вид:

ячейка 20 : 013700
ячейка 22 : 000100
ячейка 24 : следующая инструкция

Заметим, что абсолютный адрес 100 размещается в следующем за инструкцией слове, т. е. как второе слово инструкции. Процессор извлекает первое слово (код инструкции MOV) и увеличивает PC на 2, т. е. указывает на ячейку 22, которая содержит абсолютный адрес исходного операнда. После того как происходит извлечение следующего слова и увеличение PC на 2, содержимое абсолютного адреса передается в регистр 0. PC указывает на ячейку 24 (следующая инструкция).

Относительная адресация (режим адресации 6). Формат

«A» : E

Относительная адресация наиболее часто применяется при обращении к памяти. Например:

```
CLR   100           ; ОЧИСТИТЬ ЯЧЕЙКУ 100
MOV   X, Y          ; ПЕРЕСЛАТЬ СОДЕРЖИМОЕ ЯЧЕЙКИ X В ЯЧЕЙКУ Y
```

Этот вид адресации интерпретируется как индексный режим с использованием регистра 7 (PC) в качестве индексного регистра.

Смещение для вычисления адреса хранится во втором или третьем слове инструкции и задается как число, которое прибавляется к содержимому PC и дает адрес операнда. Таким образом, смещение есть E—PC; E — адрес операнда. Например, инструкция MOV 100, R3, размещенная в памяти по абсолютному адресу 20, имеет следующий вид:

```
ячейка 20:   016703
ячейка 22:   000054
ячейка 24:   следующая инструкция
```

Заметим, что константа 54 размещается во втором слове инструкции.

Процессор извлекает инструкцию MOV и добавляет 2 к PC так, что последний указывает на адрес 22, содержащий значение 54. Код режима адресации операнда (источника) есть 67, т. е. это адресация с индексацией, где роль индексного регистра выполняет счетчик инструкций PC. Значение смещения процессор извлекает по адресу, указанному в PC, а затем к PC добавляется 2. Таким образом, в PC оказывается адрес 24. Для вычисления адреса операнда-источника значение смещения прибавляется к содержимому указанного регистра, т. е. адрес операнда будет

$$\text{Смещение} + \text{PC} = 54 + 24 = 100$$

Так как язык ассемблера рассматривает «.» как адрес первого слова инструкции (см. гл. 3), эквивалентный оператор с индексацией будет

```
MOV  100 — . — 4(PC),R3
```

Этот вид адресации называется относительной адресацией, потому что адрес операнда вычисляется относительно текущего значения PC. Смещением называется расстояние (в байтах) между операндом и текущим PC. Если оператор и его операнд перемещаются в памяти так, что расстояние между оператором и данными остается постоянным, инструкция будет выполняться правильно в любом месте памяти.

Относительно-косвенная адресация (режим адресации 7). Формат

```
«A»  :@E
```

Этот вид адресации подобен относительной адресации, за исключением того, что выражение E используется в качестве указателя адреса операнда. Другими словами, операнд, следующий за инструкцией, прибавляется к содержимому PC. Эта сумма дает указатель адреса операнда. Например:

```
MOV  @X, R0     ; ПЕРЕСЛАТЬ В РЕГИСТР 0
                ; СОДЕРЖИМОЕ ЯЧЕЙКИ, АДРЕС КОТОРОЙ
                ; НАХОДИТСЯ В X
```

2.2.4. ТАБЛИЦА РЕЖИМОВ АДРЕСАЦИИ И ИХ КОДОВ

Шесть режимов адресации, приведенных в табл. 2.1, не увеличивают длину инструкции. Любой другой вид адресации увеличивает длину инструкции на одно слово.

Таблица 2.1

Режим адресации	Код, регистр	Применение
R	0N	Прямое обращение к регистру
@R или (R)	1N	Косвенное обращение к регистру
(R) +	2N	С автоувеличением
@(R) +	3N	С автоувеличением косвенное
—(R)	4N	С автоуменьшением
@—(R)	5N	С автоуменьшением косвенное

Примечание. N — номер регистра, отличный от 7.

Любой из режимов адресации, приведенных в табл. 2.2, увеличивает длину инструкции на одно слово.

Таблица 2.2

Режим адресации	Код, регистр	Применение
E(R)	6N	Индексация
@E(R)	7N	Косвенная индексация
# E	27	Прямое обращение к непосредственному аргументу
@# E	37	Абсолютное обращение к памяти
E	67	Относительная адресация
@E	77	Относительно-косвенная адресация

2.2.5. АДРЕСАЦИЯ В ИНСТРУКЦИЯХ ПЕРЕХОДА

Инструкции передачи управления однословные. В старшем байте содержится код операции, а в младшем байте — 7-битное смещение со знаком, которое определяет адрес перехода относительно содержимого PC. Адрес перехода аппаратно вычисляется следующим образом:

знак смещения расширяется по битам 8—15;

результат умножается на 2. Это позволяет выразить смещение в байтах, а не в словах;

результат складывается с содержимым PC для формирования окончательного адреса перехода.

Следует отметить, что когда смещение складывается с PC, то

PC указывает на слово, следующее за инструкцией перехода, т. е. фактически увеличенное на 2.

Смещение в словах рассчитывается по формуле

$$(E - PC)/2,$$

где E — выражение, содержащее адрес перехода.

Затем оно усекается до 8 бит. Поскольку PC=.+2, смещение в словах может быть определено по формуле

$$(E - . - 2)/2.$$

При помощи этих инструкций нельзя делать переход: из одной программной секции в другую; к ячейке памяти, определенной как внешний глобальный символ; к ячейке памяти, адрес которой имеет смещение, выходящее за границы интервала от -128_{10} до $+127_{10}$.

Если в программе встречается нарушение допустимых условий перехода такого типа, то соответствующая команда перехода помечается в листинге кодом ошибки A.

2.2.6. АДРЕСАЦИЯ ИНСТРУКЦИЙ ПРЕРЫВАНИЙ

Инструкции прерываний EMT и TRAP не используют младший байт слова. Это позволяет передавать дополнительную информацию в младшем байте. Если за EMT или TRAP в строке оператора в исходном тексте программы следует выражение, то его значение будет размещаться в младшем байте слова. Если значение слишком велико ($>377_8$), оно усекается до восьми разрядов с генерацией кода ошибки T (см. приложение 8).

Примечания к режимам адресации:

1. Символы +, —, @, #, (,) являются зарезервированными символами языка макроассемблера, что обеспечивает правильную трансляцию операндов с учетом их режимов адресации.

2. Все необходимые вычисления смещений (индексация, относительная и косвенно-относительная адресация, переходы) при программировании на языке макроассемблера выполняются транслятором автоматически.

3. При выполнении машинных инструкций для всех режимов адресации, за исключением режимов 2 и 3 с регистром PC, содержимое последнего автоматически увеличивается на длину инструкции (на 2, 4 или 6). Увеличение PC при режиме адресации 2 и 3 применительно к самому PC выполняется за счет автоувеличения PC.

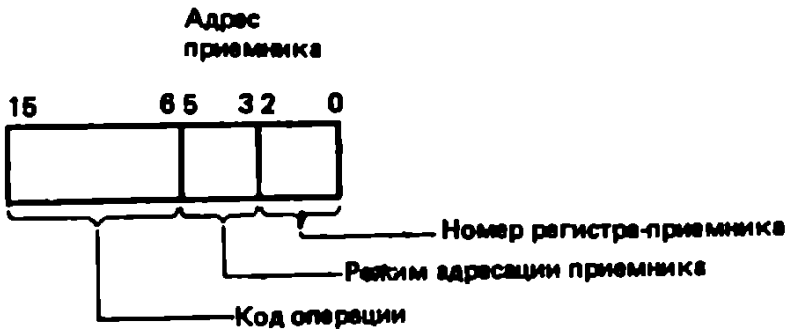
Однако и в этом случае после выполнения инструкции PC также указывает на адрес слова, следующего за последним словом инструкции. Сказанное справедливо для всех инструкций, не являющихся инструкциями переходов.

4. Использование аппаратного указателя стека SP (регистр 6) в инструкциях, работающих с байтами (для режимов адресации 2 и 4), всегда приводит к изменению его содержимого на 2.

2.3. ОДНОАДРЕСНЫЕ ИНСТРУКЦИИ

Одноадресные инструкции содержат один операнд, называемый приемником, и указание о режиме его адресации. Одноадресные инструкции могут работать как со словами, так и с байтами.

Структура одноадресных инструкций:



Примечания: 1. Если используется адресация с PC, т. е. номер регистра 7 и режим адресации 2, 3, 6 или 7, то инструкция состоит из двух слов. Во втором слове указывается непосредственно заданный операнд, адрес либо индекс.

2. Разряд 15, равный 1, означает, что инструкция работает с байтом.

В табл. 2.3 приведен полный список одноадресных инструкций и даны пояснения их работы.

Таблица 2.3

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
CLR	0050	Очистка слова	0 засылается в слово по адресу, определяемому операндом	N = 0 Z = 1 V = 0 C = 0
CLRB	1050	Очистка байта	0 засылается в байт, адрес которого определяется операндом	
COM	0051	Дополнение слова	Логическое дополнение значения, находящегося по заданному адресу, заменяет само значение	N = 0, если 15-й разряд результата равен 0 N = 1, если 15-й разряд результата равен 1 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V = 0 C = 1
COMB	1051	Дополнение байта		
INC	0052	Увеличение слова	Значение, находящееся по заданному адресу, увеличивается на 1	N = 1, если результат меньше 0 N = 0, если результат больше или равен 0 Z = 1, если результат равен 0
INCB	1052	Увеличение байта		

Продолжение

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
				V = 1, если операнд-слово содержит 077777 или операнд-байт содержит 177 C — не изменяется
DEC DECB	0053 1053	Уменьшение слова Уменьшение байта	Значение слова (байта), находящееся по заданному адресу, уменьшается на 1	N = 1, если результат меньше 0 N = 0, если результат больше или равен 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V = 1, если операнд-слово равен 100000 или операнд-байт равен 200 C — не изменяется
NEG NEGB	0054 1054	Отрицание слова Отрицание байта	Числу, находящемуся в слове (байте) по заданному адресу, присваивается знак «минус», т. е. число заменяется на свое двоичное дополнение	N = 1, если результат меньше 0 N = 0, если результат больше или равен 0 Z = 1, если результат равен 0 V = 1, если операнд-слово равен 100000 или операнд-байт равен 200 C = 0, если результат равен 0 C = 1, если результат не равен 0
TST TSTB	0057 1057	Проверка слова (байта)	Содержимое слова (байта), находящегося по заданному адресу, проверяется с вырабатыванием кодов условий в PS	N = 1, если результат меньше 0 N = 0, если результат больше или равен 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V = 0 C = 0

Продолжение

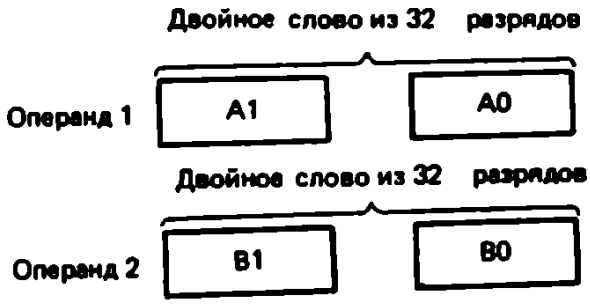
Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
ASR ASRB	0062 1062	Сдвиг вправо слова (байта)	Содержимое слова (байта), находящегося по заданному адресу, сдвигается на 1 разряд вправо. Сдвигаются все разряды, кроме знакового, т. е. разряда 15 в слове и в нечетном байте, 7 — в четном байте. Значение разряда 0 переходит в разряд C в PS. Значение знакового разряда не изменяется	N = 1, если результат меньше 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V — формируется в результате логической операции «исключающее ИЛИ» $0 \vee 0 = 0$ $0 \vee 1 = 1$ $1 \vee 0 = 1$ $1 \vee 1 = 0$ над разрядами N и C в PS C принимает значение разряда 0 операнда
ASL ASLB	0063 1063	Сдвиг влево слова (байта)	Содержимое слова (байта), находящегося по заданному адресу, сдвигается на 1 разряд влево. Знаковый разряд 15 в слове и нечетном байте, 7 — в четном байте переносятся в разряд C в PS. Разряд 0 принимает значение 0	N = 1, если результат меньше 0 N=0, если результат больше или равен 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V заполняется результатом операции «исключающее ИЛИ» над разрядами N и C C заполняется содержимым знакового разряда
ROR RORB	0060 1060	Вращение слова (байта) вправо	Все биты слова (байта) смещаются на 1 разряд вправо. Разряд 0 переходит в разряд C в PS, а содержимое разряда C переходит в разряд 15 в слове или в нечетном байте; в разряд 7 — в четном байте	N = 1, если знаковый разряд равен 1 N=0, если знаковый разряд равен 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V заполняется результатом операции «исключающее ИЛИ» над разрядами N и C в PS C заполняется содержимым разряда 0

Продолжение

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
ROL ROLB	0061 1061	Вращение слова (байта) влево	Работает аналогично ROR (RORB), за исключением того, что все разряды слова (байта) сдвигаются на 1 разряд влево. Разряд C переходит в разряд 0, а знаковый разряд — в разряд C	N, Z, V — см. ROR (RORB) C заполняется содержимым знакового разряда
SWAB	0003	Обмен	Обмен местами четного и нечетного байтов слова	N = 1, если разряд 7 четного байта равен 1 N = 0, если разряд 7 четного байта не равен 1 Z = 1, если младший байт равен 0 Z = 0, если результат не равен 0 V = 0 C = 0

Одноадресные инструкции для работы с операндами, состоящими из нескольких слов (байтов)

При выполнении операций, например сложения и вычитания, применяются специальные инструкции ADC (ADCB) и SBC (SBCB). Ниже приводится пример искусственного комбинирования операндов, состоящих из двух слов, и их сложения:



Пусть требуется сложить операнд 1 = —1 и операнд 2 = —1. Тогда

операнд 1 : 37777777777
операнд 2 : 37777777777 } — это — 1 в дополнительном коде в двух словах

Примем: A1=177777, A0=177777, B1=177777, B0=177777
Сложение производится следующим образом:

```
ADD   A0, B0    ; СКЛАДЫВАЕМ A0 и B0, В РЕЗУЛЬТАТЕ
                ; В B0 ИМЕЕМ 177776 И РАЗРЯД С В PS
                ; УСТАНОВЛЕН В 1;
ADC   B1        ; СОДЕРЖИМОЕ РАЗРЯДА С
                ; СКЛАДЫВАЕМ С B1, В B1 ПОЛУЧАЕМ
                ; 000000
ADD   A1, B1    ; СКЛАДЫВАЕМ A1 И B1, В A1 ПОЛУЧАЕМ
                ; 177777
```

Следовательно, в общей сложности в результате, состоящем из B1 и B0, имеем 37777777776, т. е. число —2.

Указанные возможности предоставляют специальные одноадресные инструкции для работы с удлиненными операндами, приведенные в табл. 2.4.

Таблица 2.4

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
ADC ADCB	0055 1055	Сложить слово (байт) с разрядом С	Содержимое разряда С в PS складывается с операндом	N = 1, если результат меньше 0 N = 0, если результат больше или равен 0 Z = 1, если результат равен 0 Z = 0, если результат не равен 0 V = 1, если значением операнда было 077777 (слово) или 177 (байт) и разряд С был 1 C = 1, если операнд был 177777 (слово) или 377 (байт) и С был 1
SBC SBCB	0056 1056	Вычесть из слова (байта) разряд С	Содержимое разряда С в PS вычитается из операнда. Применяется в случаях, аналогичных ADC (ADCB)	N = 1, если результат меньше 0 Z=1, если результат равен 0 V=1, если операнд был 100000 или 200 (байт) C=1, если операнд был 0 и С был 1

Примеры работы одноадресных инструкций с прямой адресацией

а)

Машинный код			Символический вид
005003			CLR R3
До операции			*После операции*
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	000300	R3	000000

б)

Машинный код			Символический вид
005023			CLR (R3)+
До операции			*После операции*
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	002000	R3	002002
002000	177777	002000	000000

в)

Машинный код			Символический вид
105023			CLRB (R3)+
До операции			*После операции*
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	003000	R3	003001
003000	177355	003000	177000

г)

Машинный код			Символический вид
005043			CLR —(R3)
До операции			*После операции*
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	003002	R3	003000
003000	105111	003000	000000
003002	177777	003002	177777

д)

Машинный код			Символический вид
105043			CLRB —(R3)
До операции			*После операции*
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	001776	R3	001775
001774	077301	001774	000301
001776	177777	001776	177777

е)

Машинный код			Символический вид
005063			CLR 20(R3)
000020			
До операции			*После операции*

Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	001000	R3	001000
001000	177777	001000	177777
001020	005555	001020	000000

Примеры работы одноадресных инструкций с косвенной адресацией

а)

Машинный код — **Символический вид**

005113 — COM (R3)

До операции		*После операции*	
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	003000	R3	003000
003000	013333	003000	164444

б)

Машинный код — **Символический вид**

105233 — INCB @ (R3)+

До операции		*После операции*	
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	002000	R3	002001
002000	003000	002000	003000
003000	011001	003000	011002

в)

Машинный код — **Символический вид**

005353 — DEC @ —(R3)

До операции		*После операции*	
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	002002	R3	002000
002000	003000	002000	003000
002002	155555	002002	155555
003000	007777	003000	007776

г)

Машинный код — **Символический вид**

005473
000020 — NEG @ 20(R3)

До операции		*После операции*	
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R3	002000	R3	002000
002000	177777	002000	177777
002020	003000	002020	003000
003000	000010	003000	177770

Примечание. Изменение знака числа означает замену числа на его двоичное дополнение, т.е. инверсию разрядов и добавление единицы к самому младшему разряду.

Примеры работы одноадресных инструкций с использованием адресации по PC

а)

Адрес	Машинный код	Символический вид
001000	006237	ASR @#1100
001002	001100	

До операции

Регистр/адрес	Содержимое
001000	006237
001002	001100
001100	177776
PC	001000

После операции

Регистр/адрес	Содержимое
001000	006237
001002	001100
001100	177777
PC	001004

б)

Адрес	Машинный код	Символический вид
001020	006367	ASL.B A
001022	000054	
.		.
.		.
.		.
001100		A:

До операции

Регистр/адрес	Содержимое
001020	006367
001022	000054
001100	177001
PC	001020

После операции

Регистр/адрес	Содержимое
001020	006367
001022	000054
001100	176001
PC	001024

в)

Адрес	Машинный код	Символический вид
001020	006077	ROR@A
001022	000020	
.		.
.		.
.		.
001044		A:

До операции

Регистр/адрес	Содержимое
001020	006077
001022	000020
001044	002000
002000	077776
бит С	1

После операции

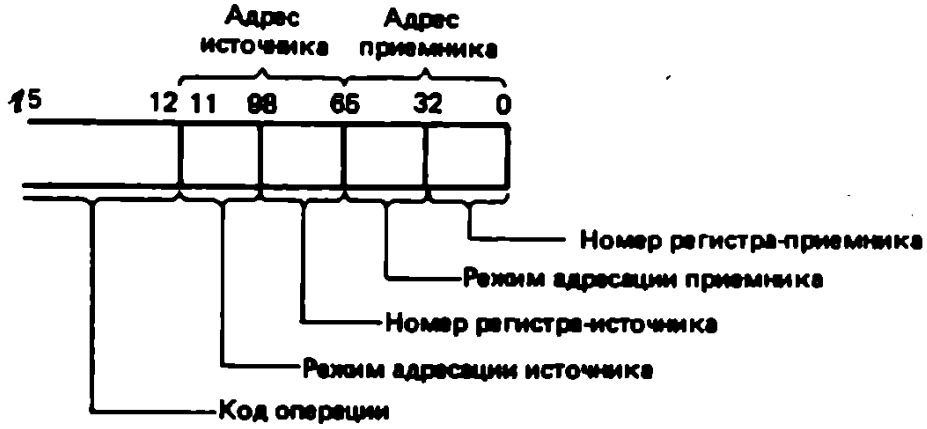
Регистр/адрес	Содержимое
001020	006077
001022	000020
001044	002000
002000	137777
бит С	0

2.4. ДВУХАДРЕСНЫЕ ИНСТРУКЦИИ

Двухадресные инструкции включают два операнда. Как уже говорилось, первый операнд называется источником, второй — приемником. При каждом из операндов устанавливается режим адресации.

В зависимости от режима адресации каждого из операндов двухадресные инструкции могут состоять из одного, двух или трех слов.

Структура первого слова инструкции:



Примечания: 1. Если используется адресация с PC или режим адресации 6 или 7 в одном из операндов, то инструкция состоит из двух слов. Во втором слове указывается непосредственно заданное число, адрес или индекс.

2. Если используется адресация с PC или режим адресации 6 или 7 в двух операндах, то инструкция состоит из трех слов. Во втором и третьем словах указывается непосредственно заданное число, адрес или индекс.

3. Значение разряда 15=1 в первом слове инструкции означает, что она выполняет операцию над байтами.

В табл. 2.5 приведен полный перечень двухадресных инструкций.

Таблица 2.5

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
MOV MOVB	01 11	Пересылка слова (байта)	Содержимое адреса, определенного 1-м операндом, пересылается по адресу, заданному 2-м операндом. Содержимое адреса, заданного 1-м операндом, не изменяется	N = 1, если источник меньше 0 N = 0, если источник больше или равен 0 Z = 1, если источник равен 0 Z = 0, если источник не равен 0 V = 0 C не изменяется
CMP CMPB	02 12	Сравнение слов (байт)	Содержимое адреса, заданного 1-м операндом, сравнивается с содержимым адреса 2-го операнда. В результате в PS вырабатываются коды условий, которые анализируются командами переходов	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 1, если имело место арифметическое переполнение (см. вычитание) C = 0, если имел место перенос из старшего разряда результата

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
			Операция сравнения выполняется путем вычитания содержимого 2-го операнда из содержимого 1-го операнда. При этом ни первый, ни второй операнды не изменяются	
BIC BICB	04 14	Очистка бит в слове (байте)	Очищает, т. е. обращает в 0 разряды 2-го операнда, которым соответствуют установленные в 1 разряды 1-го операнда	N = 1, если старший разряд результата равен 1 Z = 1, если результат равен 0 V = 0 C не изменяется
BIS BISB	05 15	Установка бит (логическое «ИЛИ») в слове (байте)	Операция логического сложения позволяет установить в 1 разряды 2-го операнда, которым соответствуют установленные в 1 разряды 1-го операнда	N = 1, если старший разряд результата равен 1 Z = 1, если результат равен 0 V = 0 C не изменяется
ADD	06	Сложение	Содержимое 1-го операнда складывается с содержимым 2-го операнда и результат пересылается по адресу 2-го операнда	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 1, если имеет место арифметическое переполнение C = 1, если имел место перенос старшего разряда результата
SUB	16	Вычитание	Содержимое адреса, указанного в 1-м операнде, вычитается из содержимого адреса, заданного 2-м операндом, и результат пересылается по адресу 2-го операнда	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 1, если имеет место арифметическое переполнение C = 0, если имел место перенос из старшего разряда результата
BIT BITB	03 13	Проверка бит в слове (байте)	Выполняет операцию логического «И» над 1-м и 2-м операндами, оставляя их неизменными, используется для проверки состояния отдельных разрядов 2-го операнда	N = 1, если старший разряд результата равен 1 Z = 1, если результат равен 0 V = 0 C не изменяется

Примеры работы двухадресных инструкций с прямой адресацией

а)

Машинный код

010103

Символический вид

MOV R1, R3

До операции

Регистр/адрес	Содержимое
R1	077777
R3	000000

После операции

Регистр/адрес	Содержимое
R1	077777
R3	077777

б)

Машинный код

112142

Символический вид

MOVB (R1)+, —(R2)

До операции

Регистр/адрес	Содержимое
R1	001001
R2	002003
001000	005177
002000	000000
002002	055177

После операции

Регистр/адрес	Содержимое
R1	001002
R2	002002
001000	005177
002000	000000
002002	055012

в)

Машинный код

026263
000010
000020

Символический вид

CMP 10(R2), 20(R3)

До операции

Регистр/адрес	Содержимое
R2	001000
R3	002000
001000	177777
001010	000005
002000	155555
002020	000005
биты NZVC	1011

После операции

Регистр/адрес	Содержимое
R2	001000
R3	002000
001000	177777
001010	000005
002000	155555
002020	000005
биты NZVC	0101

Примеры работы двухадресных инструкций с косвенной адресацией

а)

Машинный код

123152

Символический вид

CMPB@(R1)+,@—(R2)

До операции

Регистр/адрес	Содержимое
R1	001000
R2	002002
001000	003000
002000	004001
002002	005000

После операции

Регистр/адрес	Содержимое
R1	001002
R2	002000
001000	003000
002000	004001
002002	005000

003000	000177	003000	000177
004000	077400	004000	077400
биты NZVC	1011	биты NZVC	0100

б)

Машинный код	Символический вид
067172 000010 000020	ADD @ 10(R1), @ 20(R2)

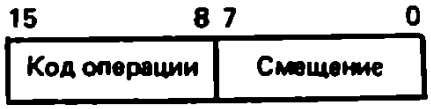
До операции		*После операции*	
Регистр/адрес	Содержимое	Регистр/адрес	Содержимое
R1	001000	R1	001000
R2	002000	R2	002000
001000	000000	001000	000000
001010	003000	001010	003000
002000	177777	002000	177777
002020	004000	002020	004000
003000	000005	003000	000005
004000	000002	004000	000007

2.5. ИНСТРУКЦИИ ПЕРЕДАЧИ УПРАВЛЕНИЯ

Передача управления обеспечивается машинными инструкциями, которые можно разделить на три группы: инструкции перехода, инструкции для работы с подпрограммами и инструкции прерываний.

2.5.1. ИНСТРУКЦИИ ПЕРЕХОДА

Данная группа инструкций позволяет осуществить передачу управления на расстояние, ограниченное возможностью представления величины смещения, задаваемого в самой инструкции. Как отмечалось ранее, каждая инструкция перехода занимает одно слово и имеет формат



Вычисление адреса перехода по смещению было приведено в 2.2.5. Ниже приводится пример вычисления адреса перехода.

Программа в машинных кодах		Программа на языке макроассемблера		
Адрес	Содержимое			
001016	005204	A2:	INC	R4
.	.	.		
.	.	.		
.	.	.		
001040	001366		BNE	A2;ИНСТРУКЦИЯ ПЕРЕХОДА
001042				НА МЕТКУ A2

Адрес перехода: 001016
Адрес инструкции перехода: 001040
Смещение $= \frac{001016 - 001040 - 2}{2} = \frac{-24}{2} = -12 = 177766$ (в дополнительном коде)

Младший байт в инструкции BNE содержит усеченное смещение —366.

Одна из инструкций перехода — BR — обеспечивает безусловную передачу управления, все остальные инструкции передают управление в соответствии с кодами условий, которые формируются в PS после выполнения каждой инструкции.

При программировании особое внимание следует обращать на формирование кодов условий V (переполнение) и C (перенос) после работы инструкций CMP (сравнение), ADD (сложение) и SUB (вычитание).

При выполнении инструкций CMP и SUB переполнение возникает (т. е. V=1), если операнды имеют разные знаки и знак результата совпадает со знаком 2-го операнда. Бит C=0, если при выполнении этих инструкций имел место перенос из старшего разряда. Например, при вычитании двух чисел 077777 и 177776 (—2) имеет место переполнение (бит V=1) и перенос из старшего разряда (бит C=0).

При выполнении инструкции ADD переполнение возникает, если операнды имеют одинаковые знаки и знак результата противоположный; C=1, если имел место перенос из старшего разряда. Например, при сложении двух чисел 077777 и 000001 возникает переполнение, а переноса нет. При сложении чисел 077777 и 177777 (— 1) имеет место перенос из старшего разряда (C=1), а переполнения не происходит.

Перечень инструкций перехода приведен в табл. 2.6.

Таблица 2.6

Мнемоника	Восьмеричный код	Название инструкции	Коды условий в PS, по которому осуществляется переход
BR	0004	Безусловный переход	
BNE	0010	Переход по неравенству нулю	$Z = 0$
BEQ	0014	Переход по равно нулю	$Z = 1$
BPL	1000	Переход по плюсу	$N = 0$
BMI	1004	Переход по минусу	$N = 1$
BVC	1020	Переход, если переполнения не было	$V = 0$
BVS	1024	Переход по переполнению	$V = 1$
BCC	1030	Переход, если переноса не было	$C = 0$
BCS	1034	Переход, если был перенос	$C = 1$
BGE	0020	Переход, если больше или равно	$N \forall V = 0$
BLT	0024	Переход, если меньше нуля	$N \forall V = 1$
BGT	0030	Переход, если больше нуля	$ZV(N \forall V) = 0$
BLE	0034	Переход, если меньше или равно	$ZV(N \forall V) = 1$
BHI	1010	Переход, если больше (без знака)	$C \vee Z = 0$

Продолжение

Мнемоника	Восьмеричный код	Название инструкции	Коды условий в PS, по которому осуществляется переход
BLOS	1014	Переход, если меньше или одинаково (без знака)	CVZ = 1
BHIS	1030	Переход, если больше или равно (без знака)	C = 0
BLO	1034	Переход, если меньше (без знака)	C = 1

Примечания: 1. Инструкции BNE, BEQ, BPL, BMI, BVC, BVS, BCC и BCS проверяют только отдельные коды условий PS.

2. Инструкции BGE, BLT, BGT и BLE проверяют комбинации кодов условий PS и позволяют делать выводы о сравнении операндов с учетом их знаков.

3. Инструкции BHI, BLOS, BHIS и BLO позволяют делать выводы о сравнении операндов как логических кодов без учета их знаков. Напомним, что старшинство следования чисел со знаками и чисел без знака (логических кодов) существенно различается.

Числа со знаком		Логические коды
max 077777		177777 max
077776		177776
.....	положительные	
000001		•
000000		•
		•
177777		000001
177776		000000 min
.....	отрицательные	
100001		
min 100000		

2.5.2. ИНСТРУКЦИИ ДЛЯ РАБОТЫ С ПОДПРОГРАММАМИ

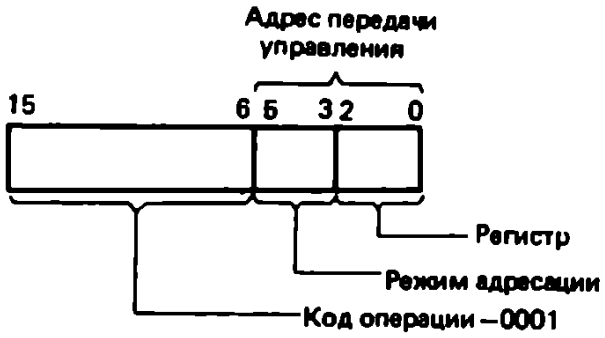
При необходимости передачи управления в любое место программы на расстояние, большее, чем то, которое обеспечивается смещением в инструкциях перехода, а также при работе с подпрограммами целесообразно использовать инструкции для работы с подпрограммами:

JMP — передача управления без запоминания адреса возврата;

JSR — передача управления подпрограмме с запоминанием адреса возврата;

RTS — выход из подпрограммы.

Инструкция JMP. Передает управление по адресу, определяемому в соответствии с заданным режимом адресации. Формат первого слова инструкции JMP:



Код адресации не может быть 0, так как в этом случае передача управления на регистр не имеет смысла.

Примеры работы инструкции JMP:

а)

Инструкция в машинном коде	Инструкция в мнемоническом виде
000121	JMP (R1)+
До операции	*После операции*
R1 = 001000	R1 = 001002
PC = 005000	PC = 001000

б)

Инструкция в машинном коде	Инструкция в мнемоническом виде
000151	JMP — (R1)
До операции	*После операции*
R1 = 003000	R1 = 002776
PC = 002000	PC = 003000

в)

Адрес	Инструкция в машинном коде	Инструкция в мнемоническом виде
001000	000167	JMP LAB
2	000020	.
4		.
.		
.		
.		
001024	005201	LAB: INC R1
До операции		*После операции*
PC = 001000		PC = 001024

Инструкция JSR. Инструкция передает управление по заданному адресу с одновременным занесением в регистр, номер которого указан в инструкции, адреса слова, следующего за инструкцией (адрес возврата). При этом содержимое регистра, в который заносится адрес возврата (регистр возврата), предварительно запоминается в аппаратном стеке. Формат инструкции:

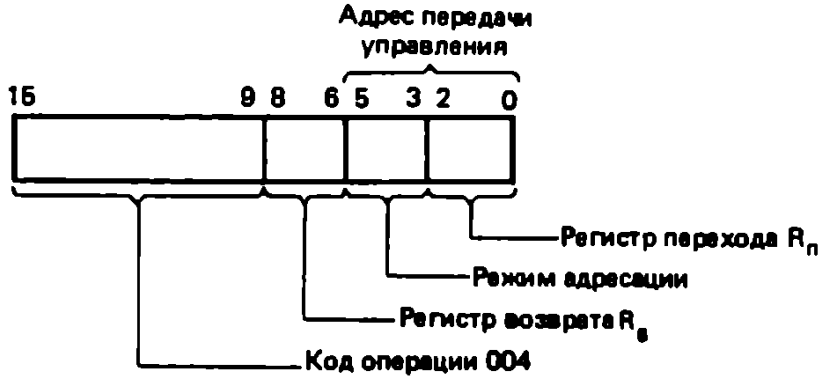


Схема выполнения инструкции:

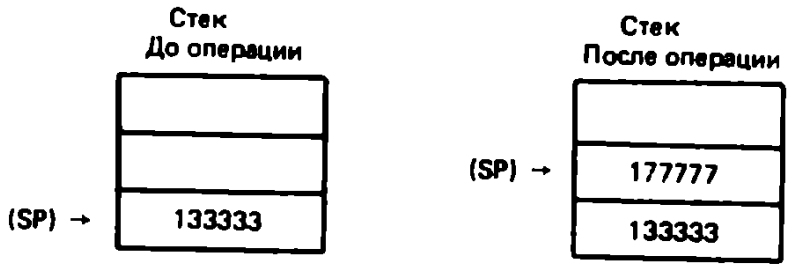
$(R_в) \rightarrow SP\downarrow$ — содержимое регистра возврата заносится в стек;

$(PC) \rightarrow R_в$ — адрес слова, следующего за инструкцией, заносится в регистр возврата;

$A_к \rightarrow PC$ — адрес инструкции, выбранный в соответствии с режимом адресации регистра перехода, заносится в PC.

Пример работы инструкции JSR:

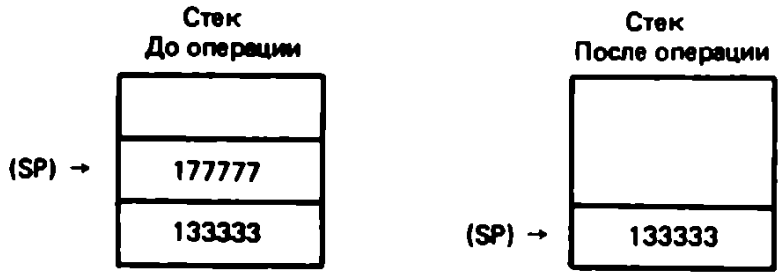
Адрес	Инструкция в машинном коде	Инструкция в мнемоническом виде
001000	004567	JSR R5, SUB
2	000020	
.		.
.		.
.		.
001024	005201	SUB: INC R1
	До операции	*После операции*
	R5=177777	PC=001024
	PC=001000	R5=001004



Инструкция RTS. Данная инструкция возвращает управление из подпрограммы на инструкцию, следующую за инструкцией обращения к подпрограмме. Адрес возврата находится в регистре, который указывается в инструкции RTS. Формат инструкции:

Схема работы инструкции RTS:
$(R_в) \rightarrow PC$ — занесение содержимого регистра возврата в PC;
$(SP) \rightarrow R_в$ — занесение элемента из стека в регистр возврата.
Пример работы инструкции RTS:

Адрес	Инструкция в машинном коде	Инструкция в мнемоническом виде
001000	004567	JSR R5, SUB
2	000020	
4	005000	CLR R0
.		.
.		.
.		
001024	005201	SUB: INC R1
...		.
001040	000205	RTS R5
	До операции	*После операции*
	R5=001024	R5=177777
	PC=001040	PC=001004



2.5.3. ИНСТРУКЦИИ ПРЕРЫВАНИЙ

Инструкции прерываний обеспечивают передачу управления при помощи программных прерываний. Каждой инструкции прерывания соответствует отдельный вектор прерывания (см. гл. 1). Передача управления при работе инструкций прерываний осуществляется по следующей схеме:

PC и PS программы, в которой работает инструкция прерывания, заносятся в стек;

из вектора прерываний соответствующей инструкции в PC и PS пересылаются адрес, по которому передается управление, и новое слово состояния процессора.

Ниже приводится перечень инструкций прерываний.

Инструкция EMT. Машинный код инструкции — число в интервале 104000—104377. Вектор прерывания EMT — ячейки 30, 32.

Инструкция TRAP. Машинный код инструкции — число в интервале 104400—104777. Вектор прерывания TRAP — ячейки 34, 36.

Как видно из приведенных форматов, структура EMT и TRAP одинакова. Данные инструкции различаются только кодами, которые задаются в разрядах 0—8. Это коды: 000—377 — для EMT,

400—777 — для TRAP[1]. Младший байт рассматриваемых инструкций обеспечивает передачу управления на требуемую ветвь программы обработки прерывания.

Инструкция EMT зарезервирована для использования в операционных системах. По этой причине применять данную инструкцию в обычных пользовательских программах не рекомендуется. При необходимости можно использовать инструкцию TRAP, которая по возможностям полностью идентична инструкции EMT.

Инструкции BPT и IOT предназначены для специальных целей и используются исключительно операционными системами. Каждая из них состоит из одного слова и содержит код операции: 000003 — для BPT, 000004 — для IOT. Векторы прерываний: ячейки 14, 16 — для BPT, 20, 22 — для IOT.

Инструкция IOT используется для вызова процедур ввода-вывода, обработки ошибок в ряде операционных систем, BPT — для трассировки программ в отладчиках.

Инструкция RTI обеспечивает возврат из программы прерывания. Этой инструкцией, как правило, оканчивается каждая программа обработки прерывания. Работа данной инструкции состоит в занесении в PC и PS из стека соответствующих величин, которые были туда ранее занесены при прерывании программы. При программировании следует заботиться о том, чтобы перед выполнением RTI указатель стека SP содержал корректный адрес.

Инструкция RTI состоит из одного слова, содержащего код операции 000002.

Следует помнить, что применение инструкции RTI не ограничивается только программами, на которые передано управление по EMT и TRAP, а обязательно для всех программ обработки прерываний.

Примечания: 1. При работе инструкций EMT, TRAP, BPT, IOT коды условий N, Z, V и C устанавливаются из вектора прерывания.

2. При работе инструкции RTI коды условий N, Z, V и C выбираются из стека.

2.6. СЛУЖЕБНЫЕ ИНСТРУКЦИИ

Служебные инструкции занимают по одному слову и состоят из кода операции. К данной группе инструкций относятся инструкции установки кодов условий и зарезервированные инструкции.

2.6.1. ИНСТРУКЦИЯ УСТАНОВКИ КОДОВ УСЛОВИЙ

Инструкции данной группы позволяют модифицировать коды условий N, Z, V и C, находящихся в PS. Перечень инструкций установки кодов условий приведен в табл. 2.7.

[1] Работа с кодами инструкций прерывания описывается в гл. 7.

Таблица 2.7

Мнемоника	Восьмеричный код	Пояснение
CLC	000241	Установка C = 0
CLV	000242	Установка V = 0
CLZ	000244	Установка Z = 0
CLN	000250	Установка N = 0
SEC	000261	Установка C = 1
SEV	000262	Установка V = 1
SEZ	000264	Установка Z = 1
SEN	000270	Установка N = 1
CCC	000257	Установка C = V = Z = N = 0
SCC	000277	Установка C = V = Z = N = 1
NOP	000240	Нет операции

2.6.2. ЗАРЕЗЕРВИРОВАННЫЕ ИНСТРУКЦИИ

Каждая инструкция этой группы состоит из одного слова, содержащего код операции. К зарезервированным инструкциям относятся:

HALT (код 000000) — инструкция останова. Вызывает останов работы процессора. Все передачи по ОШ прекращаются, PC указывает на следующую за HALT инструкцию. При этом сброса регистров устройств в начальное состояние не происходит. Работа программы может быть продолжена нажатием на пульте процестора клавиши ПРОДОЛЖЕНИЕ.

RESET (код 000005) — инструкция сброса устройств. Сброс устройств означает их приведение в исходное состояние и производится по сигналу INIT на ОШ.

WAIT (код 000001) — инструкция ожидания. Выполнение программы временно прекращается, и система переходит в состояние ожидания прерывания от внешних устройств. При поступлении прерывания адрес инструкции, следующей за WAIT, запоминается. После завершения работы программы обработки прерывания этот адрес заносится в PC, так как с него возобновляется работа программы, содержащей инструкцию WAIT.

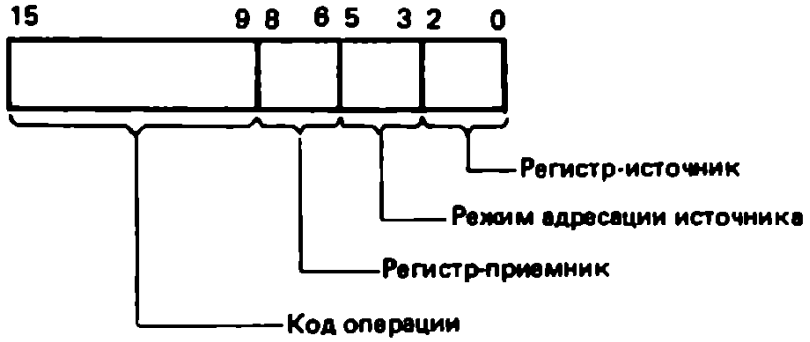
2.7. РАСШИРЕННЫЙ НАБОР ИНСТРУКЦИЙ

Расширенный набор инструкций [1] обусловлен наличием дополнительных аппаратных средств для реализации отдельных арифметических и логических операций, операций для работы с циклами и подпрограммами, которые в СМ1300 могут быть выполнены только программным путем.

[1] В СМ1300 отсутствует.

2.7.1. ДОПОЛНИТЕЛЬНЫЕ АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ИНСТРУКЦИИ

В эту группу входят следующие инструкции: MUL — умножение целых чисел, DIV — деление целых чисел, ASH — арифметический сдвиг, ASHC — арифметический сдвиг комбинированный и XOR — «исключающее ИЛИ». Перечисленные инструкции имеют следующий формат:



При выполнении операций содержимое источника, выбираемое в соответствии с режимом адресации соответствующего регистра, взаимодействует с содержимым регистра-приемника (и регистра, следующего за регистром-приемником — для операций DIV и ASHC).

Результат засылается в регистр-приемник (и следующий за ним регистр — для инструкций MUL и ASHC).

Из сказанного следует, что инструкции MUL, DIV и ASHC работают с 16-разрядными числами с фиксированной запятой, которые задаются в источнике, и с 32-разрядными числами с фиксированной запятой, которые задаются и (или) размещаются в двух смежных регистрах R и R′, первый из которых задается в инструкции как регистр-приемник.

При размещении числа в двух смежных регистрах знак числа находится в разряде 15 первого регистра. Описание дополнительных арифметических и логических инструкций приведено в табл. 2.8.

При программировании на языке макроассемблера следует учитывать, что в дополнительных арифметических инструкциях порядок следования источника и приемника такой же, как и в обычных двухадресных инструкциях, например

```
MUL S,R2
```

где S — источник;
R2 — приемник.

Таблица 2.8

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
MUL	070	Умножение целых	Содержимое регистра-приемника и содержимое источника умножаются как целые числа в дополнительном коде, результат запоминается в регистрах R и R′. Если регистр R — нечетный, то в нем запоминается младшая часть произведения	N = 1, если произведение меньше 0 Z = 1, если произведение равно 0 V = 0 C = 1, если произведение меньше -2^{15} или больше 2^{15-1}
DIV	071	Деление	32-разрядное двоичное дополнение целого числа, содержащегося в регистре-приемнике R; R′ делится на 16-разрядное содержимое источника Частное помещается в регистре R, остаток — в R′ Остаток имеет тот же знак, что и делимое Регистр R всегда четный	N = 1, если частное меньше 0 Z = 1, если частное равно 0 V = 1, если делитель (источник) равен 0 или если абсолютное значение регистра больше абсолютного значения источника. В этом случае частное занимает больше 15 бит C = 1, если имела место попытка деления на 0
ASHC	073	Комбинированный арифметический сдвиг	Двойное слово, содержащееся в регистре-приемнике R и следующем за ним регистре R′, сдвигается вправо или влево на N разрядов. N задается так же, как и для инструкции ASH. Содержимое этих регистров интерпретируется как единое 32-разрядное число. Регистр R четный. Если в операции указан нечетный регистр, то R и R′ совпадают, и в случае сдвига вправо над 16 разрядами регистра выполняется операция вращения вправо	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 1, если знак разряда во время операции изменяется C заполняется последним сдвигаемым разрядом
ASH	072	Сдвиг арифметический	Содержимое регистра-приемника R сдвигается на N позиций. Число N задается в младших пяти разрядах источника. Знак занимает 6-й разряд. Если число в N положительное, то имеет	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 1, если знак регистра изменяется в процессе сдвига C заполняется значением

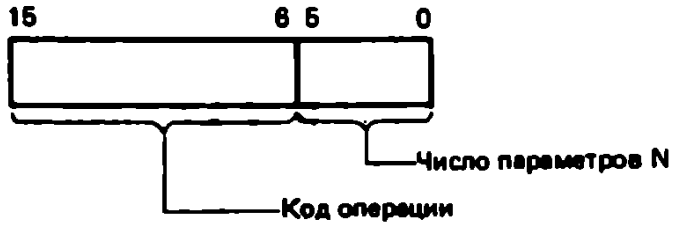
Продолжение

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
			место сдвиг влево, если отрицательное, то имеет место сдвиг вправо. При сдвиге вправо знаковый разряд не изменяется и в сдвиге не участвует	последнего сдвигаемого разряда
XOR	074	«Исключающее ИЛИ»	Над содержимым регистра-приемника и источника выполняется логическая операция «исключающее ИЛИ» по правилам: 0∀0 = 0 0∀1 = 1 1∀0 = 1 1∀1 = 0	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 0 C не изменяется

2.7.2. ИНСТРУКЦИИ ДЛЯ РАБОТЫ С ПОДПРОГРАММАМИ И ЦИКЛАМИ

Для работы с подпрограммами в моделях типа СМ-4 введена дополнительная инструкция MARK. Данная инструкция обеспечивает выход из подпрограммы с одновременным освобождением стека от параметров, которые были занесены в него при обращении к подпрограмме.

Формат инструкции:



Инструкция MARK взаимодействует вместе с инструкцией JSR и использует при своей работе регистр 5.

Схема работы инструкции:

(PC)+2 · N→SP — здесь PC указывает на адрес в стеке, где начинаются параметры, N — число параметров;

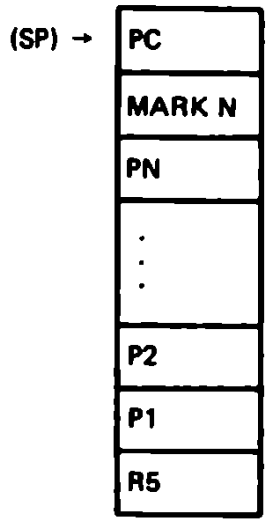
(R5)→PC — в R5 содержится адрес инструкции, следующей за JSR;

(SP)→R5 — в R5 заносится его старое значение, выбранное из стека.

Поясним работу инструкции MARK на следующем примере:

```
        MOV R5, -(SP)          ; ЗАНЕСЕНИЕ СОДЕРЖИМОГО
                               ; РЕГИСТРА 5 В СТЕК
        MOV #P1, -(SP)         ; ЗАНЕСЕНИЕ В СТЕК ПАРАМЕТРОВ
        . . .
        MOV #PN, -(SP)         ;
        MOV WMARK, -(SP)       ; ЗАНЕСЕНИЕ В СТЕК ИНСТРУКЦИИ MARK
        MOV SP, R5             ; АДРЕС ИНСТРУКЦИИ MARK В СТЕКЕ
                               ; ЗАНОСИТСЯ В РЕГИСТР 5
        JSR, PC, SUB           ; ОБРАЩЕНИЕ К ПОДПРОГРАММЕ
WMARK:  MARK N                 ; ИНСТРУКЦИЯ MARK
```

После выполнения данной последовательности инструкций содержимое стека имеет вид:



Регистр 5 содержит адрес инструкции MARK, находящейся в стеке. Выход из подпрограммы осуществляется по инструкции

RTS R5

При этом выполняются следующие действия:

Управление передается инструкции MARK, находящейся в стеке, т. е. PC будет содержать ее адрес.

В R5 из стека заносится значение PC, представляющее собой адрес инструкции, следующей за JSR (см. пример выше).

В результате выполнения инструкции MARK (см. схему):

в SP устанавливается адрес стека, где содержится старое значение регистра 5, т. е. стек освобождается от самой инструкции MARK и параметров P1, . . ., PN;

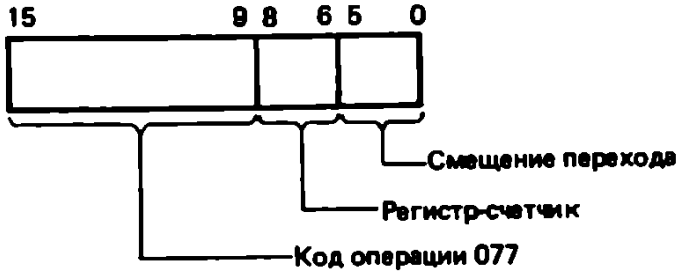
в PC заносится содержимое регистра 5, т. е. адрес инструкции, следующей за JSR;

в регистре 5 восстанавливается из стека его старое содержимое, т. е. стек полностью очищается.

Для работы с циклами в моделях типа СМ-4 введена инструкция SOB, которая обеспечивает передачу управления по задан-

ному адресу в зависимости от значения счетчика, задаваемого регистром. При передаче управления значение счетчика уменьшается на 1. Если содержимое счетчика равно 0, то передача управления не производится и выполняется инструкция, следующая за SOB.

Формат инструкции:



Смещение, задаваемое разрядами 0—5 в инструкции, всегда интерпретируется как число без знака. Это связано с тем, что инструкция SOB может передавать управление только в обратном направлении. Другими словами, если регистр-счетчик после вычитания 1 содержит отличное от 0 число, то переход осуществляется путем вычитания из PC смещения, умноженного на 2.

PC = PC — (2 * смещение)

2.8. ОПЕРАЦИИ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Аппаратное выполнение операций с плавающей запятой обеспечивается специальными устройствами, расширяющими функции процессора.

Различаются два типа таких устройств:

расширитель плавающей запятой (FIS) в СМ-4, СМ1300.01;

процессор плавающей запятой (FPP) в СМ1420, СМ1600.

Расширитель плавающей запятой — достаточно простое устройство, ориентированное на выполнение четырех арифметических операций над числами с плавающей запятой: сложения, вычитания, умножения и деления.

Процессор плавающей запятой обеспечивает не только выполнение арифметических операций над числами с плавающей запятой, но и выполнение других операций, например различного рода преобразований чисел, настройку процессора на работу с числами с двойной или одинарной точностью, операций сравнения чисел и др.

Наличие в составе комплексов СМ1600, СМ1420 процессора плавающей запятой, который функционирует параллельно с работой центрального процессора, существенно повышает эффективность комплексов при решении задач, где требуется обработка числовых данных. Повышаются эффективность компиляторов, которые ориентированы на решение вычислительных задач путем включения в их состав модулей, использующих процессор плавающей запятой, а также качество рабочих программ, получаемых

в результате работы компиляторов. В качестве примера можно привести компилятор языка Фортран-77, входящего в состав современных операционных систем СМ ЭВМ.

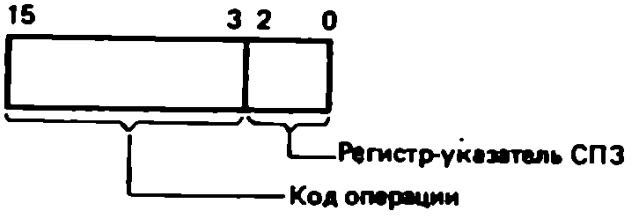
Далее приводится описание функциональных возможностей двух рассматриваемых устройств.

2.8.1. РАСШИРИТЕЛЬ ПЛАВАЮЩЕЙ ЗАПЯТОЙ (FIS)

При выполнении операций с плавающей запятой операнды задаются в стеке плавающей запятой (СПЗ) следующим образом:

P: Старшая часть операнда B
P+2: Младшая часть операнда B
P+4: Старшая часть операнда A
P+6: Младшая часть операнда A

СПЗ может размещаться в любом месте программы. При выполнении операций с плавающей запятой результат операции помещается в СПЗ на место операнда A (в ячейки P+4, P+6 СПЗ). Начальный адрес СПЗ перед выполнением операции с плавающей запятой должен быть помещен в универсальный регистр, который указывается в инструкции. Формат инструкции с плавающей запятой:



При выполнении операций сложения и вычитания порядки операндов выравниваются: производится соответствующий сдвиг мантисс, а затем происходит сложение или вычитание мантисс.

При умножении (делении) порядки складываются (вычитаются), а мантиссы перемножаются (делятся).

Если результат операции меньше, чем 2^{-128}, то на место операнда A генерируется машинный нуль.

Перечень инструкций с плавающей запятой приведен в табл. 2.9.

Таблица 2.9

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
FADD	07500	Сложение	Операнды A и B в СПЗ складываются и результат пересылается в СПЗ на место операнда A	N = 1, если результат меньше 0 Z = 1, если результат равен 0 V = 0 C = 0

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в PS
FSUB	07501	Вычитание	Операнд B вычитается из операнда A и результат пересылается в СПЗ на место A	См. FADD
FMUL	07502	Умножение	Операнды A и B перемножаются и результат засылается на место операнда A	См. FADD
FDIV	07503	Деление	Операнд A делится на операнд B и результат засылается в СПЗ на место операнда A	См. FADD

2.8.2. ПРОЦЕССОР ПЛАВАЮЩЕЙ ЗАПЯТОЙ (FPP)

FPP функционирует в двух режимах, определяющих разрядность операндов (точность вычислений):

одинарную (32 разряда или 8 значащих цифр мантиссы);

двойную (64 разряда или 17 значащих цифр мантиссы).

Отдельные инструкции FPP выполняют операции над целыми числами, которые также могут быть двух типов: одинарные, занимающие одно слово, и двойные, занимающие два слова.

Старший разряд целых чисел — знаковый.

2.8.2.1. Особенности выполнения операций

FPP рассматривает любое положительное число с нулевым смещенным порядком как нуль. При этом значение разрядов мантиссы не учитывается.

Если в результате выполнения операции с плавающей запятой смещенный порядок результата становится равным нулю при нулевом знаковом разряде, то возможны два случая:

1) прерывание по исключительной ситуации, если оно разрешено;

2) так называемый истинный нуль, если прерывания по исключительным ситуациям запрещены, все разряды мантиссы также устанавливаются в нуль.

К исключительным ситуациям, вызывающим прерывание, относятся:

недостаток, возникающий в случае, если прерывание по недостатку разрешено и порядок становится меньше, чем -177_8 (-127_{10}). В этом случае знаковый разряд и разряды порядка нулевые, но разряды мантиссы результата сохраняют свое значение, что отличает значение результата при недостатке от истинного нуля;

переполнение, возникающее в случае, если прерывание по переполнению разрешено и порядок становится больше, чем 177_8 (127_{10}), т. е. знаковый разряд числа становится равным 1, а все разряды порядка равны 0, т. е. смещенный порядок как бы становится равным 400_8, а разряды мантиссы результата сохраняются.

Значение результата, полученное при переполнении, если соответствующее прерывание разрешено, рассматривается как неопределенное и трактуется как —0. Если прерывание запрещено, то значение —0 не может быть результатом выполнения операции, так как в этом случае вырабатывается истинный нуль.

Возможность получения неопределенного значения при разрешении прерывания используется обычно для отладочных целей.

Если же значение —0 появляется в качестве одного из операндов в инструкции FPP, то операция не выполняется и возникает прерывание по —0, если оно разрешено.

Разряды, управляющие прерываниями при выполнении инструкций FPP, находятся в специальном регистре, рассматриваемом в п. 2.8.2.2.

2.8.2.2. Структура FPP

В состав FPP входят: регистры состояния и обработки исключительных ситуаций, шесть регистров-аккумуляторов, используемых при выполнении операций, арифметическое устройство.

Регистры-аккумуляторы обозначаются как AC0, AC1, ..., AC5 и содержат по 64 разряда каждый. При выполнении операций с двойной точностью используются все 64 разряда, с одинарной — старшие 32 разряда каждого регистра. Регистры AC0÷AC3 используются для обмена данными между FPP и основным процессором. Регистры AC4 и AC5 для программиста недоступны.

Выполнение инструкций с плавающей запятой осуществляется по следующей схеме:

а) основной процессор выбирает для выполнения очередную инструкцию;

б) если это инструкция с плавающей запятой, то инициируется FPP, и основной процессор выбирает для выполнения следующую инструкцию;

в) если FPP занят обработкой инструкции в тот момент, когда

поступает новая инструкция с плавающей запятой, то выполнение новой инструкции откладывается до завершения выполнения предшествующей инструкции;

Таблица 2.10

Разряды	Обозначение	Назначение
0—3		Коды условия, формирующиеся после каждой операции FPP
0	FC	Перенос старшего значащего разряда
1	FV	Переполнение
2	FZ	Результат операции — нуль
3	FN	Результат операции отрицательный
4	FMM	Используется оборудованием
5	FT	Если разряд точности установлен в 1, то результат операции усекается до допустимого числа цифр мантиссы, если — в 0, то округляется
6	FL	Формат представления целых чисел: 0 — одинарные, 1 — двойные
7	FD	Режим точности выполнения операций FPP определяет модификацию инструкций: 0 — одинарная точность, 1 — двойная точность
8	FIC	Прерывание по ошибке преобразования в целое, которая возникает, если целое, полученное в результате преобразования, не соответствует формату, установленному разрядом FL: 1 — прерывание разрешено, 0 — прерывание запрещено
9	FIV	Прерывание по переполнению: 1 — разрешено, 0 — запрещено
10	FIU	Прерывание по недостатку: 1 — разрешено, 0 — запрещено
11	FIUV	Прерывание по неопределенному значению (—0) в качестве операнда: 1 — разрешено, 0 — запрещено
12—13		Не используются
14	FID	Комплексный разряд прерываний: 1 — все прерывания запрещены, 0 — прерывания осуществляются в зависимости от установки соответствующих разрядов FPS
15	FER	Разряд ошибки, устанавливаемый при возникновении одной из ситуаций: деление на нуль, неверный код операции, а также при возникновении исключительной ситуации, в результате которой имело место прерывание

г) инструкции основного процессора выполняются параллельно с инструкциями FPP.

К основным регистрам FPP относятся:

регистр состояния (Floating Point Status—FPS);

регистр кодов исключительных ситуаций (Floating Exeption Code — FEC);

адресный регистр выхода по исключительным ситуациям (Floating Exeption Address — FEA).

Регистр FPS содержит информацию, необходимую для управления выполнением операций FPP. Структура FPS приведена в табл. 2.10.

Регистр FEC занимает четыре разряда и заполняется при прерывании одним из следующих кодов, каждый из которых соответствует определенной исключительной ситуации (причина прерывания):

2 — ошибка кода операции;
4 — деление на нуль;
6 — ошибка преобразования;
8 — переполнение;
10 — недостаток;
12 — неопределенное значение (—0).

Для прерывания от процессора плавающей запятой отводится вектор 244.

В регистр FEA заносится адрес инструкции плавающей запятой, при выполнении которой произошло прерывание.

Регистры FEC и FEA доступны только по чтению при помощи специальной инструкции с плавающей запятой.

Если прерывания в регистре FPS запрещены, но имеет место соответствующая исключительная ситуация, регистры FEA и FEC не изменяются.

2.8.2.3. Точность вычислений

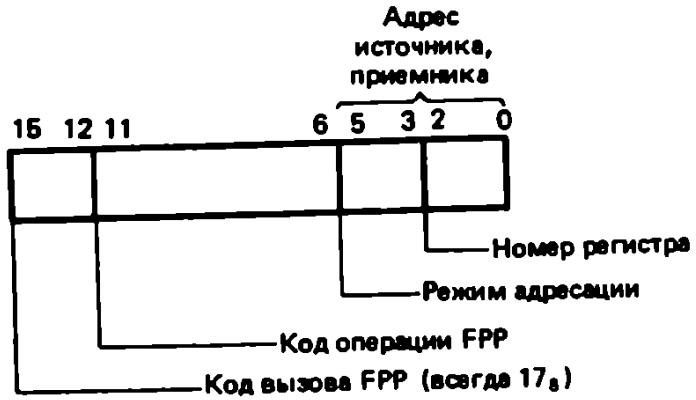
Погрешность вычислений определяется допустимой разрядностью результата в зависимости от режима работы FPP — с одинарной или двойной точностью и содержимым разряда точности FT в FPS. Результат является точным, если при усечении результата до его допустимой разрядности не теряются значащие разряды. В этом случае усеченное значение результата равно его округленному значению.

Если же при усечении результата до допустимой разрядности теряются разряды, погрешность вычисления зависит от значения первого отбрасываемого разряда, называемого разрядом округления. Результат является приближенным и его величина уменьшается при собственно усечении; уменьшается при округлении, если разряд округления равен 0; увеличивается при округлении, если разряд округления равен 1.

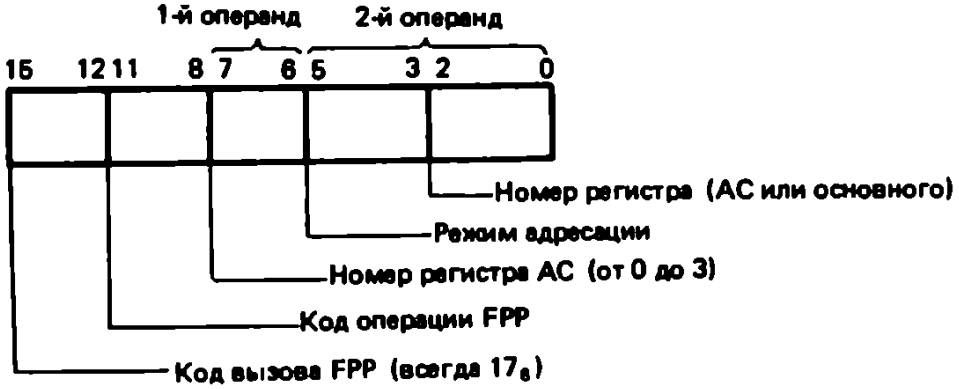
2.8.2.4. Формат инструкций

Различаются инструкции двух форматов — одноадресные и двухадресные.

Формат одноадресных инструкций:



Формат двухадресных инструкций:



При символической записи двухадресных инструкций на макроассемблере первый операнд машинной инструкции указывается в качестве второго операнда символической инструкции и наоборот.

Например, в машинной инструкции, соответствующей символической инструкции

LDF (R2)+,AC1

аккумулятор AC1 будет задан в качестве первого операнда, а регистр R2 с режимом адресации с автоувеличением — в качестве второго операнда.

Код операции основного процессора 17_8 определяет обращение к процессору плавающей запятой. Код операции процессора плавающей запятой специфицирует конкретную инструкцию.

Каждый из операндов может быть либо источником, либо приемником в зависимости от конкретной инструкции.

Второй операнд двухадресных инструкций и операнд одноадресных инструкций содержит один из восьми регистров основного процессора или один из четырех регистров процессора плавающей запятой с соответствующим режимом адресации.

2.8.2.5. Режимы адресации и регистры

Режимы адресации операндов FPP такие же, как и режимы адресации основного процессора (см. рис. 2.2). Однако для FPP характерны следующие особенности:

режим адресации 0 (регистр) указывает, что операнд находится в одном из указанных четырех регистров от AC0 до AC3, а не в регистре центрального процессора. Исключение составляют отдельные инструкции, у которых указание режима адресации 0 означает нахождение операнда в регистре основного процессора;

указание любого другого режима адресации определяет использование в операнде регистра центрального процессора;

автоувеличение, как и автоуменьшение универсальных регистров, осуществляется либо на 4 — для операндов одинарной точности, либо на 8 — для операндов двойной точности;

в режиме непосредственной адресации (режим 2 с регистром PC) в операции участвует только 16 разрядов операнда;

в двухадресных инструкциях в первом операнде задается один из регистров AC0÷AC3 без указания режима адресации. По этой причине для первого операнда двухадресных инструкций отводится два разряда.

Примеры:

а)

```
LDF  (R2)+,AC1
LDD  (R3)+,AC0
```

Данные инструкции вызывают загрузку числа с одинарной (LDF) и двойной (LDD) точностью в регистры FPP AC1 и AC0 (режим адресации 0). Само число находится в ячейках с адресами, находящимися в регистрах R2 и R3. После выполнения операции содержимое регистров R2 и R3 увеличивается на 4 и на 8 соответственно.

б)

```
LDD  -(R4),AC5
```

Данная инструкция недопустима, так как используется регистр AC5 совместно с универсальным регистром.

В отдельных инструкциях могут использоваться режимы адресации основного процессора. В этом случае указание режима адресации 0 означает обращение к регистру основного процессора, а не к регистру FPP, указание режимов автоувеличения или автоуменьшения вызывает увеличение или уменьшение соответствующего регистра основного процессора на 2.

Например, в инструкции

```
LDF  R2,AC0
```

предполагается, что нулевой режим адресации первого операнда определяет выбор универсального регистра R2, а нулевой режим адресации второго операнда — регистра FPP AC0. Выбор регистра универсального или регистра FPP определяет сама команда LDF.

В двухадресных инструкциях в машинном представлении первый и второй операнды могут быть источниками либо приемни-

ками в зависимости от типа инструкции. Однако в символьном представлении источником всегда является первый операнд и приемником — второй операнд.

2.8.2.6. Особенности выполнения инструкций

После выполнения каждой инструкции с плавающей запятой в регистре FPS формируются соответствующие коды условий (FC, FV, FZ, FN). Содержимое регистра FPS доступно программе пользователя для анализа при помощи специальной команды.

Аналогичные инструкции обеспечивают возможность изменения отдельных разрядов FPS, например, для установки необходимых масок прерываний или режима работы FPP.

После нормального выполнения инструкции с плавающей запятой основной процессор выбирает для выполнения следующую инструкцию.

Если при выполнении инструкции с плавающей запятой наступила исключительная ситуация, то возможны два случая:

прерывание по соответствующей исключительной ситуации разрешено. В этом случае управление передается соответствующей программе обработки прерывания (обычно это программа операционной системы). После выхода из программы обслуживания прерывания управление передается инструкции, которая следует за инструкцией с плавающей запятой, вызвавшей прерывание;

прерывание по соответствующей исключительной ситуации запрещено. В этом случае управление передается инструкции, следующей за инструкцией, вызвавшей исключительную ситуацию.

2.8.2.7. Одноадресные инструкции процессора плавающей запятой

Код операции одноадресных инструкций состоит из двух частей: разряды 12—15 — код операции основного процессора всегда 17_8; разряды 6—11 — код операции процессора плавающей запятой.

В табл. 2.11 приведен полный список одноадресных инструкций FPP.

Код операции каждой инструкции обозначен как восьмеричное число, образующееся в результате конкатенации разрядов 6—15.

При отсутствии указания о типе операнда подразумевается регистр-аккумулятор FPP. Использование в операнде регистра основного процессора указывается явным образом.

Последний символ F в мнемоническом обозначении инструкции означает операцию над данными с одинарной точностью, D — с двойной точностью. Соответствующий режим определяется содержимым разряда 7 в FPS.

Результат операции усекается или округляется в зависимости от содержимого разряда 5 в FPS.

Таблица 2.11

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
ABSF ABCD	1706	Вычисление абсолютного значения	Операнд инструкции является одновременно источником и приемником. Значение источника замещается его абсолютным значением. Если порядок источника равен нулю, то в результате в приемник помещается плавающий нуль	FN = 0 FZ = 1, если порядок равен 0 FZ = 0, если порядок не равен 0 FV = 0 FC = 0
CFCC	170000	Переслать коды условий	Коды условий, сформированные в FPS, пересылаются в соответствующие разряды слова состояния основного процессора. Код операции занимает все разряды — с 0 по 15	Без изменения
CLRF CZRD	1704	Установить в 0	Операнд инструкции является приемником. В поле приемника заносится нуль с одинарной или двойной точностью. Прерывание, а также недостаток или переполнение не возникают	FN = 0 FZ = 1 FV = 0 FC=0
LDFPS	1701	Загрузить FPS	Операнд инструкции является источником. Действуют режимы адресации основного процессора. Значение источника (одно слово) загружается в регистр FPS	Без изменения
NEGF NEGD	1707	Отрицание	Операнд является источником и приемником. Значение операнда замещается на противоположное. Если порядок источника равен 0, то формируется плавающий нуль. Прерывание возникает, если в FPS установлен разряд 11 и результат неопределен, т. е. равен —0. Недостаток или переполнение не возникает	FN = 1, если результат меньше 0 FN = 0, если результат больше или равен 0 FZ = 1, если результат равен 0 FZ = 0, если результат не равен 0 FV = 0 FC = 0
SETF	170001	Установка режима одинарной точности	Разряд 7 в FPS сбрасывается в 0, т. е. процессор плавающей запятой переводится в режим одинарной точности. Код	Без изменений

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
			операции занимает все 16 разрядов — с 0 по 15	
SETD	170011	Установка режима двойной точности	Разряд 7 в FPS устанавливается в 1, т. е. процессор плавающей запятой переводится в режим двойной точности. Код операции занимает все 16 разрядов — с 0 по 15	Без изменений
SETI	170002	Установка режима одинарных целых	Разряд 6 в FPS сбрасывается в 0, т. е. команды процессора плавающей запятой обрабатывают одинарные целые. Код операции занимает все 16 разрядов — с 0 по 15	Без изменений
SETL	170012	Установка режима двойных целых	Разряд 6 в FPS устанавливается в 1, т. е. команды процессора плавающей запятой обрабатывают двойные целые. Код операции занимает все 16 разрядов — с 0 по 15	Без изменений
STFPS	1702	Копирование FPS	Операнд инструкции является приемником, действует режим адресации основного процессора. В поле приемника (1 слово) копируется содержимое FPS	Без изменений
STST	1703	Копирование регистров FEC и FEA	Операнд инструкции является приемником, действуют режимы адресации основного процессора. В поле приемника (два слова) копируется содержимое регистров FEC (первое слово) и FEA (второе слово). Если используется режим адресаций типа регистр (т. е. 0) или непосредственная адресация (т. е. 2 с регистром PC), то копируется только FEC. В FEC и FEA находится информация, полученная в результате последнего исключительного события	Без изменений

Продолжение

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Код условий в FPS
TSTF[1] TSTD	1705	Проверка	Операнд является одновременно источником и приемником. Содержимое источника пересылается в приемник, т. е. источник остается без изменения. В результате операции устанавливаются коды условий в FPS. Если источник содержит —0 и разряд 11 в FPS установлен в 1, возникает прерывание. Недостаток или переполнение не возникает	FN = 1, если результат меньше 0 FN = 0, если результат больше или равен 0 FZ = 1, если порядок равен 0 FV = 0 FC = 0

[1] В качестве операнда может использоваться регистр-аккумулятор с режимом адресации 0 или регистр основного процессора с соответствующим режимом адресации.

2.8.2.8. Двухадресные инструкции процессора плавающей запятой

Код операции двухадресных инструкций состоит из двух частей: разряды 12—15 — код операции основного процессора (всегда 17_8), разряды 8—11 — код операции процессора плавающей запятой.

В табл. 2.12 приведен полный список двухадресных инструкций с плавающей запятой.

Код операции каждой инструкции обозначен как восьмеричное число, образующееся в результате конкатенации разрядов 8—15. При отсутствии указания о режиме адресации 2-го операнда подразумевается режим адресации FPP; режим адресации основного процессора указывается явным образом.

Последний символ F в мнемоническом обозначении инструкции означает операцию над данными с одинарной точностью, D — с двойной. Соответствующий режим определяется содержимым разряда 7 в FPS. Результат операции усекается или округляется в зависимости от содержимого разряда 5 в FPS.

2.9. ДИСПЕТЧЕР ПАМЯТИ[1]

При работе на СМ ЭВМ с диспетчером памяти (ДП) каждой программе независимо от того, где она размещается физически в

[1] В СМ1300 отсутствует.

Таблица 2.12

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
ADDF ADDD	364	Сложение	Первый операнд является приемником, второй операнд — источником. Содержимое приемника складывается с содержимым источника и результат записывается в регистр AC, указанный в первом операнде. Если в результате операции имеет место недостаток и разряд 10 в FPS равен 0 или если имеет место переполнение и разряд 9 в FPS равен 0, то в AC заносится 0. Сложение выполняется с двойной точностью, если разряд 7 в FPS установлен в 1, и с одинарной, если — в 0. Результат операции округляется или усекается в зависимости от содержимого разряда 5 в FPS	FN = 1, если содержимое AC меньше 0 FN = 0, если содержимое AC больше или равно 0 FZ = 1, если содержимое AC равно 0 FZ = 0, если содержимое AC не равно 0 FV = 1, если имеет место переполнение FV = 0, если переполнения нет FC = 0
CMPF CMPD	367	Сравнение	Содержимое регистра AC, указанного в первом операнде, сравнивается с содержимым второго операнда. Операнд, порядок которого равен 0, трактуется как истинный нуль	FN = 1, если разность 2-го и 1-го операндов меньше 0 FN = 0, если разность 2-го и 1-го операндов больше или равна 0 FZ = 1, если разность 2-го и 1-го операндов равна 0 FZ = 0, если разность 2-го и 1-го операндов не равна 0 FV = 0 FC = 0
DIVF DIVD	371	Деление	Содержимое регистра AC, указанного в 1-м операнде, делится на содержимое 2-го операнда и результат заносится в регистр AC, указанный в 1-м операнде. Если порядок 2-го операнда равен 0, то инструкция не выполняется (деление на 0) и имеет место прерывание, в регистр FEC заносится 4. Если порядок AC равен 0,	FN = 1, если результат меньше 0 FN = 1, если результат больше или равен 0 FZ = 1, если порядок результата равен 0 FZ = 0, если порядок результата не равен 0 FV = 1, если имеет место переполнение FV = 0, если переполнения нет FC = 0

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
			то результат деления равен 0. Точность выполнения операции и округление либо усечение результата зависит от содержимого разрядов 7 и 5 в FPS соответственно. Если имеет место недостаток или переполнение, то при разрядах 10 и 9 в FPS, установленных в 0, прерывание не возникает, и в AC заносится 0. Если же эти разряды установлены в 1, то возникает прерывание. При этом мантисса результата сохраняется, и все разряды порядка сбрасываются в 0. При переполнении знаковый разряд AC устанавливается в 1, т. е. имеет место неопределенный результат. Если 2-й операнд содержит 0 и разряд 11 в FPS установлен в 1, то прерывание возникает до выполнения операции	
LDCDF LDCFD	377	Загрузка и преобразование	Если разряд 7 в FPS установлен в 1, т. е. имеет место режим двойной точности, то содержимое 2-го операнда, представляющее собой число с одинарной точностью, преобразуется в число с двойной точностью, которое заносится в регистр AC, указанный 1-м операндом (мнемоническое обозначение инструкции LDCFD). Если разряд 7 в FPS установлен в 0, т. е. имеет место режим одинарной точности, то содержимое 2-го операнда, представляющее собой число с двойной точностью, преобразуется в число с одинарной точ-	FN = 1, если результат меньше 0 FN = 0, если результат больше или равен 0 FZ = 1, если результат равен 0 FZ = 0, если результат не равен 0 FV = 1, если имеет место переполнение FV = 0, если переполнения нет FC = 0

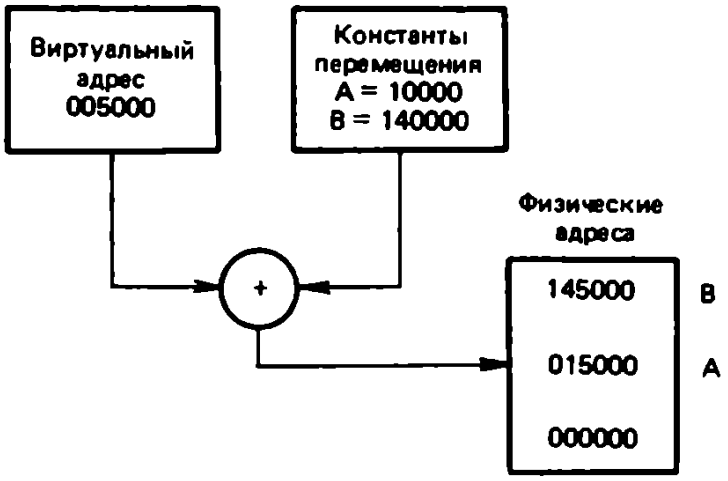
Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
			ностью, которое заносится в регистр AC, указанный первым операндом. Усечение или округление результата проводится в зависимости от содержимого разряда 5 в FPS	
LDCIF LDCID LDCLF LDCLD	376	Загрузка и преобразование целого в плавающее	Четыре модификации инструкции преобразования целого числа в число с плавающей запятой. Исходное целое число определяется в зависимости от режима адресации регистра основного процессора, задаваемого 2-м операндом. Соответствующая модификация инструкции, а следовательно, и ее мнемоническое обозначение определяется содержимым разрядов 6 и 7 в FPS: LDCIF — преобразование одинарного целого в число с плавающей запятой одинарной точности (разряд 6 = 0, разряд 7 = 0); LDCID — преобразование одинарного целого в число с плавающей запятой двойной точности (разряд 6 = 0, разряд 7 = 1); LDCLF — преобразование двойного целого в число с плавающей запятой одинарной точности (разряд 6 = 1, разряд 7 = 0); LDCLD — преобразование двойного целого в число с плавающей запятой двойной точности (разряд 6 = 1, разряд 7 = 1)	FN = 1, если результат меньше 0 FN = 0, если результат больше или равен 0 FZ = 1, если результат равен 0 FZ = 0, если результат не равен 0 FV = FC = 0
LDEXP	375	Загрузка порядка	Режим адресации 2-го операнда определяет регистр основного процессора. Число, задаваемое	FN = 1, если число в AC меньше 0 FN = 0, если число в AC больше или равно 0

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
			2-м операндом, рассматривается как порядок, который увеличивается на 200_8 и заносится в поле порядка числа с плавающей запятой, задаваемого в регистре AC в 1-м операнде. Разряды мантиссы 1-го операнда остаются без изменения. В регистр AC заносится истинный 0, если содержимое 2-го операнда больше 177_8 или меньше -177_8 и прерывание по переполнению или по недостатку в FPS (разряды 9, 10) запрещено. Если же прерывания разрешены, то содержимое разрядов 2-го операнда с 0 по 7 увеличивается на 200_8 и заносится в поле порядка 1-го операнда, после чего возникает соответствующее прерывание	FZ = 1, если порядок AC равен 0 FZ = 0, если порядок AC не равен 0 FV = 1, если 2-й операнд больше 177_8 FV = 0, если 2-й операнд меньше или равен 177_8 FC = 0
LDF LDD	365	Загрузка	Загрузка числа с плавающей запятой с одинарной (LDF) или двойной (LDD) точностью, заданного 2-м операндом, в регистр AC, который указан в 1-м операнде. Модификация инструкции LDF или LDD определяется содержимым разряда 7 в FPS	FN = 1, если результат в AC меньше 0 FN = 0, если результат в AC больше или равен 0 FZ = 1, если результат в AC равен 0 FZ = 0, если результат в AC не равен 0 FV = FC = 0
MODF MODD	363	Умножение и преобразование в целое	Умножение чисел с плавающей запятой одинарной (MODF) или двойной (MODD) точности. В результате операции выделяется целая и дробная части. Если регистр AC, заданный 1-м операндом, четный, то целая часть запоминается в регистре AC + + 1, а дробная часть — в регистре AC	FN = 1, если содержимое AC меньше 0 FN = 0, если содержимое AC больше или равно 0 FZ = 1, если содержимое AC равно 0 FZ = 0, если содержимое AC не равно 0 FV = 1, если произведение дает переполнение FV = 0, если переполнения нет FC = 0

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
MULF MULD	362	Умножение	Умножение чисел с одинарной (MULF) или двойной (MULD) точностью. Произведение заносится в регистр AC, указанный в 1-м операнде. Если возникает недостаток или переполнение и соответствующие прерывания запрещены (разряды 9, 10 в FPS), то в AC заносится истинный нуль. Произведение усекается или округляется в зависимости от содержимого разряда 5 FPS. При недостатке или переполнении, если соответствующее прерывание разрешено, мантисса произведения сохраняет полученное значение, а все разряды порядка сбрасываются в 0. Если один из операндов имеет значение —0, то прерывание возникает до выполнения операции. Результат —0 может возникнуть только вследствие недостатка или переполнения	см. MODF MODD
STCFD STCDF	374	Преобразование и запоминание	Операция, идентичная по своему характеру операции LDCFD (LDCDF) с той разницей, что преобразование выполняется над содержимым регистра AC, указанного в 1-м операнде, и преобразованный результат запоминается по адресу, указанному во 2-м операнде	см. LDCDF LDCFD
STCFI STCFL STCDI STCDL	373	Преобразование плавающего в целое и запоминание	Операции, обратные операциям LDCIF, LDCLF, LDCID, LDCLD соответственно. Результат — одинарное (если в FPS разряд 6 = 0) или двойное (если в FPS разряд 6 = 1) целое, полученное в результате пре-	N = FN = 1, если результат меньше 0 N = FN = 0, если результат больше или равен 0 Z = FZ = 1, если результат равен 0 Z = FZ = 0, если результат не равен 0 V = FV = 0

Мнемоника	Восьмеричный код	Название инструкции	Пояснение	Коды условий в FPS
			образования числа с плавающей запятой (с одинарной или двойной точностью) и находящегося в AC, запоминается по адресу 2-го операнда, в котором используется режим адресации основного процессора. Наибольшее целое число: $2^{15}-1$ — одинарное $2^{31}-1$ — двойное	C=FC=0, если результат X в AC находится в пределах: $-2^{15}-1 < X < 2^{15}+1$ $-2^{31}-1 < X < 2^{31}+1$ В противном случае C = FC = 1
STEXP	372	Запомнить порядок	Порядок числа с плавающей запятой, находящегося в регистре AC, указанном в 1-м операнде, запоминается по адресу, указанному во 2-м операнде, в котором используется режим адресации основного процессора	N = FN = 1, если результат меньше 0 N = FN = 0, если результат больше или равен 0 Z = FZ = 1, если результат равен 0 Z = FZ = 0, если результат не равен 0 V = FV = 0 C = FC = 0
STF STD	370	Запоминание	Число с плавающей запятой с одинарной (STF) или двойной (STD) точностью, заданное в регистре AC, указанном в 1-м операнде, запоминается по адресу 2-го операнда	Остаются без изменения
SUBF SUBD	366	Вычитание	Содержимое 2-го операнда вычитается из содержимого 1-го операнда и разность заносится в регистр AC, указанный в 1-м операнде. При возникновении недостатка или переполнения, если соответствующее прерывание разрешено (разряд 10 или 9 в FPS установлен в 0), в AC заносится истинный 0. Если соответствующее прерывание разрешено, то мантисса результата сохраняется, а разряды порядка сбрасываются в 0	см. ADDF ADDD

ОЗУ, отводится одно и то же поле адресов виртуальной памяти. Величина этого поля зависит от длины программы и не может превышать 32 Кслов. При работе с диспетчером памяти каждый 16-разрядный адрес программы рассматривается как виртуальный. В функции ДП входит преобразование 16-разрядных виртуальных адресов программы в соответствующие 18-разрядные физические адреса при размере ОЗУ до 128 Кслов или в 22-разрядные адреса при размере ОЗУ до 2048 Кслов. Такое преобразование проводится путем прибавления к каждому виртуальному адресу константы перемещения. Разные константы перемещения для одного и того же виртуального адреса обеспечивают его преобразование в разные физические адреса памяти. Ниже приводится пример преобразования одного и того же виртуального адреса 005000 в различные физические адреса при помощи разных констант перемещения.



Указанное преобразование виртуальных адресов программы выполняется при помощи страничной организации памяти, которая входит в функции диспетчера памяти.

При страничной организации память делится на фиксированные по размеру блоки. Размер блока — 32 слова. Несколько блоков образуют страницу. Размер страницы переменный — от 1 до 128 блоков, т. е. от 32 до 4096 слов. Адрес начала страницы всегда кратен 64.

При помощи физических констант перемещения виртуальные страницы программы, следующие друг за другом, могут отображаться в физические страницы, находящиеся в различных участках физической памяти. На рис. 2.1 приведен пример размещения в памяти емкостью свыше 128 Кслов двух программ длиной 32 Кслова с использованием страничной организации.

Несложно заметить, что программы, имеющие одинаковые виртуальные адреса, имеют разные физические адреса памяти.

Информация о страницах и константах перемещения содержится в регистрах ДП, рассматриваемых далее.

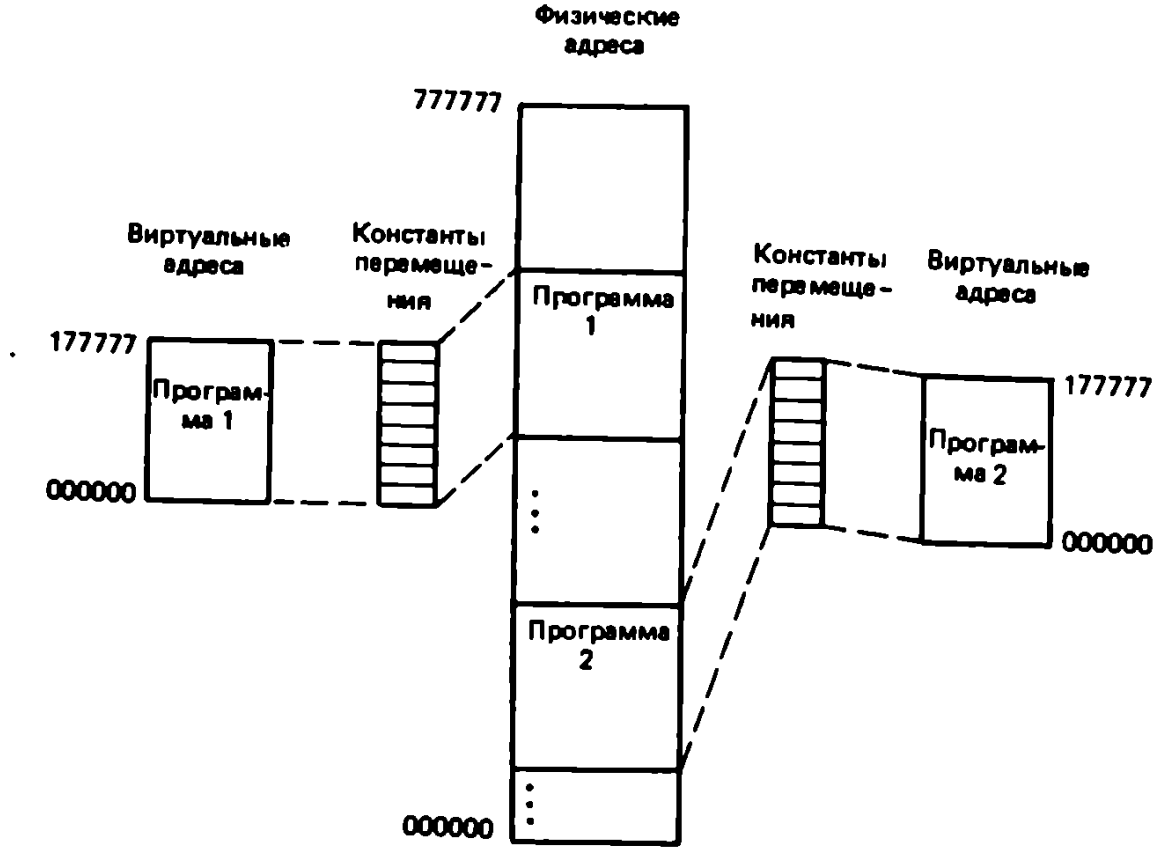


Рис. 2.1. Размещение в памяти двух программ

2.9.1. ИДЕНТИФИКАЦИЯ СТРАНИЦ

При наличии ДП процессор может работать в одном из двух режимов (см. структуру PS в главе 1): системном (привилегированном), т. е. в состоянии обработки системной программы, и пользовательском (непривилегированном). Режим работы определяется в разрядах 14—15 и 12—13 PS : 00 — системный режим, 11 — пользовательский режим.

Каждому режиму соответствует набор из 8 специальных регистров, являющихся частью ДП, которые называются регистрами активных страниц (РАС). Схематическая связь РАС с режимами работы процессора показана на рис. 2.2. Каждый РАС содержит текущую информацию об одной странице, которая необходима ДП для соответствующей настройки виртуальных адресов на физические. Страницы, информация о которых содержится в РАС, называются активными. Одновременно могут существовать 8 активных страниц для пользовательского режима и 8 активных страниц для системного режима. Как правило, один из наборов

активных страниц связан с работой одной программы. Путем изменения содержимого PAC изменяется отображение части виртуального пространства на физическую память, таким образом, могут быть использованы другие участки физической памяти.

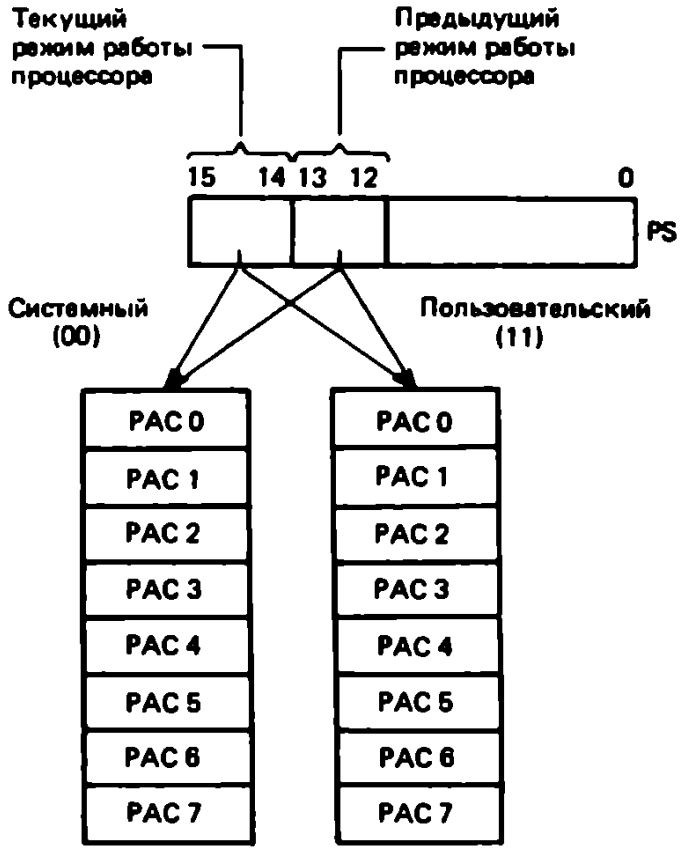
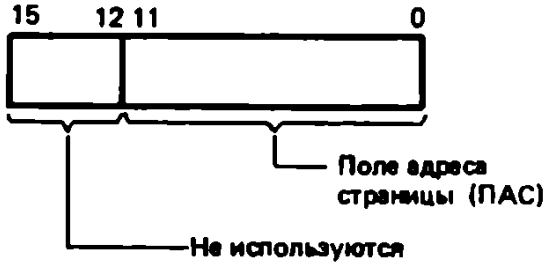


Рис. 2.2. Связь между режимами процессора и PAC

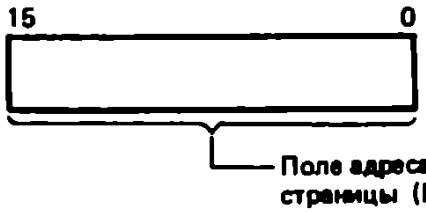
Установка начальных значений PAC и их изменение в процессе работы выполняются программным путем. Обычно это входит в функции операционных систем, использующих ДП.

Каждый PAC состоит из двух 16-разрядных регистров: регистра адреса страницы (PA) и регистра описания страницы (PO). Два варианта структуры PA представлены ниже.

1-й вариант — для комплексов с ОЗУ до 128 Кслов.



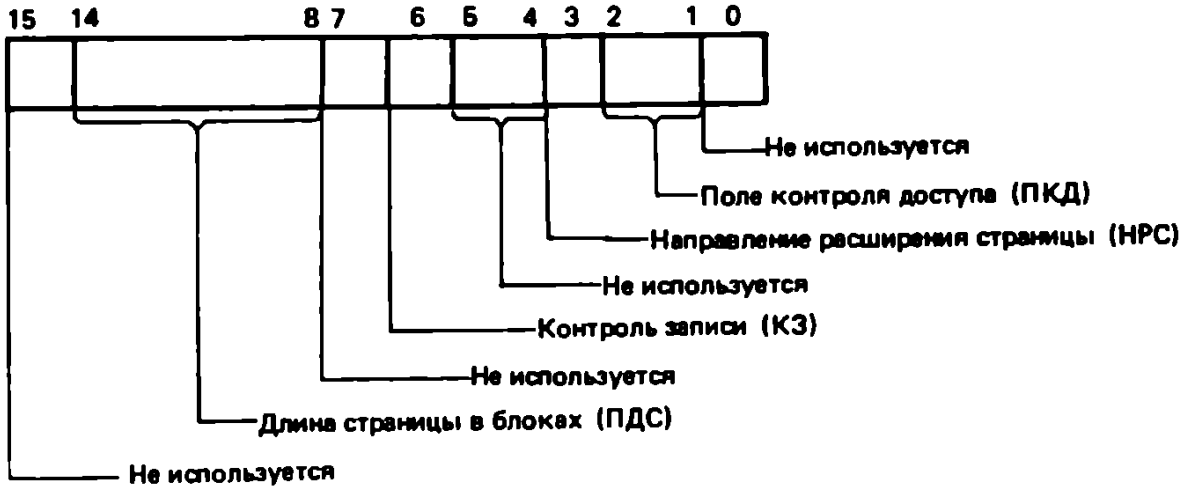
2-й вариант — для комплексов с ОЗУ свыше 128 Кслов.



Здесь ПАС содержит номер блока физической памяти, с которого начинается страница (константа перемещения).

Регистр описания страницы включает информацию, необходимую диспетчеру памяти для организации работы со страницами: их расширения, контроля доступа к страницам и т. п.

Структура РО:



Здесь ПКД — поле контроля доступа, содержит ключ доступа к странице: 00; 10 — обращения к странице запрещены, 01 — страница доступна только для чтения, 11 — страница доступна для чтения и для записи;

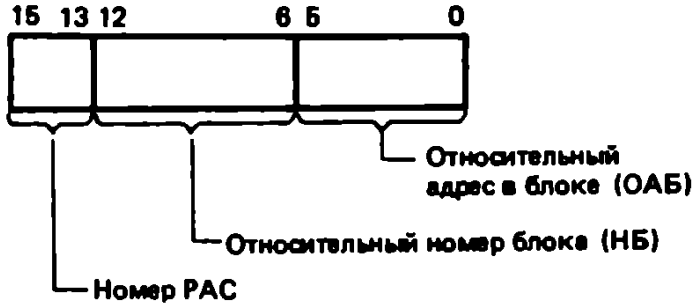
НРС — направление расширения страницы. Значение 0 указывает возможность дополнения страницы новыми блоками для увеличения адресов памяти, значение 1 — для уменьшения адресов памяти;

КЗ — значение 1 в данном разряде указывает, что в данную страницу во время работы программы проводилась запись;

ПДС — размер страницы в блоках по 32 слова.

2.9.2. ФОРМИРОВАНИЕ ФИЗИЧЕСКОГО АДРЕСА

Виртуальный 16-разрядный адрес имеет следующую структуру:

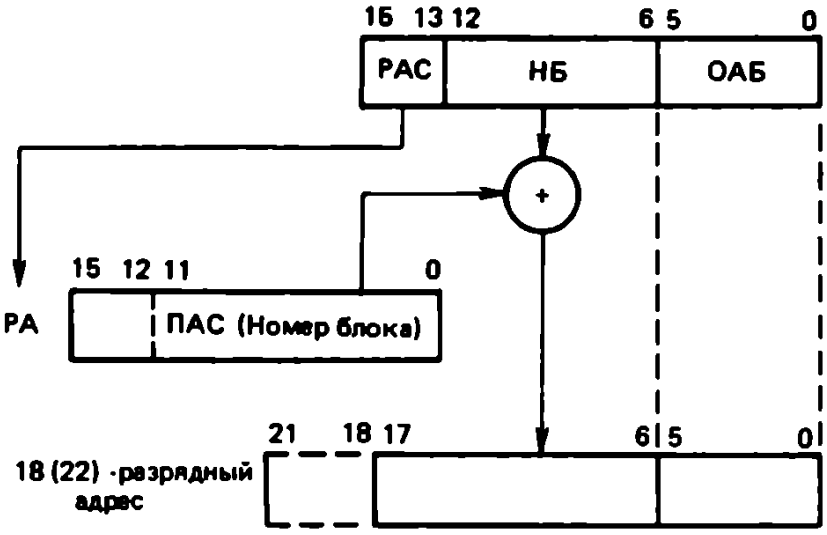


Здесь РАС — номер регистра активных страниц в интернале от 0 до 7. Данный регистр выбирается из набора, соответствующего режиму работы процессора;

НБ — относительный номер блока внутри страницы;

ОАБ — относительный адрес слова (байта) внутри блока.

Вычисление 18-разрядного или 22-разрядного физического адреса выполняется по следующей схеме:



Из набора РАС, соответствующего режиму работы процессора (системный или пользовательский), выбирается регистр адреса страницы, номер которого задан в разрядах 13—15 виртуального 16-разрядного адреса.

Разряды 0—11 или 0—15 РА из выбранного РАС суммируются с разрядами 6—12 виртуального адреса и их значения заносятся в разряды 6—17 (6—21) физического адреса.

К разрядам 6—17 (6—21) в качестве младших разрядов добавляются страницы 0—5 виртуального адреса, образуя таким образом полный 18 (22)-разрядный физический адрес.

Заметим, что формирование 22-разрядного физического адреса производится по той же схеме, что и формирование 18-разрядного адреса, с той разницей, что в РА для хранения ПАС задействованы все 16 разрядов.

2.9.3. ОТОБРАЖЕНИЕ АДРЕСОВ ШИНЫ

Как уже указывалось, адреса ОШ—18-разрядные. Для хранения физического адреса в моделях с ОЗУ емкостью до 128 Кслов также используются 18 разрядов. По этой причине нет необходимости в каких-либо специальных преобразованиях адресов при обращении к регистрам внешних устройств при организации ввода-вывода (как с использованием процессора, так и на уровне внепроцессорного обмена).

Однако ситуация меняется при организации ввода-вывода на уровне внепроцессорного обмена с ОЗУ емкостью свыше 128 Кслов на комплексах СМ1420, СМ1600. Поскольку такой обмен осуществляется без участия центрального процессора, то для преобразования 18-разрядных адресов ОШ в 22-разрядные физические адреса, что необходимо при передаче данных по прямому доступу, используется специальный преобразователь адресов ОШ, являющийся также составной частью диспетчера памяти, но расположенный вне центрального процессора.

В комплексах с ОЗУ емкостью свыше 128 Кслов старшие 128 Кслов памяти не используются. Из них 124 Кслова (с адресами от 17000000 до 17757777) зарезервированы для преобразования адресов ОШ, а верхние 4 Кслова (с адресами от 17760000 до 17777777), как и в комплексах с обычной памятью, отведены для регистров внешних устройств. Для более четкого понимания сказанного следует обратиться к рис. 1.5 (в).

Преобразователь адресов ОШ использует специальные регистры, содержащие информацию, необходимую для преобразования 18-разрядных адресов ОШ в 22-разрядные адреса памяти.

Структура этих регистров и сам механизм преобразования в данной книге не рассматриваются.

2.9.4. ПРЕРЫВАНИЯ ПРИ РАБОТЕ С ДП

Если при работе с ДП произведены какие-либо ошибочные действия (например, обращение к неактивной странице, попытка записи в страницу, доступную только для чтения, и т. п.), происходит прерывание работы программы по вектору с адресом 250. Соответствующая программа обработки прерывания может провести анализ причины прерывания, проверив содержимое двух специальных регистров состояния ДП — SR0 и SR2.

Регистр SR0 (адрес на ОШ 777572) имеет следующие значения разрядов:

Номер разряда	Значение
0	Признак работы ДП: 0 — ДП не работает; 1 — ДП работает
1—3	Номер РАС, являющегося причиной прерывания
5—6	Режим работы процессора: 00 — системный; 11 — пользовательский
7	Не используется
8	Диагностический разряд
11—12	Не используются
13	Прерывание по записи в страницу, доступную для чтения
14	Прерывание по нарушению длины страницы
15	Прерывание по обращению к неактивной странице

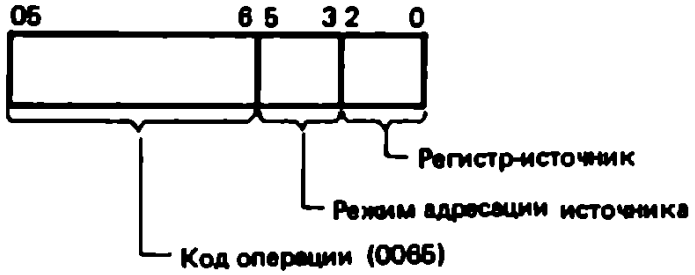
Регистр SR2 (адрес на ОШ — 777576) содержит виртуальный адрес, во время обработки которого произошло прерывание по вектору 250.

Как правило, обработка прерываний от ДП в соответствующий адрес регистров состояния ОП производится программами операционных систем.

2.9.5. ДОПОЛНИТЕЛЬНЫЕ ИНСТРУКЦИИ ПРИ РАБОТЕ С ДП

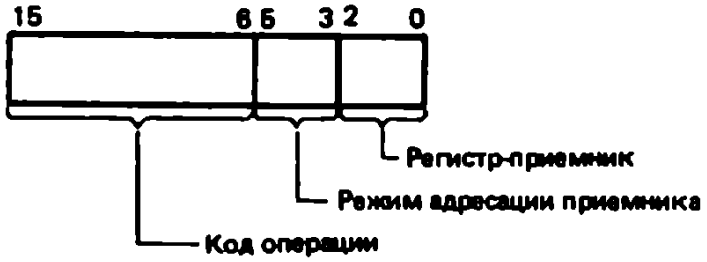
С работой ДП связана работа двух специальных дополнительных инструкций, выполняющих переход от одного режима работы процессора к другому.

Инструкция MFPI. Формат инструкции:



Данная инструкция пересылает в стек программы, работающей в соответствии с текущим режимом процессора (разряды 14—15 PS), содержимое адреса, вычисленного для предшествующего режима (разряды 12—13 PS).

Инструкция MTPI. Формат инструкции:



Данная инструкция извлекает из стека программы, работающей в текущем режиме (в соответствии с разрядами 14—15 PS), данные и пересылает их по адресу, вычисляемому для предшествующего режима (разряды 12—13 PS).

Особенность работы этих двух инструкций состоит в том, что виртуальный адрес, задаваемый для MFPI в соответствии с режимом адресации регистра-источника и для MTPI — регистра-приемника, преобразуется в физический с использованием PAC, определяемых в соответствии с режимом работы процессора, который имел место до последнего прерывания.

3

ГЛАВА

ВВЕДЕНИЕ В МАКРОАССЕМБЛЕР

Обычно слово «ассемблер» включает два понятия: собственно машинно-ориентированный язык программирования, наиболее полно учитывающий специфику ЭВМ, и транслятор с этого языка, переводящий программу в машинные коды. В СМ ЭВМ к языку уровня ассемблера применяется название макроассемблер, или, для краткости, Макро, которое используется в изложении.

Важной особенностью данного языка является то, что каждый оператор программы, составленной на языке Макро, как правило, переводится в одну машинную инструкцию. Исключение составляют макрокоманды, которые в зависимости от соответствующих им макроопределений могут переводиться в несколько машинных инструкций.

Язык Макро используется для программирования в тех случаях, когда к программам предъявляются жесткие требования по экономии оперативной памяти, быстродействию, а также когда возникает необходимость использования машинных ресурсов, недоступных языкам программирования высокого уровня.

В данной главе описываются основные конструкции языка Макро и кратко — операционная обстановка, в которой работает программист при использовании этого языка. В последующих двух главах — 4 и 5 — описываются более сложные элементы языка, такие, как директивы управления трансляцией и директивы макрокоманд. Язык Макро позволяет использовать стандартные способы написания и оформления программ и их документирования [1].

3.1. ОПЕРАТИВНАЯ ОБСТАНОВКА

Под оперативной обстановкой понимается среда операционной системы (ОС), в которой происходят подготовка и выполнение программ. В силу разнообразия операционных систем СМ ЭВМ по

[1] Рекомендации по оформлению, общие для всего программного обеспечения СМ ЭВМ, приведены в гл. 7.

своим характеристикам и средствам, предоставляемым для подготовки, отладки и работы программ, можно говорить лишь о наиболее общих чертах этого процесса и этой среды.

В целом процесс подготовки программы может быть представлен следующими шагами:

подготовка исходного текста,

трансляция с получением объектного файла,

создание загружаемого файла.

При этом, если на каком-то шаге подготовки программист (он же чаще всего и оператор) обнаруживает ошибки, операционная система дает возможность вернуться к одному из предыдущих этапов для их исправления (рис. 3.1).

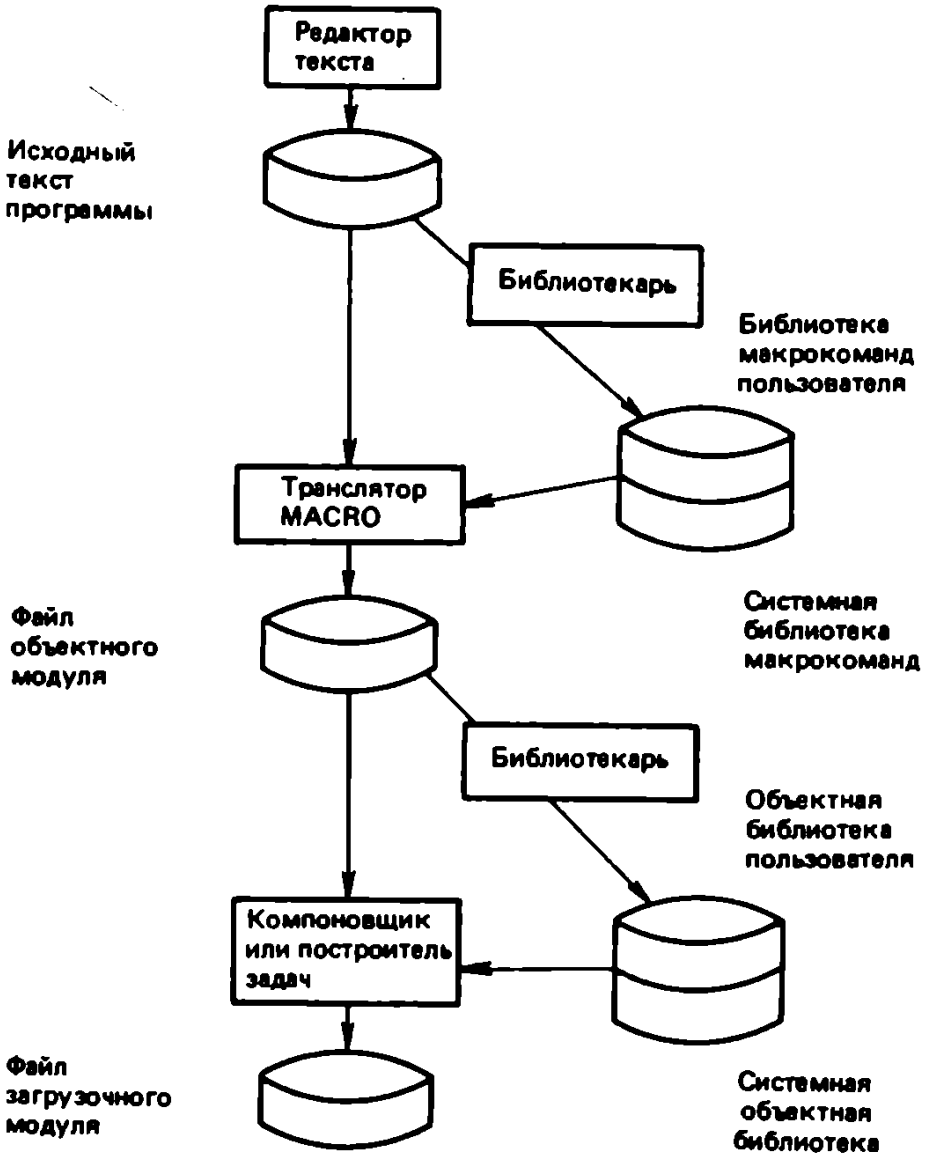


Рис. 3.1. Процесс подготовки программы в ОС

При подготовке исходного текста программы обычно используется один из редакторов текстовой информации, предоставляемых операционной системой. Этот же редактор используется и для

внесения изменений в текст программы. При всем разнообразии редакторов их базовый набор операций идентичен: перемещение «точки редактирования» по тексту, вставка (удаление) строк или знаков в строке.

Готовый текст программы преобразуется транслятором (ассемблером) в промежуточный двоичный файл — объектный файл. В этом виде в программе остается незавершенной ее настройка на адреса той области оперативной памяти, в которой она будет выполняться. Эта информация хранится в объектом коде так же, как и информация о межмодульных связях, которая обеспечивает возможность раздельной трансляции частей программы, хранящихся в отдельных текстовых файлах.

Загружаемый файл, т. е. готовый к выполнению загрузочный модуль программы, создается специальной системной программой (компоновщик LINK, построитель задач ТКВ — в зависимости от операционной системы) из отдельно оттранслированных объектных модулей. На этом же этапе в компоновку загрузочного модуля могут включаться объектные модули из библиотек. Библиотеки объектных модулей создаются предварительно программой-библиотекарем из отдельных объектных модулей и содержат наиболее часто используемые процедуры и функции. Поиск и выборка необходимого модуля из библиотеки осуществляется компоновщиком по данным о межмодульных связях (глобальным ссылкам). При построении загрузочного файла, кроме объединения объектных модулей, производится также и настройка двоичного кода на конкретные адреса той области оперативной памяти, в которой будет размещаться и выполняться программа.

Если загрузочный файл создается для отладки программы, то на стадии компоновки в него может быть включен отладчик ODT — системный модуль, обеспечивающий в диалоге с оператором возможность приостанавливать выполнение программы в любой заданной точке, проверять и (или) модифицировать содержимое любого регистра или ячейки оперативной памяти в области программы, выполнять программу в пошаговом режиме (по одной инструкции) и т. д. При работе с отладчиком программист обычно использует для указания интересующих его адресов информацию, состоящую из двух распечаток: листинга программы, создаваемого транслятором и содержащего относительные адреса инструкций и ячеек данных, и карты распределения памяти, создаваемой компоновщиком и содержащей абсолютные адреса глобальных ссылок.

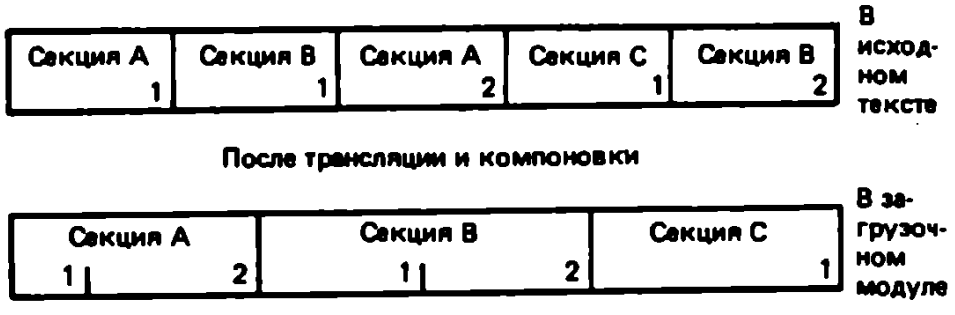
Если используется ОС, то загрузка и запуск задачи производятся при помощи средств, предоставляемых системой, причем на ОС возлагаются функции выделения памяти под задачу, предоставления ей в соответствии с той или иной дисциплиной других ресурсов ЭВМ (процессор, внешние устройства). Программа при этом обязана следовать определенным правилам общения с системой: операции на внешних устройствах, работа с файлами, общение с оператором и т. д. выполняются при помощи запросов к

операционной системе. Обычно для этих целей используется инструкция EMT, вызывающая программное прерывание, по которому управление передается в ядро операционной системы. Так как младший байт кода инструкции процессором не обрабатывается, он обычно используется для указания требуемой функции ОС. Каждая ОС имеет свой набор функций и кодов и свои правила передачи дополнительных параметров. Для упрощения программирования операционная система включает в себя системную библиотеку макрокоманд, использование которых позволяет не описывать детально весь процесс общения с ОС. Состав этой библиотеки, имена макрокоманд и их параметры зависят от используемой операционной системы.

3.2. РАСПРЕДЕЛЕНИЕ ПАМЯТИ В ПРОГРАММЕ

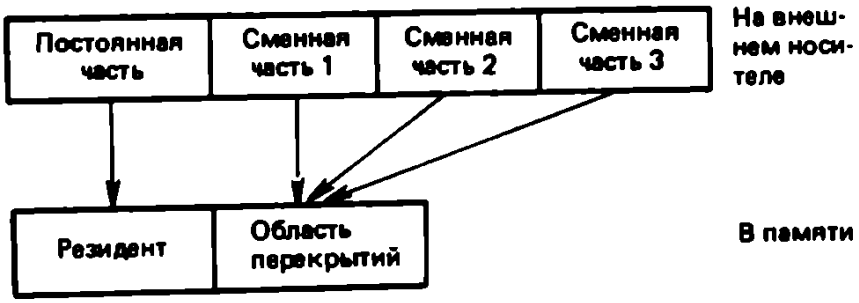
В отличие от языков программирования высокого уровня макроассемблер задачу распределения памяти в программе целиком возлагает на программиста. Информация о заданном программном распределении памяти помещается транслятором в объектный модуль и используется компоновщиком при построении загрузочного файла программы. При компоновке программы компоновщик производит объединение указанных ему объектных модулей и при необходимости модулей из объектных библиотек в единое целое, выделяя память под данные и инструкции в соответствии с описаниями, сделанными программистом.

Основная роль в этих описаниях принадлежит программной секции — непрерывной области памяти, для которой программист может задать имя и характеристики, определяющие правила, по которым компоновщик будет с ней работать. При написании программы инструкции и данные объявляются принадлежащими к одной из программных секций. В тексте программы информация, принадлежащая разным программным секциям, может следовать в любом порядке, но после трансляции и компоновки она будет собрана воедино по одноименным областям:



При этом порядок секций определяется порядком их первых объявлений в исходном тексте. При компоновке нескольких объектных модулей этот порядок накладывается на порядок указания модулей компоновщику.

При создании загрузочного файла компоновщик, объединяя модули, следит за тем, чтобы размер секций и всей задачи не превысил полного адресного пространства 64 Кбайт. Если такое переполнение происходит, это означает, что задача не может быть целиком размещена в ОЗУ. Решить эту проблему можно с помощью структуры перекрытий, при которой выделяются постоянно резидентная часть задачи и ее сменные части, размещаемые на внешнем запоминающем устройстве и считывающиеся в память по мере необходимости. Обычно при таком считывании новая сменная часть размещается в памяти на те же адреса, что и предыдущая, т. е. замещает ее.



Такие сменные части называют перекрытиями. Способы построения структуры перекрытий зависят от операционной системы, но во всех случаях при этом используются отдельно транслировавшиеся объектные модули.

Рассмотрим более детально возможности программных секций Макроассемблер предоставляет программисту директиву .PSECT, которая объявляет начало (или продолжение) программной секции, ее имя и характеристики. Для каждой программной секции транслятор ведет отдельный счетчик ячеек, значения которого присваиваются меткам инструкций и полей данных, размещаемых в этой секции. Счет при этом ведется от относительного нуля — начала секции. При компоновке программы все ячейки, содержащие ссылки на такое относительное значение, корректируются компоновщиком.

Программа может содержать абсолютную программную секцию, неименованную перемещаемую программную секцию и до 254 именованных абсолютных или перемещаемых программных секций. Абсолютная программная секция обычно используется для доступа к фиксированным ячейкам памяти, таким, как векторы прерываний и регистры внешних устройств, или для определения констант. Перемещаемая программная секция не связана с фиксированными ячейками, ее положение (базовый адрес) определяется компоновщиком при построении загрузочного модуля. Управляя размещением секций при компоновке, можно разместить в разных областях оперативной памяти секции, содержащие, например, только инструкции процессора (чистый код), и секции, содержащие только данные. В системах коллективного пользования это дает возможность нескольким пользователям выполнять

один и тот же программный код (т. е. одновременно использовать одну и ту же копию программы) на собственных данных.

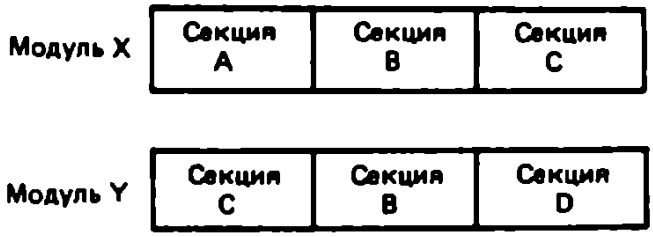
Директива .PSECT позволяет указать следующие характеристики программной секции.

Доступ. Секция может быть доступна только на чтение или на чтение-запись. В настоящее время эта характеристика не учитывается операционными системами.

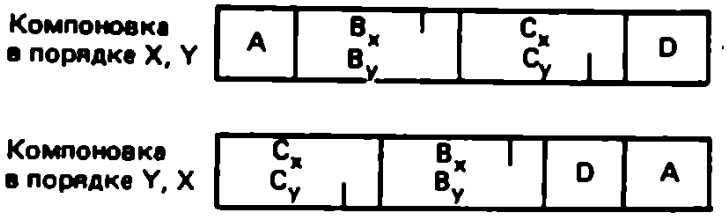
Содержимое (инструкции или данные). Эта характеристика позволяет при компоновке задачи различать обращения к точкам входа в процедуры от обращения к данным.

Перемещаемость. Позволяет различать перемещаемые и абсолютные секции.

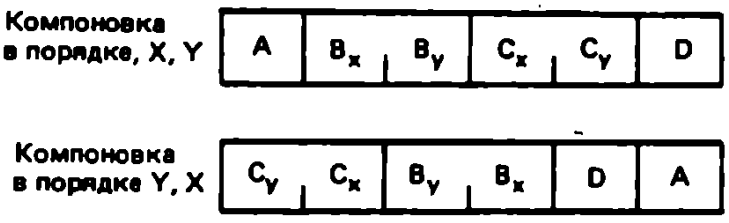
Размещение. Программная секция может быть конкатенируемой или перекрывающейся. В первом случае одноименные секции из разных объектных модулей при компоновке соединяются последовательно, и размер требуемой области памяти равен сумме размеров секции в модулях. Во втором случае одноименные секции из разных модулей размещаются с одного базового адреса (т. е. образуют общее поле памяти) и размер требуемой области равен максимуму из размеров секции в модулях:



Все секции конкатенируемые



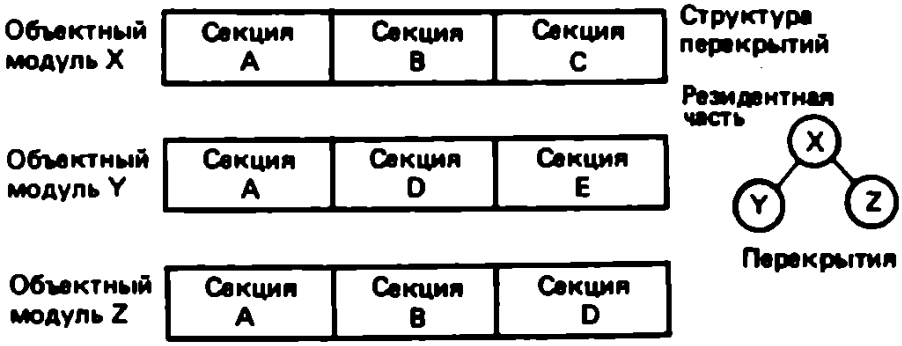
Все секции перекрывающиеся



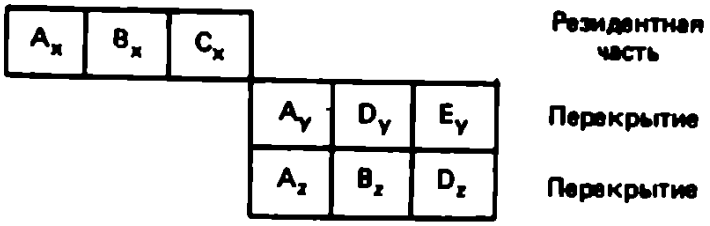
На приведенном рисунке показано размещение конкатенирующихся и перекрывающихся секций при компоновке в зависимости от порядка указания компоновщику объединяемых объектных модулей.

Локальность. Характеристика учитывается компоновщиком только при построении задачи, имеющей структуру перекрытий. Для локальной программной секции память выделяется в том перекрытии, в котором заявлен содержащий ее объектный модуль, т. е. если несколько объектных модулей, входящих в разные перекрытия, содержат одноименные локальные секции, память под эти секции выделяется в каждом перекрытии отдельно. Для глобальной программной секции память выделяется в резидентной части программы либо в перекрытии, находящемся в памяти одновременно с перекрытиями, в объектных модулях которых содержится эта секция.

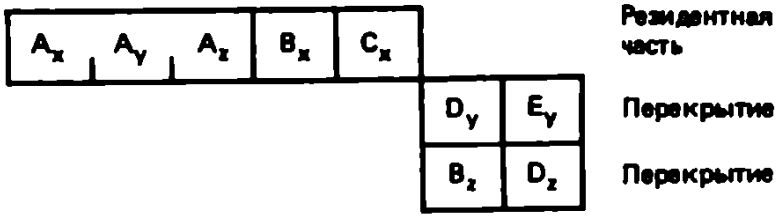
На рисунках показано поведение программных секций при построении программы с перекрытиями в зависимости от присутствия или отсутствия свойства локальности.



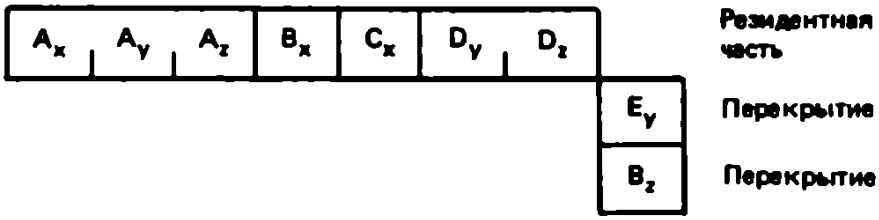
Загрузочный модуль
Все секции локальные и конкатенирующиеся



Секция A объявлена глобальной



Секции A, D объявлены глобальными



3.3. ФОРМАТ ОПЕРАТОРА

Программа, написанная на языке Макро, состоит из последовательности символьных строк. Каждая строка содержит один оператор языка и имеет длину не более 132 знаков. Однако, учитывая формат строки листинга и размер строки терминала, рекомендуется ограничивать длину строки до 80 знаков.

Оператор языка может содержать до четырех полей. Назначение этих полей определяется порядком их появления в строке и (или) разграничительными знаками между ними. Оператор макроассемблера имеет формат

МЕТКА: ОПЕРАЦИЯ ОПЕРАНДЫ ;КОММЕНТАРИЙ

Поле метки и поле комментария не обязательны.

Поле операции и поле операнда являются взаимозависимыми, т. е. если эти поля присутствуют в исходном операторе, то семантика полей взаимосвязана. Оператор может содержать поле операции и не содержать поля операнда, но не наоборот. Оператор, содержащий операнд и не содержащий поля операции, интерпретируется макроассемблером как неявная директива .WORD.

Макроассемблер интерпретирует и обрабатывает операторы исходной программы последовательно один за другим, генерируя одну или несколько двоичных инструкций процессора, слов данных или выполняя указанные действия по трансляции и (или) созданию листинга.

Оператор языка макроассемблера должен занимать не более одной строки, т. е. не может иметь строк продолжения.

Для большей наглядности текста программы в листинге трансляции рекомендуется использовать следующее форматирование исходной строки:

МЕТКА	— начинается с колонки 1
ОПЕРАЦИЯ	— начинается с колонки 9
ОПЕРАНДЫ	— начинается с колонки 17
КОММЕНТАРИЙ	— начинается с колонки 33

Для размещения поля согласно приведенному формату в качестве ограничителя поля может использоваться знак горизонтальной табуляции <TAB>. Например, оператор

```
ABC:<TAB>MOV<TAB>#MASK,R3<TAB> ;ПЕРЕСЫЛКА
<TAB>    <TAB>    <TAB>    <TAB> ;КОНСТАНТЫ
```

будет размещен в листинге в соответствии со стандартным форматом представления строки:

```
ABC:        — с колонки 1
MOV         — с колонки 9
#MASK, R3   — с колонки 17
;...        — с колонки 33
```

Хотя при программировании на языке макроассемблера такое форматирование исходных строк не является обязательным, однако желательно использование этого соглашения для всех исходных текстов программ.

Пустые строки распечатываются в листинге и могут использоваться для выделения участков программы.

3.3.1. ПОЛЕ МЕТКИ

Метка представляет собой определяемое пользователем имя (символ), значение которого при трансляции приравнивается к текущему значению счетчика адреса и заносится транслятором в таблицу символов.

Счетчик адреса является средством, с помощью которого транслятор определяет адреса памяти для операторов исходной программы по мере их трансляции. Значение адреса метки будет абсолютным или относительным в зависимости от того, является ли программная секция, в которой определяется эта метка, абсолютной или перемещаемой. В случае абсолютной программной секции значение счетчика адреса является абсолютным. Аналогично значение счетчика адреса в перемещаемой программной секции является относительным. Чтобы установить фактический абсолютный адрес метки, компоновщик вычисляет абсолютное смещение программной секции и прибавляет его к значению счетчика адреса.

Метка является средством символического обозначения ячейки внутри программы. Если метка имеется, она всегда стоит первой в операторе и должна быть ограничена двоеточием. Например, если текущее значение счетчика адреса будет 100 (абсолютное), то в операторе

```
ABCD: CLR  X  ;ПЕРЕСЫЛКА
```

макроассемблер присвоит метке ABCD значение 100. Последующее обращение к этой метке является обращением к абсолютному адресу 100. Если бы значение счетчика адреса в этом примере было относительным, окончательное значение метки ABCD было бы равно 100+K. K представляет собой величину смещения программной секции, вычисленную компоновщиком.

В одном поле метки может быть несколько меток. Все метки в поле имеют одно и то же значение. Например, если текущее значение счетчика адреса равно 100, то каждой из меток в операторе

```
AB: ⊙KL: C3.21: MOV  A,B[1]
```

присваивается значение 100.

Метки одного оператора могут размещаться также на последовательных строках. Например, тот же самый оператор может быть записан следующим образом:

---

[1] По технологическим причинам знак ⌑ в тексте книги заменен на ⊙.

```
AB:
⊙KL:
C3.21: MOV A,B
```

Всем трем меткам в этом случае присваивается одно и то же значение счетчика адреса. Из двух методов определения меток, представленных выше, последний является предпочтительным, так как сохранение стандартного формата размещения полей оператора в исходной программе облегчает ее понимание.

Двойное двоеточие (::) определяет метку как глобальную. На такую метку может ссылаться другой, независимо транслируемый программный модуль. Ссылки в этой метке из других модулей будут вычислены компоновщиком при связывании объектных модулей в загрузочный.

Например, оператор

```
KLM:: MOV A,B
```

объявляет метку KLM глобальным символом. Отличительным свойством глобального символа является то, что на него можно ссылаться из других модулей, в которых символ не определен, но объявлен глобальным.

По системным соглашениям знаки денежной единицы (⊙) и точка (.) зарезервированы для имен в системном программном обеспечении.

Метка может содержать любое число знаков. Однако значащими являются только первые шесть знаков, комбинация которых должна быть единственной для всех меток в исходной программе. Все метки заканчиваются двоеточием (:), которое рассматривается как ограничитель. Если первые шесть знаков в двух или более метках совпадают, то в листинге трансляции появляется флаг ошибки M (см. приложение 8).

Символ, используемый как метка, не может быть переопределен внутри исходной программы. Многократное переопределение метки порождает в листинге трансляции флаг ошибки M. Кроме того, любой оператор в исходной программе, который ссылается на многократно определенную метку, дает в листинге трансляции флаг ошибки D.

3.3.2. ПОЛЕ ОПЕРАЦИИ

Поле операции в исходном операторе следует за полем метки. Это поле может содержать мнемоническое обозначение инструкции, директиву транслятора или вызов макрокоманды.

Поле операции не может предшествовать метке; ему может предшествовать одна или несколько меток, а за ним может следовать один или несколько операндов и (или) комментарий. Пробелы и знаки табуляции в начале и конце поля операции не несут смыслового значения и служат только для отделения поля операции от предшествующего и следующего за ним полей.

Если оператор является мнемоническим обозначением инструкции, то генерируется код машинной инструкции. Затем макроассемблер вычисляет адреса операндов. Если оператор является директивой, то макроассемблер выполняет соответствующие этой директиве действия. Если оператор является вызовом макрокоманды, то транслятор вставляет коды, сгенерированные как расширение вызываемой макрокоманды. Ограничителями оператора могут быть пробел, знак табуляции, а также любой знак, не входящий в алфавит кода RADIX-50 (см. приложение 5).

Пример

```
MOV @A, B          ; ОПЕРАЦИЯ MOV ОГРАНИЧЕНА
                   ; ПРОБЕЛОМ
MOV     @A, B      ; ОПЕРАЦИЯ MOV ОГРАНИЧЕНА
                   ; ЗНАКОМ ТАБУЛЯЦИИ
MOV@A, B           ; ОПЕРАЦИЯ MOV ОГРАНИЧЕНА
                   ; ЗНАКОМ @
```

Несмотря на то что все вышеприведенные операторы эквивалентны по своим действиям, рекомендуется использовать вторую форму записи, так как она соответствует принятому стандарту оформления программы.

3.3.3. ПОЛЕ ОПЕРАНДА

Если поле операции содержит мнемоническое обозначение инструкции (код операции), то поле операнда определяет в программе те переменные, над которыми оператор будет производить действия. Поле операнда может быть использовано также для передачи соответствующих аргументов директив транслятора и макрокоманд.

Операнды могут быть выражениями или символическими аргументами. Если в поле операндов указываются несколько выражений, они должны быть разделены запятыми. Аналогично этому несколько символических аргументов должны быть разделены любыми из установленных разделителей — запятой, знаком табуляции и (или) пробелом. Операнду должно предшествовать поле операции; если поле операции опущено, то оператор интерпретируется как неявная директива .WORD.

Если поле операции содержит директиву макроассемблера или макрокоманду, то соответствующие операнды являются символическими аргументами, например

```
.MACRO COMPL ARG1, ARG2
```

Тип и число требуемых операндов определяются соответствующей директивой макроассемблера.

Поле операнда ограничивается точкой с запятой, если за ним следует комментарий. Если поле комментария отсутствует, то поле операнда ограничивается концом исходной строки. Например, в операторе

```
LABEL: MOV A,B ;ПОЛЕ КОММЕНТАРИЯ
```

знак табуляции между MOV и A ограничивает поле операции и определяет начало поля операнда, запятая разделяет операнды A и B, а точка с запятой ограничивает поле операнда и определяет начало поля комментария.

3.3.4. ПОЛЕ КОММЕНТАРИЯ

Поле комментария всегда начинается точкой с запятой (;) и, как правило, с колонки 33 и распространяется до конца строки. Это поле необязательно и может содержать любые знаки символьного кода за исключением управляющих знаков:

<DEL> — забой
<CR> — возврат каретки
<LF> — перевод строки
<VT> — вертикальная табуляция
<FF> — перевод формата

Все другие знаки, появляющиеся в поле комментария, даже специальные знаки, зарезервированные для использования в макроассемблере, проверяются только на допустимость в символьном коде и включаются в листинг трансляции в том виде, в каком они пишутся в исходной строке.

Комментарии можно продолжать на следующей строке. Такая строка должна начинаться со знака «;». При этом более наглядно, если продолжение комментария идет с той же позиции, что и начальная часть комментария. Строка комментария может быть включена как разделительная строка между строками операторов:

```
;
;   ПРИМЕР КОММЕНТАРИЯ ЗАГОЛОВКА
;
        MOV     A,B                     ;ПРИМЕР
                                        ;КОММЕНТАРИЯ
                                        ;НА НЕСКОЛЬКИХ СТРОКАХ
```

Комментарии не влияют на процесс трансляции или выполнение программы.

3.3.5. ДИРЕКТИВА .REM

Директива .REM является признаком начала последовательности строк комментариев. Формат директивы

```
.REM    знак
```

где знак — печатный символ, отмечающий конец последовательности строк комментариев.

Строки, следующие за этой директивой до появления указанного в этой директиве знака, транслятор считает строками комментария. Эти строки не обязаны содержать в своем начале признак комментария — точку с запятой, например

```
.REM !
```

Весь текст до второго восклицательного знака воспринимается транслятором как комментарий. При этом в тексте могут использоваться любые печатные знаки, кроме восклицательного.

3.3.6. УПРАВЛЕНИЕ ФОРМАТОМ

Горизонтальное или строчное форматирование исходной программы определяется с помощью знаков пробела и (или) табуляции. Эти знаки можно использовать для записи исходной программы в удобном для чтения виде, как показано в следующем примере:

```
POP: MOV (SP)+,R3        ; ОПЕРАНД ИЗ СТЕКА
POP: MOV      (SP)+,R3   ; ОПЕРАНД ИЗ СТЕКА
```

В последнем операторе содержатся знаки горизонтальной табуляции, т. е. элементы оператора разделены на 4 поля и легче распознаются, чем в предыдущем операторе.

Страничное форматирование листинга трансляции описано в гл. 6.

3.4. СИМВОЛЫ ЯЗЫКА

В этом параграфе описываются различные компоненты инструкций макроассемблера; допустимый набор знаков, соглашения, принятые для имен, использование чисел, операций, термов и выражений.

3.4.1. НАБОР ЗНАКОВ

В языке макроассемблера допустимы следующие знаки:

латинские буквы и буквы кириллицы (последние преобразуются в латинские буквы, если нет соответствующей директивы);

цифры от 0 до 9;

знак денежной единицы (⊙) и точка (.).

Эти знаки резервируются для использования в символах системного программного обеспечения. Допустимы также специальные знаки, которые приводятся ниже:

Знак	Пояснение и название
:	Двоеточие (ограничитель метки)
: :	Два знака двоеточия, следующие подряд (ограничитель метки, определяющий метку как глобальный символ)
=	Знак равенства (операция прямого присваивания)
= =	Два знака равенства, следующие подряд (операция прямого присваивания, определяющая символ как глобальный)
%	Процент (индикатор терма регистра)
<TAB> <пробел>	Горизонтальная табуляция, пробел (ограничитель аргумента макрокоманды или поля оператора)
#	Номер (индикатор непосредственного выражения)
@	Коммерческое «эт» (индикатор косвенной адресации)
(	Левая круглая скобка (индикатор начала терма регистра)
)	Правая круглая скобка (индикатор конца терма регистра)
,	Запятая (разделитель поля операнда)
;	Точка с запятой (индикатор начала поля комментария)
<	Левая угловая скобка (индикатор начала аргумента или выражения)
>	Правая угловая скобка (индикатор конца аргумента или выражения)
+	Плюс (знак операции «арифметическое сложение» или индикатор режима автоувеличения в машинных инструкциях)
—	Минус (знак операции «арифметическое вычитание» или индикатор режима автоуменьшения в машинных инструкциях)
*	Звездочка (знак операции «арифметическое умножение»)
/	Дробная черта (знак операции «арифметическое деление»)
&	Коммерческое «И» (знак логической операции «И»)
!	Восклицательный знак (знак логической операции «ИЛИ»)
"	Кавычки (индикатор символьного кода для двух знаков)
↑	Стрелка вверх (индикатор унарной операции или индикатор аргумента)
'	Апостроф (индикатор символьного кода для одного знака или индикатор конкатенации в макрокомандах)
\	Обратная дробная черта (индикатор символьного изображения числового значения терма)

В строке в качестве разделителей могут использоваться следующие знаки:

<пробел>, <TAB>	Разделители между полями инструкций, пробел может быть разделителем между символическими аргументами в поле операндов; пробелы внутри выражения игнорируются

, (запятая)	Разделитель для аргументов в поле операндов
< >	Парные угловые скобки используются для ограничения аргумента (если этот аргумент содержит разделительные знаки), парные угловые скобки могут быть использованы в любом месте программы для выделения выражения, которое будет обрабатываться как терм
	Предшествует аргументу, ограниченному парными печатными знаками; конструкция имеет вид ↑X...X; она эквивалентна <...> и применяется в случае, если сам аргумент содержит угловые скобки; при этом X в аргументе не должен встречаться

Знак считается запрещенным в одном из двух случаев:

если он не является допустимым знаком языка Макро. Такой знак вызывает немедленное завершение обработки текущей строки и вывод флага ошибки I в листинге трансляции;

если допустимый в языке знак является запрещенным в контексте данного оператора, такой знак вызывает появление в листинге трансляции флага ошибки О.

В языке макроассемблера допустимы унарные и бинарные операции. Унарная операция выполняется над одним термом (аргументом или операндом). Унарная операция относится к тому терму, перед которым она указана. После выполнения операции терм получает значение, которое может использоваться и как самостоятельная величина, и как элемент выражения.

Знаки унарных операций:

\+ унарный плюс, не меняет знака терма (+А есть само А);

— унарный минус, меняет знак терма (—А есть значение А в дополнительном коде). Например, —24 (восьмеричное) есть 177754 (восьмеричное), а —05204 (восьмеричное) есть 72574 (восьмеричное);

↑ универсальный индикатор унарной операции, используется для разового указания формата представления или основания системы счисления; вид преобразования терма определяется знаком, следующим за знаком ↑ :

- ↑F — задает однословное представление числа с плавающей запятой;
- ↑C — задает представление числа в обратном коде;
- ↑D — задает десятичное число;
- ↑O — задает восьмеричное число;
- ↑B — задает двоичное число.

К одному терму, аргументу или операнду могут относиться несколько унарных операций.

Ниже приводятся примеры задания унарных операций:

↑F3.0 — интерпретирует 3.0 как число с плавающей запятой и дает его однословное представление;

↑C24 — дает значение числа 24 (восьмеричное) в обратном коде, т. е. 177753 (восьмеричное);

↑B1110010 — интерпретирует 1110010 как двоичное число;

—↑D50 — эквивалентно выражению —<↑D50>;

↑C↑012 — эквивалентно выражению ↑C<↑012>.

В отличие от унарных операций бинарные операции указывают действия, которые нужно выполнить над элементами или термами внутри выражения. Знаки бинарных операций разделяют

элементы или термы внутри выражения для того, чтобы указать действия, которые необходимо выполнить над ними во время вычисления выражения в процессе трансляции.

Знаки бинарных операций:

\+ Сложение;
— Вычитание;
× Умножение (результат — 16-разрядная величина);
/ Деление (результат — 16-разрядная величина);
& Поразрядное логическое «И»;
! Поразрядное логическое «ИЛИ».

Все бинарные операции имеют один и тот же приоритет. Элементы можно сгруппировать в пределах выражения, заключив их в угловые скобки. Термы в угловых скобках обрабатываются первыми, а остальные операции выполняются слева направо. Например, результатом вычисления выражения $1+2\cdot 3$ будет восьмеричное число 11, а результатом вычисления $1+<2\cdot 3>$ — восьмеричное число 7.

3.4.2. СИМВОЛЫ МАКРОАССЕМБЛЕРА

В языке различают символы трех типов: постоянные символы, символы, определяемые пользователем, и символы макрокоманд. Соответственно этому транслятор работает с таблицами символов трех типов: PST — таблица постоянных символов, UST — таблица символов пользователя и MST — таблица символов макрокоманд.

Таблица PST содержит все постоянные символы и является частью загрузочного модуля транслятора. UST и MST заполняются при трансляции программы. Символы, определенные пользователем, вносятся в эти таблицы, когда встречаются в тексте транслируемого модуля.

В таблицу постоянных символов входят мнемонические обозначения инструкций и директив. Эти символы являются постоянной частью транслятора и не нуждаются в определении до использования.

Символы пользователя могут определяться несколькими способами. Они могут использоваться в качестве меток. В этом случае их переопределение в данном модуле запрещено и приводит к сообщению об ошибке. Символы пользователя могут определяться операцией прямого присваивания, которая допускает переопределение их значений. Эти символы включаются в таблицу символов пользователя по мере того, как они встречаются в процессе трансляции.

Символы макрокоманд представляют собой имена макрокоманд. Они включаются в таблицу символов макрокоманд по мере того, как определения макрокоманд встречаются в процессе трансляции.

Символы, определяемые пользователем, и символы макрокоманд могут состоять только из алфавитно-цифровых знаков, знаков ⊙ и точки. Все остальные знаки запрещены.

При составлении символов и имен макрокоманд необходимо придерживаться следующих правил:

первый знак не должен быть цифрой;

первые шесть знаков не должны совпадать с первыми шестью знаками других символов;

символ может состоять более чем из шести разрешенных знаков, но седьмой и последующие знаки проверяются только на допустимость и транслятором игнорируются. Их использование целесообразно для обеспечения наглядности программы;

пробелы, знаки табуляции и запрещенные знаки не следует включать в символ.

Значение символа зависит от его использования в программе. Символ в поле операции может относиться к любому из трех вышеприведенных типов. Для определения типа и значения символа, находящегося в поле операции, макроассемблер проводит поиск в таблицах в следующем порядке: таблица имен макрокоманд, таблица постоянных символов, таблица символов, определенных пользователем.

Этот порядок поиска позволяет переопределить постоянные символы как имена макрокоманд. Однако во избежание неправильной интерпретации пользователь должен помнить последовательность, в которой осуществляется поиск.

Если символ появляется в поле операнда, то сначала поиск ведется в таблице символов, определенных пользователем, а потом — в PST.

В зависимости от использования символов, определяемых пользователем, в исходной программе они являются либо локальными, либо глобальными.

Обычно транслятор воспринимает все символы, определяемые пользователем, как внутренние, т. е. их определение ограничивается модулем, в котором они появляются. Однако имена могут быть явно объявлены глобальными символами одним из трех способов: директивой .GLOBL, двойным двоеточием (::) в определении метки и двумя знаками равенства (==) в операторе прямого присваивания.

Все символы внутри модуля, которые не определены к концу трансляции, объявляются глобальными по умолчанию.

Символы, не определенные в конце трансляции, получают значение нуля и помещаются в таблицу UST как неопределенные глобальные символы. Однако если действует директива .DSABL GBL, она отменяет функцию автоматического объявления неопределенных символов глобальными, и операторы, содержащие неопределенные символы, отмечаются в листинге трансляции флагом ошибки U.

Через глобальные символы устанавливается связь между независимо транслируемыми модулями внутри загрузочного модуля. Глобальные символы, определенные, например, как метки внутри одного программного модуля, могут служить адресами точек входа

из других программных модулей. На такие символы ссылаются из других модулей, например, для передачи управления или для обращения к данным.

3.4.3. ОПЕРАТОР ПРЯМОГО ПРИСВАИВАНИЯ

Оператор прямого присваивания ставит в соответствие символу заданное значение. Если оператор прямого присваивания определяет символ впервые, этот символ заносится в таблицу символов пользователя. Символ может быть переопределен в последующем операторе прямого присваивания. Формат оператора прямого присваивания

СИМВОЛ = ВЫРАЖЕНИЕ

или

СИМВОЛ == ВЫРАЖЕНИЕ

Как отмечалось ранее, оператор прямого присваивания, содержащий два знака равенства, следующие подряд, определяет символ как глобальный.

Специальный знак прямого присваивания «= :» при попытке переопределения заданного им символа приводит к сообщению об ошибке. Для запрещения переопределения глобальных символов используется соответственно знак прямого присваивания «==:».

Примеры операторов прямого присваивания:

```
A       = 1                     ; СИМВОЛУ "A" ПРИСВОЕНО ЗНАЧЕНИЕ 1
B       = A - 1 & MASKLOW       ; СИМВОЛУ "B" ПРИСВОЕНО
                                ;   ЗНАЧЕНИЕ ВЫРАЖЕНИЯ
        D = .                   ; СИМВОЛУ "D" ПРИСВАИВАЕТСЯ ТЕКУЩЕЕ
                                ;   ЗНАЧЕНИЕ СЧЕТЧИКА АДРЕСА, Т.Е.
        MOV   #1, SLC           ; АДРЕС ИНСТРУКЦИИ MOV НА
                                ; СЛЕДУЮЩЕЙ СТРОКЕ
```

Примечание. Использование символа «.» приведено в п. 3.4.6.

Для операторов прямого присваивания действуют следующие правила:

один оператор прямого присваивания может определить только один символ;

за оператором прямого присваивания может следовать только поле комментария;

допускается только один уровень ссылки вперед.

В примере показано недопустимое использование двухуровневой ссылки вперед:

```
X=Y   ; НЕПРАВИЛЬНАЯ ССЫЛКА ВПЕРЕД
Y=Z
Z=1
```

Данный пример вызвал бы в листинге трансляции в строке, содержащей неверную ссылку вперед, флаг ошибки U.

Перечисленные особенности языка связаны с тем, что Макро является двухпроходным транслятором. В приведенном выше примере на первом проходе трансляции символы X, Y, Z будут записаны в таблицы символов пользователя, причем символ Z получит значение 1, а значения символов X, Y не будут определены. На втором проходе трансляции символ Y получит значение 1, а символ X останется неопределенным, что и приведет к сообщению об ошибке в строке, содержащей символ X.

Один уровень ссылок вперед допустим только для локальных символов. Глобальные символы, определяемые оператором прямого присвоения, не могут содержать ссылок вперед, т. е. выражение, определяющее значение глобального символа, не может само содержать символы, определяемые через другие. Такие ссылки вперед вызывают в листинге трансляции флаг ошибки A.

3.4.4. СИМВОЛЫ РЕГИСТРОВ

Восемь универсальных регистров нумеруются цифрами (0—7) и могут быть записаны в исходной программе как R1, R2, R3, R4, R5, R6 или SP, R7 или PC.

Программисту рекомендуется использовать указанные стандартные символические имена при всех обращениях к регистрам. В связи с особым назначением регистров 6 и 7 в системе они имеют специальные обозначения —SP и PC соответственно.

По усмотрению программиста символ регистра может быть определен в программе через оператор прямого присваивания. Выражение, определяющее символ регистра, должно быть допустимой абсолютной величиной.

Пользователь может сам переопределить символы регистров, отменяя через директиву .DSABL REG стандартное определение регистров. Однако это делать не рекомендуется. Попытка пользователя переопределить символы регистров без использования директивы .DSABL REG вызывает в листинге трансляции в строке с оператором присваивания флаг ошибки R.

Все нестандартные обозначения регистров должны быть определены прежде, чем к ним будет произведено обращение в исходной программе.

Выражение, определяющее регистр и имеющее значение меньше нуля или больше семи, отмечается в листинге трансляции флагом ошибки R.

Любому разрешенному терму или выражению, определяющему регистр, должен предшествовать знак %. Например, оператор CLR %3+1 эквивалентен по функции оператору CLR %4.

Каждая из этих команд очищает регистр 4. В противоположность этому оператор CLR 4 очищает содержимое ячейки с адресом 4.

3.4.5. ЛОКАЛЬНЫЕ СИМВОЛЫ

Локальными символами называются символы специального формата, используемые как метки в блоке локальных символов.

Блок локальных символов представляет собой набор строк исходной программы, ограниченный одним из следующих способов:

а) символическими метками, составленными как обычно. Заметим, что оператор типа

XXX=ВЫРАЖЕНИЕ

представляет собой оператор прямого присваивания и не создает метку, ограничивающую диапазон блока;

б) директивами .PSECT, .CSECT или .ASECT;

в) диапазон блока локальных символов может быть указан следующими директивами: начало — директивой .ENABL LSB, окончание — директивой .DSABL LSB.

Локальные символы имеют форму N⊙; N — десятичное целое число от 1 до 65535 включительно[1]. Они могут быть использованы только на границах слов, т. е. ими могут отмечаться только четные адреса.

Если не использовалась директива .ENABL LSB, локальная метка не может быть установлена далее чем за -128_{10} или $+127_{10}$ слов от начала блока локальных символов. Пример локальных символов:

```
1⊙
27⊙
32120⊙
59⊙
124⊙
```

Локальные символы удобны для использования в инструкциях перехода, так как позволяют уменьшать вероятность появления многократно определенных символов в программах пользователя. Они легко отличимы от символов входных точек и локальных символов, так как к последним не может быть обращений извне блока локальных символов. В других блоках могут быть локальные метки с такими же именами, как и в данном блоке. Это не приводит к конфликтным ситуациям.

Использование локальных меток имеет еще и то преимущество, что для них требуется меньше места в таблице символов, чем для обычных меток. Ниже приведены примеры использования локальных символов:

```
1                                .TITLE  EXAMPLE  1
2                       ;
3                       ;
4                                .PSECT  X
5  000000  012700  XXX:  MOV     #CONST1, R0
           000000G
6  000004  005020  1⊙:   CLR     (R0)+
7  000006  022700         CMP     #END1, R0
           000000G
```

[1] В предшествующей версии макроассемблера число N могло принимать значения от 1 до 128.

```
 8 000012  100774          BMI     1⊙
 9 000000                  .PSECT  Y
10 000000  012700  YYY:    MOV     #CONST2, R0
           000000G
11 000004  005020  1⊙:     CLR     (R0)+
12 000006  022700          CMP     #END2, R0
           000000G
13 000012  100774          BMI     1⊙
14 000000                  .PSECT
15 000000  012700  ZZZ:    MOV     #CONST3, R0
           000000
16 000004  005020  1⊙:     CLR     (R0)+
17 000006  022700          CMP     #END3, R0
           000000G
18 000012  101374          BHI     1⊙
19         000001          .END
```

Программисту выделены локальные символы от 1⊙ до 29999⊙, а символы от 30000⊙ до 65535⊙ используются транслятором для автоматической генерации меток. Такие локальные символы используются в макрорасширениях.

Если локальный символ многократно определен в одном блоке локальных символов, то это вызывает появление в листинге трансляции флага ошибки P.

3.4.6. СЧЕТЧИК АДРЕСОВ ПРОГРАММЫ

Точка (.) используется как символ текущего значения счетчика адресов программы. При использовании ее в поле операнда инструкции она обозначает адрес первого слова инструкции. При использовании точки в поле операнда директивы макроассемблера она представляет адрес текущего байта или слова, например

```
A: MOV # .,R0
```

В данном примере точка будет иметь значение адреса метки A, т. е. адреса первого слова инструкции MOV. Эта запись эквивалентна инструкции

```
A: MOV #A, R0
```

Если

```
. = 500
VAL = 0
.WORD   177535, .+4, VAL
```

то счетчику адреса присваивается начальное значение 500_8. В процессе трансляции Макро по директиве .WORD резервирует память, начиная с адреса 500. В зарезервированной памяти сохраняются значения операндов. Значение первого операнда 177535_8 запоминается по адресу 500. Значение второго операнда (.+4) запоминается по адресу 502. Это значение вычисляется как текущее значение счетчика адреса (которое теперь 502) плюс абсолютное

значение 4. Таким образом, по адресу 502 будет записано значение 506. Значение VAL, равное 0, сохраняется по адресу 504.

В начале каждого прохода макроассемблер очищает счетчик адресов программы. Обычно последовательные адреса памяти присваиваются каждому байту генерируемой при трансляции программы. Значение счетчика адресов может быть изменено с помощью оператора прямого присваивания вида

.=ВЫРАЖЕНИЕ

Подобно другим символам языка символ счетчика адресов (.) является относительным или абсолютным в зависимости от текущей секции программы. Это свойство счетчика адресов не может быть изменено. Значение выражения, присваиваемое счетчику адресов, не должно содержать ссылок к другой программной секции, даже если тип программной секции не меняется.

Выражение, определяющее счетчик адресов, не должно содержать ссылок вперед или символов, значение которых изменяется от одного прохода трансляции к другому. Такие нарушения вызывают появление в листинге трансляции флага ошибки A.

При трансляции абсолютной секции программы счетчик адресов получает значение, определенное в программе. Начальным значением счетчика адреса по умолчанию является 0. При трансляции относительной секции начальное значение — всегда 0. При компоновке относительные секции могут перемещаться. Пример использования счетчика адресов приводится ниже.

```
        .ASECT
        .=300           ; СЧЕТЧИК АДРЕСОВ ПОЛУЧАЕТ
ONE:  MOV  .+10, GOU    ; АБСОЛЮТНОЕ ЗНАЧЕНИЕ 300
                        ; (ВОСЬМЕРИЧНОЕ).
                        ; МЕТКА ONE ИМЕЕТ ЗНАЧЕНИЕ
                        ; 300 (ВОСЬМЕРИЧНОЕ) .+10 = 310
                        ; (ВОСЬМЕРИЧНОЕ);
                        ; СОДЕРЖИМОЙ ЯЧЕЙКИ 310
                        ; (ВОСЬМЕРИЧНОЕ) БУДЕТ
                        ; ПОМЕЩЕНО В ЯЧЕЙКУ GOU
        .=320           ; СЧЕТЧИК АДРЕСОВ ПОЛУЧАЕТ
                        ; АБСОЛЮТНОЕ ЗНАЧЕНИЕ 320
                        ; (ВОСЬМЕРИЧНОЕ)
TOR:  MOV  ., INDEX     ; МЕТКА TOR ИМЕЕТ ЗНАЧЕНИЕ
                        ; 320 (ВОСЬМЕРИЧНОЕ)
                        ; СОДЕРЖИМОЕ ЯЧЕЙКИ 320
                        ; (ВОСЬМЕРИЧНОЕ), Т.Е.
                        ; ДВОИЧНЫЙ КОД САМОЙ
                        ; ИНСТРУКЦИИ БУДЕТ
                        ; ПОМЕЩЕН В ЯЧЕЙКУ INDEX
        .PSECT
        .=.+20          ; СЧЕТЧИК АДРЕСОВ
                        ; НЕИМЕНОВАННОЙ
                        ; ПРОГРАММНОЙ
                        ; СЕКЦИИ ПОЛУЧАЕТ
                        ; ОТНОСИТЕЛЬНОЕ ЗНАЧЕНИЕ
                        ; 20 (ВОСЬМЕРИЧНОЕ)
THR:  .WORD 0           ; МЕТКА THR ИМЕЕТ
                        ; ОТНОСИТЕЛЬНОЕ ЗНАЧЕНИЕ 20 (ВОСЬМЕРИЧНОЕ)
```

Счетчик адресов может быть использован в программе для резервирования памяти. Например, если текущее значение счетчика адресов есть 1000_8, то каждый из операторов

```
.= .+40
.BLKB 40
.BLKW 20
```

резервирует в исходной программе 40_8 байт памяти. Однако для резервирования памяти предпочтительнее использование директив .BLKB и .BLKW, которые описываются в следующей главе.

3.4.7. ЧИСЛА

В языке Макро принято соглашение о том, что все числа в исходной программе интерпретируются в восьмеричной системе счисления, если нет никаких специальных указаний относительно системы счисления.

Принятая по умолчанию восьмеричная система счисления может быть изменена директивой .RADIX. Кроме того, отдельные числа по указанию программиста могут обрабатываться как числа в десятичной, двоичной или восьмеричной системах счисления.

Если оператор исходной программы содержит число с цифрой, не допустимой в текущей системе счисления, то этот оператор отмечается в листинге трансляции флагом ошибки N. При этом транслятор вычисляет каждое встретившееся число с недопустимыми цифрами как десятичное.

Отрицательным числам предшествует знак «минус». В процессе трансляции отрицательные числа представляются в дополнительном коде. Положительным числам может предшествовать знак «плюс», хотя это необязательно.

Число, занимающее больше 16 разрядов, т. е. большее, чем 177777 (восьмеричное), усекается слева и отмечается флагом ошибки T в распечатке программы. Числа всегда представляют собой абсолютные величины.

3.5. ТЕРМЫ

Терм является компонентом выражения. Терм может быть:

1) числом; значение числа не должно превышать 177777_8;

2) символом; символы интерпретируются согласно следующим правилам:

точка, указанная в выражении, означает использование текущего значения счетчика адреса;

для символа, определенного пользователем, используется значение символа из таблицы UST;

имя неопределенного символа размещается в таблице симво-

лов, определенных пользователем. Ему присваиваются значение 0 и свойство неопределенного глобального символа.

Если действует директива .DSABL GBL (функция автоматического объявления неопределенных символов глобальными запрещена), в листинге трансляции появляется флаг ошибки U;

3) одним знаком либо двумя знаками в символьном коде. Одному знаку предшествует апостроф, двум знакам — кавычки;

4) выражением, заключенным в угловые скобки. Любая величина, заключенная в угловые скобки, обрабатывается прежде, чем оставшаяся часть выражения, в котором находится эта величина; угловые скобки используются, например, для изменения обычного порядка выполнения операций (для отличия A*B+C от A*<B+C>);

5) унарным оператором, за которым следует символ или число.

Примеры термов:

A, −2, <17! MASK + 2>

3.6. ВЫРАЖЕНИЯ

Выражение является комбинацией термов, разделенных знаками операций. Значение выражения не должно превосходить 16-разрядной величины. Вычисление выражения включает определение его перемещаемости. Результирующее значение выражения может быть любым из четырех типов: абсолютным, относительным, внешним или сложным относительным.

Выражения обрабатываются слева направо, за исключением обработки унарной операции, которая предшествует обработке бинарных операций. Там, где это необходимо, термы обрабатываются до их использования в выражениях. Допустимо использование подряд нескольких знаков унарных операций. Например, −+−A эквивалентно −<+<−A>>.

Отсутствующий терм, выражение или внешний глобальный символ интерпретируются как нуль. В случае отсутствия в выражении знака операции, а также при недопустимом знаке операции анализ выражения прекращается. В листинге трансляции в зависимости от контекста самого выражения указывается флаг ошибки A или Q либо одновременно оба. Например, TAG!LA 1776 интерпретируется как TAG!LA, так как первый отличительный от пробела знак, следующий за LA, не является ни знаком операции, ни разделителем выражений (т. е. запятой), ни ограничителем поля операции (т. е. точкой с запятой или ограничителем строки). Пробелы внутри выражения игнорируются.

Значением внешнего выражения, вычисленного транслятором, является значение абсолютной части выражения. Например, если EXT — внешний символ, то EXT+A после трансляции имеет значение символа A. Компоновщик модифицирует это значение таким образом, чтобы оно стало равным EXT+A. Выражение может быть: абсолютным, относительным, внешним (или глобальным), сложным относительным.

Выражение является абсолютным, если его значение фиксировано. Выражение, термы которого являются числами или знаками символьного кода, имеет абсолютное значение. Выражение — относительный терм минус относительный терм, где оба элемента относятся к одной и той же программной секции, также является абсолютным, поскольку такое выражение преобразуется транслятором при завершении анализа выражения к единичному терму. Например, выражение TMP1—TMP2, где TMP1 и TMP2 определены в одной и той же секции программы, является абсолютным выражением. Терм, который содержит метку, определенную в абсолютной секции, будет иметь абсолютное значение.

Выражение является относительным, если его значение фиксируется относительно базового адреса программной секции, в которой оно появилось, так как оно корректируется во время компоновки. Выражения, термы которых содержат метки, определенные в перемещаемых секциях, или счетчик адреса, определяемый в перемещаемой секции, имеют относительное значение.

Выражение, термы которого содержат глобальный символ, не определенный в текущей программе, является внешним или глобальным. Его назначение определяется при трансляции только частично. Окончательно оно определяется компоновщиком.

Выражение является сложным относительным, если выполняется одно из следующих условий:

выражение содержит глобальную ссылку и относительный символ;

выражение содержит относительные термы, определенные в различных программных секциях;

величина, полученная в результате вычисления выражения, требует более одного перемещения. Например, если относительные символы TML1 и TML2 находятся в одной программной секции и образуют выражение вида TML1+TML2, то будут выполнены два перемещения, так как каждый символ определяется относительно базы программной секции;

операция, отличная от сложения, задана для неопределенного глобального символа;

операции, отличные от сложения, вычитания, изменения знака, дополнения, определены для относительной величины.

Вычисление относительных, внешних, сложных относительных выражений завершается компоновщиком.

4

ГЛАВА

ОСНОВНЫЕ ДИРЕКТИВЫ МАКРОАССЕМБЛЕРА

Знание основных директив и возможностей транслятора позволяет достаточно эффективно писать программы на языке Макро. Все директивы, рассматриваемые в данной главе, разбиты в соответствии с функциональными возможностями на четыре группы и обеспечивают:

управление распределением памяти в модуле и занесение начальных значений в ячейки оперативной памяти;

условную трансляцию отдельных фрагментов текста программы на входном языке;

средства идентификации модулей в листинге и объектном коде;

секционирование программ.

4.1. УПРАВЛЕНИЕ ПАМЯТЬЮ

Управление памятью осуществляется с помощью следующих директив и знаков:

.BYTE	' (апостроф)
.WORD	" (кавычки)
.ASCII	↑B
.ASCIZ	↑C
.FLT2	↑D
.FLT4	↑F
.RAD50	↑O
	↑R

Перечисленные директивы позволяют не только распределить память под данные, но и определить начальные значения, которые будут занесены в соответствующие ячейки оперативной памяти.

4.1.1. ДИРЕКТИВА .BYTE

Директива .BYTE используется при задании последовательности байтов данных. Директива

.BYTE ВЫРАЖЕНИЕ

записывает восьмеричное значение выражений в последовательные байты, начиная с адреса директивы .BYTE. Например:

```
VAM=5
.   =410
.BYTE   ↑D48, VAM   ; В ЯЧЕЙКЕ ПАМЯТИ 410 ДОЛЖНО БЫТЬ
                    ; ЗАПИСАНО 060 — ВОСЬМЕРИЧНЫЙ
                    ; ЭКВИВАЛЕНТ 48 (ДЕСЯТИЧНОЕ).
                    ; В ЯЧЕЙКЕ 411 ДОЛЖНО БЫТЬ 005
```

Значения термов, входящих в выражение в директиве .BYTE, рассматриваются как слова. Когда значение выражения подсчитано, результат усекается до 8 младших разрядов. Причем старший байт слова перед усечением должен содержать 0 или —1 (если значение байта отрицательное, разряд знака распространяется на весь старший байт). Если старший байт значения выражения отличен от 0 или от —1, оно также усекается до младших 8 бит. Строка с этим оператором отмечается флагом ошибки Т. Если выражение относительное, то возможно, что при компоновке задачи выражение получит значение, содержащее более 8 бит. В этом случае компоновщик выдает соответствующее диагностическое сообщение. Например:

```
A:  .BYTE    23   ; ЗАПИСЫВАЙТЕ В БАЙТ ВОСЬМЕРИЧНОЕ
                  ; ЧИСЛО 23 И
    .BYTE    A    ; ОТНОСИТЕЛЬНОЕ ЗНАЧЕНИЕ
                  ; СИМВОЛА «А»
```

Пустой операнд директивы .BYTE интерпретируется как нуль, например:

```
.=420               ; В БАЙТЫ 420, 421, 422 и 423
.BYTE    ,,,        ; ЗАПИСЫВАЮТСЯ НУЛИ
```

Три запятые в поле операнда разделяют подразумеваемые четыре нуля. В объектном модуле резервируются 4 байта; каждый байт содержит значение 0.

```
.BYTE    'С,'Т,'Р,'О,'К,'А
```

приводит к выделению 6 байт памяти, в которые записываются коды внутреннего представления знаков слова «СТРОКА».

4.1.2. ДИРЕКТИВА .WORD

Директива .WORD используется для занесения в тело программы данных в виде последовательно расположенных слов. Директива имеет вид:

```
.WORD    ; ВЫРАЖЕНИЕ
```

Она записывает восьмеричное значение выражения в слово, адрес которого — адрес директивы .WORD.

.WORD ВЫРАЖЕНИЕ1, ВЫРАЖЕНИЕ2, ... записывает вось-

меричные значения выражений 1, 2 и т. д. в последовательно расположенные слова, начиная с адреса директивы .WORD.

Значение операнда директивы .WORD должно занимать не более 16 разрядов. Значение выражения запоминается в отдельном слове объектной программы. Если имеется несколько операндов, они разделяются запятыми, и их значения запоминаются в последовательно расположенных словах. Например:

```
CAL   =0
.     =500
.WORD  177535,.+4, CAL      ;ЗАПИСЫВАЮТСЯ 177535, 506
                            ;и 0 ПО АДРЕСАМ 500, 502 и 504
```

Если директива .WORD имеет пустой операнд, то он будет интерпретироваться как нуль, например:

```
.     =500
.WORD  ,5,      ;0, 5 и 0 БУДУТ ЗАПИСАНЫ В ЯЧЕЙКАХ
                ;500, 502 и 504
```

Поле операции, оставшееся пустым (любая операция, не опознанная транслятором ни как вызов макрокоманды, ни как код операции, ни как директива или точка с запятой), будет интерпретироваться как директива .WORD. Например:

```
.       =440            ;ЗНАЧЕНИЕ 100 ЗАПОМИНАЕТСЯ
LABEL:  100, LABEL      ;ПО АДРЕСУ 440, И ЗНАЧЕНИЕ 440
                        ; ЗАПОМИНАЕТСЯ ПО АДРЕСУ 442
```

Необходимо отметить, что директива .WORD допустима только при четном значении счетчика ячеек транслятора. В противном случае транслятор даст сообщение об ошибке.

4.1.3. ПРЕДСТАВЛЕНИЕ ОДНОГО ИЛИ ДВУХ ЗНАКОВ В СИМВОЛЬНОМ КОДЕ

Знаки апостроф (') и кавычки (") используются для генерации в объектном модуле символьного кода для одного и для двух знаков исходного текста соответственно.

Если за апострофом следует какой-либо знак, то образуется слово, в котором 8-разрядное представление знака в символьном коде помещено в младший байт, а нуль — расширение знакового разряда младшего байта — помещен в старший байт. Например, в результате

```
MOV #'A,R0
```

в младший байт регистра R0 пересылается символьный восьмеричный код A — 101, в старший байт — 0.

После знака ' недопустимы знаки: <CR> — возврат каретки, <NUL>—пусто, <DEL>—стирание, <LF>—перевод строки, <FF> — перевод формата.

Если эти знаки используются после знака ', то в листинге транслятора устанавливается флаг ошибки A. Если за кавычками

следуют два знака, то образуется слово, в котором помещаютс 7-разрядные представления двух знаков в символьном коде. Например, в результате выполнения операции

```
MOV #"AB,R0
```

в младший байт регистра R0 пересылается символьный восьмеричный код «A» — 101, в старший байт — код «B», равный 102

После знака " недопустимы знаки <CR>, <NUL>, <DEL> <LF> или <FF>. Если эти знаки появляются после знака " то в листинге трансляции устанавливается флаг ошибки A. Список знаков КОИ-7 приведен в приложении 4.

4.1.4. ДИРЕКТИВА .ASCII

Директива .ASCII заносит строку знаков в символьном коде в последовательные байты объектной программы. Эта директива имеет формат

```
.ASCII /строка-знаков/
```

где строка-знаков — строка любых допустимых в символьном коде печатных знаков;

/ /— ограничительные знаки: ограничителем может быть любой печатный знак, за исключением знаков < , =, ; и любого знака, используемого в строке текста.

Если ограничительные знаки не составляют пару или используется неверный ограничительный знак, то в листинге трансляции директива .ASCII отмечается флагом ошибки A.

Недопустимыми знаками являются <NUL>, <DEL>, <LF> и все другие не выдаваемые на печать знаки КОИ-7, исключая <CR> и <FF>. Использование недопустимых знаков в строке вызывает появление в листинге трансляции флага ошибки I. По знакам <CR> и <FF> заканчивается сканирование исходной строки. Преждевременное появление этих знаков в директиве .ASCII приводит к тому, что макроассемблер не может закончить сканирование парного ограничителя в конце строки знаков. В этом случае в листинге устанавливается флаг ошибки A.

Не выводимые на печать знаки могут быть представлены в строке знаков .ASCII соответствующими символьными кодами, заданными в текущем основании. Они должны быть ограничены парными угловыми скобками. В угловые скобки разрешается также заключать любое дозволенное описание выражений. Значение этого выражения записывается в соответствующий байт, например

```
.ASCII<15>/ABC/<A+2>/DEF/<5><4>
```

В данном примере выражения <15>, <A+2>, <5> и <4> дают значения непечатаемых знаков. Абсолютные величины усекаются до 8 бит в соответствии с теми же правилами, что и для директивы .BYTE.

Угловые скобки могут быть включены в строку между ограничительными знаками; использованные таким образом угловые скобки теряют свой обычный смысл как разграничители для непечатаемых знаков. Например, оператор

.ASCII /ABC<ВЫРАЖЕНИЕ> DEF/

содержит единственную строку знаков и не выполняет вычисление выражения, заключенного в угловые скобки. Ниже приведены примеры использования директивы

```
.ASCII /<AB>/     ; ЗАПИСЫВАЕТ СИМВОЛЬНЫЕ КОДЫ
                  ; ЧЕТЫРЕХ ЗНАКОВ <, A, B,> В
                  ; ПОСЛЕДОВАТЕЛЬНО РАСПОЛОЖЕННЫЕ
                  ; БАЙТЫ
```

Знаки ; и = могут быть использованы как ограничивающие знаки строки, но требуется особая внимательность при использовании этих знаков (они используются как индикатор комментария и знак операции присваивания соответственно). Например:

```
.ASCII; ABC; /DEF/    ; ЗАПОМИНАЕТ ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ
                      ; ЗНАКОВ A, B, C, D, E И F В ШЕСТИ
                      ; ПОСЛЕДОВАТЕЛЬНЫХ БАЙТАХ; ПРИМЕНЯТЬ
                      ; ТАКУЮ КОНСТРУКЦИЮ НЕ РЕКОМЕНДУЕТСЯ
.ASCII /ABC/; DEF;    ; ЗАПОМИНАЕТ ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ
                      ; ЗНАКОВ A, B И C В ТРЕХ ПОСЛЕДО-
                      ; ВАТЕЛЬНЫХ БАЙТАХ; ЗНАКИ D, E,
                      ; F И ";" ТРАКТУЮТСЯ КАК КОММЕНТАРИЙ
.ASCII /ABC/ = DEF =  ; ЗАПОМИНАЕТ ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ
                      ; ЗНАКОВ A, B, C, D, E, F В ШЕСТИ
                      ; ПОСЛЕДОВАТЕЛЬНЫХ БАЙТАХ; ПРИМЕНЯТЬ
                      ; НЕ РЕКОМЕНДУЕТСЯ
```

Знак равенства трактуется как оператор прямого присваивания, если он появляется первым в строке .ASCII, как показано в следующем примере:

```
ASCII = DEF =     ; ВЫПОЛНЯЕТСЯ ОПЕРАТОР ПРЯМОГО
                  ; ПРИСВАИВАНИЯ .ASCII = DEF =
                  ; И ВЫДАЕТСЯ ФЛАГ ОШИБКИ Q,
                  ; КОГДА ВСТРЕЧАЕТСЯ ВТОРОЙ ЗНАК "="
```

4.1.5. ДИРЕКТИВА .ASCIZ

Директива .ASCIZ эквивалентна директиве .ASCII с автоматическим добавлением нулевого байта в качестве последнего знака строки символов.

При обработке строки текста, созданной с помощью директивы .ASCIZ, обнаружение байта, содержащего значение 0, означает конец строки.

Для директивы .ASCIZ действуют те же правила относительно ограничителей и допустимых знаков в строке, что и для .ASCII.

Пример использования директивы .ASCIZ:

```
   CR = 15              ; ПРИСВОИТЬ CR ЗНАЧЕНИЕ 15
   LF = 12              ; ПРИСВОИТЬ LF ЗНАЧЕНИЕ 12
K: .ASCIZ (CR) (LF)       /ВВЕСТИ СИМВОЛ?/
   ...
                        ; ПЕЧАТЬ ПУСТОЙ СТРОКИ И ВОПРОСА
                        ; "ВВЕСТИ СИМВОЛ?" ЯВЛЯЕТСЯ
   .PRINT # K           ; МАКРОКОМАНДОЙ, РАСПЕЧАТЫВАЮЩЕЙ
   ...                  ; СТРОКУ, ОГРАНИЧЕННУЮ БАЙТОМ,
                        ; СОДЕРЖАЩИМ НУЛЬ
```

4.1.6. ДИРЕКТИВА .RAD50

Директива .RAD50 позволяет кодировать данные в коде RADIX-50. Эта кодировка дает возможность записи трех знаков в 16 разрядах. Форма директивы

.RAD50 /СТРОКА 1/.../СТРОКА N/

где СТРОКА — список преобразуемых в код RADIX-50 знаков;
/ / — ограничительные знаки.

Строки могут состоять из букв от A до Z, цифр от 0 до 9, символа ⊙, точки (.) и пробела.

Три последовательных знака строки кодируются в одно слово. Если в строке или в последней группе меньше трех знаков, то такая группа дополняется справа до трех знаков пробелами.

Если в строке встречаются запрещенные печатные знаки, они вызывают флаг ошибки Q в листинге трансляции.

Как и в директиве .ASCII, знаки <VT>, <LF>, <DEL> и все другие непечатаемые знаки, за исключением <CR> и <LF>, являются запрещенными. Их появление в строке знаков отмечается в листинге трансляции флагом ошибки I. Аналогично .ASCII знаки <CR> и <FF> приводят к флагу ошибки A, так как эти знаки являются признаком конца строки и не позволяют макроассемблеру обнаружить парный ограничитель.

Ограничительными могут быть любые парные печатные знаки, за исключением знака равенства (=), левых угловых скобок (<) или точки с запятой (;). Ограничительные знаки не должны встречаться внутри текста самой строки. Если ограничительные знаки не являются парными или в качестве ограничителей используются недопустимые символы, директива .RAD50 отмечается в листинге трансляции флагом ошибки A.

```
. RAD50    /ABC/          ; ЗАПИСАТЬ "ABC" В ОДНО СЛОВО
. RAD50    /AB /          ; ЗАПИСАТЬ "AB" И ПРОБЕЛ В ОДНО СЛОВО
. RAD50    /  /           ; ЗАПИСАТЬ 3 ПРОБЕЛА В ОДНО СЛОВО
. RAD50    /ABCD/         ; ЗАПИСАТЬ "ABC"  В ОДНО СЛОВО,
                          ; А "D" И ДВА ПРОБЕЛА ВО ВТОРОЕ СЛОВО
```

Каждый знак преобразуется в соответствующий код в RADIX-50. Коды RADIX-50 приведены ниже.

Символ	Код RADIX-50 (восьмеричный)
(пробел)	0
A — Z	1—32
⊙	33
.	34
резервируется	35
0—9	36—47

Для кода 35, который в настоящее время не используется, может быть определен еще один знак.

Запись трех знаков в коде RADIX-50 (с кодами C1, C2, C3) в слово производится следующим образом:

$$\text{ЗНАЧЕНИЕ} = = ((C1 * 50) + C2) * 50 + C3$$

Например, значение ABC в коде RADIX-50 есть $(1 * 50 + 2) * *50 + 3$ или 3223.

С помощью таблицы из приложения 5 можно быстро определить упакованное значение в коде RADIX-50.

Когда в строку текста нужно вставить специальные коды, рекомендуется применять угловые скобки, например:

```
        . ASCII   <101>                    ; ЭКВИВАЛЕНТНО . ASCII /A/
        . EVEN
        . RAD50   /AB/<35>                 ; ЗАПИСЫВАЕТ 3255 В СЛОВО
CHR1    =1
CHR2    =2
CHR3    =3
        . RAD50   <CHR1><CHR2><CHR3>       ; ЭКВИВАЛЕНТНО . RAD50 /ABC/
```

4.1.7. ОПЕРАТОР ВРЕМЕННОГО УКАЗАНИЯ КОДА RADIX-50

Оператор ↑R указывает, что аргумент преобразовывается в формат RADIX-50. Он позволяет три символа запомнить в одном слове. Оператор ↑R имеет вид

↑RCCC

где CCC — не более трех символов, которые должны быть преобразованы в 16-битное значение.

Если указано больше трех символов, любой символ, следующий за третьим символом, игнорируется. Если указано менее трех

символов, то предполагается, что последние символы — пробелы. Следующий пример показывает использование оператора для записей спецификации трехсимвольного типа файла в 16-битное слово со знаком:

```
MOV #↑RFOR, FILTYP
```

Знак # используется как индикатор непосредственной адресации, т. е. данные транслируются непосредственно в объектный код. ↑R указывает, что символы FOR преобразуются в RADIX-50.

4.1.8. УПРАВЛЕНИЕ ВНУТРЕННИМ ПРЕДСТАВЛЕНИЕМ ЧИСЕЛ

Значения чисел или выражений, встречающиеся в исходных программах, по умолчанию рассматриваются как восьмеричные. Однако часто бывает необходимо временное изменение основания системы счисления в программе или для переменных внутри одного оператора. Например, при объявлении основания системы счисления применительно ко всей программе необходимо, чтобы отдельные числа или выражения интерпретировались в процессе трансляции как двоичные, восьмеричные или десятичные. Иногда полезно бывает получить дополнения значения числа или выражения. Эти возможности языка описаны ниже.

Числа, используемые в исходной программе, первоначально рассматриваются как восьмеричные числа. Однако программист может выбрать также основания по умолчанию, равные 2, 8 и 10. Выбор основания делается с помощью директивы .RADIX, имеющей вид

```
.RADIX N
```

где N — одно из допустимых оснований, приведенных выше.

Аргумент директивы .RADIX интерпретируется всегда в десятичном основании.

Если в начале программы директива указания основания отсутствует или в директиве нет аргумента, то принимается восьмеричное основание. Любое основание, объявленное в исходной программе с помощью директивы .RADIX, остается в силе, пока не будет изменено другой директивой .RADIX. Например:

```
.RADIX 10      ; ОБЪЯВЛЯЕТ ДЕСЯТИЧНОЕ ОСНОВАНИЕ
.RADIX 8       ; ВОЗВРАЩАЕТ К ВОСЬМЕРИЧНОМУ
               ; ОСНОВАНИЮ
```

Любые значения, отличные от 2, 8 и 10, указанные как аргументы в директиве .RADIX, вызывают флаг ошибки A в листинге трансляции.

Если указано основание для какого-либо участка программы или решено использовать по умолчанию восьмеричное основание, то можно обнаружить ряд случаев, когда более удобным яв-

ляется переменное основание (особенно в макроопределениях). Например, маску лучше всего формировать с двоичным основанием.

В макроассемблере имеются три унарные операции, обеспечивающие при заданном в программе основании системы счисления интерпретацию числа в другом основании, как показано ниже:

↑DX — число X интерпретируется по основанию 10;
↑OX — число X интерпретируется по основанию 8;
↑BX — число X интерпретируется по основанию 2.

Таким образом, чтобы удовлетворить локальное требование исходной программы, следует временно объявить другое основание. Такое объявление может быть сделано в любое время независимо от того, предполагалось ли по умолчанию восьмеричное основание или в данном месте программы было объявлено другое основание. Другими словами, область действия оператора временного указания основания ограничена термом или выражением, непосредственно следующим за оператором. Любая величина, заданная в операторе временного указания основания, вычисляется в процессе трансляции как 16-битная величина.

Если две или более унарные операции появляются вместе, изменяя один и тот же терм, то операции применяются к терму справа налево.

Ниже приведены примеры написания операторов временного указания основания:

```
↑D127
↑O14
↑B00100100
↑O <B+2>
```

Следует обратить внимание на то, что знак ↑ и буква, указывающая основание, не могут быть разделены, а директива указания основания .RADIX может быть отделена от числа, указывающего основание, с помощью пробелов или табуляции (это делается для форматирования). Если терм или выражение должны интерпретироваться по другому основанию, его необходимо заключить в угловые скобки, как показано в последнем приведенном примере.

Следующий пример показывает использование угловых скобок для ограничения выражения, которое должно будет интерпретироваться с другим основанием:

```
.RADIX    10
    A  =12
.WORD    ↑O<A+10> *10
```

Вычисление выражения в данной директиве .WORD с учетом оператора указания временного основания будет эквивалентно оператору

```
.WORD 310
```

Временные указания основания могут быть размещены в любом месте программы, где используются числовые значения.

В языке восьмеричное основание может быть временно преобразовано в десятичное. Для этого число с десятичным основанием указывается с «десятичной точкой», например:

```
144.    ;ЭТИ ЧИСЛА РАССМАТРИВАЮТСЯ В ДЕСЯТИЧНОМ
1515.   ;ОСНОВАНИИ НЕЗАВИСИМО ОТ ЗАДАННОГО В ПРОГРАММЕ
120.    ;ОСНОВАНИЯ
```

Предыдущие примеры эквивалентны следующей записи:

```
↑D 144
↑D 1515
↑D 120
```

4.1.9. УПРАВЛЕНИЕ ФОРМАТОМ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Имеются несколько директив, обеспечивающих внутреннее представление числа с плавающей запятой (запись числа в исходной программе — это строка десятичных знаков). В строке (она может состоять и из одной цифры) может содержаться десятичная точка, а за ней следовать указатель порядка — буква «Е» и десятичный показатель степени со знаком.

Числа не должны содержать пробела, знаков табуляции или угловых скобок и не должны быть выражениями. Строка, имеющая недопустимые знаки, отмечается в листинге трансляции одним или несколькими флагами ошибок A или Q. Ниже приводятся семь различных правильных представлений одного и того же числа с плавающей запятой:

```
5
5.
5.0
5.0E0
5E0
.5E1
500E — 2
```

Очевидно, что этот список может быть продолжен (например, 5000E-3, .05E2 и т. д.). Знак «плюс», стоящий впереди, может опускаться (т. е. 5.0 рассматривается как +5.0). Стоящий впереди знак «минус» меняет знак числа. Другие унарные и (или) бинарные операторы не разрешены (например, 5.0+N — неправильно).

Числа с плавающей запятой вычисляются как 64-разрядные числа в формате: разряды 0—54 (55 бит) — мантисса, разряды 55—62 (8 бит) — экспонента, разряд 63 (1 бит) — знак мантиссы.

Транслятор размещает числа в соответствии с размером и точностью, задаваемыми директивами работы с плавающей запятой.

При представлении чисел с плавающей запятой во внутреннем формате числа обычно округляются. Это значит, что, если внутреннее представление числа с плавающей запятой выходит за пределы поля, в котором оно должно быть записано, старший бит

невместившейся части прибавляется к младшему биту размещенной части числа. Например, если число должно быть записано в двухсловном поле, а его значение не умещается в 32 разряда, то старший разряд несохраняемой части прибавляется к младшему биту (0) размещенной части.

Для обеспечения возможности усечения чисел с плавающей запятой применяется директива .ENABL FPT. Директива .DSABL FPT возвращает транслятор к округлению чисел с плавающей запятой.

Имеются две директивы для записи чисел с плавающей запятой во внутреннем представлении. Эти директивы аналогичны директиве .WORD и имеют вид:

```
.FLT2 ARG1, ARG2,...
.FLT4 ARG1, ARG2,...
```

где ARG1, ARG2, ... — одно или несколько чисел с плавающей запятой;
.FLT2 дает представление каждого аргумента в двух словах (32 разряда);
.FLT4 дает представление каждого аргумента в четырех словах — представление с двойной точностью (64 разряда).

4.1.10. ВРЕМЕННОЕ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ЧИСЕЛ

Подобно операциям временного указания основания системы счисления в макроассемблере имеются две унарные операции временного указания вида внутреннего представления чисел ↑C и ↑F.

Операция ↑C обеспечивает вычисление в процессе трансляции обратного кода для заданного аргумента. Эта операция может использоваться для любого допустимого выражения в любом месте программы. Прежде чем строить дополнение, макроассемблер вычисляет выражение как 16-разрядную двоичную величину, например

```
TMP: .WORD ↑C172
```

Результатом операции ↑C будет дополнительный код числа 172_8, т. е. число 177605_8.

Поскольку ↑C является унарной операцией, то операция и ее аргумент рассматриваются как терм. К единичному терму может быть применено более одной унарной операции, например конструкция вида ↑C↑O25 вычисляет в процессе трансляции дополнительный код числа 25. Он равен 177752_8.

Терм со знаком унарной операции может использоваться самостоятельно или в выражении. Например, конструкция вида ↑C7+15 эквивалентна <↑C7>+15.

Эти выражения вычисляются в процессе трансляции как дополнительный код числа 7 плюс абсолютное значение 15. При сложении этих термов происходит перенос за старший значащий разряд. Таким образом, значение выражения равно 000005 (восьмеричное).

При использовании внутри выражения термов со знаком операции временного указания представления чисел рекомендуется записывать такие термы в угловых скобках, что обеспечивает большую наглядность.

Другая унарная операция временного указания внутреннего представления чисел ↑F обеспечивает представление чисел с плавающей запятой в однословном формате. Такие числа обрабатываются программным способом.

Формат однословного представления чисел с плавающей запятой: разряды 0—6 (7 бит) — мантисса, разряды 7—14 (8 бит) — порядок, разряд 15 (1 бит) — знак мантиссы. Например, при трансляции команды

```
ABBA: CMP #↑F 3.5,R0
```

операция ↑F даст значение числа 3.5 в однословном формате. Оно будет размещено по адресу ABBA+2.

В следующем примере показаны результаты операций:

```
  ↑F    1.0-040200
  ↑F  -1.0-140200
- ↑F    1.0-137600
- ↑F  -1.0-037600
```

Значение, полученное как результат применения унарной операции ↑F к ее аргументу, является также термом, который может использоваться самостоятельно или в выражении. В этих случаях рекомендуется использовать угловые скобки; выражения, применяемые в качестве термов или аргументов унарных операций, должны быть явно сгруппированы, например

```
↑C↑F3.73
```

эквивалентно

```
^C<↑F3.73>
```

4.1.11. ДИРЕКТИВА .PACKED

Директива .PACKED позволяет запоминать целые десятичные числа в упакованном формате. Две десятичные цифры упаковываются в один байт. Арифметические действия над упакованными данными аналогичны операциям над числовыми строками. Директива имеет формат

```
.PACKED  десятичные-цифры [,SYM]
```

где десятичные цифры — десятичное число, содержащее от 0 до 31 цифры. Каждая цифра должна быть в пределах от 0 до 9. Число может иметь знак;

SYM — символ, которому, если он указан, присваивается значение, равное числу цифр в десятичном числе, без учета знака.

Примеры использования директивы .PACKED:

```
.PACKED -12,PCKD        ; PCKD=2
.PACKED +391,LEN        ; ЗАПИСЫВАЕТ ЧИСЛО 391, LEN=3
.PACKED 923E+03         ; ОШИБКА-E НЕДОПУСТИМО
```

4.1.12. ДИРЕКТИВА УПРАВЛЕНИЯ СЧЕТЧИКОМ АДРЕСОВ

Язык Макро требует, чтобы некоторые директивы и инструкции транслировались по четному адресу. Однако несколько директив макроассемблера могут привести к нечетному значению счетчика адресов. Это директивы .BYTE, .BLKB, .ASCII или .ASCIZ и оператор прямого присваивания вида .=.+ВЫРАЖЕНИЕ.

В этом случае директива .EVEN приводит к автоматической настройке счетчика адресов на четный адрес. Если эту директиву не использовать, возникает ошибка, отмечаемая флагом B. Однако транслятор проводит выравнивание счетчика адресов, как бы выполняя отсутствующую директиву.

Директива .EVEN обеспечивает четность счетчика адресов программы следующим образом: если содержимое счетчика нечетно, она прибавляет к нему единицу и оставляет содержимое без изменения в случае его четности. Директива не требует операндов. Появление любого аргумента в директиве .EVEN приведет к появлению в распечатке трансляции флага ошибки Q, например:

```
XY1: .ASCII   /номер/     ; НЕЧЕТНОЕ ЧИСЛО СИМВОЛОВ

     .EVEN                ; ДИРЕКТИВА РАЗМЕСТИТ ЗНАЧЕНИЕ
     .WORD XY1            ; XY1 С АДРЕСА НА ГРАНИЦЕ СЛОВА
```

Функцию, обратную функции директивы .EVEN, выполняет директива .ODD. Директива .ODD обеспечивает нечетное значение счетчика адресов программы: если значение счетчика четное, добавляет к нему единицу; если значение нечетное, то оставляет его без изменения. Директива .ODD не требует операндов. Любой аргумент в директиве .ODD вызывает появление в распечатке трансляции флага ошибки Q.

Блоки памяти в объектной программе могут быть зарезервированы с помощью директив .BLKB и .BLKW. Директива .BLKB предназначена для резервирования последовательности байтов, а .BLKW — резервирования последовательности слов.

Директивы имеют формат:

```
.BLKB     ВЫРАЖЕНИЕ
.BLKW     ВЫРАЖЕНИЕ
```

В этих директивах в качестве аргумента может стоять любое допустимое выражение, полностью определенное во время трансляции и получающее абсолютное значение. Значение выражения указывает число резервируемых байтов или слов. Если выражение, указанное в директиве этого типа, не является абсолютным, то в распечатке трансляции выдается флаг ошибки A. Использование этих директив без аргументов не рекомендуется. Однако, если аргумент отсутствует, предполагается значение, равное 1.

Директива .BLKB производит такое же действие, как и оператор присваивания .=.+ВЫРАЖЕНИЕ, но ее легче интерпретировать в контексте исходной программы.

Примечание. Следует отметить, что эти директивы только резервируют области памяти. Начальные значения в отличие от директив .BYTE и .WORD не присваиваются.

4.1.13. ДИРЕКТИВА ГРАНИЦ ПРОГРАММЫ

Часто необходимо знать верхний и нижний граничные адреса загрузочного модуля. Если в исходной программе указана директива .LIMIT, транслятор резервирует два слова памяти в объектном модуле.

Позже в процессе компоновки младший адрес программы помещается в первое зарезервированное слово, а адрес первого, следующего за загрузочным модулем, свободного слова оперативной памяти помещается во второе слово, зарезервированное по директиве .LIMIT. При компоновке размер модуля округляется с точностью до двух слов.

4.2. УСЛОВНАЯ ТРАНСЛЯЦИЯ

Директивы условной трансляции дают программисту возможность включать или не включать части исходного текста программы в процесс трансляции. Этот широко используемый способ позволяет формировать несколько вариантов объектного кода программы из одного текста исходной программы.

4.2.1. ДИРЕКТИВА БЛОКА УСЛОВНОЙ ТРАНСЛЯЦИИ

Общий формат блока условной трансляции:

```
.IF  CND, ARG      ; ЗАГОЛОВОК БЛОКА
                   ; ТЕЛО УСЛОВНОГО БЛОКА
.ENDC              ; КОНЕЦ УСЛОВНОГО БЛОКА
```

где CND — условие, которое должно быть выполнено, чтобы блок транслировался; условия, допустимые для директив условной трансляции, приведены в табл. 4.1;

, (запятая) — разделитель, допускается также пробел, и (или) знак табуляции;

ARG — аргументы, для которых проверяется условие (см. табл. 4.1);

тело условного блока — исходный текст программы, который транслируется или нет в зависимости от выполнения условия;

.ENDC — директива, завершающая блок условной трансляции. Эта директива должна быть поставлена в конце условного блока. Условия, отличные от указанных в табл. 4.1, недопустимые аргументы или отсутствие аргумента в директиве .IF вызывают в распечатке трансляции флаг ошибки A.

Примечание. Аргумент макрокоманды должен быть заключен в угловые скобки, например <A, B, C>, или в конструкцию со стрелкой, направленной вверх, например ↑/1234/.

Таблица 4.1

Обозначение условия		Аргумент	Блок транслируется при условии
основное	дополнительное		
EQ	NE	Выражение	Выражение = 0 (или ≠0)
GT	LE	Выражение	Выражение > 0 (или <=0)
LT	GE	Выражение	Выражение <0 (или >=0)
DF	NDF	Символический аргумент	Символ определен в программе (или не определен)
B	NB	Аргумент макрокоманды	Аргумент пропущен (или не пропущен)
IDN	DIF	Два аргумента макрокоманды	Аргументы идентичны (или различны)
Z	NZ	Выражение	Так же, как и EQ/NE
G	L	Выражение	Так же, как и GT/LE

Пример директивы условной трансляции:

```
.IF EQ CNT-3   ; ТРАНСЛИРОВАТЬ БЛОК, ЕСЛИ CNT-3
   ...
.ENDC
```

При условии DF и NDF для группировки символических аргументов используются два знака логических операций: & — знак логической операции «И», ! — знак логической операции «ИЛИ». Например:

```
.IF DF SYMV1&SYMV2
   ...
.ENDC
```

В данном примере трансляция условного блока выполняется, если оба аргумента определены.

Вложенные условные директивы имеют вид:

```
ДИРЕКТИВА УСЛОВНОЙ ТРАНСЛЯЦИИ
ДИРЕКТИВА УСЛОВНОЙ ТРАНСЛЯЦИИ
   ...
.ENDC
.ENDC
```

Например:

```
.IF DF SYMV1
.IF DF SYMV2
   ...
.ENDC
.ENDC
```

В данном примере предполагается, что если условие в первом (внешнем) операторе условной трансляции не выполняется, то ни один вложенный условный оператор проверяться не будет. Для лучшего восприятия программы рекомендуются строки, находящиеся внутри блока условной трансляции, выделить дополнительными пробелами:

```
.IF     DF SYMV1
        .IF DF SYMV2
            ...
        .ENDC
.ENDC
```

Каждый блок условной трансляции должен заканчиваться директивой .ENDC. Директива .ENDC, встретившаяся вне блока условной трансляции, дает в листинге трансляции флаг ошибки О. Макроассемблер допускает глубину вложения до 16_{10} блоков условной трансляции. Превышение допустимой глубины вложения условных блоков вызывает в листинге трансляции флаг ошибки О.

4.2.2. ПОДДИРЕКТИВЫ УСЛОВНОЙ ТРАНСЛЯЦИИ

Поддирективы условной трансляции могут быть помещены в блоки условной трансляции для того, чтобы разрешить трансляцию части кодов условного блока, если условие блока указывает, что блок не должен быть транслирован; трансляцию не смежных с заголовком кодов условного блока, если условие блока выполняется; безусловную трансляцию кодов внутри условного блока.

Если поддирективы условной трансляции появляются вне блока условной трансляции, то в распечатке трансляции возникает флаг ошибки О.

Ниже описываются три поддирективы условной трансляции:

.IFF — означает, что часть тела блока, следующая за данной поддирективой до следующей поддирективы условной трансляции или до конца блока, транслируется в случае, если условие, проверенное при входе в условный блок, было ложным;

.IFT — означает, что часть тела блока, следующая за данной поддирективой до следующей поддирективы условной трансляции или до конца блока, транслируется в случае, если условие, проверенное при входе в условный блок, было истинным;

.IFTF — означает, что часть тела блока, следующая за данной поддирективой, до следующей поддирективы условной трансляции или до конца блока, транслируется независимо от того, выполняется или нет условие, проверенное при входе в условный блок.

Подразумеваемым аргументом поддирективы условной трансляции является значение условия при входе в охватывающий блок условной трансляции. Поддирективы трансляции по условию проверяются всегда в блоках условной трансляции внешнего

уровня. Если условие внешнего блока не выполнено, поддирективы трансляции по условию игнорируются во вложенных блоках.

Пример 1. Использование поддиректив условной трансляции в условном блоке:

```
.IF NDF VAL      ; ТРАНСЛИРОВАТЬ БЛОК, ЕСЛИ VAL НЕ ОПРЕДЕЛЕНО
  ...
.IFF             ; ТРАНСЛИРОВАТЬ СЛЕДУЮЩИЕ КОДЫ, ТОЛЬКО
  ...            ; ЕСЛИ ОПРЕДЕЛЕНО VAL
.IFTF            ; ТРАНСЛИРОВАТЬ СЛЕДУЮЩИЕ КОДЫ БЕЗУСЛОВНО
.IFT             ; ТРАНСЛИРОВАТЬ ОСТАВШИЕСЯ КОДЫ, ЕСЛИ VAL
  ...            ; НЕ ОПРЕДЕЛЕНО
.ENDC
```

Трансляция поддиректив условной трансляции во вложенных блоках показана на примерах 2—4.

Пример 2. В данном примере символ X определен, символ Y не определен. В этом случае условие внешнего блока выполняется и поддирективы во вложенном условном блоке будут транслироваться.

```
.IF DF, X        ; УСЛОВИЕ ВНЕШНЕГО БЛОКА ИСТИННО
.IF DF, Y        ; Y НЕ ОПРЕДЕЛЕНО, УСЛОВИЕ ЛОЖНО
  ...            ; ЧАСТЬ ВЛОЖЕННОГО УСЛОВНОГО БЛОКА ДО
                 ; ДИРЕКТИВЫ .IFF НЕ ТРАНСЛИРУЕТСЯ
.IFF             ; ДИРЕКТИВА .IFF РАЗРЕШАЕТ ТРАНСЛЯЦИЮ КОДОВ
  ...            ; ВЛОЖЕННОГО УСЛОВНОГО БЛОКА
.IFT             ; ДИРЕКТИВА IFT ЗАПРЕЩАЕТ ТРАНСЛЯЦИЮ КОДОВ
  ...            ; ВЛОЖЕННОГО УСЛОВНОГО БЛОКА
.ENDC
.ENDC
```

Пример 3. Допустим, что символ A определен. Символ B не определен:

```
.IF DF A         ; "A" ОПРЕДЕЛЕНО. УСЛОВИЕ ИСТИННО,
  ...            ; СЛЕДУЮЩИЕ КОДЫ ТРАНСЛИРУЮТСЯ
.IFF             ; УСЛОВИЕ НА ВХОДЕ БЛОКА ВЫПОЛНЕНО,
  ...            ; СЛЕДУЮЩИЕ КОДЫ НЕ ТРАНСЛИРУЮТСЯ
.IF NDF B        ; ВЛОЖЕННАЯ УСЛОВНАЯ ДИРЕКТИВА ТРАНСЛИРУЕТСЯ
.ENDC
.ENDC
```

Пример 4. Предположим, что символ X не определен, а символ Y определен:

```
.IF DF X         ; УСЛОВИЕ ЛОЖНО, СИМВОЛ "X" НЕ ОПРЕДЕЛЕН
                 ; СЛЕДУЮЩИЕ КОДЫ НЕ ТРАНСЛИРУЮТСЯ
.IF DF Y         ; ВЛОЖЕННАЯ УСЛОВНАЯ ДИРЕКТИВА НЕ
                 ; ТРАНСЛИРУЕТСЯ
.IFF             ; ВЛОЖЕННАЯ ПОДДИРЕКТИВА УСЛОВНОЙ
  ...            ; ТРАНСЛЯЦИИ НЕ РАССМАТРИВАЕТСЯ
.IFT             ; ВЛОЖЕННАЯ ПОДДИРЕКТИВА УСЛОВНОЙ
  ...            ; ТРАНСЛЯЦИИ НЕ РАССМАТРИВАЕТСЯ
.ENDC
.ENDC
```

4.2.3. ДИРЕКТИВА НЕПОСРЕДСТВЕННОЙ УСЛОВНОЙ ТРАНСЛЯЦИИ

Директива непосредственной условной трансляции является средством записи однострочного условного блока. Для этой формы условных блоков оператор .ENDC не требуется. Условие записывается в одной строке с оператором, являющимся телом блока.

Директива непосредственной условной трансляции имеет формат:

.IIF CND, ARG, ОПЕРАТОР

где CND — одно из условий, определенных для условных блоков (см. табл. 4.1);

, (первая запятая) — разделитель, допускается также пробел и (или) знак табуляции;

ARG — аргумент, записанный в форме, допустимой для аргументов условных блоков, т. е. либо символ, либо аргумент макрокоманды (см. табл. 4.1);

ОПЕРАТОР представляет собой оператор, который должен транслироваться, если условие выполняется;

, (вторая запятая) — разделитель между условным аргументом и полем оператора. Если предшествующий аргумент является выражением, то должна быть использована только запятая, в противном случае это может быть запятая, пробел и (или) знак табуляции.

Например:

.IIF EQ GOO, BNE THAT

вызывает трансляцию инструкции

BNE THAT

если символ GOO определен в исходной программе и равен 0.

Как и для директивы .IF, условия, отличные от указанных в табл. 4.1, недопустимый аргумент или отсутствие аргумента в директиве приводят к флагу ошибки A в листинге трансляции.

4.3. ИДЕНТИФИКАЦИЯ МОДУЛЕЙ

Языком Макро предусмотрен набор директив, обеспечивающих именование и идентификацию программных модулей, листингов трансляции (в целях документирования) и результатов трансляции.

4.3.1. ДИРЕКТИВА .TITLE

Директива .TITLE используется для присвоения имени объектному модулю. Имя состоит из первых шести (или менее) отличных от пробела знаков, следующих за этой директивой. Все знаки имени должны быть допустимы в кодировке RADIX-50. Знаки, следующие за первыми шестью, проверяются на допустимость в символьном коде, но не используются макроассемблером в имени объектного модуля.

Например, директива .TITLE EXAMPL присваивает объектному модулю имя EXAMPL (не следует путать это имя с именем файла, указанным в командной строке для транслятора). Имя объектного модуля записывается в карте загрузки, создаваемой компоновщиком. Это имя может использоваться также библиотекарем при поиске модуля в библиотеке.

Если оператор .TITLE отсутствует, макроассемблер по умолчанию присваивает объектному модулю имя .MAIN.

Все знаки табуляции и (или) пробелы, встречающиеся до первого знака, отличного от пробела-табуляции, за директивой .TITLE игнорируются.

Если в исходной программе имеется более одной директивы .TITLE, последняя из них дает имя объектному модулю.

Если директива .TITLE указана без имени объектного модуля или первый знак в имени объектного модуля, отличный от пробела или табуляции, не является допустимым для кода RADIX-50, то в листинге трансляции директива отмечается флагом ошибки A. Кодировка в RADIX-50 приведена в приложении 5.

4.3.2. ДИРЕКТИВА .SBTTL

Директива .SBTTL используется для задания строки информационного сообщения, распечатываемого во второй строке заголовка страницы листинга. Текст, следующий за директивой .SBTTL, повторяется на всех страницах листинга до тех пор, пока не встретится следующая директива .SBTTL, например

```
.SBTTL SUBROUTINE
```

Текст печатается во второй строке каждой из последующих страниц распечатки программы.

Макроассемблер строит из текстов директив .SBTTL таблицу оглавления листинга. Во время первого прохода трансляции макроассемблер печатает оглавление листинга, в котором содержатся номер страницы, порядковый номер строки и текст каждой директивы .SBTTL в программе.

Вывод оглавления запрещается путем указания ключа /N:TOC в спецификации файла листинга программы в командной строке трансляции или директивой .NLIST TOC в исходном тексте программы. Пример распечатки оглавления листинга для программы:

```
EXAMPL MACRO 14 — JUN — 86 19:32:07 PAGE1
        TABLE OF CONTENTS
    1—3 — ПРОВЕРКА ДИРЕКТИВ;
    1—7 — НАЧАЛО БЛОКА;
    1—20 — КОНЕЦ БЛОКА.
```

Текст оглавления берется из директивы .SBTTL, в первых двух столбцах указываются номер страницы и номер строки исходного текста, где встретилась директива .SBTTL.

4.3.3. ДИРЕКТИВА .IDENT

Директива является средством, позволяющим дополнительно именовать объектный модуль, получаемый в результате трансляции. В дополнение к названию, присвоенному объектному модулю директивой .TITLE, в директиве .IDENT может быть указана строка знаков (до шести знаков, допустимых в RADIX-50), задающая номер версии программы. Строка знаков в .IDENT должна быть записана в парных ограничителях. Эта директива имеет вид:

.IDENT /STRING/

где STRING состоит из шести или меньше допустимых знаков и задает номер версии программы.

Этот номер включается в список глобальных символов объектного модуля и печатается в карте загрузки в каталоге библиотеки.

Две косые черты // являются ограничительными знаками. Кроме этих знаков ограничителями могут быть любые печатные знаки, отличные от знака равенства (=), левых угловых скобок (<) или точки с запятой (;). Ограничительный символ не должен использоваться в строке. Если ограничительные знаки не составляют пару или если они являются недопустимыми, в листинге трансляции директива отмечается флагом ошибки A.
Например, по директиве

.IDENT /V01.29/

строка знаков «V01.29» вводится в коде RADIX-50 в список глобальных символов объектного модуля.

Номер версии объектного модуля указывается также в распечатке карты загрузки, создаваемой компоновщиком, и в распечатке карты библиотеки.

Если в данной программе встречается более одной директивы .IDENT, последняя из них определяет имя, являющееся частью идентификации объектного модуля.

4.3.4. ДИРЕКТИВА .END

Директива .END указывает на логический конец данного модуля. Формат директивы

.END [ВЫРАЖЕНИЕ]

где ВЫРАЖЕНИЕ — необязательный аргумент, указывающий стартовый адрес модуля.

По директиве .END транслятор заканчивает текущий проход трансляции. Дополнительный текст, который может появиться за директивой .END в исходном файле, будет игнорироваться.

Если загрузочный модуль создается из нескольких объектных модулей, то только единственный объектный модуль может заканчиваться директивой .END ВЫРАЖЕНИЕ, указывающей стартовый адрес задачи. Все другие объектные модули должны

заканчиваться директивой .END без аргумента. В противном случае компоновщик будет выдавать сообщение об ошибке. Если ни в одном объектном модуле адрес запуска не указан, то выполнение задачи будет начинаться с адреса, равного 1, в результате чего немедленно возникает ошибка по нечетности адресации.

Директива .END не может быть использована внутри макрорасширения или условного блока трансляции; нарушение условия использования отмечается в распечатке трансляции флагом ошибки Q. Директива .END может быть использована, однако, в однострочном операторе условной трансляции.

Если исходная программа не заканчивается директивой .END, то в распечатке трансляции появляется код ошибки E.

4.4. СЕКЦИОНИРОВАНИЕ ПРОГРАММ

Директивы (.PSECT, .CSECT, .ASECT) секционирования программ используются для объявления секций и имен секций программ, используемых при компоновке.

4.4.1. ДИРЕКТИВА .PSECT

Директива .PSECT обеспечивает управление распределением памяти программы на этапе компоновки задачи с помощью аргументов этой директивы, передаваемых компоновщику.

Программист может указать, что данная программная секция состоит только из инструкций или содержит только данные. Программная секция может быть объявлена доступной только для чтения, что обеспечивает защиту повторно входимой секции. С помощью аргумента GBL явно указывается размещение программной секции в структуре перекрытий. Директива .PSECT имеет формат

.PSECT ИМЯ, ARG1, ..., ARGN

где ИМЯ — это символическое имя программной секции;

, (запятая) — разделитель; допускается также знак табуляции и (или) пробел;

ARG1, ..., ARGN — один или несколько допустимых символических аргументов, описанных ниже. Аргументы должны быть разграничены допустимыми разделительными знаками. Любой символический аргумент, отличный от описанных, вызовет в листинге трансляций флаг ошибки A.

Косая черта, разграничивающая каждую пару символических аргументов, указывает, что эти аргументы взаимоисключающие, т. е. может указываться один или другой аргумент, но не оба сразу.

Символические аргументы директивы

ИМЯ (по умолчанию пробел) указывает имя секции программы. Оно может состоять не более чем из шести знаков, допустимых кодом RADIX-50. Если этот аргумент опущен, то вместо этого параметра ставится запятая.

RO/RW (по умолчанию RW) — определяет разрешенный тип доступа к программной секции: RO — только чтение, RW — чтение/запись.

I/D (по умолчанию I) — указывает, что программная секция содержит только инструкции I (секция чистого кода) либо данные D. Этот аргумент позволяет компоновщику отличать глобальные символы, являющиеся точками входов программной секции I, от глобальных символов — значений данных (D).

Если программная секция имеет атрибут I, а программа (задача) имеет оверлейную структуру (задача с перекрытиями), то все вызовы этой секции преобразуются в ссылки на драйвер перекрытий, размещенный в резидентной части задачи. Если секция с атрибутом I является конкатенируемой (атрибут CON), то конкатенация происходит по четному адресу (т. е. размер секции в байтах определяется до четности).

Если программная секция имеет атрибут D, то все ее вызовы обрабатываются «напрямую». Размер конкатенируемой секции с атрибутом D никогда не округляется.

GBL/LCL (по умолчанию LCL) — определяет область размещения программной секции; этот признак впоследствии интерпретируется компоновщиком. При построении односегментных (неоверлейных) программ аргументы GBL/LCL не используются компоновщиком, так как вся программа является резидентной.

Аргументы GBL/LCL используются только в случае построения перекрытий.

Если объектный модуль содержит локальную программную секцию, то память для этой секции будет выделяться в сегменте, содержащем этот модуль. Если несколько модулей одного сегмента содержат объявления одной и той же локальной программной секции, то выделяемый для такой секции объем памяти определяется либо конкатенацией секций, либо наложением внутри сегмента в зависимости от аргумента CON/OVR в .PSECT.

Если объектный модуль содержит глобальную программную секцию, то память для такой секции выделяется только один раз и только в том сегменте, где встретилось первое объявление этой глобальной секции.

В разных ОС могут существовать отличия в интерпретации атрибутов GBL/LCL.

ABS/REL (по умолчанию REL) — определяет признак перемещаемости программной секции.

ABS — признак абсолютной (неперемещаемой) секции. Аргумент ABS указывает компоновщику, что секция абсолютная, т. е. не требует перемещения; такая программная секция транслируется и загружается с абсолютного адреса 0.

В абсолютной секции данные должны размещаться внутри границ сегмента, содержащего секции программы. В противном случае при компоновке выдается сообщение об ошибке. Например, коды

```
.PSECT ABCD, ABS
.=+100000
.WORD X
```

не вызывают ошибки при трансляции. Однако компоновщик будет выдавать сообщение об ошибке, если адрес размещения символа X, определенный при компоновке, окажется вне границ соответствующего сегмента.

Если указан аргумент REL (признак перемещаемой секции), то компоновщик вычисляет смещение и прибавляет его ко всем ссылкам программной секции, т. е. ссылки в секции программы должны быть настроены на физический адрес секции.

CON/OVR (по умолчанию CON) — определяет размещение программной секции.

CON — конкатенация. Программная секция строится как объединение всех объявленных программных секций с тем же именем в данной. Таким образом, объем памяти, выделяемой для программной секции, равен суммарной длине всех ее вхождений.

OVR — перекрытие. Программная секция строится как наложение всех программных секций с тем же именем в данную область памяти. Таким образом, объем памяти, выделяемый для программной секции, равен длине самой большой программной секции с тем же именем.

Примечание. Не следует путать перекрытие сегментов и перекрытие секций, для которых указан аргумент OVR. Перекрывающиеся сегменты используют одну и ту же область памяти программы за счет их физического вызова из внешней памяти при обращении к ним. Перекрытие секций — совместное использование одной и той же памяти за счет отображения адресов данных, описанных в программных секциях.

SAV/NOSAV (по умолчанию NOSAV) — определяет способ выделения памяти. Если используется атрибут SAV, то программная секция всегда будет размещена в резидентной части задачи. В противном случае используются обычные правила выделения памяти.

Только аргумент ИМЯ в директиве .PSECT должен занимать строго определенную позицию. Если он опушен, то позиция этого аргумента должна быть отделена запятой. Например, директива .PSECT, GBL задает аргумент ИМЯ пробелом и аргумент GBL.

Значения других аргументов в случае их отсутствия выбираются по умолчанию.

Свойства программной секции объявляются первой директивой .PSECT; транслятор предполагает, что эти свойства остаются действительными для всех последующих объявлений этой же программной секции, встречающихся в данном модуле.

Макроассемблер позволяет иметь до 256 программных секций, из них одну абсолютную программную секцию, имя которой по умолчанию .ABS.; одну неименованную секцию и до 254 именованных программных секций.

Директива .PSECT позволяет создавать программные секции и размещать инструкции и данные в разных программных секциях. Для каждой подразумеваемой или явно заданной программной секции макроассемблер фиксирует следующую информацию:

имя программной секции, текущее значение счетчика адреса, конечное значение счетчика адреса, атрибуты программной секции, т. е. аргументы директивы .PSECT.

Макроассемблер всегда начинает трансляцию исходных операторов с неименованной программной секции и счетчиком адреса, равным 0, т. е. первым оператором исходной программы всегда подразумевается директива .PSECT.

Первое проявление директивы .PSECT с заданным именем предполагает установку текущего счетчика адреса в нуль. Действие этой директивы продолжается до тех пор, пока не будет указана директива, объявляющая другую программную секцию. Появление имени программной секции в последующих операторах .PSECT вызывает возобновление трансляции секции с тем счетчиком адреса, на котором закончилась трансляция предыдущего вхождения этой секции. Например:

```
        .PSECT          ; ОБЪЯВЛЕНИЕ НЕИМЕНОВАННОЙ ПЕРЕМЕЩАЕМОЙ
                        ;   ПРОГРАММНОЙ СЕКЦИИ
A:      .WORD   0       ; ТРАНСЛИРУЕТСЯ В ОТНОСИТЕЛЬНЫЕ
B:      .WORD   0       ;   АДРЕСА 0,  2,  4
C:      .WORD   0
        .PSECT  TWO     ; ОБЪЯВЛЕНИЕ ПЕРЕМЕЩАЕМОЙ ПРОГРАММНОЙ СЕКЦИИ
X:      .WORD   0       ; ТРАНСЛИРУЕТСЯ В ОТНОСИТЕЛЬНЫЕ АДРЕСА
Y:      .WORD   0       ;   0 И 2
        .PSECT          ; ВОЗВРАТ К НЕИМЕНОВАННОЙ ПРОГРАММНОЙ СЕКЦИИ
D:      .WORD   0       ; ТРАНСЛЯЦИЯ ПРОДОЛЖАЕТСЯ С ОТНОСИТЕЛЬНОГО
                        ;   АДРЕСА 6
```

Программная секция полностью определяется при первом объявлении. Новое объявление этой секции можно делать, указывая только ее имя. Например, директива

```
.PSECT SECOND, ABS, OVR
```

объявляет абсолютную программную секцию .SECOND. Новое объявление этой секции может быть сделано директивой .PSECT без аргументов:

```
.PSECT SECOND
```

Сохраняя счетчики адресов для каждой программной секции, макроассемблер обеспечивает возможность загрузки в последовательные адреса памяти операторов, размещенных в тексте исходной программы в различных местах, но в одних программных секциях. Например:

```
        .PSECT  SEGM1, REL      ; НАЧАЛО ПЕРЕМЕЩАЕМОЙ ПРОГРАММНОЙ
                                ;   СЕКЦИИ SEGM1
A:      .WORD   0               ; ТРАНСЛИРУЕТСЯ В ОТНОСИТЕЛЬНЫЕ
B:      .WORD   0               ;   АДРЕСА 0,  2  И 4
C:      .WORD   0
ST:     CLR     A               ; МАШИННЫЕ КОМАНДЫ, ТРАНСЛИРУЮТСЯ
        CLR     B               ;   В ОТНОСИТЕЛЬНЫЕ АДРЕСА 6 – 20
        CLR     C
        .PSECT  SEGM2, ABS      ; НАЧАЛО АБСОЛЮТНОЙ ПРОГРАММНОЙ СЕКЦИИ
                                ;   С ИМЕНЕМ SEGM2
        .WORD   .+2, A          ; ТРАНСЛИРУЕТСЯ В АБСОЛЮТНЫЕ АДРЕСА 0 И 2
```

```
.PSECT   SEGM1          ; ВОЗВРАТ К ОТНОСИТЕЛЬНОЙ ПРОГРАММНОЙ
                        ;   СЕКЦИИ SEGM1
INC      A              ; ТРАНСЛИРУЕТСЯ В ОТНОСИТЕЛЬНЫЕ АДРЕСА
                        ;   22 – 26
BR       ST             ; ПЕРЕХОД НА МЕТКУ ST
```

Все метки в абсолютной программной секции абсолютные. Соответственно все метки перемещаемой программной секции относительные. Символ текущего счетчика адреса (.) может быть относительным или абсолютным в зависимости от того, к какой секции программы он относится.

Метка перед директивой .PSECT (.ASECT, .CSECT) рассматривается как метка, принадлежащая секции, предшествующей директиве .PSECT (.ASECT, .CSECT). Ей присваивается значение текущего счетчика адреса предшествующей секции прежде, чем директива .PSECT (или .ASECT, .CSECT) будет действовать. Например, если первый оператор программы

```
A:  .PSECT ESC, REL
```

то метке A присваивается значение относительного нуля — значение счетчика адреса неименованной программной секции, с которой по умолчанию начинается модуль.

Так как во время трансляции не известно, куда будет загружена перемещаемая программная секция, то ссылки между перемещаемыми секциями транслируются в адреса относительно базы секций, содержащей имя, к которому делается ссылка.

Таким образом, макроассемблер передает компоновщику всю необходимую информацию для расчета взаимных ссылок между различными программными секциями во время компоновки задачи. Такая информация не требуется для абсолютной программной секции, так как все команды в такой секции транслируются в абсолютные адреса.

В приведенном ниже примере ссылки к именам X и Y транслируются в адреса относительно базы перемещаемой программной секции SEND:

```
        .PSECT   EXT, ABS
        . = .+1000
A:      CLR      X             ; X ТРАНСЛИРУЕТСЯ КАК <ОТНОСИТЕЛЬНАЯ
                               ;   БАЗА СЕКЦИИ SEND + 10>
        JMP      Y             ; Y ТРАНСЛИРУЕТСЯ КАК <ОТНОСИТЕЛЬНАЯ
                               ;   БАЗА СЕКЦИИ SEND + 6>
        .PSECT   SEND, REL
        MOV      R0, R1
        JMP      A             ; ЭТА КОМАНДА ТРАНСЛИРУЕТСЯ
                               ;   КАК <JMP 1000>
Y:      HALT
X:      .WORD    0
```

Использование в данном примере константы вместе с символом текущего счетчика адреса (.) в виде .=1000 привело бы к ошибке, потому что константы являются абсолютными и всегда связанными с .ASECT (с именем «.ABS»).

Именованные перемещаемые программные секции с аргументами GBL и OVR выполняют ту же роль, что и блоки COMMON в Фортране, т. е. одноименные секции различных программ с аргументами GBL и OVR компонуются по одному и тому же адресу. Программные секции с аргументом CON объединяются, т. е. компонуются в последовательные адреса.

Необходимо заметить, что не возникает конфликтов между внутренними символическими именами и именами программных секций, т. е. разрешается использовать одно и то же имя для той и другой цели подобно тому, как в Фортране допустима конструкция

```
COMMON /Y/ A, B, C, Y
```

где Y представляет собой имя блока COMMON и именует четвертый элемент данных блока.

4.4.2. ДИРЕКТИВЫ .SAVE И .RESTORE

Директивы .SAVE и .RESTORE позволяют сохранять и восстанавливать параметры текущей программной секции. Сохранение параметров, выполняемое директивой .SAVE, происходит в специальном стеке, организуемом макроассемблером в процессе трансляции. Размер стека обеспечивает хранение информации о 16 программных секциях. Директива .RESTORE восстанавливает значения параметров программной секции, сохраненных по последней директиве .SAVE. Формат директив

```
.SAVE
.RESTORE
```

Если стек параметров заполнен, то появление директивы .SAVE вызовет флаг ошибки A. Если использовать директиву .RESTORE когда стек пуст, это приведет к ошибке с флагом A.

Обычно данные директивы используются в макрокомандах. Например:

```
.MACRO TYPMES  TEXT
        .SAVE                       ; СОХРАНИТЬ ТЕКУЩУЮ СЕКЦИЮ
        .PSECT  @@@TXT, D, GBL
@@@     =.
        .ASCIZ  "TEXT"
        .RESTORE                    ; ВЕРНУТЬСЯ К ПРЕРВАННОЙ СЕКЦИИ
        MOV     #@@@, R0            ; ЗАПОМНИТЬ АДРЕС СООБЩЕНИЯ
        CALL    PRINT               ; НАПЕЧАТАТЬ ЕГО
.ENDM TYPMES
```

4.4.3. ДИРЕКТИВЫ .ASECT И .CSECT

В программах рекомендуется использовать только директиву .PSECT, так как она обеспечивает все возможности директив .ASECT и .CSECT. Однако макроассемблер допускает использо-

вание директив .ASECT и .CSECT со свойствами, определенными по умолчанию (табл. 4.2).

Таблица 4.2

Аргумент	Значения по умолчанию		
	.ASECT	.CSECT именованная	.CSECT неименованная
Имя	.ABS	Имя	Пробел
Доступ	RW	RW	RW
Тип	I	I	I
Область действия	GBL	GBL	LCL
Перемещаемость	ABS	REL	REL
Способ размещения	OVR	OVR	CON
Выделение памяти	NOSAV	NOSAV	NOSAV

Такая возможность обеспечивает совместимость между ассемблерными программами, написанными для различных операционных систем, включая ДОС СМ.

Допустимый формат директив .ASECT и .CSECT

```
.ASECT
.CSECT
.CSECT  СИМВОЛ
```

Например, оператор .CSECT BOB тождествен оператору .PSECT BOB, RW, I, REL, OVR, так как для именованной программной секции, объявляемой директивой .CSECT, будут выбраны значения аргументов по умолчанию — RW, I, GBL, REL, OVR.

4.4.4. ДИРЕКТИВА .GLOBL

Транслятор создает перемещаемый объектный модуль и файл распечатки, содержащий листинг программы и таблицу символов. Компоновщик объединяет отдельно оттранслированные объектные модули в загрузочный модуль программы. Объектные модули перемещаются в зависимости от базы размещения, определяемой для модуля.

Объектные модули связаны между собой через глобальные символы. На глобальный символ, определенный в одном модуле, можно ссылаться из других модулей. Все символы, на которые будут производиться ссылки из других модулей, должны быть объявлены глобальными в модулях, в которых они определяются.

Глобальный символ может быть определен через оператор присваивания == либо как глобальная метка ::, либо через директиву .GLOBL.

Директива .GLOBL используется для объявления глобальных символов, если они не определены другим способом. Например, если действует директива .DSABL GBL, то для компоновки мо-

дуля с программами какой-либо библиотеки имена этих программ должны быть объявлены глобальными в исходном тексте директивой .GLOBL.

Для объявления глобальности символов A, B, C директива

```
.GLOBL  A, B, C
```

эквивалентна операторам:

```
A==  ВЫРАЖЕНИЕ  ; или A::
B==  ВЫРАЖЕНИЕ  ; или B::
C==  ВЫРАЖЕНИЕ  ; или C::
```

Формат директивы

```
.GLOBL  SYMB1, SYMB2, ..., SYMBN
```

где SYMB — разрешенные символические имена, разделенные запятыми и (или) знаком табуляции.

Строка директивы .GLOBL может содержать метку в поле метки и комментарий в поле комментария.

В конце первого прохода транслятор устанавливает, определен ли данный символ в текущем программном модуле, иначе рассматривает его как внешний символ. Все локальные символы в данной программе должны быть определены к концу первого прохода, иначе по умолчанию они будут рассматриваться как глобальные ссылки.

Например, данная программа имеет две точки входа и вызывает внешнюю подпрограмму Z:

```
        .PSECT                  ; ОБЪЯВЛЯЕТ НЕИМЕНОВАННУЮ ПРОГРАММНУЮ
                                ; СЕКЦИЮ
        .GLOBL  A               ; ОПРЕДЕЛЯЕТ "A" КАК ГЛОБАЛЬНЫЙ СИМВОЛ
X:      MOV     @ (R5) +, R0
        MOV     #A, R1
A:      JSR     PC, Z           ; ВЫЗОВ ВНЕШНЕЙ ПОДПРОГРАММЫ Z
        RTS     R5              ; ВЫХОД ИЗ ДАННОЙ ПОДПРОГРАММЫ
Y::     MOV     @ (R5) +, R1
        CLR     R2
        BR      X
        .END
```

В данном примере A и Y являются входными символами. Символ A объявлен директивой .GLOBL глобальным. Символ Y определен как глобальная метка через двойное двоеточие ::. Так как символ Z не определен внутри текущего модуля, он является ссылкой к внешнему символу.

Внешние символы могут использоваться в поле операнда инструкции или директивы макроассемблера в виде прямой ссылки:

```
.WORD   EXTERN
CLR     EXTERN
CLR @   EXTERN
```

Они могут использоваться так же, как термы в выражениях, например:

```
.WORD   EXTERN+A
CLR     EXTERN+A
CLR  @  EXTERN+A
```

Нельзя использовать неопределенный внешний символ в операторах прямого присваивания или как аргумент в директивах условной трансляции.

Глобальным символам, которые определены в транслируемом модуле (т. е. встречаются в поле метки, ограниченной знаками ::, или в левой части оператора присваивания со знаками ==), транслятором присваиваются соответствующие числовые значения. Эти значения в дальнейшем могут быть откорректированы компоновщиком операционной системы.

Глобальным символом, встречающимся только в поле операндов транслируемых модулей, при трансляции присваиваются нулевые значения, которые также подлежат корректировке в процессе компоновки.

Указанный процесс вычисления фактических значений глобальных символов при компоновке модулей называется разрешением внешних ссылок.

4.4.5. ДИРЕКТИВА .WEAK

Директива .WEAK используется для определения символа как внешнего (т. е. определенного в другом модуле) или глобального (определенного в данном модуле и доступного в других модулях). Эта директива запрещает поиск указанных символов в библиотеках.

Если директивой .WEAK специфицирован символ, определенный в другом модуле, то символ рассматривается как глобальный. Если компоновщик находит определение этого символа в другом модуле, то использует найденное определение. Если это определение не будет найдено, символ получит значение 0. Компоновщик не занимается поисками такого символа в библиотеке, но если модуль, извлеченный из библиотеки по другим причинам, содержит его определение, компоновщик использует это определение.

Однако, если данный модуль включается в объектную библиотеку, символ, указанный в директиве .WEAK, в каталог библиотеки не попадает. Поэтому при поиске в каталоге по такому имени модуль найден не будет.

Формат директивы .WEAK совпадает с форматом .GLOBL

```
.WEAK A1, S2, ABC
```

Строка с директивой .WEAK может также содержать метку и комментарий.

5

ГЛАВА

ДИРЕКТИВЫ МАКРОКОМАНД

При программировании на языке макроассемблера можно формировать обращение к часто повторяющейся последовательности команд при помощи одного оператора. Для того чтобы обеспечить возможность такого обращения, последовательность команд объявляется первоначально с использованием формальных аргументов как макроопределение.

Формально описанная последовательность команд называется *макроопределением*. Макроопределение и команда обращения к макроопределению (вызов макрокоманды) определяют макрорасширение.

Вызов макрокоманды содержит фактические аргументы, которыми заменяются соответствующие формальные аргументы в теле макроопределения.

5.1. ОПРЕДЕЛЕНИЕ И ВЫЗОВ МАКРОКОМАНД

Первым оператором макроопределения должна быть директива .MACRO, имеющая формат

LAB: .MACRO NAME, FLIST

где LAB — необязательная метка оператора;

NAME — символическое имя макрокоманды; этим именем может быть любой разрешенный символ. Выбранное имя может использоваться и как метка где-либо в программе, что не приведет к конфликтной ситуации с учетом алгоритма поиска по таблицам. Разделителем между именем и списком аргументов может быть любой разрешенный разделитель: запятая, пробел и(или) знак табуляции;

FLIST — любые разрешенные символы, которые могут появиться где-либо в теле макроопределения.

Символ, применяемый как формальный аргумент, может использоваться также в качестве метки в программе пользователя, что не вызывает конфликта с макроопределением. Формальные аргументы разделяются любым разрешенным разделителем (обычно запятой). Макроопределение может не иметь формальных аргументов.

В операторе, содержащем директиву .MACRO, за списком формальных аргументов может следовать комментарий. Например:

```
.MACRO ADDR A,B ; ОПРЕДЕЛИТЬ МАКРОКОМАНДУ С ИМЕНЕМ ADDR
                ; И С ДВУМЯ АРГУМЕНТАМИ ''A'' И ''B''
```

Хотя метка допустима в строке с директивой .MACRO, такая конструкция нежелательна, особенно при использовании вложенных макроопределений, потому что недопустимые метки или метки, построенные конкатенацией символов, будут приводить к игнорированию директивы .MACRO, расположенной на этой строке. Это замечание относится также к директивам .IRP, IRPC, .REPT.

Последним оператором каждого макроопределения должна быть директива .ENDM, имеющая формат

```
.ENDM NAME
```

где NAME — имя макрокоманды, макроопределение которой заканчивается директивой .ENDM; NAME — необязательный аргумент, например:

```
.ENDM       ; ЗАКАНЧИВАЕТ ТЕКУЩЕЕ МАКРООПРЕДЕЛЕНИЕ
.ENDM ADDR  ; ЗАКАНЧИВАЕТ МАКРООПРЕДЕЛЕНИЕ
              МАКРОКОМАНДЫ ADDR
```

Если в операторе .ENDM указано имя, оно должно соответствовать имени макрокоманды, определенному в операторе .MACRO. В противном случае этот оператор порождает в листинге трансляции флаг ошибки A. В любом случае .ENDM завершает текущее макроопределение. Спецификация имени макрокоманд в операторе .ENDM позволяет транслятору обнаружить отсутствие операторов .ENDM во вложенных макроопределениях или неправильно вложенные макроопределения.

Оператор .ENDM может содержать поле комментария, но не должен содержать метку. Метки у директивы .ENDM игнорируются, а ошибки в поле метки будут вызывать игнорирование директивы. Пример макроопределения:

```
. MACRO   UNIT
          MOV     @#JOCH, R1
          MOV     #BUFF, R3
          CALL    KODE          ; БЛОК КОДИРОВКИ И
          CALL    PRINT         ; ПЕЧАТИ
          . ENDM  UNIT
```

Если макроассемблер встретил директиву .ENDM вне макроопределения, то в листинге трансляции появляется флаг ошибки O.

5.1.1. ДИРЕКТИВА .MEXIT

Директива позволяет завершить генерацию макрорасширения раньше, чем достигнут конец макроопределения, и применяется для указания конца логической ветви макроопределения при исполь-

зовании директив условной трансляции. Эта директива допустима также в блоках повторения.

Наиболее полезной директива .MEXIT оказывается при работе с вложенными макроопределениями. Она завершает текущее макрорасширение подобно директиве .ENDM. Оставшаяся часть макроопределения игнорируется. В случае вложенных макроопределений директива .MEXIT вызывает переход к макрорасширению более высокого уровня. В макроопределении может быть несколько директив .MEXIT.

.MEXIT вне макроопределения и блоков повторений вызывает в листинге трансляции флаг ошибки О. Например:

```
.MACRO   ALTR    N,A,B
         ...
.IF  EQ, N
         CALL    ABORT          ; НАЧАЛО УСЛОВНОГО БЛОКА
         .MEXIT                 ; ВЫЗОВ ПОДПРОГРАММЫ ЗАВЕРШАЕТ
         ...                    ;  ГЕНЕРАЦИЮ МАКРОРАСШИРЕНИЯ
```

В программе, где формальный аргумент N принимает значение 0, директива .MEXIT заканчивает формирование макрорасширения.

5.1.2. ФОРМАТИРОВАНИЕ МАКРООПРЕДЕЛЕНИЙ

В макроопределениях могут быть использованы знак перевода формата <FF> и директива .PAGE.

Знак перевода формата в макроопределении вызывает перевод страницы листинга при обработке самого макроопределения. Перевод страницы не выполняется при построении макрорасширения.

Директива .PAGE игнорируется при просмотре самого макроопределения и вызывает перевод страницы листинга при распечатке построенного макрорасширения.

5.2. ВЫЗОВЫ МАКРОКОМАНД

Макрокоманда должна быть описана соответствующим макроопределением до первого обращения к ней. Вызов макрокоманды имеет вид:

```
LABEL: NAME LIST   ;КОММЕНТАРИЙ
```

где LABEL — метка; в вызове макрокоманды метки могут отсутствовать;

NAME — имя макрокоманды, указанное в директиве .MACRO соответствующего макроопределения. Между именем и списком аргументов может быть любой разрешенный разделитель (запятая, пробел или знак табуляции); разделитель не требуется, если нет аргументов;

LIST — символы, выражения и константы, которые заменяют формальные аргументы, использованные в директиве .MACRO. Разделяются с помощью любого разрешенного разделителя (обычно запятой).

Если имя макрокоманды одинаково с меткой, появление символа в поле операции интерпретируется транслятором как вызов макрокоманды, а появление символа в поле операнда интерпретируется как обращение к метке. Например:

```
ABC:  MOV R, R1        ; ABC ОПРЕДЕЛЯЕТСЯ КАК МЕТКА
      ...
      BR   ABC         ; ABC ИСПОЛЬЗУЕТСЯ КАК МЕТКА
      ...
      ABC#4, EMT, L    ; ВЫЗОВ МАКРОКОМАНДЫ ABC С ТРЕМЯ
                       ; АРГУМЕНТАМИ
```

Аргументы в вызове макрокоманды рассматриваются как строки знаков, использование которых определяется макроопределением.

5.3. АРГУМЕНТЫ МАКРОКОМАНД

В вызове макрокоманды для разделения аргументов может быть использован любой из разделителей, например

```
.MACRO FEN A B C          ; РАЗДЕЛИТЕЛЬ АРГУМЕНТОВ — ПРОБЕЛ
FEN ARG1, ARG2, <C1,C2>   ; РАЗДЕЛИТЕЛЬ АРГУМЕНТОВ — ЗАПЯТАЯ
```

Аргументы, которые содержат разделительные знаки, должны быть заключены в угловые скобки. Аргументы, заключенные в скобки, редко используются в макроопределениях, но чаще используются при вызовах макрокоманд. Например, макровызов

```
FEN <CMP X,Y>, #15, NEW
```

приведет к тому, что строка CMP X, Y заменит формальный аргумент A во всех случаях его появления в макроопределении. Использование конструкции со знаком ↑ обеспечивает возможность включения угловых скобок в строку знаков, составляющих фактический аргумент, например

```
FEN ↑/CMP X,Y/, #15, NEW
```

В данном примере в качестве первого аргумента передается строка <CMP X, Y>. Однако макровызов

```
FEN #15, NEW ↑/CMP X,Y/
```

содержит только два аргумента #15 и NEW↑/CMP X, Y/, так как ↑ является указателем операций с одним членом и ему не предшествует разделитель аргументов.

5.3.1. АРГУМЕНТЫ ВЛОЖЕННЫХ МАКРОКОМАНД И МАКРООПРЕДЕЛЕНИЙ

Макрокоманда называется вложенной, если обращение к этой макрокоманде (ее макровызов) содержится в макроопределении другой макрокоманды.

Глубина вложения макрокоманд зависит от размера динамической памяти, используемой при трансляции исходной программы. Аргументы макроопределения, передаваемые вложенным макрокомандам, должны быть заключены в угловые скобки на каждом уровне вложения.

При расширении вложенных макрокоманд производится удаление внешней пары угловых скобок из передаваемых аргументов на каждом уровне вложения. Угловые скобки поэтому желательно указывать в макроопределении, а не в вызове внешней макрокоманды.

Например, макрокоманда LEVEL2 вызывается в макроопределении LEVEL1 и формальный аргумент макроопределения, передаваемый в макровызове LEVEL2, заключается в угловые скобки.

```
.MACRO   LEVEL    DUM1, DUM2
         LEVEL2   <DUM1>
         LEVEL2   <DUM2>
         .ENDM
.MACRO   LEVEL2   DUM3
         DUM3
         CMP      #10.,   R3
         SUB      R3,  (R1)+
         .ENDM
```

Вызов макрокоманды

```
LEVEL <MOV X, R0>, <MOV R2. R0>
```

имеет аргументы, содержащие знаки-разделители, поэтому эти аргументы должны быть заключены в угловые скобки. Макрорасширение этой макрокоманды имеет вид:

```
MOV      X, R0
CMP      #10., R3
SUB      R3, (R1)+
MOV      R2, R0
CMP      #10., R3
SUB      R3, (R1)+
```

Макроопределение называется вложенным, если одно макроопределение полностью содержится внутри другого макроопределения.

Внутреннее макроопределение не определяет макрокоманду, если внешняя макрокоманда не была вызвана, например

```
.MACRO   LV1      A, B        ; ВНЕШНЕЕ МАКРООПРЕДЕЛЕНИЕ
         ...
.MACRO   LV2      C           ; ВНУТРЕННЕЕ МАКРООПРЕДЕЛЕНИЕ
         ...
         ...
.ENDM    LV2
.ENDM    LV1
```

Макрокоманда LV2 не может быть использована до вызова макрокоманды LV1. Аналогично любая макрокоманда, определенная внутри макроопределения LV2, не может быть использована до вызова макрокоманды LV2.

5.3.2. ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ЗНАКОВ В АРГУМЕНТАХ

Аргумент, содержащий специальные знаки, может не заключаться в угловые скобки, если среди специальных знаков нет пробела, знака табуляции, точки с запятой или запятой. Например, в случае

```
.MACRO   PUSH    ARG
         MOV     ARG, -(SP)
         ENDM
         PUSH    X+3(R2)
```

транслятор формирует следующий код макрорасширения для макровызова:

```
MOV  X+3(R2), -(SP)
```

5.3.3. СИМВОЛЬНОЕ ПРЕДСТАВЛЕНИЕ ЧИСЛОВОГО АРГУМЕНТА

При задании аргументов в вызове макрокоманды может быть использована возможность передачи в макрорасширение числового значения аргумента в символьном виде. Для этого перед соответствующим фактическим аргументом ставится при вызове знак «\» (обратная косая черта). Числовое значение аргумента берется в текущем основании. Такая возможность может быть использована для формирования символьной метки, например

```
. MACRO   DCL     A, B
          LABEL   A\B
          B=B+1
          . ENDM  DCL
. MACRO   LABEL   A, B
A'B:      . WORD  4
          . ENDM  LABEL'
...
          C=0
          DCL     X, C        ; ВЫЗОВ МАКРОКОМАНДЫ
```

Макрокоманда будет транслироваться в X0: .WORD 4.

В этом макрорасширении метка X0 является конкатенацией двух фактических аргументов. Знак апострофа (') в A'B приводит к конкатенации строк фактических аргументов X и C, даваемых в макрорасширение[1]. Для повторного вызова той же самой макрокоманды макрорасширение будет иметь вид:

```
X1:   .WORD 4
```

---

[1] Конкатенация аргументов подробно описана в 5.3.7.

Для дальнейших вызовов

```
XN:   .WORD 4
```

В данном примере необходимы два макроопределения, чтобы обеспечить возможность использования фактического значения аргумента B: как числа для вычисления арифметического выражения (в макрокоманде DCL) и как строки цифр в символьном коде для формирования метки (в макрокоманде LABEL).

В макрорасширение передается строка, состоящая только из значащих цифр числового представления значения аргумента. Если фактический аргумент равен 0, то в макрорасширение передается один знак 0.

Представление числового значения аргумента как строки символов может также использоваться для идентификации листинга исходной программы. Например, варианты программы, созданные посредством условной трансляции одного исходного текста программы, могут быть идентифицированы следующим образом:

```
.MACRO   OUT     ARG
         .IDENT  /V03B.'ARG/
         .ENDM   OUT
         I=3
         OUT     \I
```

Макрорасширение макрокоманды OUT дает

```
.IDENT /V03B.3/
```

5.3.4. ЧИСЛО АРГУМЕНТОВ В ВЫЗОВЕ МАКРОКОМАНДЫ

Если в вызове макрокоманды задано больше аргументов, чем в макроопределении, лишние аргументы игнорируются и в листинге трансляции появляется флаг ошибки Q. Если в вызове макрокоманды задано меньше аргументов, чем в макроопределении, отсутствующие аргументы получают значение ПУСТО (пустая строка). Для обнаружения недостающих аргументов в макрокоманде могут быть использованы условные директивы .IF B и .IF N B.

Число аргументов может быть определено также с помощью директивы .NARG. Возможно определение макрокоманд, не имеющих аргументов.

5.3.5. АВТОМАТИЧЕСКИ СОЗДАВАЕМЫЕ ЛОКАЛЬНЫЕ СИМВОЛЫ

Как отмечалось ранее, транслятор может автоматически создавать локальные символы вида N⊙, где N — десятичное целое в интервале от 30000 до 65535 включительно. Такие локальные символы создаются макроассемблером в порядке возрастания числа N:

```
30000⊙
30001⊙
  .
  .
  .
65534⊙
65535⊙
```

Автоматически создаваемые локальные символы особенно полезны, когда в макрорасширении необходима метка. Создание таких меток в макрорасширении позволяет, с одной стороны, уменьшить число аргументов в вызове макрокоманды (может быть опущен фактический аргумент, определяющий метку в макрорасширении); с другой стороны, позволяет избежать многократного определения меток, возможного при каждом использовании макрокоманды, если соответствующий фактический аргумент задан некорректно.

Область допустимости автоматически создаваемых локальных символов лежит между двумя явными метками. Каждая новая нелокальная метка объявляет начало нового блока локальных символов.

Локальные метки создаются транслятором при каждом вызове макрокоманды, макроопределение которой содержит формальный аргумент с предшествующим знаком вопроса ?, а соответствующий фактический аргумент в вызове макрокоманды либо отсутствует, либо пропущен. Если в вызове макрокоманды указан фактический аргумент, формирование локального символа запрещается и выполняется нормальная замена. Например:

```
.MACRO   BETA    A, ?B
         BIT     #100, A
         BNE     B
         SUB     #12, A
B:
         .ENDM   BETA
```

Рассмотрим макрорасширение для разных вызовов этой макрокоманды.

1. Формирование локального символа в случае отсутствия аргумента:

```
         BETA    R1          ; МАКРОВЫЗОВ СОДЕРЖИТ ОДИН АРГУМЕНТ
         BIT     #100, R1    ; МАКРОРАСШИРЕНИЕ ИМЕЕТ ВИД
         BNE     30000⊙
         SUB     #12,  R1
30000⊙:
```

2. Если локальный символ не формируется, макрорасширение выглядит следующим образом:

```
         BETA    R2, ABC     ; МАКРОВЫЗОВ СОДЕРЖИТ ОБА АРГУМЕНТА
         BIT     #100,  R2   ; В МАКРОРАСШИРЕНИИ СФОРМИРОВАНА
         BNE     ABC         ;  ОБЫЧНАЯ МЕТКА
         SUB     #12,  R2
ABC:
```

Возможность автоматического создания локальных символов предусматривается только для первых шестнадцати аргументов макроопределений. При автоматическом создании локальных символов в макрорасширении нежелательно использование директив .ENABL LSB и .DSABL LSB, так как изменение границ блока локальных символов может привести к появлению в листинге трансляции флага ошибки P.

5.3.6. КЛЮЧЕВЫЕ СЛОВА

Макрокоманды могут определяться и (или) вызываться с аргументами в формате ключевого слова. Синтаксис описания ключевого слова:

NAME=STRING

где NAME — ключевое слово (имя формального параметра);
STRING — значение фактического аргумента по умолчанию (фактический аргумент).

Описание ключевого слова не может содержать разграничителей вложенных аргументов, если только они разделены не по правилам.

Если описание ключевого слова появляется в списке формальных аргументов макроопределения, то указанная строка STRING становится значением по умолчанию аргумента NAME макроопределения.

Если аргумент содержится в списке фактических аргументов вызова макрокоманд в формате ключевого слова, указанная строка становится фактическим значением формального аргумента, имя которого точно совпадает с указанным независимо от того, был ли определен этот формальный аргумент в макроопределении в формате ключевого слова. Если имя не совпадает, вся спецификация воспринимается как следующий позиционный фактический аргумент.

Аргумент в формате ключевого слова может указываться в любом месте списка аргументов макроопределения. Он является частью позиционного расположения аргументов. С другой стороны, аргумент в форме ключевого слова может указываться где-либо в списке фактических аргументов вызова макрокоманды, и при этом он не влияет на позиционное соответствие оставшихся аргументов. Например:

```
        .LIST     ME
;
; РАСШИРЕНИЕ МАКРОКОМАНДЫ, СОДЕРЖАЩЕЙ КЛЮЧЕВЫЕ
; СЛОВА В СПИСКЕ АРГУМЕНТОВ
;
.MACRO  EXAM      BUFF=1, BLOCK, COUNT=TEMP
        .WORD     BUFF
        .WORD     BLOCK
```

```
        .WORD   COUNT
        .ENDM
;
; .ПРИМЕРЫ МАКРОВЫЗОВОВ И ИХ МАКРОРАСШИРЕНИЙ
;
        EXAM    A, B, C
        .WORD   A
        .WORD   B
        .WORD   C

        EXAM    COUNT=20, BLOCK=30, BUFF=40
        .WORD   40
        .WORD   30
        .WORD   20

        EXAM    BLOCK=5
        .WORD   1
        .WORD   5
        .WORD   TEMP

        EXAM    BUFF=5, COUNT=VARIAB
        .WORD   5
        .WORD
        .WORD   VARIAB

        EXAM
        .WORD   1
        .WORD
        .WORD   TEMP

        EXAM    COUNT=A&B
        .WORD   1
        .WORD
        .WORD   A&B
```

5.3.7. КОНКАТЕНАЦИЯ АРГУМЕНТОВ МАКРООПРЕДЕЛЕНИЯ

Апостроф (') в макроопределении используется как знак конкатенации строковых значений, получаемых формальными аргументами. В результате выполнения конкатенации производится объединение фактических значений формальных аргументов. Знак , который предшествует и (или) следует за формальным аргументом в макроопределении, удаляется, и выделенный формальный аргумент заменяется фактическим аргументом. Например, макрокоманда, заданная определением

```
        .MACRO  TEST    A, B, C
A'B:    .ASCII  /C/
        .EVEN
        .WORD   "A, "B
        .ENDM
```

при вызове

```
TEST X,Y, <PROGR>
```

расширяется следующим образом:

```
XY: .ASCII /PROGR/
    .EVEN
    .WORD   'X,'Y
```

При обработке оператора макроопределения

```
A'B:  .ASCII /C/
```

сканирование заканчивается при обнаружении первого знака '. Так как A является формальным аргументом, знак ' удаляется, A заменяется фактическим аргументом. Сканирование возобновляется с B. B опознается как другой формальный аргумент, и производится конкатенация фактических значений A и B, т. е. в макрорасширение вводится метка XY. Третий формальный аргумент входит в операнд директивы.

Аргумент директивы .WORD рассматривается следующим образом. Сканирование начинается со знака '. Так как этот знак не стоит перед формальным аргументом и не следует за ним, он остается в макрорасширении. В дальнейшем при сканировании обнаруживается второй знак ', сопровождаемый формальным аргументом. Этот апостроф удаляется, сканирование аргумента A заканчивается при обнаружении запятой. Следующий знак не стоит перед формальным аргументом и не следует за ним: он остается в макрорасширении. Четвертый (последний) знак апострофа, за которым следует формальный аргумент B, удаляется.

5.4. ВСТРОЕННЫЕ МАКРОКОМАНДЫ

Блок неопределенных повторений является конструкцией, аналогичной макроопределению. Он представляет собой макроопределение, которое имеет только один формальный аргумент и расширяется последовательно для каждого фактического аргумента. Заголовок блока неопределенных повторений одновременно содержит формальный и фактические аргументы.

Блоки неопределенных повторений могут использоваться вне и внутри других блоков неопределенных повторений, блоков повторений, макроопределений. Правила написания блока неопределенных повторений аналогичны правилам написания макроопределения. Синтаксис аргументов блока неопределенных повторений тот же, что и у аргумента макрокоманды.

Блок неопределенных повторений содержит один формальный аргумент (формальный аргумент данного блока неопределенных повторений) и список фактических аргументов. Число расширений

блока неопределенных повторений зависит от числа фактических аргументов: формальный аргумент принимает последовательно значение каждого фактического аргумента из списка и для каждого значения строится расширение блока. Создание таких блоков возможно по директивам .IRP и .IRPC.

5.4.1. ДИРЕКТИВА .IRP

Блок неопределенных повторений имеет следующую структуру:

```
LABEL: .IRP NAME, <LIST>
        тело блока
       .ENDM
```

где LABEL — необязательная метка оператора;

NAME — формальный аргумент, который последовательно заменяется фактическими аргументами из списка, заключенного в угловые скобки. Если формальный аргумент не указан, то директива .IRP получает в листинге трансляции флаг ошибки A;

, (запятая) представляет собой разделитель, в качестве которого допускается также пробел и(или) знак табуляции;

LIST — список фактических аргументов, которые должны быть использованы при построении расширений блока неопределенных повторений; должен быть заключен в угловые скобки. Фактический аргумент может состоять из одного или нескольких знаков. Аргументы разделяются любыми разрешенными разделителями (запятая, пробел, знак табуляции). Если фактический аргумент не указан, никаких действий не производится;

тело блока — последовательность команд, которая повторяется для каждого фактического аргумента списка. Тело блока может включать макроопределение, блоки повторений, блоки неопределенных повторений. В блоке неопределенных повторений допустима директива .MEXIT;

.ENDM — директива, означает конец тела блока неопределенных повторений.

Пример использования директивы .IRP приведен в 5.4.2.

5.4.2. ДИРЕКТИВА .IRPC

Возможен второй тип блока неопределенных повторений — посимвольный. Для этого типа блока список фактических аргументов является строкой символов. Строка должна быть заключена в угловые скобки, если она содержит разделители; в остальных случаях ограничение строки угловыми скобками необязательно, однако рекомендуется их использовать. Формат блока:

```
LAB: .IRPC NAME, <STRING>
        тело блока
     .ENDM
```

При каждом расширении блока неопределенных повторений формальный аргумент принимает последовательно значение каждого знака в строке. Пример использования директив .IRP и .IRPC:

```
1                                       .LIST   ME
2                                       .IRP    X,<N,K,L,M>
3                                       MOV     X,  (R0)+
4                                       .ENDM
  000000  016720  000000G              MOV     N,  (R0)+
  000004  016720  000000G              MOV     K,  (R0)+
  000010  016720  000000G              MOV     L,  (R0)+
  000014  016720  000000G              MOV     M,  (R0)+
5                                       .IRPC   X,<ABCD>
6                                       MOVB    #X,-(R1)
7                                       .ENDM
  000020  112741  000000G              MOVB    #A,-(R1)
  000024  112741  000000G              MOVB    #B,-(R1)
  000030  112741  000000G              MOVB    #C,-(R1)
  000034  112741  000000G              MOVB    #D,-(R1)
8         000001                        .END
```

5.4.3. БЛОКИ ПОВТОРЕНИЙ

Иногда необходимо повторить некоторую последовательность команд заданное число раз. Это обеспечивается созданием блока повторений, который имеет структуру:

```
LAB: .REPT EXPR
        тело блока повторений
     .ENDM ;или .ENDR
```

где LAB — необязательная метка оператора;

EXPR — любое разрешенное выражение, задающее число повторений тела блока. Если выражение равно 0, то тело блока повторений не транслируется. Если выражение не является абсолютной величиной, то в листинге трансляции задается флаг ошибки A;

тело блока — последовательность строк, которая должна быть повторена столько раз, сколько указано выражением EXPR.

Тело блока может содержать макроопределения, блоки неопределенных повторений, блоки повторений. Последним оператором в блоке повторений должен быть оператор .ENDM или .ENDR. В области блока повторений также разрешена директива .MEXIT.

5.5. СЛУЖЕБНЫЕ ДИРЕКТИВЫ

Три директивы макроассемблера позволяют пользователю определить некоторые атрибуты макрокоманды: количество аргументов в вызове макрокоманды, количество знаков в аргументе, код режима адресации аргумента в вызове макрокоманды.

Эти директивы могут использоваться для определения характеристик макрокоманды и (или) ее аргументов, которые могут быть необходимы при условной трансляции макрорасширения.

5.5.1. ДИРЕКТИВА .NARG

Директива .NARG дает возможность определить количество аргументов, заданных в вызове макрокоманды. Формат директивы

.NARG NAME

где NAME — любой разрешенный символ, который получает значение, равное количеству неключевых аргументов в вызове макрокоманды, транслируемой в данный момент. Если символ не указан, то директива порождает ошибку с флагом в листинге трансляции А.

Эта директива допустима только в теле макроопределения, иначе в листинге трансляции появляется сообщение об ошибке О.

Пример пользования директивы:

```
ABC: .NARG N ;N ПОЛУЧИТ ЗНАЧЕНИЕ ЧИСЛА АРГУМЕНТОВ В
             ;ТРАНСЛИРУЕМОЙ В ДАННЫЙ МОМЕНТ
             МАКРОКОМАНДЕ
```

5.5.2. ДИРЕКТИВА .NCHR

Директива .NCHR дает возможность определить число знаков в знаковой строке. Формат директивы

.NCHR NAME, <STRING>

где NAME — любой разрешенный символ, который получает значение, равное числу знаков в указанной знаковой строке. Символ отделяется от знаковой строки любым разрешенным разделителем. Если символ не указан в директиве .NCHR, то в листинге трансляции устанавливается флаг ошибки А;

STRING — строка печатных знаков. Должна быть заключена в угловые скобки или знаки †/.../, если она содержит разрешенные разделители (запятая, пробел или знак табуляции). Если знаки, ограничивающие строку, являются недопустимыми или в строке знаков не распознан концевой ограничитель и закончена обработка строки, то директива .NCHR отмечается флагом ошибки А в листинге трансляции.

Директива .NCHR может появляться в любом месте программы, написанной на языке Макро. Например, директива

XY: .NCHR X, <ABC, D>

присвоит символу X значение 5 (число знаков в строке, заключенной в угловые скобки).

5.5.3. ДИРЕКТИВА .NTYPE

Директива .NTYPE дает возможность при формировании макрорасширения определить вид адресации фактического аргумента. Форма директивы

.NTYPE NAME, EXPR

где NAME — любой разрешенный символ, которому при построении макрорасширения присваивается значение 6-разрядного поля режима адресации фактиче-

ского аргумента. Этот символ должен быть отделен от аргумента разрешенным разделителем: запятая, пробел и(или) знак табуляции. Если символ не указан, то директива NTYPE отмечается в листинге трансляции флагом ошибки A;

EXPR — любое допустимое адресное выражение. Если аргумент не указан, то результатом будет 0. Эта директива может использоваться только в макроопределении; появление ее вне макроопределения приводит к сообщению об ошибке с флагом O в листинге трансляции.

Пример использования .NTYPE в макроопределении:

```
. MACRO   SAVE     ARG
          . NTYPE  SYM, ARG
. IF      EQ,  SYM&70
          MOV      ARG    TEMP     ; АРГУМЕНТ – РЕГИСТР
. IFF
          MOV      @#ARG,  TEMP    ; АРГУМЕНТ – ЯЧЕЙКА ПАМЯТИ
          ...
          SAVE     R1              ; ВЫЗОВ МАКРОКОМАНДЫ БУДЕТ
                                   ;  РАСШИРЕН КАК MOV R1, TEMP
```

Полная информация о типах адресации содержится в главе 2.

5.5.4. ДИРЕКТИВА .MCALL

Макроопределения могут размещаться в самой программе и в файлах на устройстве прямого доступа, называемых макробиблиотеками. В программе макроопределения могут находиться в любом месте, но каждое из них должно появиться раньше обращения к соответствующей макрокоманде. Пользовательские библиотеки макроопределений создаются специальной системной программой «Библиотекарь». Макробиблиотеки должны быть созданы до трансляции исходной программы. Правила и способы оформления и создания макробиблиотек зависят от конкретной операционной системы и описываются в соответствующей эксплуатационной документации.

Директива .MCALL позволяет программисту объявить имена макрокоманд, которые будут использоваться в исходной программе, но макроопределения которых размещены в макробиблиотеке на диске. При трансляции соответствующие макроопределения вызываются из макробиблиотеки. Директива .MCALL имеет формат

.MCALL ARG1, ARG2,...

где ARG — имена макроопределений, вызываемых из макробиблиотеки.

В исходной программе директива .MCALL, описывающая имя макрокоманды, должна появляться раньше первого обращения к этой макрокоманде.

Библиотеки, в которых макроассемблер осуществляет поиск макроопределений, перечисленных в директиве .MCALL, описывается макроассемблеру как входные файлы в команде. Если в исходной программе встречается макровызов, транслятор начи-

нает поиск имени макроопределения с библиотеки пользовательских макроопределений и, если необходимо, продолжает поиск в библиотеке системных макроопределений.

Если имя макроопределения не найдено в библиотеках макроопределений, т. е. макрокоманда не определена, то директива .MCALL отмечается в листинге трансляции флагом ошибки U, а все вызовы этой макрокоманды отмечаются в листинге трансляции флагом ошибки O.

Организация библиотек макроопределений и их состав зависят от особенностей конкретной операционной системы. Обычно в составе операционных систем пользователям поставляется несколько системных макробиблиотек:

общесистемная макробиблиотека;

макробиблиотеки отдельных компонентов операционных систем.

Общесистемные макробиблиотеки содержат макроопределения, облегчающие программирование на языке Макро операций ввода-вывода, обращения к монитору операционной системы для выполнения системных функций, обращения к файловому процессору.

Макробиблиотеки отдельных компонентов состоят из макроопределений, реализующих функции компонентов, например работу с записями файлов, поиск данных по заданным критериям в базе данных и т. п.

Макроопределения, перечисленные директивами .MCALL, хранятся в памяти, размер которой ограничен. Если количество макроопределений велико и (или) макроопределения велики сами по себе, то возрастает вероятность нехватки памяти. В обоих случаях время трансляции может сильно увеличиться.

Директива .MDELETE предназначена для удаления из памяти макроопределений, необходимость в которых отпала, т. е. они в оставшейся части модуля не будут использоваться. Процесс удаления ненужных макроопределений освобождает память для других макроопределений и приводит к сокращению времени трансляции. Формат директивы .MDELETE полностью совпадает с форматом .MCALL.

5.6. ДИРЕКТИВЫ УПРАВЛЕНИЯ ФАЙЛАМИ

Директивы управления файлами предназначены для спецификаций файлов, содержащих макробиблиотеки, в которых может осуществляться поиск требуемых макроопределений, а также для включения текста из указанного файла в текст транслируемого модуля.

5.6.1. ДИРЕКТИВА .LIBRARY

Директива .LIBRARY позволяет указать в транслируемом модуле спецификацию файла, содержащего макробиблиотеку. Список файлов, задаваемый директивами .LIBRARY, используется

жащих ошибки). Если содержимое счетчика равно 0, то строки листинга создаются либо не создаются в зависимости от параметров управления листингом, заданных в данный момент для транслятора.

Например, следующее макроопределение использует директивы .LIST и .NLIST для выборочной распечатки частей макрорасширения.

```
        .MACRO  LITES  ; ТЕСТ РАСПЕЧАТКИ
; A-СТРОКА 1            ; СЧЕТЧИК=0
        .LIST
; B-СТРОКА 2            ; СЧЕТЧИК = 1
        .NLIST
; C-СТРОКА 3            ; СЧЕТЧИК = 0
        .NLIST
; D-СТРОКА 4            ; СЧЕТЧИК=-1
        .LIST
; E-СТРОКА 5            ; СЧЕТЧИК = 0
        .ENDM
        .LIST     ME   ; РАСПЕЧАТЫВАТЬ МАКРОРАСШИРЕНИЯ
        LITES          ; МАКРОВЫЗОВОВ
; A-СТРОКА 1            ; СЧЕТЧИК = 0
; B-СТРОКА 2            ; СЧЕТЧИК = 1
; C-СТРОКА 3            ; СЧЕТЧИК = 0
; E-СТРОКА 5            ; СЧЕТЧИК = 0
```

Основным назначением .LIST и .NLIST без аргументов является обеспечение возможности избирательной распечатки макрорасширения с выходом на уровень, имевший место во время вызова макрокоманды.

Директивы управления распечаткой, имеющие аргумент, не влияют на содержимое счетчика. Однако некоторые аргументы позволяют обеспечить избирательную распечатку листинга при нулевом значении счетчика, например:

```
.MACRO    XX
    ...
.LIST             ; РАСПЕЧАТАТЬ СЛЕДУЮЩУЮ СТРОКУ И
X         =.      ; НЕ РАСПЕЧАТЫВАТЬ ОСТАВШУЮСЯ ЧАСТЬ
.NLIST            ; МАКРОРАСШИРЕНИЯ
    ...
.ENDM
.NLIST    ME      ; АРГУМЕНТ ME В .NLIST ЗАПРЕЩАЕТ РАСПЕЧАТКУ
                  ; МАКРОРАСШИРЕНИЯ.
                  ; СЧЕТЧИК РАВЕН НУЛЮ
XX                ; ВЫЗОВ МАКРОКОМАНДЫ XX
X         =.      ; СТРОКА МАКРОРАСШИРЕНИЯ РАСПЕЧАТЫВАЕТСЯ,
                  ; ТАК КАК СЧЕТЧИК = +1, ОСТАЛЬНЫЕ СТРОКИ
                  ; МАКРОРАСШИРЕНИЯ НЕ РАСПЕЧАТЫВАЮТСЯ ПО
                  ; АРГУМЕНТУ ME ПРИ СЧЕТЧИКЕ, РАВНОМ 0
```

Допустимо использование в директиве как одного, так и нескольких аргументов; разделителями аргументов могут быть знак табуляции, запятая, пробел. Аргументы, не указанные в директи-

вах управления листингом, получают значение по умолчанию («распечатывать» или «не распечатывать»).

Ниже приводится функциональное назначение аргументов директив управления листингом .LIST и .NLIST.

SEQ — управляет распечаткой порядковых номеров строк. Если распечатка поля нумерации запрещается директивой .NLIST SEQ, транслятор выводит знак табуляции, который сохраняет формат выводной строки. Поле нумерации выделяется, но заполняется пробелами;

LOC — управляет распечаткой счетчика адресов (обычно это поле не блокируется). Однако если распечатка поля счетчика адресов запрещена директивой .NLIST LOC, транслятор в отличие от .NLIST SEQ не выводит знака табуляции и не выделяет место для этого поля. Таким образом, при запрещении распечатки этого поля все последующие поля сдвигаются влево и начинаются с той позиции, с которой должно было размещаться поле счетчика адресов;

BIN — управляет распечаткой создаваемого двоичного кода; если это поле запрещено директивой .NLIST BIN, то исходные поля сдвигаются влево так же, как и при .NLIST LOC;

BEX — управляет распечаткой второго и третьего слов двоичного кода; является частным случаем аргумента BIN;

COM — управляет распечаткой комментария; может использоваться для уменьшения времени распечатки и места, отводимого на нее, в случаях, когда комментарии не нужны;

MD — управляет распечаткой макроопределений и определений блоков повторений;

MC — управляет распечаткой вызовов макрокоманд и расширений блоков повторений;

ME — управляет распечаткой макрорасширений. Директива дает полную распечатку макрорасширения и двоичного кода, генерируемого макрорасширением;

MEB — управляет распечаткой двоичного кода макрорасширения. Директива .LIST MEB вызывает распечатку только директив и операторов макрорасширения, в результате трансляции которых создается двоичный код. Является частным случаем аргумента ME.

CND — дает возможность управлять распечаткой блоков условной трансляции в случае, если не выполнены условия их трансляции;

SRC — управление распечаткой исходного текста транслируемого модуля;

LD — управляет распечаткой всех директив управления листингом, в которых нет аргументов, т. е. директив, изменяющих значения счетчика распечатки;

TOC — управляет распечаткой оглавления при первом проходе трансляции. Этот аргумент не влияет на распечатку полного листинга трансляции во время второго прохода;

TTM — управляет форматом выходных строк листинга. Директива .LIST TTM вызывает усечение выходных строк до 80 знаков. Сгенерированный двоичный код для каждого оператора печатается в один столбец. .NLIST TTM обеспечивает вывод листинга со строками по 132 знака;

SYM — управляет распечаткой таблицы символов программы.

При отсутствии директив с аргументами SEQ, LOC, BIN, SRC, BEX, COM, MD, MC, CND, TOC, SYM выполняется распечатка соответствующих элементов листинга.

Если в директиве .NLIST указаны одновременно аргументы SEQ, LOC, BIN, SRC, т. е. если запрещены все четыре поля строки листинга, то соответствующая строка пробелов не выводится.

Любой аргумент, указанный в директивах .LIST/.NLIST, отличный от перечисленных, отмечается в листинге флагом ошибки A.

В листинге трансляции, выведенном в формате TTM (длина строки 80 знаков), только первое слово двоичного кода печатается в одной строке с исходным оператором. Дополнительные слова двоичного кода печатаются на последующих строках.

Управление форматированием листинга может также осуществляться с помощью командной строки транслятору. Ее использование позволяет отменить действия директив управления листингом с соответствующими аргументами, указанных в исходной программе.

6.1.2. ЗАГОЛОВОК СТРАНИЦЫ

Макроассемблер оформляет страницы листинга в формате, определенном параметром TTM. На первой строке каждой страницы листинга транслятор печатает (слева направо) имя объектного модуля, взятое из директивы .TITLE, идентификацию версии макроассемблера, дату, время и номер страницы.

На второй строке каждой страницы листинга печатается текст подзаголовка, указанный в последней встреченной на странице директиве .SBTTL.

6.1.3. ДИРЕКТИВА .PAGE

Существует несколько способов перехода на новую страницу при распечатке программы, написанной на языке макроассемблер.

Первый способ. После того как число строк достигает 58, макроассемблер автоматически переходит на новый лист распечатки в случае вывода на АЦПУ или дополняет распечатку знаком «перевод формата», если вывод производится на терминал. Номер страницы не изменяется.

Второй способ. Перевод страницы происходит, если знак перевода формата используется в качестве ограничителя строки (или является единственным знаком в строке). Страницы перево-

дятся также в случае, если знак перевода формата используется в макроопределении. Перевод страницы при этом имеет место в процессе трансляции макроопределения, но не при расширении макровызова. Если перевод страницы производится по знаку перевода формата, то номер каждой новой страницы увеличивается на единицу.

Третий способ. Перевод страницы производится по директиве .PAGE. Эта директива используется в исходной программе, и перевод страницы происходит в точке ее появления. В таком случае номер страницы увеличивается на единицу. Директива .PAGE не распечатывается в листинге. Формат директивы

.PAGE

Эта директива не использует никаких аргументов и вызывает переход к началу следующей страницы. Если директива .PAGE встречается в макроопределении, то она игнорируется. Если директива .PAGE встречается при обработке макрорасширения, то происходит переход на новую страницу. В этом случае номер новой страницы также увеличивается на единицу.

Четвертый способ. Переход к каждому новому исходному файлу вызывает увеличение номера страницы.

6.2. ПАРАМЕТРЫ ТРАНСЛЯЦИИ

Управление некоторыми функциями транслятора осуществляется с помощью директив .ENABL и .DSABL, использующих аргументы для обозначения требуемой функции. Эти директивы применяются в исходной программе, чтобы разрешать или запрещать выполнение определенных функций и действий в процессе трансляции. Форматы директив

```
.ENABL ARG
.DSABL ARG
```

где ARG — один или несколько символических аргументов, описываемых ниже.

Если в директивах .ENABL/.DSABL указываются несколько аргументов, они должны разделяться знаками табуляции, пробелами или запятыми.

Любой аргумент в директивах .ENABL/ .DSABL, отличный от указанных ниже, вызывает в листинге трансляции флаг ошибки A.

Ниже приводится функциональное назначение аргументов директив .ENABL и .DSABL.

ABS (по умолчанию — запрещение) — генерируется абсолютный двоичный код.

LCM (по умолчанию — запрещение) — оказывает влияние на директивы условной трансляции .IF IDN и .IF DIF. В случае .ENABL LCM аргументы в этих директивах условной трансляции не преобразуются в верхний регистр. В противном случае перед сравнением аргументы преобразуются в верхний регистр (т. е. буквы кириллицы — в буквы латыни).

MCL (по умолчанию — запрещение) — при использовании директивы .ENABL MCL транслятор начинает искать в заданных макробиблиотеках все неопределенные символы, появившиеся в поле операции. По умолчанию строки, содержащие неопределенный символ в поле операции, отмечаются флагом ошибки U или объявляются глобальными (в зависимости от .ENABL/ .DSABL GBL).

AMA (по умолчанию — запрещение) — все относительные адреса (код адресации 67) транслируются как абсолютные адреса (код адресации 37). Аргумент полезно использовать во время отладки программы.

CDR (по умолчанию — запрещение) — в результате действия оператора .ENABL CDR (колонки оператора с 73-й интерпретируются как комментарии, что позволяет указывать порядковые номера на картах в колонках — 73—80).

FPT (по умолчанию — запрещение) — вызывает усечение чисел с плавающей запятой. При .ENABL FPT производится округление чисел с плавающей запятой.

LC (по умолчанию — разрешение) — макроассемблер принимает входную информацию с буквами кириллицы, не преобразовывая их в латинские.

LSB (по умолчанию — запрещение) — если обычно блок локальных символов заканчивается при обнаружении новой символической метки или директивы .PSECT, то .ENABL LSB обеспечивает продолжение блока локальных символов до тех пор, пока не появится символическая метка или директива .PSECT, следующая за оператором .DSABL LSB. Несмотря на то что директива .ENABL LSB позволяет блоку локальных символов переходить границу секции .PSECT, локальные символы могут быть определены только в той секции, в которой начат этот блок. Расширение блока локальных символов на другую программную секцию возможно лишь в случаях, когда новая секция используется для размещения данных, а затем следует возврат в исходную секцию. Попытка определить локальные символы в другой программной секции отмечается в листинге флагом ошибки P.

PNC (по умолчанию — разрешение) — директива .DSABL PNC запрещает вывод двоичной информации до тех пор, пока не появится .ENABL PNC.

REG (по умолчанию — разрешение) — директива .DSABL REG подавляет определения регистров по умолчанию: R0, R1, ..., R5, SP, PC. Оператор .ENABL REG может быть использован как логическое дополнение директивы .DSABL REG. Однако использование этих директив не рекомендуется. Программист обычно использует определения регистров по умолчанию.

GBL (по умолчанию — разрешение, зависит от операционной системы) — по директиве .ENABL CBL макроассемблер трактует все символические ссылки, не определенные в конце 1-го прохода трансляции, как глобальные. По директиве .DSABL GBL макроассемблер трактует все такие ссылки как неопределенные символы. Во 2-м проходе трансляции, если еще действует функция

.DSABL GBL, такие неопределенные символы отмечаются флагом ошибки U.

CRF (по умолчанию — разрешение) — блокировка этой функции запрещает генерацию вывода для таблицы перекрестных ссылок. Эта функция имеет смысл, если генерация вывода таблицы перекрестных ссылок определена в командной строке.

6.3. ДИРЕКТИВЫ .CROSS И .NOCROSS

Формат директив

```
.CROSS
.CROSS      SYM1,...
.NOCROSS
.NOCROSS    SYM1,...
```

где SYM,... — одно или несколько символических имен. Если указываются несколько символов, они должны разделяться в строке одним из допустимых разделителей (запятая, пробел или табуляция).

Директивы .CROSS и .NOCROSS определяют, какие символы войдут в распечатку таблицы перекрестных ссылок, генерируемую транслятором. Эти директивы действуют, если в командной строке транслятору дан ключ /C или ключ /CROSS.

По умолчанию таблица перекрестных ссылок содержит записи о точках определения, использования и изменения для всех символов, определенных пользователем. Сбор информации для таблицы может быть запрещен или разрешен для всех символов директивами .ENABL/.DSABL CRF.

Если директива .NOCROSS используется без аргументов, то она действует как .DSABL CRF. Сбор информации может быть продолжен по директиве .CROSS, которая действует аналогично .ENABL CRF.

Директива .NOCROSS, использованная с аргументами (символическими именами), запрещает сбор информации для заданных символов. Для продолжения сбора информации используется директива .CROSS с аргументами.

В следующем примере определение метки L1 и ссылки на символы M1, M2 фиксироваться в таблице не будут.

```
        .NOCROSS            ; ЗАПРЕТ СБОРА ДАННЫХ
L1:     ADD         M1, M2
        .CROSS              ; СНЯТИЕ ЗАПРЕТА
```

Использование символа M1 в приведенном ниже примере в таблице перекрестных ссылок не фиксируется.

```
.NOCROSS    M1
ADD         M1, M2
.CROSS      M1
```

При этом использование M2 и определение L1 будут зафиксированы.

Директива .CROSS, использованная без списка, не воздействует на сбор информации о символах, указывавшихся в списке директивы .NOCROSS. Аналогично, если сбор информации полностью запрещен, использование директивы .CROSS со списком не дает требуемого эффекта до тех пор, пока не будет использована директива .CROSS без списка.

6.4. РЕГИСТРАЦИЯ ОШИБОК

Директивы .ERROR и .PRINT используются для вывода сообщений во время 2-го прохода трансляции. Они могут быть использованы в любом месте программы для вывода сообщений об ошибочных вызовах макрокоманд либо для вывода предупреждающих сообщений при условной трансляции.

Если не указан файл листинга, то сообщение по директиве .ERROR выводится на устройство, с которого подаются команды. Директива .ERROR может использоваться в любом месте исходной программы. Она имеет следующую форму:

```
LABEL: .ERROR EXPR; TEXT
```

где LABEL — необязательная метка оператора;

EXPR — допустимое выражение, значение которого выводится при выполнении директивы .ERROR во время трансляции;

; — обозначает начало текста, который должен быть напечатан;

TEXT — сообщение, выводимое по директиве .ERROR; строка текста заканчивается ограничителем строки.

В результате выполнения директивы .ERROR распечатывается строка, содержащая: флаг ошибки P; порядковый номер строки исходного текста, содержащей директиву .ERROR, текущее значение счетчика адреса; значение выражения EXPR, если оно указано; исходную строку, содержащую директиву .ERROR. Например:

```
.ERROR A      ;НЕДОПУСТИМЫЙ МАКРОАРГУМЕНТ
```

дает в листинге следующую строку

```
P 512 005642 000076  .ERROR A  ;НЕДОПУСТИМЫЙ МАКРОАРГУМЕНТ
```

Директива .PRINT тождественна по функциям директиве .ERROR за исключением того, что она не выводит флаг ошибки P в листинге трансляции. При обработке этой директивы значение счетчика ошибки не увеличивается на 1 (как это происходит при обработке директивы .ERROR).

7

ГЛАВА

ТЕХНИКА ПРОГРАММИРОВАНИЯ

Главы 7 и 8 посвящены применению наиболее часто используемых средств и возможностей языка Макро. Рассмотрим рекомендации по оформлению программ и типичные для начинающих программистов ошибки. Изложенный материал иллюстрируется примерами сравнительно простых программ, понимание которых не должно вызывать затруднений.

7.1. СОГЛАШЕНИЯ И РЕКОМЕНДАЦИИ

Использование единых соглашений при программировании на языке Макро для оформления и кодирования программ облегчает сопровождение и передачу программ, понимание их другим программистом. Рассмотрим один из возможных вариантов таких соглашений и рекомендаций.

7.1.1. ОФОРМЛЕНИЕ СТРОКИ

При написании программ рекомендуется, чтобы все строки содержали не более 80 знаков. Это ограничение полезно при выводе листингов на терминал и при работе с редакторами текстов в случае внесения изменений, например при отладке и сопровождении программы.

Как отмечалось в главе 3, для строки рекомендуется следующий формат:

поле метки должно начинаться с первой позиции;

поле операции — с 9-й позиции (после одного знака табуляции);

поле операндов — с 17-й позиции (после двух табуляций);

комментарий занимает позиции 33—80 (после четырех табуляций).

7.1.2. КОММЕНТАРИИ

Комментарии могут использоваться для пояснения роли инструкции в алгоритме данного фрагмента программы. Такой комментарий располагается в строке, содержащей инструкцию. Комментарий, который начинается в строке инструкции и требует продолжения, может быть продолжен на следующей строке. При этом рекомендуется начинать его с той же позиции, что и в предыдущей строке, например:

```
CLR  CNTPAG      ; ОЧИСТИТЬ СЧЕТЧИК СТРАНИЦ ДЛЯ
                 ; ВТОРОГО ПРОХОДА
MO\  #1, CNTSTR  ; ЧИСЛО СТРОК РАВНО 1
```

Если поле операнда выходит за 32-ю позицию, то в строке оставляют один пробел и печатают комментарий:

```
MOV  R1, R0                  ; СКОПИРОВАТЬ АККУМУЛЯТОР
BIT  #ERRBIT!DONBIT, 4 (R5) ; ПРОВЕРИТЬ СОСТОЯНИЕ
BEO  20©                     ; СОСТОЯНИЕ НЕ ИЗМЕНИЛОСЬ
```

В отдельных случаях удобно начинать все комментарии с 40-й позиции (после пяти табуляций).

Если трудно кратко пояснить назначение инструкции, то этой строке или фрагменту текста может предшествовать «абзац» комментариев. Такой комментарий начинается с первой позиции и отделяется от остального текста строками, содержащими только точку с запятой, например

```
        MOV     R1, R0                          ; СКОПИРОВАТЬ АККУМУЛЯТОР
;
; ПРОВЕРИТЬ ВНЕШНЕЕ УСТРОЙСТВО НА ГОТОВНОСТЬ
; ИЛИ НА НАЛИЧИЕ ОШИБКИ В РАБОТЕ
;
        BIT     #ERRBIT!DONBIT, 4(R5)           ; ПРОВЕРИТЬ СОСТОЯНИЕ
```

Пояснительный текст, описывающий форматы данных, алгоритмы, программные локальные переменные, рекомендуется оформлять для большей наглядности следующим образом:

```
; +
; В ДАННОМ СЕГМЕНТЕ РАЗМЕЩЕНЫ ПРОГРАММЫ ПРЕОБРАЗОВАНИЙ
; ДАННЫХ ИЗ ФОРМАТА А В ФОРМАТ Б ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ
;
; ДЛЯ ПРЕОБРАЗОВАНИЯ ИСПОЛЬЗУЕТСЯ АЛГОРИТМ...
;
; ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ: ...
; -
```

7.1.3. ФОРМИРОВАНИЕ СИМВОЛОВ

Символические обозначения, используемые в программе (имена), можно условно разбить на три группы: обозначения ре-

гистров аппаратуры, символы, используемые для связи между модулями, и локальные символы.

К регистрам аппаратуры относятся регистры процессора, регистры внешних устройств и т. д. При использовании универсальных регистров процессора рекомендуется всегда применять обозначения, приведенные в 3.4.4.

Эти имена не следует использовать для каких-либо других целей. Не рекомендуется также при обозначении универсальных регистров применять иные обозначения, поскольку это затрудняет понимание программы другими программистами.

Регистры внешних устройств должны иметь имена, идентичные их обозначениям в технической документации. Например, если слово состояния процессора обозначается в документации PS, то в программах желательно использовать ту же самую мнемонику.

Для обозначения отдельных разрядов регистров аппаратуры рекомендуется использовать аналогичные правила. Например, для проверки и изменения приоритета процессора целесообразно применять символы:

```
PR0=0     ; ПРИОРИТЕТ 0
PR1=40    ; ПРИОРИТЕТ 1
 ...
PR7=340   ; ПРИОРИТЕТ 7
```

Приведенным символам присвоены значения, соответствующие значениям приоритетных разрядов слова состояния процессора.

Необходимо отметить, что только стандартные имена универсальных регистров процессора определены в самом Макро, а все другие символы, обозначающие аппаратные регистры и их разряды, должны определяться в тексте программы. Такие определения стандартных имен могут находиться вне текста программы (отдельный файл, макробиблиотека, объектная библиотека).

При выборе мнемоники для обозначения часто используемых символов целесообразно пользоваться следующими обозначениями:
AXXXXX — внутренний символ модуля;
SXXXXX — глобальный символ;
ASXXXX — глобальное смещение (например, в элементе очереди);
ACSXXX — глобальное обозначение разряда (разрядов) — маска, где A — любая буква латинского алфавита, C — буква или цифра, S — знак ⊙ или . (точка), X — необязательный символ (может быть буквой, цифрой или вообще опущен).

Примеры внутренних символов:

```
ABC2
LABEL4
BUFFER
```

Примеры глобальных символов:

```
⊙BASE
.ENTRY
⊙CNTR
```

Примеры глобальных смещений:

A.SHFT
C⊙LINK
X.PNT

Примеры обозначений глобальных разрядов и масок:

CH.OPN
DV⊙DON
JB⊙RUN

В качестве локальных символов программы рекомендуется использовать локальные символы Макро, которые, как описано в главе 3, имеют формат n⊙.

Обычно локальные символы используются в качестве меток в инструкциях переходов. Локальные символы (метки) рекомендуется размещать в теле блока локальных символов таким образом, чтобы их номера возрастали последовательно как на отдельной странице, так и от страницы к странице.

Использование конструкции

```
10⊙     TSTB    @#CSR
        BPL     10⊙
```

предпочтительнее использования конструкции

```
TSTB    @#CSR
BPL     .-4
```

В последнем варианте затрудняется понимание логики программы и повышается вероятность ошибки при последующих модификациях программы.

7.1.4. ОФОРМЛЕНИЕ МОДУЛЯ

Текст программы (модуля) на языке Макро рекомендуется разбивать на несколько разделов: вводная часть, определения, спецификация и текст программы.

Вводная часть должна включать наименование и версию модуля, дату создания и коррекций, имена разработчиков и т. д. Желательно, чтобы вводная часть размещалась на одной странице листинга. Информацию, входящую в вводную часть, можно располагать, например, в следующем порядке:

имя модуля, задаваемое обычно директивой .TITLE;

номер версии и (или) коррекции, указываемый директивой .IDENT;

дата создания и имена разработчиков;

даты внесения изменения (коррекций) с пояснением и указанием имени программиста, проводившего эти изменения; эту информацию рекомендуется размещать в хронологическом порядке;

краткое описание функций модуля.

В раздел определений входят описания используемых символов и переменных, определения локальных макрокоманд и

структура локальных данных (данные, внутренние для модуля). Этот раздел может иметь следующую структуру:

перечень библиотечных макрокоманд, указываемых директивой .MCALL;

описание и определения локальных макрокоманд;

имена и краткая характеристика используемых модулей;

описание глобальных символов и программных секций, используемых в модуле;

описание и определения внутренних символов и структуры локальных данных, включая описания каждого элемента (тип, размер и т. д.), специфика размещения в памяти и пр.;

использование регистров процессора и внешних устройств.

Раздел спецификаций может иметь вид:

описание и структура входных данных, ожидаемых модулем, включая установленные разряды слова состояния процессора и значения глобальных переменных, а также вызывающую последовательность, если она имеет какие-либо особенности;

описание и структура выходных данных, таких как основные результаты и установленные разряды PS;

подробное описание функций, выполняемых модулем, включая применяемые алгоритмы, со ссылками на соответствующую литературу;

подробное описание побочных эффектов, поясняющее возможные изменения состояния системы, которое явно не предвидится в вызывающей последовательности или является неявным для того, кто использует функцию.

В случае необходимости, если к модулю предъявляются особые требования, в этот раздел включаются также данные об использовании рабочей памяти, размеры необходимого стека, временные характеристики для разных моделей технических средств и т. д.

Последний раздел, содержащий текст программы на языке Макро, рекомендуется снабжать подробными комментариями. Текст программы следует разделять на области инструкций и данных, размещая их в различных программных секциях. Данные, доступные только для чтения, рекомендуется отделять от данных, доступных для чтения-записи, и размещать в программных секциях с соответствующими параметрами.

Соблюдение рекомендуемых правил оформления модуля обеспечивает высокое качество документации по сопровождению программы («самодокументированность» программы).

7.1.5. ВЗАИМОДЕЙСТВИЕ МОДУЛЕЙ

Использование при проектировании и написании программы принципа модульности позволяет снизить затраты на отладку и сопровождение, облегчить ее понимание другими программистами. При использовании этого принципа особое значение приобретают

соглашения о связях между отдельными модулями, поскольку применение единых правил для связи всех модулей позволяет уменьшить вероятность ошибок при реализации взаимодействия.

Для вызова модуля рекомендуется использовать инструкцию

```
JSP PC, SUBR
```

а для завершения работы и возврата управления

```
RTS PC
```

Исключение из этого правила допустимо только для программ сохранения и восстановления регистров.

Для удобства и наглядности в языке Макро содержатся специальные мнемонические обозначения таких команд:

```
CALL    SUBR  ; ЭКВИВАЛЕНТНО JSR PC, SUBR
RETURN        ; ЭКВИВАЛЕНТНО RTS PC
```

В случае отсутствия в трансляторе приведенных мнемоник соответствующие конструкции можно определить с помощью макрокоманд:

```
.MACRO  CALL    SUBR
        JSR     PC, SUBR
        .ENDM
.MACRO  RETURN
        RTS     PC
        .ENDM
```

Регистры, которые предполагается использовать в модуле, должны быть сохранены (как правило, в стеке) в начале и восстановлены в конце работы модуля. Исключение могут составлять регистры, через которые возвращаются результаты.

Для передачи аргументов и возврата результатов могут быть использованы любые регистры. Однако предпочтительным является использование регистров с младшими номерами. Например, если передаются два аргумента, то целесообразно использовать R0, R1, а не R5, R3.

Если результат работы модуля может быть представлен бинарной функцией, то для его возврата рекомендуется использовать бит C слова состояния процессора. Этот разряд также может быть использован для указания нормального (C=0) или ненормального (C=1) завершения работы. Другими словами, бит C может быть использован для указания на ошибку при выполнении модуля.

В том случае, когда модуль использует какие-либо входные параметры, проверка их правильности должна производиться самим модулем. Это правило позволяет локализовать такую проверку вместо того, чтобы осуществлять ее в каждой точке обращения к модулю в вызывающих программах.

Необходимо помнить, что некоторые «критические» фрагменты

модуля желательно особо выделить в тексте программы. Средством такого выделения могут служить комментарии. Примером подобной ситуации может быть участок программы, на котором запрещены прерывания.

```
        MOV     #FLAG, R1  ; ПРЕРЫВАНИЯ РАЗРЕШЕНЫ
        BISB    #PR7, @#PS ; ; ; НА ЭТОМ
        TST     @R1        ; ; ; УЧАСТКЕ
        BEQ     10⊙        ; ; ; ПРОГРАММЫ
        CALL    QUEUE      ; ; ; ЗАПРЕЩЕНЫ
10⊙:    CLRB    @#PS       ; ; ; ПРЕРЫВАНИЯ
```

7.1.6. ЗАМЕЧАНИЯ

Замечания, рассматриваемые далее, касаются некоторых нежелательных вариантов использования инструкций, вызывающих побочные эффекты.

Не рекомендуется использовать в текущей инструкции следующую инструкцию или индексное слово в качестве литерала, например

```
MOV     @PC, R3
BIC     A, B
```

Команда MOV использует инструкцию BIC как литерал. Использование такого приема, хотя он может показаться удобным и эффективным, не рекомендуется, поскольку он ухудшает наглядность программы и может порождать ошибки при изменении инструкции BIC или режима адресации одного из операндов.

Не рекомендуется для передачи управления использовать инструкцию MOV вместо инструкции JMP, например

```
MOV     #SUBR, PC
```

Подобный прием программирования нежелателен, поскольку увеличивается время выполнения программы (команда MOV выполняется примерно в два раза дольше, чем JMP). Регистрация точек передачи управления, возможная при использовании JMP, затруднительна для команд MOV. По этим же причинам нежелательно использование и других инструкций, модифицирующих PC, таких, как ADD, SUB, BIC и т. д.

Особое внимание следует обратить на использование инструкций условных переходов. Необходимо четко осознавать различие между знаковыми и беззнаковыми инструкциями условных переходов:

Знаковые	Беззнаковые
BGE	BHIS(BCC)
BGT	BHI
BLE	BLOS(BCS)
BLT	BLO

При использовании инструкции перехода по знаку возможна ошибка, если перед этим сравниваются два адреса памяти. Переход будет осуществлен правильно, если адреса имеют одинаковый знаковый разряд. Как только один из адресов получает значение, большее 32 К (100000_8), передача управления происходит неправильно. Ошибка этого типа обычно возникает при перекомпоновке программы на другие адреса или изменении размера программы, например

```
        MOV     #BUFF, R0       ; НАЧАЛО БУФЕРА
        MOV     #BUFEND, R1     ; КОНЕЦ БУФЕРА
1⊙:     CLR     (R0)+           ; ОЧИСТИТЬ СЛОВА
        CMP     R0, R1          ; СРАВНИТЬ АДРЕСА
        BLT     1⊙              ; ВОЗМОЖНА ОШИБКА
        BLO     1⊙              ; ВСЕГДА ПРАВИЛЬНО
```

7.2. ПРИЕМЫ ПРОГРАММИРОВАНИЯ

Рассмотрим на ряде примеров некоторые приемы программирования на языке Макро для СМ ЭВМ. Большинство рассматриваемых примеров с незначительными изменениями могут быть использованы в прикладных или системных программах. Предлагаемый перечень приемов не исчерпывает всех возможных случаев и предназначен в основном для демонстрации особенностей использования некоторых инструкций и средств языка Макро. Для подробного ознакомления с приемами программирования и полезными алгоритмами можно рекомендовать [2].

Использование Макро для реализации вычислительных задач с использованием численных методов не рассматривается, поскольку эти задачи с высокой степенью эффективности решаются языками высокого уровня типа Фортран IV, Паскаль, Бейсик, входящими в большинство операционных систем СМ ЭВМ. При необходимости реализации таких задач на языке Макро в качестве подсобного материала может быть использован листинг соответствующей программы на языке Фортран IV, так как компилятор с этого языка может создавать распечатку сгенерированного кода на языке Макро.

7.2.1. ПРИМЕР ЦИКЛА

Использование циклов упрощает обработку массивов, таблиц строк и подобных по структуре данных. Различные конструкции циклов и возможности их реализации в языках программирования подробно описываются в [2, 3].

Рассмотрим возможные варианты организации цикла на примере фрагмента программы, который производит очистку (занесение нулей в ячейки) области памяти.

```
        MOV     #ARRAY, R0 ; ПЕРЕСЛАТЬ АДРЕС МАССИВА В R0
        MOV     #155., R1  ; ПЕРЕСЛАТЬ ДЛИНУ МАССИВА В R1
CLR:    MOVB    #0,@R0     ; ОЧИСТИТЬ ОЧЕРЕДНОЙ БАЙТ
        ADD     #1, R0     ; СДВИНУТЬ УКАЗАТЕЛЬ К СЛЕДУЮЩЕМУ БАЙТУ
        SUB     #1, R1     ; УМЕНЬШИТЬ СЧЕТЧИК НА 1
        CMP     #0, R1     ; СРАВНИТЬ СЧЕТЧИК С 0
        BNE     CLR        ; ПОВТОРИТЬ ОЧИСТКУ, ЕСЛИ
                           ; СЧЕТЧИК НЕ 0
        ...
ARRAY:  .BLKB   155.       ; ОЧИЩАЕМЫЙ МАССИВ
```

В приведенном примере инструкция MOVB используется для записи нуля в очередной байт очищаемой области. Эта инструкция занимает два слова памяти, и при ее выполнении производятся три обращения процессора к ОЗУ: считывание первого слова инструкции, считывание второго слова инструкции (константа 0) и запись этой константы по адресу, находящемуся в регистре 0. Данную инструкцию можно заменить более короткой и быстрой инструкцией CLRB. Далее, инструкции ADD и SUB также можно заменить на эквивалентные им в данном случае инструкции INC и DEC соответственно, а инструкцию CMP — на TST. В результате этих изменений фрагмент приобретает следующий вид:

```
        ...
CLR     CLRB    @R0 ; ОЧИСТИТЬ ОЧЕРЕДНОЙ БАЙТ
        INC     R0  ; СДВИНУТЬ УКАЗАТЕЛЬ
        DEC     R1  ; УМЕНЬШИТЬ СЧЕТЧИК НА 1
        TST     R1  ; СРАВНИТЬ СЧЕТЧИК С 0
        BNE     CLR ; ПОВТОРИТЬ ОЧИСТКУ, ЕСЛИ НЕ 0
        ...
```

Это позволило сократить объем фрагмента на 4 слова и уменьшить число обращений к памяти для каждой итерации цикла также на 4.

В последнем варианте программы можно исключить инструкцию INC за счет использования в инструкции CLRB режима адресации с автоувеличением. Этот режим обеспечивает автоматическое увеличение содержимого регистра после его использования в качестве адреса операнда на 1 при работе с байтами (и на 2 при работе со словами). Инструкция BNE выполняет условный переход на начало цикла, если код условия Z в слове состояния процессора PS установлен в 0. Для определения этого кода условия в программе используется инструкция TST. Использование этой инструкции в данном примере необязательно — соответствующая установка кодов условий PS осуществляется инструкцией DEC, предшествующей инструкции TST. Внесенные изменения позволили сократить программу на 6 слов (по сравнению с первым вариантом).

```
CLR:    CLRB    (R0)+ ; ОЧИСТИТЬ БАЙТ И СДВИНУТЬ УКАЗАТЕЛЬ
        DEC     R1    ; УМЕНЬШИТЬ СЧЕТЧИК
        BNE     CLR   ; ПОВТОРИТЬ, ЕСЛИ НЕ 0
```

7.2.2. УСЛОВНАЯ ТРАНСЛЯЦИЯ

Последний вариант программы будет одинаково выполняться на СМ1300 и на СМ-4. Используя расширенный набор инструкций СМ-4 (инструкцию SOB), можно заменить последовательность DEC-BNE одной инструкцией:

```
CLR:    CLRB    (R0)+    ; ОЧИСТИТЬ БАЙТ
        SOB     R1, CLR  ; ПОВТОРИТЬ, ЕСЛИ СЧЕТЧИК НЕ 0
```

Полученный вариант занимает всего два слова ОЗУ, но может выполняться только на СМ-4.

Для того чтобы один исходный текст программы после трансляции максимально эффективно выполнялся на обоих комплексах, можно воспользоваться средствами условной трансляции.

```
        ...
.IF     EQ      CM-3    ; НАЧАЛО УСЛОВНОГО БЛОКА ДЛЯ СМ 1300
CLR:    CLRB    (R0)+
        DEC     R1
        BNE     CLR
        .ENDC           ; КОНЕЦ ДЛЯ СМ 1300
.IF     EQ      CM-4    ; НАЧАЛО УСЛОВНОГО БЛОКА ДЛЯ СМ-4
CLR:    CLRB    (R0)+
        SOB     R1, CLR
        .ENDC           ; КОНЕЦ ДЛЯ СМ-4
```

Для получения варианта программы, работающей только на СМ-4, необходимо в начале программы включить оператор

```
CM=4        ; РАБОТА НА КОМПЛЕКСЕ СМ—4
```

Тогда при трансляции приведенного фрагмента во время проверки условий в начале первого блока условной трансляции разность значений символа СМ и константы 3 не равна нулю, и тело блока не транслируется. Во втором блоке условие EQ выполняется, и текст транслируется. Для получения программы, работающей на СМ-4 и на СМ1300, в начале текста необходимо задать СМ-3. При этом будет транслироваться первый блок и не будет транслироваться второй.

Поскольку программа рассчитана либо на СМ1300, либо на СМ-4, используя поддирективы условной трансляции, фрагмент можно записать в виде:

```
        ...
CLR:    CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1, CLR ; ДЛЯ СМ-4
.IFF
        DEC     R1      ; ДЛЯ СМ1300
        BNE     CLR     ; ДЛЯ СМ1300
.ENDC
```

7.2.3. ИСПОЛЬЗОВАНИЕ МАКРОКОМАНД

Операция очистки массива достаточно часто встречается в программах. Для того чтобы многократно не переписывать один и тот же текст, целесообразно воспользоваться средствами макрокоманд. Определим рассмотренный выше фрагмент как макрокоманду с именем CLEAR:

```
.MACRO CLEAR
CLR:    CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1,CLR
.IFF
        DEC     R1
        BNE     CLR
.ENDC
.ENDM   CLEAR
```

Чтобы использовать эту макрокоманду, необходимо поместить показанное определение макрокоманды в начало текста программы.

Пусть необходимо очистить массивы XXX и YYY размером 200 и 300 байт соответственно. Использовав макрокоманду CLEAR, получаем:

```
        ...
        MOV     #XXX,R0
        MOV     #200.,R1
        CLEAR
        MOV     #YYY,R0
        MOV     #300.,R1
        CLEAR
        ...
```

При условии, что CM=4, получаем:

```
        MOV     #XXX,R0
        MOV     #200.,R1
CLR:    CLRB    (R0)+
        SOB     R1,CLR
        MOV     #YYY,R0
        MOV     #300.,R1
CLR:    CLRB    (R0)+
        SOB     R1,CLR
```

Очевидно, что полученная последовательность инструкций (а следовательно, и макроопределение CLEAR) содержит ошибку: метка CLR определена дважды. Кроме того, неудобно перед обращением к макрокоманде повторять инструкции MOV, пересылающие адрес и длину массива. Избежать этого можно, используя аргументы макрокоманд, как показано ниже.

```
.MACRO  CLEAR   ARR,LEN,LABEL
        MOV     #ARR,R0
        MOV     #LEN,R1
```

```
LABEL:  CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1, LABEL
.IFF
        DEC     R1
        BNE     LABEL
.ENDC
.ENDM   CLEAR
```

Обращение к макрокоманде для рассматриваемого случая приобретает вид

```
...
CLEAR XXX, 200., LAB1
CLEAR YYY, 300., LAB2
...
```

а макрорасширение —

```
        MOV     #XXX, R0
        MOV     #200., R1
LAB1:   CLRB    (R0)+
        SOB     R1, LAB1
        MOV     #YYY, R0
        MOV     #300., R1
LAB2:   CLRB    (R0)+
        SOB     R1, LAB2
```

Используемое макроопределение имеет явный недостаток: при вызове

```
...
CLEAR   @R5, R4, LAB3
```

в макрорасширении появляются строки инструкции, содержащие неправильное указание аргументов.

```
...
MOV     #@R5, R0
MOV     #R4, R1
...
```

Эти ошибки связаны с тем, что в макроопределении фиксируется режим адресации аргументов — предполагается использование только непосредственной адресации (режим 27, т. е. #n). Вторым недостатком является необходимость в каждом макровызове явным образом указывать имя метки, внутренней для макроопределения. Перечисленные недостатки отсутствуют в макроопределении:

```
.MACRO  CLEAR   ARR,  LEN,  ?LABEL
        MOV     ARR, R0
        MOV     LEN, R1
LABEL:  CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1, LABEL
.IFF
        DEC     R1
        BNE     LABEL
.ENDC
.ENDM   CLEAR
```

Для этого макроопределения вызовы

```
...
CLEAR   #XXX, #200.
CLEAR   @R5, R4
...
```

приведут к расширению (для СМ-4):

```
                ...
                MOV     #XXX, R0
                MOV     #200., R1
30000◎:         CLRB    (R0)+
                SOB     R1, 30000◎
                MOV     @R5, R0
                MOV     R4, R1
30001◎:         CLRB    (R0)+
                SOB     R1, 30001◎
                ...
```

Этот пример показывает использование возможности автоматической генерации транслятором локальных меток.

Данная макрокоманда имеет еще один недостаток, который заключается в том, что она неявно использует и изменяет содержимое регистров R0 и R1. Это может привести к алгоритмическим ошибкам при работе программы. Для того чтобы избавить программиста от необходимости помнить о таком побочном явлении, достаточно в начале макрокоманды сохранять, а после выполнения восстанавливать содержимое этих регистров. Для этих целей в СМ ЭВМ обычно используется стек. Необходимо отметить, что в отдельных случаях сохранение регистров необязательно.

Поэтому в макрокоманде желательно предусмотреть возможность управления сохранением регистров. Ниже показано возможное решение данной проблемы с использованием ключевого аргумента макрокоманды.

```
.MACRO  CLEAR   ARR,  LEN,  ?LABEL,  SAVE=YES
.IF     IDN     SAVE, YES
        MOV     R0,-(SP)
        MOV     R1,-(SP)
.IFTF
        MOV     ARR, R0
        MOV     LEN, R1
LABEL:  CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1, LABEL
.IFF
        DEC     R1
        BNE     LABEL
.ENDC
.IFT
        MOV     (SP)+, R1
        MOV     (SP)+, R0
.ENDC
.ENDM   CLEAR
```

При использовании такой макрокоманды обращения

```
CLEAR   #XXX, #200.
CLEAR   #YYY, #300., SAVE=YES
CLEAR   @R5, R4, SAVE=NO
```

порождают следующее расширение:

```
                ...
                MOV     R0, -(SP)
                MOV     R1, -(SP)
                MOV     #XXX, R0
                MOV     #200., R1
30000⊙:         CLRB    (R0)+
                SOB     R1, 30000⊙
                MOV     (SP)+, R1
                MOV     (SP)+, R0
                MOV     R0, -(SP)
                MOV     R1, -(SP)
                MOV     #YYY, R0
                MOV     #300., R1
30001⊙:         CLRB    (R0)+
                SOB     R1, 30001⊙
                MOV     (SP)+, R1
                MOV     (SP)+, R0
                MOV     @R5, R0
                MOV     R4, R1
30002⊙:         CLRB    (R0)+
                SOB     R1, 30002⊙
```

7.2.4. ОПТИМИЗАЦИЯ МАКРОКОМАНД

В полученном макрорасширении обращает на себя внимание тот факт, что последние две инструкции первого макрорасширения и первые две второго

```
...
MOV     (SP)+, R1
MOV     (SP)+, R0
MOV     R0, -(SP)
MOV     R1, -(SP)
...
```

являются избыточными, так как могут быть удалены из программы без изменения результата ее работы.

Рассмотрим новый вариант макрокоманды, в которой проводится некоторая оптимизация генерируемой последовательности машинных инструкций за счет дополнительного анализа на этапе трансляции.

```
.MACRO  CLEAR   ARR,  LEN,  ?LABEL,  SAVE=YES
.IF     IDN     SAVE,  YES
.IFF    NDF     FLAG,  FLAG=.+1
.IF     EQ      FLAG-.
        .=.-4
```

```
.IFF
        MOV     R0,-(SP)
        MOV     R1,-(SP)
.ENDC
.IFTF
        MOV     ARR,R0
        MOV     LEN,R1
LABEL:  CLRB    (R0)+
.IF     EQ      CM-4
        SOB     R1,LABEL
.IFF
        DEC     R1
        BNE     LABEL
.ENDC
.IFT
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        FLAG=.
.ENDC
.ENDM   CLEAR
```

Эта оптимизация проводится только для случаев, когда необходимо сохранение (восстановление) регистров R0, R1. Процессом оптимизации управляет символ FLAG.

При первом вызове макрокоманды определяется начальное значение символа FLAG директивой

```
.IIF NDF FLAG, FLAG=.+1
```

В приведенной директиве условной трансляции, если FLAG не определен, проводится его определение путем присваивания ему значения. Это значение выбирается так, чтобы оно не совпадало со значением счетчика адреса (.) — точка. При последующих вызовах макрокоманды значение FLAG будет определено и рассматриваемой директивой не изменяется.

В следующем блоке условной трансляции

```
.IF     EQ      FLAG-.
        .=.-4
.IFF
        MOV     R0,-(SP)
        MOV     R1,-(SP)
.ENDC
```

производится сохранение регистров, если значение FLAG не совпадает с текущим значением счетчика адреса. Таким образом, при первом вызове макрокоманды сохранение выполняется. В конце макрокоманды осуществляется восстановление регистров, а FLAG получает значение счетчика адреса с помощью оператора прямого присваивания

```
FLAG=.
```

При следующих вызовах CLEAR директива

```
.IF EQ FLAG-.
```

фактически проверяет наличие машинных инструкций между двумя последовательными обращениями к макрокоманде. Если значение FLAG не совпадает со счетчиком адреса, операция сохранения производится. В противном случае оператор .=.—4 возвращает счетчик адреса на два слова назад, тем самым удаляя полученные в предыдущем вызове CLEAR инструкции

```
MOV     (SP)+, R1
MOV     (SP)+, R0
```

Например:

```
CLEAR   A, #5
CLEAR   B, #7
```

генерирует для СМ-4 следующую инструкцию:

```
                MOV     R0, -(SP)
                MOV     R1, -(SP)
                MOV     A, R0
                MOV     #5, R1
30000⊙:         CLRB    (R0)+
                SOB     R1, 30000⊙
                MOV     B, R0
                MOV     #7, R1
30001⊙:         CLRB    (R0)+
                SOB     R1, 30001⊙
                MOV     (SP)+, R1
                MOV     (SP)+, R0
```

В некоторых случаях средства Макро позволяют проводить дополнительную оптимизацию в зависимости от фактических аргументов при вызове макрокоманды, а также выявлять семантические ошибки. Например, макрокоманда

```
.MACRO  SEND    ARR,  LEN,  COD,  ?LAB
        MOV     ARR, R0
        MOV     LEN R1
LAB:    MOVB    COD (R0)+
        SOB     R1, LAB
.ENDM   SEND
```

при вызове

```
SEND  R0,  R1, # 0
```

генерирует малоэффективную последовательность инструкций

```
                MOV     R0, R0
                MOV     R1, R1
30000⊙:         MOVB    #0, (R0)+
                SOB     R1, 30000⊙
```

а вызов

```
SEND#MASS, #0, #NOP
```

генерирует последовательность

```
        MOV     #MASS, R0
        MOV     #0, R1
30001⊙: MOVB    #NOP, (R0)+
        SOB     R1, 30001⊙
```

содержащую семантическую ошибку. Ошибка заключается в том, что задается длина массива, равная 0. Это приведет к тому, что тело цикла будет выполнено 65536 (2 в степени 16) раз, поскольку инструкция осуществляет проверку содержимого регистра на 0 после вычитания 1 (таким образом, после первого вычитания 1 будет получено 177777, после второго — 177776 и т. д. до 0).

Перечисленные недостатки устранены в макрокоманде

```
.MACRO  SEND    ARR,    LEN,    COD,    ?LAB
.IF     IDN     <LEN>,  <#0>
        .ERROR  LEN
        .MEXIT
.ENDC
.IIF    DIF     <ARR>, <R0> MOV ARR, R0
.IIF    DIF     <LEN>, <R1> MOV LEN, R1
LAB:
.IF     IDN     <COD>, <#0>
        CLRB    (R0)+
.IFF
        MOVB    COD,    (R0)+
.ENDC
        SOB     R1, LAB
.ENDM   SEND
```

7.2.5. РАБОТА СО СТРОКАМИ

При обработке текстовой информации основной единицей обрабатываемых данных является строка знаков. Строка, хранящаяся в оперативной памяти, обычно содержит от 0 (пустая строка) до 132 знаков. Каждому знаку строки отводится один байт оперативной памяти, причем смежные знаки строки размещаются в последовательных байтах. Признаком конца строки является байт, содержащий 0.

Обработка строк будет рассмотрена на примерах наиболее используемых операций. Достаточно часто производится операция определения длины строки. Пусть адрес начала строки находится в регистре R1, тогда последовательность инструкций

```
        ...
LEN:    MOV     #-1, R0 ;  УСТАНОВИТЬ НАЧАЛЬНОЕ
                        ;  ЗНАЧЕНИЕ СЧЕТЧИКА
1⊙      INC     R0      ;  УВЕЛИЧИТЬ СЧЕТЧИК
        TSTB    (R1)+   ;  ПРОВЕРИТЬ НА ПРИЗНАК КОНЦА
        BNE     1⊙      ;  ПОВТОРИТЬ ДЛЯ СЛЕДУЮЩЕГО ЗНАКА
```

подсчитает количество знаков в строке и поместит результат в регистр R0. Отметим, что после выполнения этого фрагмента программы регистр R1 будет содержать адрес байта, следующего за концом строки (байтом, содержащим 0). Следующий фрагмент:

```
LEN:    CLR     R0
1⊙:     TSTB    (R1)+
        BEQ     2⊙
        INC     R0
        BR      1⊙
2⊙:
```

выполняет ту же операцию. Но хотя оба фрагмента занимают одинаковый объем оперативной памяти, последний выполняется дольше, так как на каждом шаге цикла содержит на одну инструкцию больше. Следует отметить при этом, что байт, содержащий 0, в подсчет числа знаков в строке не входит.

Примером другой часто используемой операции со строками является копирование, которое может быть реализовано с помощью следующей программы:

```
        ...
; +
; R0 СОДЕРЖИТ АДРЕС ПРИЕМНИКА (КУДА)
; R1 СОДЕРЖИТ АДРЕС ИСТОЧНИКА (ОТКУДА)
; -
COPY:   MOVB    (R1)+, (R0)+
        BNE     COPY
        ...
```

Операция конкатенации может быть выполнена следующим образом:

```
; +
; R0 = АДРЕС ПРИЕМНИКА (СТРОКИ-РЕЗУЛЬТАТА)
; R1 = АДРЕС ПЕРВОЙ СТРОКИ
; R2 = АДРЕС ВТОРОЙ СТРОКИ
; -
CONC:
1⊙:     MOVB    (R1)+, (R0)+    ПЕРЕСЫЛКА СТРОКИ 1
        BNE     1⊙
        DEC     R0
2⊙:     MOVB    (R2)+, (R0)+;   ПЕРЕСЫЛКА СТРОКИ 2
        BNE     2⊙
```

В показанном примере инструкция DEC используется для того, чтобы пропустить ограничитель (нулевой байт) первой строки.

При вводе текстовой информации с перфокарт часто приходится сталкиваться с необходимостью отбрасывания незначащих пробелов, которые расположены в конце введенной строки. Эту процедуру может реализовать приводимый фрагмент программы:

```
;  +
;  R1 = АДРЕС СТРОКИ
;  -
TRIM:
1⊙:     TSTB    (R1)+           ; ПОИСК КОНЦА СТРОКИ
        BNE     1⊙
        DEC     R1              ; ПРОПУСТИТЬ 0
2⊙:     CMPB    #40, -(R1)      ; ДА - ПОВТОРИТЬ
        INC     R1              ; НЕТ - ПРОПУСТИТЬ НЕПРОБЕЛ
        CLRB    (R1)+           ; ПОСТАВИТЬ ОГРАНИЧИТЕЛЬ
```

В начале приведенного фрагмента осуществляется поиск конца строки, затем указатель (регистр R1) перемещается по строке в обратном направлении до тех пор, пока не встретится знак, отличный от пробела (код знака «пробел» равен 40_8). После этого последний встретившийся пробел замещается нулем.

Эта программа будет правильно работать, если обрабатываемая строка содержит хотя бы один знак, отличный от пробела.

Фрагмент программы

```
;  +
;  R0 - РАБОЧИЙ БУФЕР
;  R1 - АДРЕС СТРОКИ
;  R2 - ВСПОМОГАТЕЛЬНЫЙ УКАЗАТЕЛЬ
;  -
RBLANK: MOV     R1, R2          ; СКОПИРОВАТЬ АДРЕС СТРОКИ
1⊙:     MOVB    (R2)+, R0       ; ИЗВЛЕЧЬ ЗНАК В БУФЕР
        BEQ     2⊙              ; КОНЕЦ СТРОКИ?
        CMPB    #40, R0         ; НЕТ - СРАВНИТЬ БУФЕР С ПРОБЕЛОМ
        BEQ     1⊙              ; СОВПАЛО - НЕ ПЕРЕСЫЛАТЬ
2⊙:     MOVB    R0, (R1)+       ; НЕ СОВПАЛО ИЛИ КОНЕЦ - ПЕРЕСЫЛАТЬ
        BNE     1⊙              ; ЕСЛИ НЕ КОНЕЦ - ПОВТОРИТЬ
```

реализует операцию удаления пробелов из строки. При этой операции строка копируется сама в себя через промежуточный буфер. Пробелы, обнаруженные в промежуточном буфере, в строку не возвращаются. Следующий пример показывает реализацию более сложной функции: замену нескольких подряд идущих пробелов в строке на один. При этом строка также копируется на себя через рабочий буфер:

```
RBLANK2:
        MOV     R1, R2          ; СКОПИРОВАТЬ АДРЕС СТРОКИ
        MOVB    #40, R3         ; КОД ПРОБЕЛА В РЕГИСТР - ДЛЯ
                                ; СКОРОСТИ
1⊙:     MOVB    (R2)+, R0       ; ИЗВЛЕЧЬ ЗНАК В БУФЕР
        BEQ     3⊙              ;   КОНЕЦ?
        CMPB    R3, R0          ; НЕТ - СРАВНИТЬ С ПРОБЕЛОМ
        BNE     3⊙              ; НЕПРОБЕЛ - ОБОЙТИ ОБРАБОТКУ
                                ; ПРОБЕЛА
        MOVB    R0, (R1)+       ; ЗАПИСАТЬ ПЕРВЫЙ ПРОБЕЛ
2⊙:     MOVB    (R2)+, R0       ; ИЗВЛЕЧЬ
        BEQ     3⊙              ; КОНЕЦ?
        CMPB    R3, R0          ; НЕТ - МОЖЕТ БЫТЬ ПРОБЕЛ?
        BEQ     2⊙              ; ДА - ПРОПУСТИТЬ ЕГО
        MOVB    R0, (R1)+       ; НЕПРОБЕЛ ИЛИ КОНЕЦ - ПЕРЕСЛАТЬ
        BNE     1⊙              ; ЕСЛИ НЕ КОНЕЦ - ПОВТОРИТЬ
```

Этот пример имеет одну особенность по сравнению с предыдущим: для ускорения работы программы константа (код пробела) предварительно заносится в регистр R3. На размер программы операция занесения не повлияла (14 слов оперативной памяти).

7.2.6. ПЕРЕКОДИРОВКА

При работе программы числа и адреса представлены в ячейках оперативной памяти в двоичном коде. Этот вид представления данных удобен для вычислений, но им невозможно пользоваться при вводе информации или выводе ее с терминала или устройства печати. Это связано с тем, что внешнее представление символов имеет специальную форму в соответствии с кодом КОИ-7. Поэтому для обмена с этими устройствами двоичную информацию необходимо преобразовать в символьную (для вывода) или наоборот (при вводе).

Простейшим примером такого преобразования могут служить перекодировки чисел. Рассмотрим фрагмент программы, которая выполняет преобразование трехразрядного двоичного числа в символьный вид.

```
;  +
;  R0 – ЧИСЛО В ДИАПАЗОНЕ ОТ 0 ДО 7
;  –
        MOVB    TABL(R0), R0        ; КОМАНДА ПЕРЕКОДИРОВКИ
;  +
;  R0 – КОД ВОСЬМЕРИЧНОЙ ЦИФРЫ
;  –
        ...
TABL:   .ASCII /01234567/           ; ТАБЛИЦА ПЕРЕКОДИРОВКИ
```

В данном примере содержимое регистра R0 используется для указания смещения по байтной таблице до кода символа. После выполнения команды MOVB регистр будет содержать код соответствующего символа. В этой программе не предполагается упорядоченности кодов символов цифр, а поэтому данный способ может быть рекомендован для самых различных перекодировок, в которых выполняется преобразование байт—байт.

Поскольку в КОИ-7 символы цифр от 0 до 7 имеют последовательные коды от 60 до 67 соответственно, то можно использовать другой (более эффективный, но менее общий) способ перекодировки:

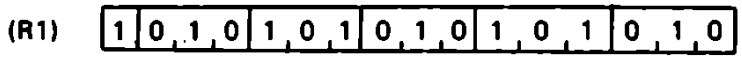
```
ADD #60, R0 ;ИЛИ ADD #'0, R0
```

Используя тот или другой метод перекодировки, можно решить более общую задачу: перекодировку 16-разрядного двоичного числа в восьмеричное символьное.

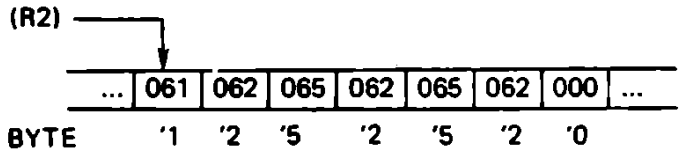
Перекодировку будем вести последовательно, выделяя триады двоичных разрядов и применяя к ним второй способ перекодировки.

```
;  +
;  ПРОГРАММА ПЕРЕКОДИРУЕТ ДВОИЧНОЕ ЧИСЛО ИЗ РЕГИСТРА R1 В
;  СИМВОЛЬНУЮ СТРОКУ, АДРЕС НАЧАЛА КОТОРОЙ НАХОДИТСЯ В R2
;  ИСПОЛЬЗУЮТСЯ РЕГИСТРЫ R0, R3
;  –
DECOD:  ADD     #6, R2          ;  СМЕСТИТЬ УКАЗАТЕЛЬ В КОНЕЦ
        CLRB    @R2             ;  УСТАНОВИТЬ ПРИЗНАК КОНЦА СТРОКИ
        MOV     #5, R0          ;  ЧИСЛО ПЕРЕКОДИРУЕМЫХ ЦИФР
1⊙:     MOV     R1, R3          ;  СКОПИРОВАТЬ ДЛЯ ОБРАБОТКИ
        BIC     #^C7, R3        ;  ВЫДЕЛИТЬ МЛАДШИЕ ТРИ РАЗРЯДА
        ADD     #'0, R3         ;  ПЕРЕКОДИРОВАТЬ
        MOVB    R3, –(R2)       ;  И ПЕРЕСЛАТЬ В БУФЕР
        ASR     R1              ;  СДВИНУТЬ ЧИСЛО
        ASR     R1              ;  НА ТРИ РАЗРЯДА
        ASR     R1              ;  ВПРАВО
        DEC     R0              ;  УМЕНЬШИТЬ СЧЕТЧИК
        BNE     1⊙              ;  ВСЕ ПЕРЕКОДИРОВАНО?
        BIC     #^C1, R1        ;  ДА – ВЫДЕЛИТЬ ПОСЛЕДНИЙ РАЗРЯД
        ADD     #'0, R1         ;  ПЕРЕКОДИРОВАТЬ ЕГО
        MOVB    R1, –(R2)       ;  И ПЕРЕСЛАТЬ В БУФЕР
```

В результате выполнения этой программы число



будет перекодировано и записано в буфер в символьном виде.



Необходимо отметить, что после выполнения регистр содержит адрес начала полученной строки. При перекодировке триады разряды выделялись, начиная с младших.

Рассмотрим пример, в котором выделение разрядов для перекодировки производится, начиная со старших.

```
        ...
DECOD:  MOV     #6, R0          ;  УСТАНОВИТЬ СЧЕТЧИК ЦИФР
        CLR     R3              ;  ОЧИСТИТЬ РАБОЧИЙ РЕГИСТР
        BR      2⊙              ;  ПЕРЕКОДИРОВАТЬ СТАРШИЙ РАЗРЯД
1⊙:     ROL     R1              ;  ПЕРЕНЕСТИ
        ROL     R3              ;  ТРИ СТАРШИХ РАЗРЯДА
        ROL     R1              ;  ИЗ R1 НА МЕСТО
        ROL     R3              ;  ТРЕХ МЛАДШИХ РАЗРЯДОВ
2⊙:     ROL     R1              ;  В РЕГИСТРЕ R3
        ROL     R3              ;  ЧЕРЕЗ БИТ "C" В PS
        ADD     #'0,R3          ;  ПЕРЕКОДИРОВАТЬ В ЦИФРУ
        MOVB    R3,(R2)+        ;  И ПЕРЕСЛАТЬ В БУФЕР
        DEC     R0              ;  ВСЕ ЦИФРЫ ПОЛУЧЕНЫ?
        BNE     1⊙              ;  НЕТ – ПРОДОЛЖИТЬ
        CLRB    @R2             ;  ДА – УСТАНОВИТЬ ПРИЗНАК КОНЦА
        SUB     #6, R2          ;  УКАЗАТЕЛЬ НА НАЧАЛО СТРОКИ
```

В представленной программе для переноса разрядов из регистра R1 в регистр R3 в качестве рабочего буфера неявно использовался бит C из PS. Действия, выполняемые обеими программами, идентичны, хотя первая программа, занимающая на три слова оперативной памяти больше, выполняется несколько быстрее.

7.2.7. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Арифметические операции умножения и деления, реализованные аппаратно в процессоре СМ-4, требуют программного выполнения при работе на СМ1300.

Рассмотрим пример программы, выполняющей умножение целого числа в регистре R0 на 13.

```
MUL13:  MOV     R0,-(SP)        ; R0=X, В СТЕКЕ-X
        ASL     R0              ; R0=2*X
        ADD     @SP,R0          ; R0=3*X
        ASL     R0              ; R0=6*X
        ASL     R0              ; R0=12*X
        ADD     (SP)+,R0        ; R0=13*X
```

В примере показана возможность реализации целочисленного умножения путем комбинирования операций умножения на 2 и сложения. На СМ-4 это можно было бы выполнить одной инструкцией.

```
MUL #13., R0
```

Другим примером программной реализации этих операций может служить умножение на $^3/_4$.

```
; +
; ПЕРВЫЙ ВАРИАНТ
; -
A:      ASR     R0              ; X/2
        MOV     R0,-(SP)        ; ВРЕМЕННО СОХРАН
        ASR     R0              ; X/4
        ADD     (SP)+,R0        ; X/4*3=X/4+X/2
; +
; ВТОРОЙ ВАРИАНТ
; -
B:      MOV     R0,-(SP)        ; R0=X
        ASL     R0              ; 2*X
        ADD     (SP)+,R0        ; 3*X
        ASR     R0              ; 3*X/2
        ASR     R0              ; 3*X/4
```

Рассмотрим более сложный пример умножения двух произвольных 8-разрядных чисел без знака. В результате умножения получается 16-разрядное целое число также без знака:

```
;  +
;  ПРОГРАММА УМНОЖЕНИЯ ВОСЬМИРАЗРЯДНЫХ ЧИСЕЛ
;  БЕЗ ЗНАКА
;  ИСПОЛЬЗУЕМЫЕ РЕГИСТРЫ:
;  R0 – МНОЖИМОЕ
;  R1 – МНОЖИТЕЛЬ
;  R2 – РЕЗУЛЬТАТ (АККУМУЛЯТОР)
;  R3 – СЧЕТЧИК ИТЕРАЦИЙ
;  -
MUL8:   MOV     #8., R3     ;  УСТАНОВИТЬ СЧЕТЧИК
        CLR     R2          ;  ОЧИСТИТЬ НАКОПИТЕЛЬ
1⊙:     ASR     R1          ;  ПРОВЕРИТЬ СЛЕДУЮЩИЙ РАЗРЯД
                            ;  МНОЖИТЕЛЯ
        BCC     2⊙          ;  РАЗРЯД СОДЕРЖИТ НУЛЬ?
        ADD     R0, R2      ;  НЕТ– ВЫПОЛНИТЬ СЛОЖЕНИЕ
2⊙:     ASL     R0          ;  УМНОЖИТЬ МНОЖИМОЕ НА 2
        DEC     R3          ;  УМНОЖЕНИЕ ОКОНЧЕНО?
        BNE     1⊙          ;  НЕТ – ПОВТОРИТЬ ДЛЯ СЛЕДУЮЩЕГО
                            ;  РАЗРЯДА
```

Предложенная программа фактически реализует алгоритм двоичного умножения «в столбик». Это можно продемонстрировать сравнением последовательности действий, выполняемых программой, и действий умножения «в столбик», например чисел 153 и 211.

```
                01 101 011      (153)
                10 001 001      (211)
               ------------
                01 101 011
                       0        ; БИТ .C. ЧИСТ
                      0         ; ПРОПУСТИТЬ СЛОЖЕНИЕ
           01 101 001           ;
                 0              ;
                0               ;
               0                ;
     011 010 11                 ;
-------------------------------------------
0 011 100 100 110 011           ; (034463)
```

7.3. ПОДПРОГРАММЫ

Основой модульного программирования являются подпрограммы, которые позволяют разделять большие и сложные программы на меньшие по размеру и более простые модули. С другой стороны, подпрограммы составляют основу прикладных и системных библиотек, которые могут использоваться несколькими программами, снижая тем самым затраты на их разработку.

Архитектура СМ ЭВМ предоставляет программисту различные средства для работы с подпрограммами.

7.3.1. ПОДПРОГРАММЫ БЕЗ ПАРАМЕТРОВ

В простейшем случае подпрограмма не предполагает наличия параметров. Пример структуры программы, использующей подпрограмму:

```
        .TITLE  PROGRAM
START:  ...                     ; НАЧАЛО ПРОГРАММЫ
        JSR     PC, SUBR        ; ВЫЗОВ ПОДПРОГРАММЫ
        ...
        JSR     PC, SUBR        ; ВЫЗОВ ПОДПРОГРАММЫ
        ...
SUBR:   ...                     ; НАЧАЛО ПОДПРОГРАММЫ
        RTS     PC              ; ВОЗВРАТ ИЗ ПОДПРОГРАММЫ
        .END    START
```

Приведенная программа содержит одну подпрограмму SUBR и дважды обращается к ней. Вызов подпрограммы осуществляется инструкцией JSR, которая запоминает в стеке адрес возврата (адрес инструкции, следующей за JSR) и передает управление на подпрограмму. Инструкция JSR всегда использует аппаратный стек. Возврат из подпрограммы осуществляется по инструкции RTS, которая всегда является последней выполняемой инструкцией подпрограммы. При этом возврат осуществляется по адресу, хранящемуся в вершине аппаратного стека.

В приведенном примере структуры текст подпрограммы содержится (и соответственно транслируется одновременно) вместе с программой, которая ее использует. Для того чтобы подпрограмма стала доступна другим программам, ее необходимо оформить как отдельный модуль, например

```
        .TITLE  SUBROUTINE
SUBR::  ...                     ; ТОЧКА ВХОДА В ПОДПРОГРАММУ
        RETURN
        .END
```

Оформленная таким образом подпрограмма транслируется отдельно и может быть помещена в библиотеку (прикладную или системную) объектных модулей. Точка (или точки, если их несколько) входа в подобную подпрограмму должна быть определена как глобальный символ. В приведенном примере для этой цели используется конструкция «::».

Модуль, использующий такую внешнюю подпрограмму, должен содержать определение точки входа в подпрограмму как глобального символа

```
.GLOBL SUBR
```

Такое явное определение глобального символа рекомендуется выполнять в начале текста программы. Существует способ неявного определения внешнего глобального символа. Для этого используется директива .ENABL GBL, которая определяет все символы, не определенные в данном модуле, как глобальные. Напри-

мер, в приведенной ниже программе символы A и B автоматически объявляются глобальными, а символ C приведет к сообщению об ошибке, если он не будет определен в программе.

```
          .TITLE   PROGRAM
          .ENABL   GBL          ; СИМВОЛ НЕ ОПРЕДЕЛЕН =>
                                ; ГЛОБАЛЬНЫЙ
PROG2:    ...
          ...
          CALL     A
          ...
          CALL     B
          ...
          .DSABL   GBL          ; СИМВОЛ НЕ ОПРЕДЕЛЕН =>
                                ; ОШИБКА
          ...
          CALL     C
          ...
          .END     PROG2
```

Как отмечалось выше, нормальный выход и выход по ошибке из подпрограммы рекомендуется различать по биту C слова состояния процессора. Рассмотрим подпрограмму, имеющую две точки выхода.

```
SUBR0:    ...                 ; НАЧАЛО ПОДПРОГРАММЫ
          ...
NORMA:    CLC                 ; 0 -> C
          RETURN              ; НОРМАЛЬНОЕ ЗАВЕРШЕНИЕ РАБОТЫ
ERROR:    SEC                 ; 1 -> C
          RETURN              ; ВЫХОД ПО ОШИБКЕ
```

При использовании подпрограмма может иметь вид:

```
          ...
          CALL     SUBR0      ; ВЫЗОВ ПОДПРОГРАММЫ
          BCS      BAD        ; ПЕРЕЙТИ НА ОБРАБОТКУ ОШИБКИ
GOOD:     ...                 ; НОРМАЛЬНОЕ ПРОДОЛЖЕНИЕ РАБОТЫ
          ...
          ...
BAD:      ...                 ; ОБРАБОТАТЬ ОШИБКУ
```

При этом управление на метку BAD передается только в случае, если бит C в PS установлен в 1, т. е. произошел выход из подпрограммы SUBR0 по ошибке. В приведенном примере используется тот факт, что выполнение команды RTS не изменит PS.

При написании подпрограмм, которые используют универсальные регистры процессора, возникает проблема сохранения содержимого этих регистров. Процедура сохранения и восстановления регистров может выполняться в вызывающей и в вызываемой программе. В соответствии с рекомендациями, приведенными в начале главы, и с целью уменьшения объема вызывающей про-

граммы более эффективно выполнять эти процедуры в подпрограмме. Например:

```
;  +
;  ПОДПРОГРАММА ИСПОЛЬЗУЕТ ДЛЯ РАБОТЫ РЕГИСТРЫ 0 И 1
;  –
SUBR1:  MOV     R0, – (SP)          ; СОХРАНИТЬ
        MOV     R1, – (SP)          ; РЕГИСТРЫ
        ...
NORMA:  CLC
        BR      RET
ERROR:  SEC
RET:    MOV     (SP) +, R1          ; ВОССТАНОВИТЬ
        MOV     (SP) +, R0          ; РЕГИСТРЫ
        RETURN                      ; ВЕРНУТЬ УПРАВЛЕНИЕ
```

В этой подпрограмме используется свойство инструкции не изменять значение бита С.

В случае большого числа подпрограмм, использующих регистры, наиболее целесообразно выделение процедур сохранения и восстановления в отдельные подпрограммы. Это позволяет за счет некоторой избыточной для тех или иных подпрограмм работы со всеми регистрами уменьшить общий объем программы. Для этих целей может быть предложена следующая подпрограмма:

```
;  +
;  ПРОГРАММА СОХРАНЯЕТ И ВОССТАНАВЛИВАЕТ РЕГИСТРЫ 0 – 5
;
;  ФОРМАТ ВЫЗОВА:     JSR    R0, SAVE05
;  –
SAVE05:  .IRPC   X, 12345
         MOV     R'X, – (SP)
         .ENDR
L3:      CALL    @R0
L4:      .IRPC    X, 543210
         MOV     (SP) +, R'X
         .ENDR
L5:      RETURN
```

Примером использования этой подпрограммы может служить программа:

```
         CALL    SUBR2               ; ВЫЗОВ ПОДПРОГРАММЫ
L1:      BCS     ERROR
         ...
SUBR2:   JSR     R0, SAVE05          ; СОХРАНЕНИЕ РЕГИСТРОВ
L2:      ...
NORMA:   CLC                         ; ВЫХОД ИЗ ПОДПРОГРАММЫ
         RETURN                      ; С ВОССТАНОВЛЕНИЕМ
                                     ; РЕГИСТРОВ
ERROR:   SEC                         ; ОТ R0 ДО R5
         RETURN                      ; ВКЛЮЧИТЕЛЬНО
```

При вызове подпрограммы SUBR2 в вершине стека записан адрес возврата (рис. 7.1, *а*, адрес L1). Первая инструкция подпрограммы производит обращение к SAVE05. При этом в стек

заносится содержимое регистра R0, а в регистр записывается адрес возврата (рис. 7.1, б, адрес L2). Далее в стек заносится содержимое регистров 1—5 (рис. 7.1, в, точка L3) и производится обращение к подпрограмме по адресу, находящемуся в R0, т. е. передается управление на метку L2, а в стек заносится адрес возврата (рис. 7.1, г, адрес L4). При выполнении в программе SUBR2 инструкции RETURN управление передается на метку L4 (рис. 7.1, д) и происходит восстановление регистров R0—R5. После восстановления регистров в вершине стека находится адрес L1 (рис. 7.1, е), и по инструкции RETURN из подпрограммы SAVE05 управление возвращается в вызвавшую программу.

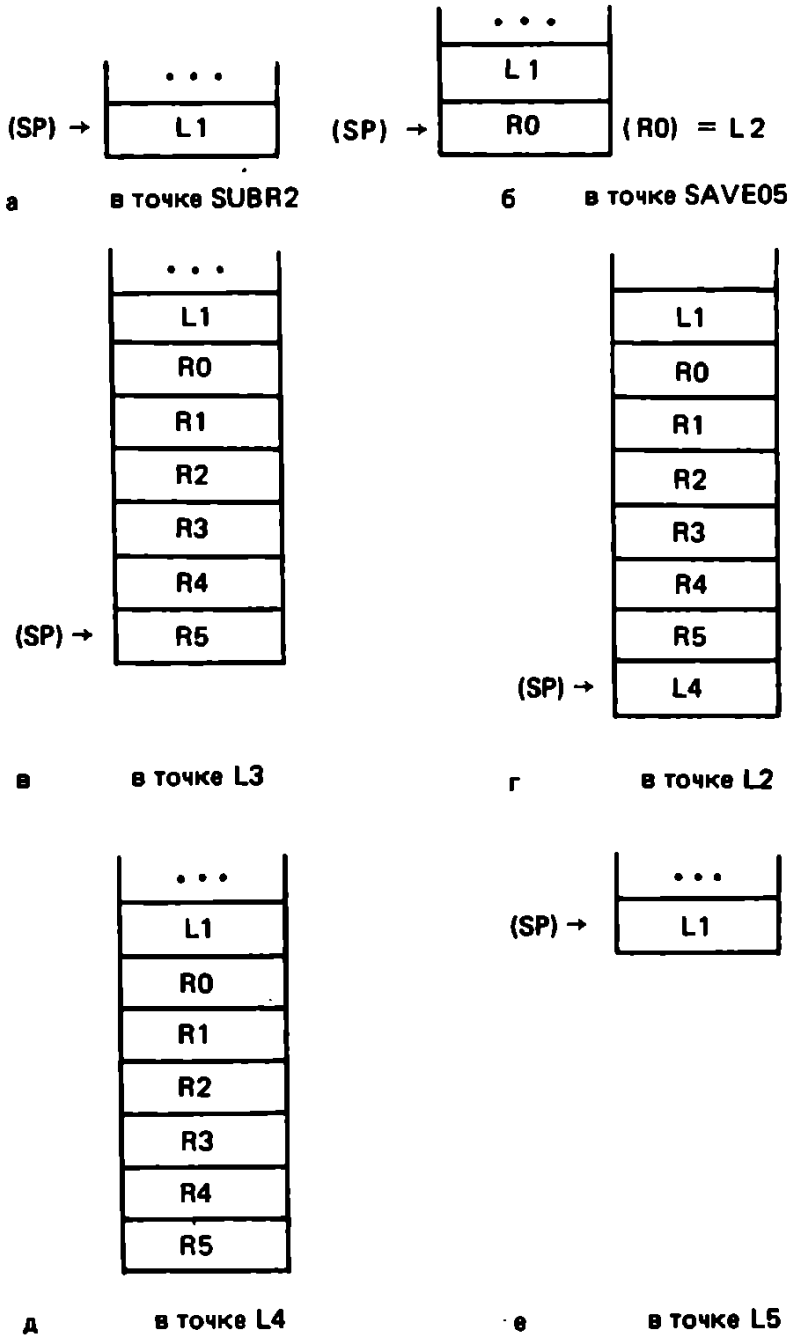


Рис. 7.1. Последовательные состояния стека при работе с подпрограммой SAVE05

Во время выполнения этих действий стек последовательно принимает вид, изображенный на рисунке.

Необходимо отметить, что при использовании подпрограммы SAVE05 для сохранения регистров передавать данные в подпрограмму через регистр R0 не следует, так как при обращении к SAVE05 его содержимое будет изменено. Для того чтобы подпрограмма SUBR2 получила значение, находившееся в регистре, ей необходимо обращаться по адресу 14(SP). Возвращать результаты работы подпрограммы в вызывающую программу через регистры невозможно, так как их содержимое восстанавливается.

Рассмотренные примеры подпрограмм не использовали явным образом параметры для обмена информацией с вызвавшей ее подпрограммой. Из этого не следует, что подобные программы вообще не обмениваются информацией, поскольку для этой цели могли использоваться глобальные (т. е. доступные из нескольких модулей) переменные или области памяти. В языке Макро в качестве подобной глобальной среды ссылок можно использовать, например ячейки с метками, объявленными глобальными, или программные секции (аналогично блокам COMMON в языке Фортран). Использование глобальной среды ссылок для обмена информацией между модулями позволяет повысить скорость выполнения программ, но затрудняет их отладку по сравнению с явной передачей параметров. Достаточно детально вопросы передачи управления и данных обсуждаются в [3].

7.3.2. ПОДПРОГРАММЫ С ПАРАМЕТРАМИ

Простейшим способом передачи параметров в подпрограмму является использование универсальных регистров. Регистры при этом могут использоваться как для передачи непосредственно данных, так и для передачи их адресов. В случае применения регистров для возврата результатов они используются последовательно, начиная с R0. Например, если подпрограмма возвращает два слова, то используются регистры R0, R1.

Более сложным способом передачи параметров является использование таблицы, содержащей значения или адреса параметров. Для хранения адреса этой таблицы обычно используется регистр R5.

```
        ...
        JSR     R5, SUM
        .WORD   100
        .WORD   200
        MOV     R0, RES
        ...
SUM:    MOV     (R5)+, R0       ; ПЕРЕСЛАТЬ АРГУМЕНТ 1
        ADD     (R5)+, R0       ; ПРИБАВИТЬ АРГУМЕНТ 2
        RTS     R5
```

После выполнения инструкции JSR содержимое регистра R5 занесено в вершину стека, а в R5 находится адрес возврата, т. е. адрес слова, следующего за инструкцией. Это дает возможность подпрограмме, использовав режим автоувеличения для пятого регистра, получить доступ к обрабатываемым данным. После извлечения и использования данных R5 изменяется и будет содержать новый адрес возврата. Таким образом, после выполнения инструкции RTS R5 управление будет передано на инструкцию MOV, следующую за таблицей (содержимое R5 восстанавливается из стека).

Преимуществом продемонстрированного способа является высокая скорость доступа к параметрам вызова. Вместе с тем необходимо отметить и целый ряд особенностей: предполагается фиксированное число аргументов; за каждой точкой вызова должна следовать таблица аргументов, в которой размещаются сами аргументы, а не их адреса.

Рассмотрим еще один случай использования таблицы параметров при вызове подпрограммы

```
        ...
        JSR     R5,SUM1
        BR      5Ø
        .WORD   ARG1
        ...
        .WORD   ARGN
5Ø:
```

В этом варианте после вызова подпрограммы пятый регистр указывает на инструкцию BR, с помощью которой производится «обход» таблицы. В этих условиях возможен доступ к аргументам при помощи индексации R5.

```
2(R5)           ; ПЕРВЫЙ АРГУМЕНТ
4(R5)           ; ВТОРОЙ АРГУМЕНТ
....
2*N(R5)         ; АРГУМЕНТ НОМЕР N
```

При этом содержимое R5 не изменяется, и при выполнении инструкции RTS R5 управление передается на BR. Во время работы подпрограммы доступ к аргументам не изменяется в силу сохранения соответствия между смещением от указателя до аргументов и аргументами.

Учитывая двоичный формат инструкции BR, подпрограмма может получить информацию о числе аргументов. Поскольку предполагается, что каждый аргумент занимает в таблице одно слово, а BR осуществляет переход на слово, следующее за таблицей, младший байт инструкции BR, содержащий смещение, является счетчиком аргументов.

Рассмотрим пример подпрограммы, использующей переменное число аргументов.

```
SUM1:   MOV     (R5)+, R1       ; ВЗЯТЬ ИНСТРУКЦИЮ BR В R1
        CLR     R0              ; ОЧИСТИТЬ АККУМУЛЯТОР
1⊙:     ADD     (R5)+, R0       ; ПРИБАВИТЬ ОЧЕРЕДНОЙ АРГУМЕНТ
        DECB    R1              ; УМЕНЬШИТЬ СЧЕТЧИК
        BNE     1s              ; ПОВТОРИТЬ, ЕСЛИ СЛАГАЕМЫЕ НЕ ВСЕ
        RTS     R5              ; ВЕРНУТЬСЯ ЗА ТАБЛИЦУ
```

В этом примере младший байт первого регистра используется как счетчик аргументов, позволяя тем самым подпрограмме работать с переменным числом аргументов без модификаций алгоритма. Алгоритм подпрограммы SUM1 строится из расчета, что число аргументов находится в пределах от 1 до 127. Это ограничение связано с форматом инструкции BR, которая не может передать управление больше чем на 127 слов вперед. Можно отказаться от инструкции BR и использовать для счетчика все 16 разрядов занимаемого ею слова.

```
        JSR     R5, SUM2
        .WORD   <5⊙-.-2>/2      ; ЧИСЛО АРГУМЕНТОВ
        .WORD   ARG1
        ...
        .WORD   ARGN
5⊙:     ...
        ...
SUM2:   CLR     R0
        MOV     (R5)+, R1
        BEQ     10⊙             ; ЕСЛИ ТАБЛИЦА ПУСТА
5⊙:     ADD     (R5)+, R0
        DEC     R1
        BNE     5⊙
10⊙:    RTS     R5
```

В последнем примере производится проверка случая нулевого числа аргументов.

Рассмотрим случай, когда передаются не сами аргументы, а их адреса:

```
        JSR     R5, SUM3
        BR      5⊙
        .WORD   ADR1            ; АДРЕС АРГУМЕНТА 1
        ...
        .WORD   ADRN            ; АДРЕС АРГУМЕНТА N
5⊙      ...
        ...
ADR1:   .WORD   ARG1            ; АРГУМЕНТ 1
        ...
ADRN:   .WORD   ARGN            ; АРГУМЕНТ N
        ...
SUM3:   MOV     (R5)+, R1
        CLR     R0
1⊙:     ADD     @(R5)+, R0
        DECB    R1
        BNE     1⊙
        RTS     R5
```

В самой подпрограмме SUM3 по сравнению с подпрограммой SUM1 присутствует только одно изменение: в инструкции ADD адресация источника ведется не через (R5)+, а через @ (R5)+.

Общим недостатком рассмотренных способов передачи параметров является необходимость размещения таблицы аргументов или их адресов непосредственно за точкой вызова подпрограмм.

С целью избежания этого ограничения рекомендуется использовать для вызова подпрограммы и возврата управления регистр PC как регистр связи, а для передачи адреса таблицы по-прежнему использовать пятый регистр. При этом таблица аргументов может размещаться в любом месте программы, в том числе и в аппаратном стеке.

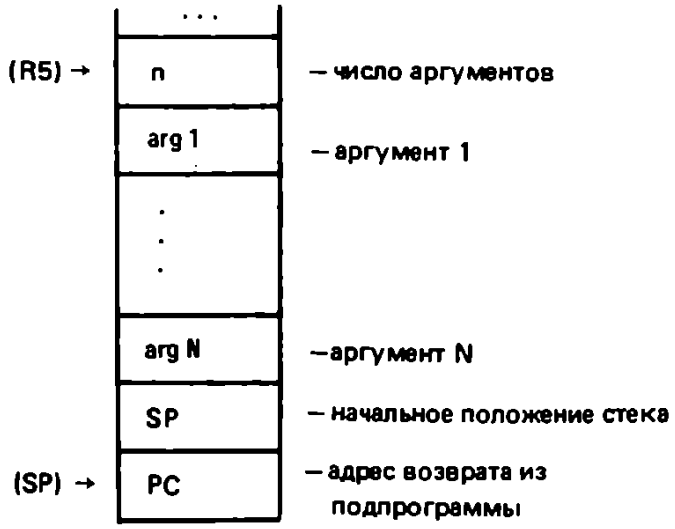
```
            ...
            MOV     #TABARG, R5
            CALL    SUM4                ; ПЕРВЫЙ ВЫЗОВ
            ...
            MOV     # TABARG, R5
            CALL    SUM4                ; ВТОРОЙ ВЫЗОВ
            ...
TABARG:     .WORD   <TAEND-2>/2         ; ЧИСЛО АРГУМЕНТОВ
            .WORD   AGR1
            ...
            .WORD   ARGN
TAEND:      ...                         ; МЕТКА КОНЦА ТАБЛИЦЫ
            ...
SUM4:       CLR     R0
            MOV     (R5)+, R1
            BEQ     10⊙
5⊙:         ADD     (R5)+, R0
            SOB     R1,5⊙               ; ТОЛЬКО ДЛЯ СМ-4
10⊙:        RETURN
```

Приведенный пример показывает возможность разных вызовов подпрограммы с одной таблицей. Изменить этот пример для вызовов подпрограммы с использованием разных таблиц или разных подпрограмм с одной таблицей аргументов несложно.

Рассмотрим теперь пример динамического построения таблицы аргументов в стеке при вызове той же подпрограммы.

```
...
MOV     SP, R5              ; ЗАФИКСИРОВАТЬ ПОЛОЖЕНИЕ СТЕКА
SUB     #N*2, SP            ; ЗАРЕЗЕРВИРОВАТЬ МЕСТО ДЛЯ
                            ; ТАБЛИЦЫ
MOV     R5,-(SP)            ; ЗАПОМНИТЬ ПОЛОЖЕНИЕ СТЕКА В
                            ; СТЕКЕ
MOV     #N, -(R5)           ; ЗАПИСАТЬ ЧИСЛО АРГУМЕНТОВ В
                            ; ТАБЛИЦУ
MOV     R5,-(SP)            ; ЗАПОМНИТЬ НАЧАЛО ТАБЛИЦЫ
MOV     #ARG1,-(R5)         ; ЗАПИСАТЬ 1-й АРГУМЕНТ
...
MOV     #ARGN,-(R5)         ; ЗАПИСАТЬ N-й АРГУМЕНТ
MOV     (SP)+, R5           ; R5 = НАЧАЛО ТАБЛИЦЫ
CALL    SUM4                ; ВЫЗВАТЬ ПОДПРОГРАММУ
MOV     @SP, SP             ; ВОССТАНОВИТЬ ПЕРВОНАЧАЛЬНЫЙ
                            ; СТЕК
                            ; БИТ "С" НЕ ИЗМЕНЯЕТСЯ
```

При входе в подпрограмму стек будет иметь вид:



Такую последовательность инструкций полезно оформить в виде макрокоманды:

```
.MACRO  SCALL   SUBR, ARGS
.IRP    Z, <ARGS>               ; ПРИСВОИТЬ СИМВОЛУ N
                                ; ЗНАЧЕНИЕ ЧИСЛА АРГУМЕНТОВ
        .NARG   N               ; ВЫЗОВА ПОДПРОГРАММЫ
        .MEXIT
.ENDR
        MOV     SP, R5
        SUB     #N*2, SP
        MOV     R5, -(SP)
        MOV     #N, -(R5)
        MOV     R5, -(SP)
.IRP    Z, <ARGS>
        MOV     Z, -(R5)
.ENDR
        MOV     (SP)+, R5
        CALL    SUBR
        MOV     @SP, SP
.ENDM
```

Обращение к этой макрокоманде может иметь следующий вид:

SCALL SUM4, <R3,#6, ABC, TAB(R1)>

Эта макрокоманда приведет к построению в аппаратном стеке таблицы из пяти слов, вызовет подпрограмму SUM4 и после возврата управления очистит стек.

7.3.3. ТАБЛИЧНЫЕ ПОДПРОГРАММЫ

Описанные примеры работы с подпрограммами не исчерпывают всех возможностей архитектуры СМ ЭВМ. Примером нетрадиционного подхода к организации подпрограмм могут служить табличные подпрограммы.

Любую программу (или ее фрагмент) можно представить в виде последовательности вызовов подпрограмм, например

```
CALL    SUBR1
CALL    SUBR2
...
CALL    SUBRN
...
```

В этом случае обращает на себя внимание тот факт, что инструкции вызова подпрограмм CALL являются избыточными в смысле описания логики работы. Поэтому может возникнуть естественное желание убрать из программы инструкции CALL, оставив только последовательность имен вызываемых подпрограмм. Эту последовательность имен можно организовать в виде таблицы адресов точек входа вызываемых подпрограмм. Тогда таблица будет иметь вид:

```
...
.WORD   SUBR1
.WORD   SUBR2
...
.WORD   SUBRN
...
```

При работе с такой таблицей необходимо иметь указатель, при помощи которого можно следить, в каком месте таблицы (т. е. на каком этапе действий) находится программа. Таким указателем может служить адрес элемента таблицы, показывающий, какая подпрограмма в данный момент выполняется или какая будет выполняться следующей. Последовательно перемещая этот указатель по таблице и вызывая соответствующие подпрограммы, можно реализовать требуемый алгоритм.

Пусть в качестве указателя используется регистр R4, указывающий на подпрограмму, которая будет выполняться следующей. Тогда для передачи управления между подпрограммами можно использовать инструкцию

```
JMP@ (R4)+
```

Структура каждой подпрограммы при этом будет иметь вид

```
SUBR: ...
      ...
      JMP@ (R4)+
```

При выполнении инструкции JMP управление передается на подпрограмму, адрес которой указывается регистром R4, а содержимое R4 увеличивается на 2 и указывает адрес следующей подпрограммы.

Начало и конец последовательности могут быть оформлены, как показано ниже.

```
          ...
START:    MOV     #1⊙, R4          ; ЗАПУСК
          JMP     @(R4)+           ; "ЦЕПОЧКИ"
1⊙:       .WORD   SUBR1            ; ПОДПРОГРАММА 1
          ...
          .WORD   SUBRN            ; ПОДПРОГРАММА N
          .WORD   .+2              ; КОНЕЦ (ВЫХОД ИЗ "ЦЕПОЧКИ")
```

Или для удобства могут быть записаны с помощью макрокоманд:

```
          .MACRO  START   ?LABEL
                  MOV     #LABEL, R4
                  JMP     @(R4)+
          LABEL:
          .ENDM   START
          .MACRO  FINISH
                  .WORD   .+2
          .ENDM   FINISH
          .MACRO  NEXT
                  JMP     @(R4)+
          .ENDM   NEXT
```

Обращение к макрокомандам выглядит следующим образом:

```
                  ...
START:    START
                  SUBR1
                  SUBR2
                  ...
                  SUBRN
          FINISH
          ...
SUBR1:    ...
          NEXT
          ...
```

Необходимо заметить, что таблица, по которой производится вызов подпрограмм, может содержать не только адреса подпрограмм, но и значения или адреса аргументов. Форма записи программы при этом остается неизменной.

```
START:    START
          SUBR1   A1, ..., AI
          SUBR2   B1, ..., BJ
          ...
          SUBRN   C1, ..., CK
          FINISH
```

Примером подпрограммы, использующей такой вызов и передачу параметров, может служить подпрограмма, находящая сумму трех чисел, заданных адресами.

```
SUM10:    MOV     @(R4)+, R0
          ADD     @(R4)+, R0
          ADD     @(R4)+, R0
          NEXT
```

Подпрограммы, использующие стандартные соглашения (CALL, RETURN), также могут быть организованы в таблицу при использовании следующих макрокоманд:

```
 MACRO  START1 ?LABEL0,   ?LABEL1
        MOV    #LABEL1, R4
LABEL0: CALL   @(R4)+
        BR     LABEL0
LABEL1:
 .ENDM  START1
 .MACRO  FINIS1 ?LABEL2
         .WORD LABEL2
LABEL2: TST     (SP)+
 .ENDM  FINIS1
```

Наконец, для того чтобы подпрограмму, использующую стандартные соглашения, вызвать из «цепочки», нужна подпрограмма

```
USUAL: CALL@ (R4)+
       NEXT          ; JMP @ (R4)+
```

которая вызывается из цепочки, как показано ниже

```
       ...
USUAL  SUBRI
```

где SUBRI — адрес подпрограммы, использующей стандартные соглашения, на которую необходимо передать управление.

7.3.4. TRAP-ПОДПРОГРАММЫ

TRAP-инструкции в архитектуре СМ ЭВМ обеспечивают альтернативный JSR способ работы с подпрограммами. Они обладают теми же возможностями, что и JSR. Инструкция TRAP занимает одно слово, в то время как JSR — почти всегда два слова. Кроме того, младший байт инструкции TRAP может использоваться для передачи в программу обработки дополнительной информации. Однако на выполнение инструкции TRAP затрачивается больше времени. Поэтому TRAP-подпрограммы рекомендуется использовать в тех случаях, когда точек вызова этих подпрограмм достаточно много и необходимо экономить память, однако управление этим подпрограммам передается сравнительно редко и в ситуациях, не критичных во времени. Характерным примером такого использования инструкции TRAP может служить реализация распечатки диагностических сообщений в компиляторе с языка Фортран IV.

При выполнении инструкции TRAP происходит (как уже отмечалось) программное прерывание по вектору с адресом 34: текущее слово состояния процессора PS и счетчик инструкций PC запоминаются в аппаратном стеке; новое PS выбирается из второго слова вектора (ячейка 36), а новое PC — из первого (ячейка 34). Таким образом, управление передается на программу обработки

прерывания. Возврат управления в вызывающую программу производится по инструкции RTI, восстанавливающей прежние PC и PS из стека.

Параметры в подпрограмму, вызванную с помощью TRAP, могут передаваться любым из рассмотренных выше способов.

Используя младший байт инструкции TRAP для идентификации подпрограмм, можно с помощью этой инструкции обращаться не к одной, а к нескольким различным подпрограммам. Выбор нужной подпрограммы по значению, записанному в младшем байте, осуществляется в таком случае специальной программой, называемой TRAP-диспетчером.

TRAP-диспетчер может иметь следующую принципиальную структуру:

```
          .ASECT
          .=34                          ; ВЕКТОР ИНСТРУКЦИИ
           .WORD   DISPIR               ; АДРЕС TRAP-ДИСПЕТЧЕРА
           .WORD   0                    ; PS
           .CSECT
DISPIR:    .IRPC   X,012345             ; ВХОД В ПРЕРЫВАНИЕ ПО TRAP
          MOV      R'X,-(SP)            ; СОХРАНИТЬ R'X В СТЕКЕ
           .ENDR
          MOV      14(SP),R0            ; АДРЕС ВОЗВРАТА => R0
          MOV      -(R0),R0             ; ПЕРЕСЛАТЬ TRAP-
                                        ; ИНСТРУКЦИЮ
          BIC      #177600,R0           ; ВЫДЕЛИТЬ МЛАДШИЙ
                                        ; БАЙТ
          ASL      R0                   ; ПОЛУЧИТЬ СМЕЩЕНИЕ
                                        ; ПО ТАБЛИЦЕ
          CALL     @TABL(R0)            ; ВЫЗВАТЬ НУЖНУЮ
                                        ; ПОДПРОГРАММУ
          .IRPC    X,543210
          MOV      (SP)+,R'X            ; ВОССТАНОВИТЬ R'X
          .ENDR
          RTI                           ; ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ
TABL:     .WORD    SUBR0
          .WORD    SUBR1
          ...
          .WORD    SUBRN
SUBR0:    ...
          RETURN
SUBR1:    ...
          RETURN
          ...
SUBRN:    ...
          RETURN
```

Обращение к подпрограмме, находящейся в этом диспетчере, производится по инструкции

TRAP n

где n — номер строки в таблице подпрограмм диспетчера.

Таким образом, обращение к подпрограмме SUBRX, которое обычно выполняется с помощью CALL SUBRX и занимает два

слова, в случае использования TRAP имеет вид TRAP 12 и занимает одно слово оперативной памяти. Это обращение предполагает, что SUBRX внесена в TRAP-диспетчер и имеет номер 12.

При выполнении этой инструкции управление передается диспетчеру (на точку DISPTR). В диспетчере производится сохранение регистров R0—R5. Затем из стека извлекается адрес возврата, с помощью которого в регистр R0 пересылается слово, содержащее саму TRAP-инструкцию. Эта пересылка необходима для того, чтобы после очистки старшего байта, содержащего код инструкции TRAP, получить в младшем байте номер строки в таблице, содержащей адрес необходимой подпрограммы. Управление на подпрограмму передается инструкцией CALL и после выполнения подпрограммы производится восстановление регистров и выход из прерывания по инструкции RTI.

Для удобства обозначений можно номер строки в таблице определить как символ, задав его значение операторам прямого присваивания NSUBRX=12. При этом обращение имеет вид:

```
TRAP NSUBRX
```

Еще один способ использования символических имен CSUBRX==TRAP+NSUBRX с вызовом подпрограммы по мнемонике выглядит так:

```
CSUBRX      ;ВЫЗОВ SUBRX ЧЕРЕЗ TRAP 12
```

7.4. ПОЗИЦИОННО НЕЗАВИСИМОЕ ПРОГРАММИРОВАНИЕ

Результатом работы транслятора является перемещаемый объектный модуль. Компоновщик объединяет один или несколько модулей и создает выполняемый загрузочный модуль. Однажды созданный загрузочный модуль обычно может быть загружен и выполнен только на адресах памяти, указанных компоновщиком. Это связано с тем, что компоновщик изменяет некоторые инструкции, чтобы отразить распределение памяти, в которой должна выполняться программа. Такая программа называется *позиционно зависимой* (т. е. зависимой от адресов, на которые она настроена). Процессоры СМ ЭВМ обеспечивают режимы адресации, которые дают возможность писать программы, не зависящие от адресов памяти. Такие программы называются *позиционно независимыми* и могут загружаться и выполняться с любого адреса.

Для мультипрограммных операционных систем важно, чтобы несколько задач могли разделять некоторую область общих кодов, например библиотеку подпрограмм. Такие разделяемые программы должны быть позиционно независимыми. Написание позиционно независимых программ требует правильного использования режимов адресации. Все способы адресации, включающие только ссылки на регистры, являются позиционно независимыми. К ним относятся: R — прямое обращение к регистру; (R) — косвен-

ное обращение к регистру; (R)+ — адресация с автоувеличением; @(R)+ — косвенная адресация с автоувеличением; —(R) — адресация с автоуменьшением и @—(R) — косвенная адресация с автоуменьшением. Если используются эти способы адресации, гарантируется позиционная независимость.

Относительные способы адресации являются также позиционно независимыми, если ссылка на относительный адрес стоит в перемещаемой инструкции. Относительные способы адресации следующие: E — относительная адресация, @E — относительно косвенная адресация.

Относительные способы адресации являются позиционно зависимыми, если из перемещаемой инструкции ссылаются на абсолютный адрес. В этом случае, чтобы сделать ссылку позиционно независимой, следует использовать абсолютную адресацию вида @#E.

Индексные способы адресации также могут быть позиционно независимыми либо позиционно зависимыми. Возможны следующие индексные способы адресации: E(R) — индексная, @E(R) — косвенная индексация.

Если база E является абсолютным значением (например, смещением в таблице), то ссылка будет позиционно независимой, например

```
        MOV     2(SP), R0       ; ПОЗИЦИОННО НЕЗАВИСИМАЯ
N=20                            ; АДРЕСАЦИЯ
        MOV     N(R4), R1       ; ПОЗИЦИОННО НЕЗАВИСИМАЯ
                                ; АДРЕСАЦИЯ
```

Если же X — относительный адрес, ссылка позиционно зависимая, например

```
CLR ADDR(R1)        ;ПОЗИЦИОННО ЗАВИСИМАЯ АДРЕСАЦИЯ
```

Непосредственная адресация может быть позиционно независимой или позиционно зависимой. Формат непосредственной адресации:

#N — прямое обращение к непосредственному аргументу

Если значение N определяется абсолютным выражением, то это позиционно независимая адресация. Если значение N есть относительное выражение, имеет место позиционно зависимая адресация. Таким образом, непосредственная адресация является позиционно независимой, если N — абсолютное значение.

Косвенное обращение к непосредственному аргументу является позиционно независимым только в том случае, когда имеет место ссылка на абсолютный адрес. Формат косвенного обращения к непосредственному аргументу:

@#E — абсолютная адресация

Язык Макро обеспечивает способ проверки позиционно независимых кодов. Слово, которое требует от компоновщика выполнения коррекции, транслятор отмечает в листинге знаком апост-

рофа (′). В некоторых случаях этот знак означает позиционно зависимую инструкцию, в других случаях он только обращает внимание на использование символа, который может быть как позиционно зависимым, так и позиционно независимым.

Знак ′ появляется в листинге трансляции в следующих случаях:

1. Абсолютная адресация помечается знаком ′, если ссылка является относительной; ссылки не помечаются, если они являются позиционно независимыми (т. е. абсолютными), например

```
MOV     @#ADDR, R0      ; ПОЗИЦИОННО НЕЗАВИСИМЫЙ КОД
                        ; ТОЛЬКО ЕСЛИ ADDR –
                        ; АБСОЛЮТНЫЙ АДРЕС
```

2. Индексация и косвенная индексация помечаются знаком ′ если смещение относительное, например

```
MOV     ADDR(R1), R2    ; ПОЗИЦИОННО ЗАВИСИМЫЙ КОД
MOV     @ADDR(R1), R2   ; ЕСЛИ ADDR – ОТНОСИТЕЛЬНАЯ
                        ; ВЕЛИЧИНА
```

3. Относительная и косвенная относительная адресации помечаются знаком ′, если указанный адрес определен в другой программной секции, например

```
MOV     ADDR, R2        ; ПОЗИЦИОННО ЗАВИСИМЫЕ КОДЫ
MOV     @ADDR, R2       ; ЕСЛИ ADDR ПРИНАДЛЕЖИТ
                        ; ДРУГОЙ ПРОГРАММНОЙ СЕКЦИИ
```

4. Непосредственная адресация к относительным ячейкам также помечается знаком ′, например

```
MOV     #3, R1          ; ВСЕГДА ПОЗИЦИОННО
                        ; НЕЗАВИСИМЫЙ КОД
MOV     #ADDR, R1       ; ПОЗИЦИОННО ЗАВИСИМЫЙ КОД
                        ; ЕСЛИ ADDR – ОТНОСИТЕЛЬНАЯ
                        ; ВЕЛИЧИНА
```

Есть один случай, когда Макро не помечает позиционно зависимую ссылку. Это происходит в том случае, если относительная ссылка сделана на абсолютную ячейку из перемещаемой инструкции.

Те ссылки, которые требуют от компоновщика сложной коррекции, помечаются в листинге трансляции особым образом. Простые глобальные ссылки помечаются буквой G. Ссылки, определенные как сложные относительные выражения, помечаются буквой С. В этих случаях бывает трудно установить, какие из ссылок такого типа являются позиционно независимыми. Однако в общем случае для определения адреса, получаемого компоновщиком, можно руководствоваться принципами, которые здесь описывались.

8

ГЛАВА

ПРОГРАММИРОВАНИЕ ВВОДА-ВЫВОДА

Рассмотрим способы и приемы программирования внешних устройств. Описание проводится без ориентации на особенности конкретной операционной системы.

Распределение адресов регистров ВУ и ячеек памяти на ОШ [1]

Адреса регистров устройств	760000—777777
Оперативная память	000000—757777
Векторы ловушек и прерываний	000000—000777

Распределение адресов регистров ВУ

Адреса стандартных устройств СМ ЭВМ	770000—777777
Адреса нестандартных устройств	764000—767777
Адреса устройств, разрабатываемых пользователем	760000—763777
Адреса диагностических устройств	760000—760007

Внешние устройства, входящие в состав комплекса СМ ЭВМ, могут быть разделены на две группы в зависимости от наличия или отсутствия прямого доступа к оперативной памяти со стороны устройства. Наличие прямого доступа позволяет устройству производить обмен данными с памятью без участия центрального процессора. Обычно такой обмен осуществляется большими порциями информации (блоками), достигающими в отдельных случаях нескольких тысяч слов. Метод прямого доступа используется, как правило, с устройствами внешней памяти, обладающими высоким быстродействием.

При отсутствии прямого доступа обмен осуществляется по одному слову (или байту). Этот способ применяется для медленных устройств ввода-вывода. При этом центральный процессор контролирует передачу каждого слова или байта. Очевидно, что на-

[1] Здесь и далее предполагаются комплексы с объемом ОЗУ 128 Кслов.

личие большого количества устройств без прямого доступа приводит к существенной загрузке процессора. Преимуществом данного способа по сравнению с прямым доступом является простота аппаратного исполнения интерфейса.

Программа может работать с внешними устройствами двумя способами: по опросу готовности или по прерываниям. Выбор метода управления внешним устройством принципиально не зависит от способа обмена информацией между устройством и оперативной памятью.

При работе по готовности программа должна выполнить следующие действия: инициировать операцию на устройстве; перейти в состояние ожидания, периодически проверяя завершенность операции; дождавшись завершения операции, проверить отсутствие ошибки при ее выполнении.

Недостатком этого метода является то, что во время ожидания процессор, как правило, лишен возможности выполнять какую-либо другую «полезную» работу. В отдельных случаях во время ожидания процессор может выполнять другую работу, периодически «отвлекаясь» на проверку состояния устройства, например по таймеру.

При работе по прерываниям действия выполняются в такой последовательности:

инициируется операция и разрешаются прерывания от этого устройства;

происходит переход к выполнению других действий до момента прерывания;

происходит переход по прерыванию на программу обработки, в которой определяется отсутствие ошибки при выполнении операции, после завершения обработки прерывания происходит возврат на прерванную программу.

Достоинство этого метода — отсутствие необходимости загружать процессор проверкой завершенности операции. Поскольку устройство само сообщает об изменении своего состояния, прерывая работу процессора, то процессор между прерываниями имеет возможность выполнить какую-либо другую работу.

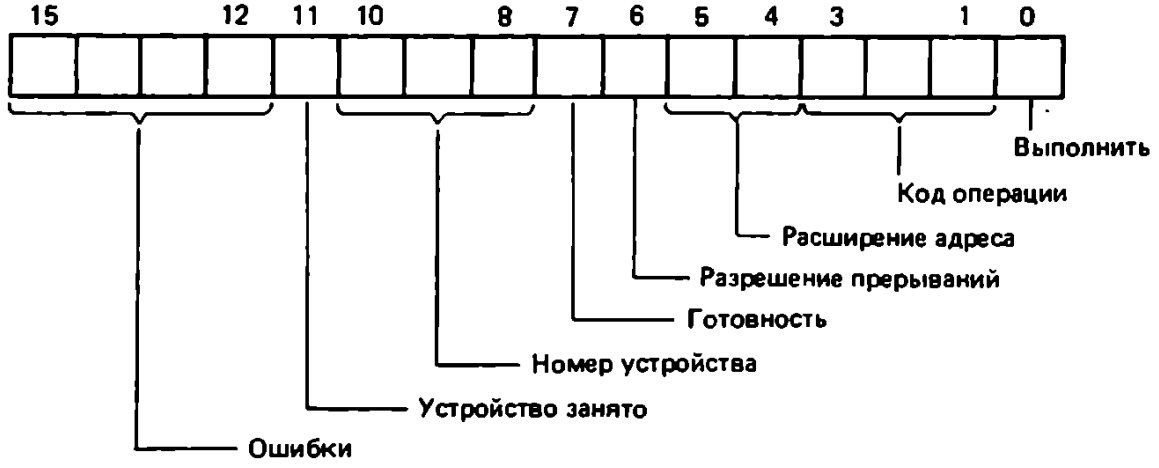
Однако работа по прерываниям требует дополнительного расхода времени процессора на переключение с одной программы на другую, сохранение и восстановление регистров процессора и т. д. Когда частота прерываний велика, интервал между прерываниями меньше времени обработки прерывания, темп обмена данными с внешними устройствами будет ниже, чем при работе по готовности. Кроме того, поскольку прерывания приводят к асинхронному выполнению отдельных модулей программы, процедура отладки подобных программ существенно осложняется.

Таким образом, выбор того или иного способа обмена с внешними устройствами должен определяться с учетом различных факторов, часть которых здесь рассматривалась.

8.1. РЕГИСТРЫ УСТРОЙСТВ

Каждое внешнее устройство представляет на ОШ набор регистров, которые адресуются как ячейки оперативной памяти. Существуют два типа регистров: регистр команд и состояния, обычно обозначаемый CSR (Control and Status Register), и регистр данных, называемый также буферным регистром.

Внешнее устройство может иметь один регистр (или несколько регистров типа CSR), который содержит все необходимое для взаимодействия программы с устройством. Общая структура регистра CSR имеет вид:



Разряд 0 (выполнить) используется программой для запуска операции на устройстве путем установки его в 1, например

```
BIS  #1,@#CSR        ;ВЫПОЛНИТЬ ОПЕРАЦИЮ
```

или

```
INC @ #CSR           ;ВЫПОЛНИТЬ ОПЕРАЦИЮ
```

При чтении регистра CSR этот разряд всегда содержит 0.

Для указания операции, которая должна быть выполнена устройством, используются разряды 1—3 (код операции). Записанная в эти разряды информация может быть получена считыванием CSR.

Разряды 4—5 (расширение адреса) используются устройством с прямым доступом в память в качестве разрядов 16—17 адреса памяти на ОШ и предполагают использование диспетчера памяти. Эти разряды доступны для записи и чтения.

Управление прерываниями работы процессора внешним устройством осуществляется разрядом 6 (разрешение прерывания) CSR. Если разряд устансвлен в 1, прерывания разрешены в соответствии с приоритетами процессора и устройства. Разряд доступен для чтения и записи. Прерывания, если они разрешены, обычно выполняются при изменении состояния устройства: завершении операции, обнаружении ошибки, переходе в состояние готовности и т. д.

Разряд 7 (готовность) предназначен только для чтения, устанавливается и обрабатывается внешним устройством. Установка этого разряда свидетельствует о завершении одной операции и о готовности выполнить следующую.

Разряды 8—10 (номер устройства) устанавливаются программно и используются контроллером для выбора одного из нескольких устройств одинакового типа, подключенных к этому контроллеру. Например, к одному контроллеру может быть подключено до 8 дисководов. Если возможно подключение только одного устройства, эти разряды не используются.

Разряд 11 (занято) устанавливается устройством и свидетельствует о том, что устройство выполняет операцию, т. е. занято.

Разряды 12—15 (ошибка) устанавливаются устройством для индикации специфической ошибки, имевшей место при выполнении операции. Обычно разряд 15 используется для индикации ошибочной ситуации, а разряды 12—14 — для уточнения, например типа ошибки. Как правило, установка разряда 15 в 1 влечет за собой прерывание, если оно разрешено.

Следует отметить, что не каждое устройство использует все 16 разрядов CSR, рассмотренные здесь. Некоторым устройствам (например, магнитной ленте) могут потребоваться дополнительные регистры для индикации состояния и получения управляющей информации.

Регистр данных (буферный регистр) используется для хранения информации, передаваемой между программой и внешним устройством.

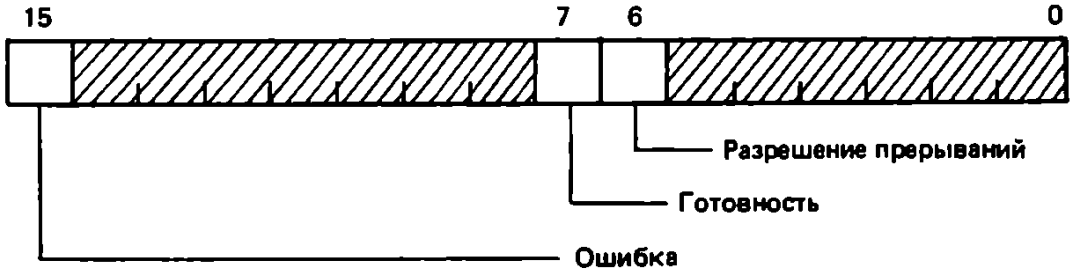
Приемы и способы программирования внешних устройств будут показаны на примерах устройства печати, алфавитно-цифрового видеотерминала, перфоленточного устройства ввода-вывода и одного устройства с прямым доступом — диска с плавающими головками СМ5400 (ИЗОТ-1370).

8.1.1. РЕГИСТРЫ УСТРОЙСТВА ПЕЧАТИ

Устройство печати предназначено для получения копий документов, листингов программ, результатов счета и т. д. При выводе информации на устройство используется 7-битный код КОИ-7, который является общим для всех устройств СМ ЭВМ, работающих с символьной информацией.

Устройство печати представлено на ОШ двумя регистрами — CSR и регистром данных, которые имеют адреса 777514 и 777516 соответственно. Устройство печати подключается к линии запроса прерывания (обычно с приоритетом 4) и имеет вектор прерывания с адресом 200.

Регистр команд и состояния устройства печати имеет следующее распределение разрядов:



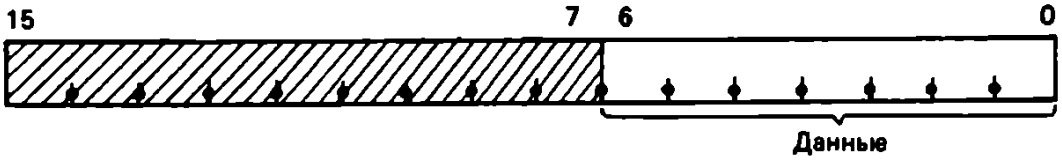
Назначение разрядов

Разряд 6 (разрешение прерывания) доступен для чтения и записи. Если разряд установлен в 1, устройству разрешено прерывать работу процессора при его готовности или наличии ошибки.

Разряд 7 (готовность) устанавливается и сбрасывается только самим устройством. Если он установлен в 1, это означает, что устройство готово к приему следующего знака или управляющего кода. Если устройство находится в автономном режиме, разряд считывается как 0. Установка этого разряда в 1 при разрешении прерываний вызывает запрос на прерывание.

Разряд 15 (ошибка) устанавливается и сбрасывается устройством. Установка его в 1 сигнализирует об ошибке. Возможны следующие ошибки: отключение питания, отсутствие бумаги, переключение в автономный режим. Сбрасывается после исправления ошибки. Если разряд устанавливается в 1 при разрешении прерываний, это вызывает запрос на прерывание.

В буферном регистре данных разряды распределены, как показано ниже.



Разряды 0—6 предназначены для приема кодов знаков и управляющих кодов. Эти разряды предназначены только для записи и читаются всегда как 0. Запись в эти разряды информации воспринимается устройством как требование выполнить команду. Если записанный код находится в пределах от 40 до 176, командой считается «Выполнить печать» символа с соответствующим кодом. Остальные коды воспринимаются как управляющие (например, «Закончить строку») или игнорируются. Среди управляющих кодов необходимо выделить:

12_8 (LF — Line Feed) — продвинуть бумагу на один шаг (т. е. перейти на следующую строку);

14_8 (FF — Form Feed) — продвинуть бумагу к началу следующего листа (переход на новую страницу);

15_8 (CR — Carige Return) — продолжить печать с начала строки.

Управляющие коды <CR> и <LF> обычно используются вместе, поскольку <LF> не возвращает к началу строки, т. е. если на строке последний знак выведен на 20-ю позицию, то по <LF> печать переходит на следующую строку на 21-ю позицию. Полная строка обычно содержит 128 или 132 знака.

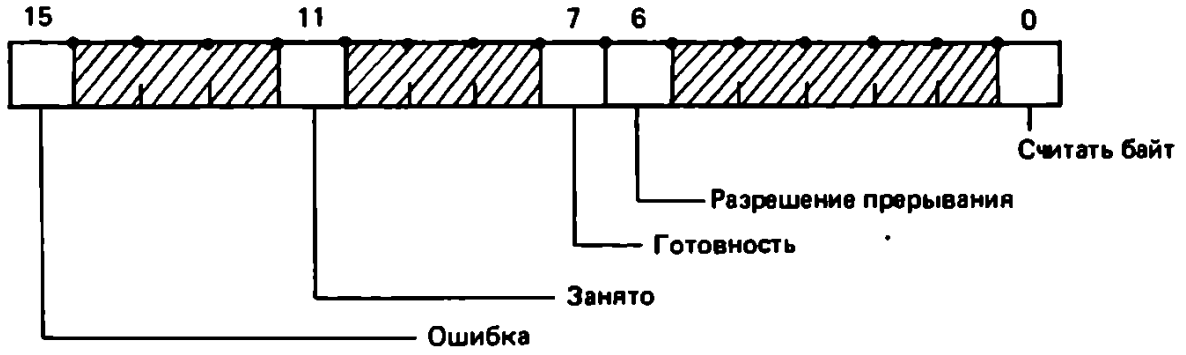
Страница печати, как правило, содержит 72 строки, после которых устройство автоматически может переходить в начало новой страницы. Код <FF> используется при необходимости выполнить этот переход раньше.

8.1.2. РЕГИСТРЫ ПЕРФОЛЕНТОЧНОГО УСТРОЙСТВА

Перфоленточное устройство ввода-вывода предназначено для считывания данных с перфоленты (8-дорожечной) и для вывода данных на перфоленту. Это устройство состоит из двух логически независимых компонентов: считывателя и перфоратора.

Считыватель представлен на общей шине регистром CSR (777550) и буферным регистром (777552), подключен к линии запроса прерывания с приоритетом 4 и имеет вектор прерывания с адресом 70.

Регистр CSR считывателя имеет следующую структуру:



Назначение разрядов

Разряд 0 (считать байт) предназначен только для записи и считывается всегда как 0. Устанавливается программно и означает необходимость чтения с перфоленты следующего байта.

Разряд 6 (разрешение прерываний) доступен для чтения и записи. Если установлен в 1, устройству разрешено прерывать работу процессора в случае его готовности (бит 7) или ошибки (бит 15).

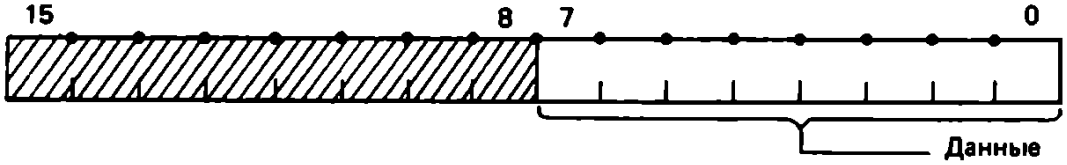
Разряд 7 (готовность) устанавливается и сбрасывается самим устройством. Если установлен в 1, это означает, что в буферном

регистре находится прочитанный байт. Установка этого разряда в 1 при разрешении прерывания вызывает запрос на прерывание.

Разряд 11 (занято) устанавливается и сбрасывается устройством. Устанавливается в 1 только при выполнении устройством операции считывания байта в буферный регистр.

Разряд 15 (ошибка) устанавливается и сбрасывается только устройством. Если разряд содержит 1, это означает, что перфолента отсутствует или кончилась, считыватель находится в автономном режиме, не включено питание и т. д. Если устанавливается в 1 при разрешении прерывания, вызывает запрос на прерывание.

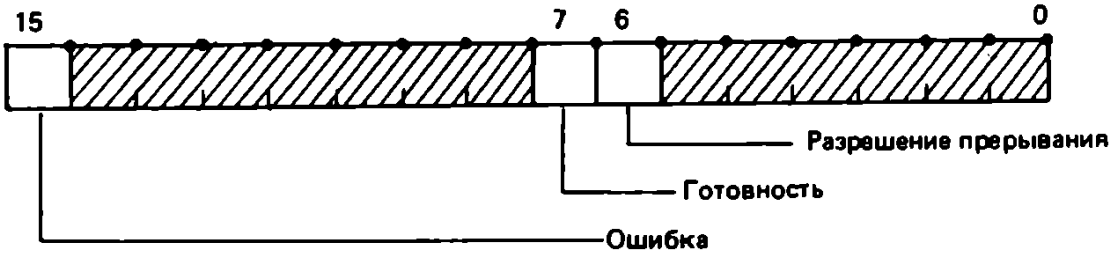
Буферный регистр считывания имеет следующую структуру:



Разряды 0—7 содержат байт, считанный с перфоленты, и доступны только для чтения. Эти разряды очищаются при установке в 1 нулевого разряда CSR.

Перфоратор представлен на общей шине регистром CSR (777554) и буферным регистром (777556), подключен к линии запроса прерывания с приоритетом 4 и имеет вектор прерывания с адресом 74.

Регистр CSR перфоратора имеет следующую структуру:



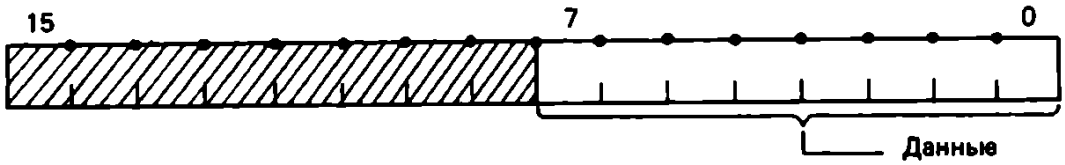
Назначение разрядов

Разряд 6 (разрешение прерываний) доступен для чтения и записи. Если установлен в 1, устройству разрешено прерывать работу процессора в случае его готовности или наличия ошибки.

Разряд 7 (готовность) устанавливается и сбрасывается только самим устройством. Если установлен в 1, то устройство готово к перфорации следующего байта. Установка этого разряда в 1 при разрешении прерывания вызывает запрос на прерывание.

Разряд 15 (ошибка) устанавливается и сбрасывается устройством. Его назначение сигнализировать об ошибке: нет питания, не заправлена лента и т. д. Если разряд устанавливается в 1 при разрешенном прерывании, это вызывает запрос на прерывание.

Буферный регистр перфоратора имеет структуру:



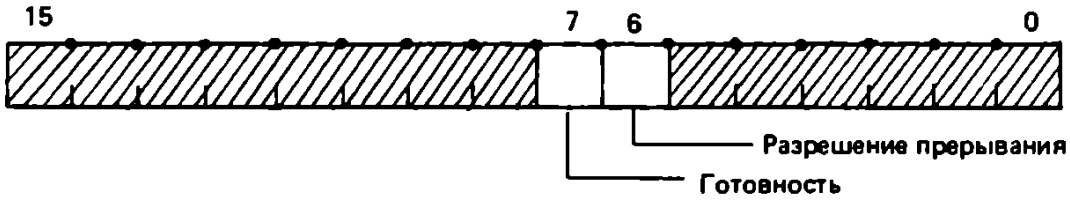
Разряды 0—7 буферного регистра предназначены для записи байта, который должен быть выведен на перфоленту. При считывании эти разряды всегда воспринимаются как 0. Запись любой информации в этот регистр приводит в действие перфоратор.

8.1.3. РЕГИСТРЫ ТЕРМИНАЛА

Терминал предназначен для связи оператора (программиста) с ЭВМ. Это устройство состоит из двух логически независимых компонентов: клавиатуры и экрана.

Клавиатура представлена на ОШ регистром CSR (777560) и буферным регистром (777562), подключена к линии запроса прерываний с приоритетом 4 и имеет вектор прерываний с адресом 60.

Регистр CSR клавиатуры имеет следующее распределение разрядов:

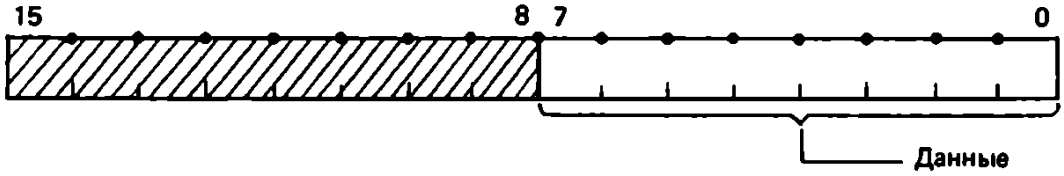


Назначение разрядов

Разряд 6 (разрешение прерывания) доступен для чтения и записи; если установлен в 1, устройству разрешено прерывать работу процессора в случае его готовности.

Разряд 7 (готовность) устанавливается и сбрасывается самим устройством. Если установлен в 1, то это означает, что в буферном регистре находится код введенного с клавиатуры знака. Установка этого разряда в 1 при разрешении прерывания вызывает запрос на прерывание.

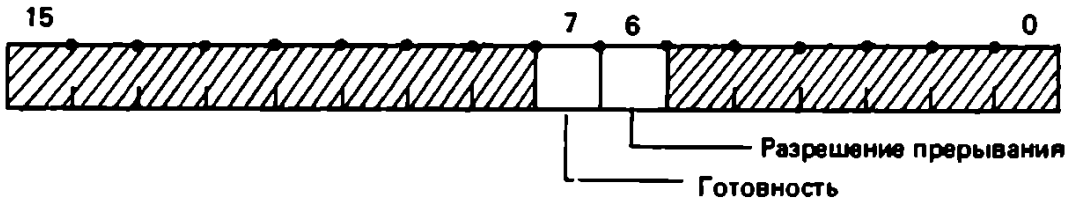
Буферный регистр клавиатуры имеет вид:



Разряды 0—7 буферного регистра содержат код знака, считанного с клавиатуры, и доступны только для чтения. Восьмой разряд введенного байта (бит 7) для некоторых типов терминалов используется в качестве разряда четности (нечетности) для контроля правильности передачи. В программах этот разряд обычно не используется и сбрасывается в 0 после считывания байта.

Экран представлен на общей шине регистром CSR (777564) и буферным регистром (777566), подключен к линии запроса прерываний с приоритетом 4 и имеет вектор прерывания с адресом 64.

Регистр CSR экрана имеет следующий формат:

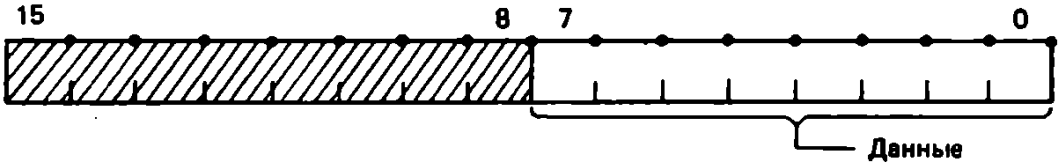


Назначение разрядов

Разряд 6 (разрешение прерывания) доступен для записи и чтения. Если он установлен в 1, устройству разрешено прерывать работу процессора в случае готовности.

Разряд 7 (готовность) устанавливается и сбрасывается устройством. Установка этого разряда в 1 при разрешении прерываний вызывает запрос на прерывание. Если разряд установлен в 1, это означает, что устройство готово к выводу следующего знака на экран.

Буферный регистр экрана имеет вид:



Разряды 0—7 предназначены для записи кода знака, который должен быть выведен на экран. При считывании эти разряды воспринимаются как 0. Запись любой информации в этот регистр приводит в действие аппаратуру экрана.

Простейшими управляющими кодами для экрана являются <CR> и <LF>, имеющие тот же смысл, что и для устройства печати.

8.1.4. РЕГИСТРЫ ДИСКА

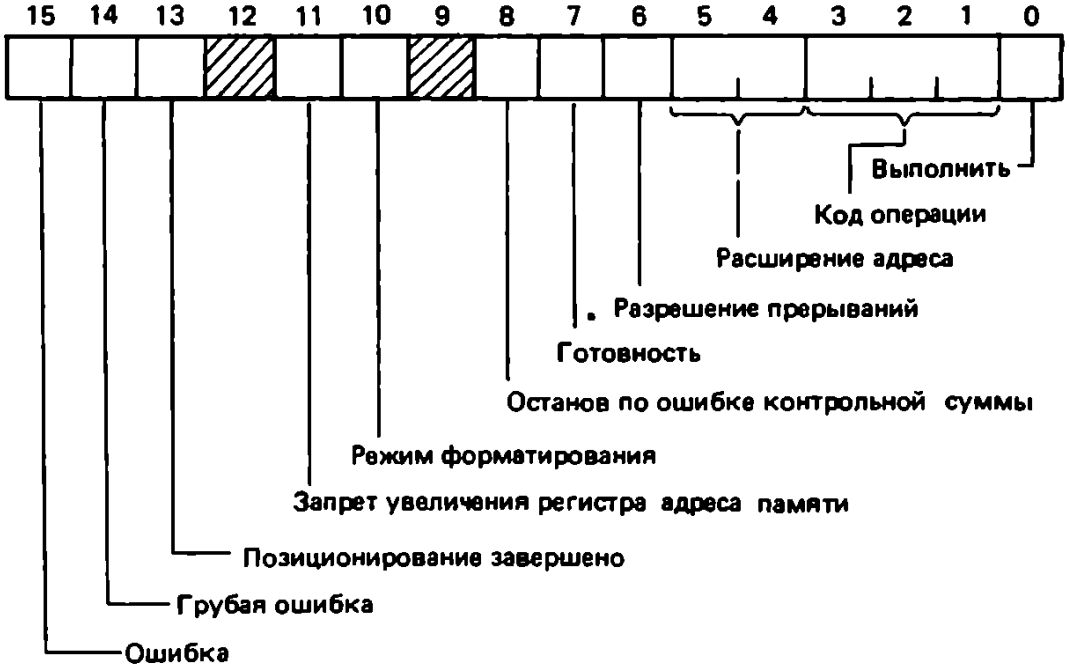
Диск предназначен для хранения и быстрого доступа к сравнительно большим объемам данных. Управление диском осуществляется контроллером, к которому может быть подключено до четырех или восьми дисководов.

Контроллер типа СМ5400 осуществляет обмен с оперативной памятью по прямому доступу, подключен к линии запроса прерываний с приоритетом 5 и имеет вектор прерывания с адресом 220.

На общей шине контроллер имеет семь регистров: регистр состояния диска (777400), регистр ошибок (777402), регистр команд и состояния контроллера (777404), счетчик слов (777406), адрес буфера в памяти (777410), регистр адреса на диске (777412), буферный регистр (777414).

Поскольку в книге рассматриваются общие принципы программирования различных устройств, остановимся на описании тех регистров, которые потребуются для дальнейшего изложения. При этом будем описывать только необходимые для последующих примеров разряды.

Регистр команд и состояния контроллера имеет следующий формат:



Назначение разрядов

Разряд 0 (выполнять) доступен для записи, считывается как 0, запускает выполнение операции, код которой указан в разрядах 1—3.

Разряды 1—3 (код операции) доступны для чтения и записи, определяют операцию, которую должен выполнить контроллер: 001 — запись на диск, 010 — чтение с диска.

Разряд 6 (разрешение прерываний) доступен для записи и чтения. Если он установлен в 1, контроллеру разрешено прерывать работу процессора в случае готовности или ошибки.

Разряд 7 (готовность) устанавливается и сбрасывается контроллером. Установка этого разряда в 1 при разрешении прерываний вызывает запрос на прерывание. Если разряд установлен в

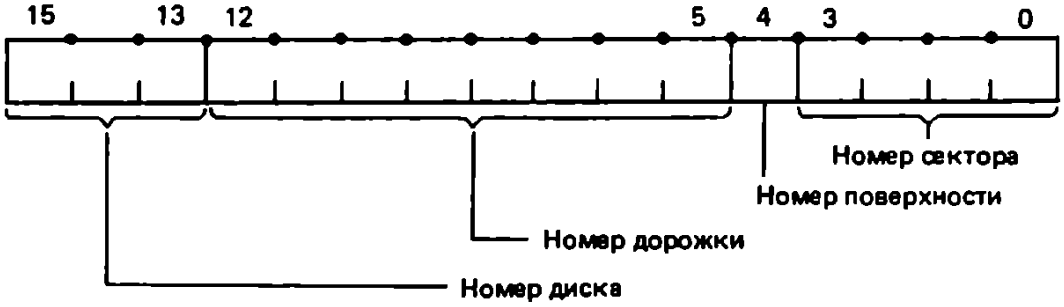
1, это означает, что контроллер свободен и готов выполнять следующую операцию.

Разряд 15 (ошибка) устанавливается и сбрасывается контроллером. Установка его в 1 сигнализирует об ошибке, возникшей в процессе работы контроллера и(или) диска. Если разряд устанавливается в 1 при разрешении прерываний, это вызывает запрос на прерывание.

Регистр счетчика слов перед запуском операции на выполнение должен содержать отрицательный 16-разрядный счетчик слов, участвующих в заданной операции. После передачи очередного слова контроллер увеличивает содержимое регистра на 1. Регистр доступен для чтения и записи.

Регистр адреса буфера в памяти перед запуском операции на выполнение должен содержать 16-разрядный адрес. После передачи очередного слова контроллер увеличивает содержимое регистра на 2. Регистр доступен для чтения и записи.

Регистр адреса на диске имеет следующую структуру:



Назначение разрядов

Разряды 0—3 (номер сектора) содержат номер сектора диска, число в диапазоне $0—12_{10}$.

Разряд 4 (номер поверхности) указывает номер поверхности (0 или 1).

Разряды 5—12 (номер дорожки) определяют номер используемой дорожки в диапазоне от 0 до 199_{10} ($0—312_8$).

Разряды 13—15 (номер диска) определяют один из восьми (0—7) дисков, с которыми будет выполняться операция.

Регистр адреса на диске всегда доступен для чтения. Запись в регистр адреса на диске возможна, только когда контроллер находится в состоянии готовности. При выполнении операции контроллер увеличивает содержимое регистра, осуществляя автоматический переход с сектора на сектор, с поверхности на поверхность, с одной дорожки на другую.

8.2. РАБОТА БЕЗ ПРЕРЫВАНИЙ

Рассмотрим работу с внешним устройством без прерываний на примере программы, выводящей строку текста на устройство печати.

Для реализации этой цели используются две подпрограммы. Подпрограмма OSTRLP выводит текстовую строку, адрес которой будет передаваться ей в регистре 0. Признаком конца строки будет байт, содержащий 200. Если строка заканчивается байтом 0, то после ее вывода будет осуществляться переход на новую строку, т. е. на устройство печати будут выведены управляющие знаки <CR> (возврат каретки) и <LF> (перевод строки). Подпрограмма OSTRLP будет использовать подпрограмму OCHRLP, выводящую на устройство печати один знак.

Подпрограмма OSTRLP может иметь следующий вид:

```
                .ENABL  LC
                .TITLE  OSTRLP                  ; ПЕЧАТЬ СТРОКИ
                .IDENT  /V01.00/
;  +
;  ВЫЗОВ:   MOV    #STR, R0
;           CALL   OSTRLP
;  STR:     .ASCII      /СТРОКА/<200>   ; ИЛИ
;  СТР:     .ASCIZ      /СТРОКА/й       ; ПЛЮС CR, LF
;  ИСПОЛЬЗУЕМЫЕ ПОДПРОГРАММЫ:
;           OCHRLP  – ВЫВОД ЗНАКА НА ПЕЧАТЬ
;  –
CR = 15                                 ; ВОЗВРАТ КАРЕТКИ
LF = 12                                 ; ПЕРЕХОД НА НОВУЮ СТРОКУ
                .GLOBL  OCHRLP
OSTRLP::
                MOV     R1,–(SP)        ; СОХРАНИТЬ РАБОЧИЙ РЕГИСТР
                MOV     R0, R1          ; R1 => АДРЕС СТРОКИ
10⊙:            MOVB    (R1)+, R0       ; R0 => ОЧЕРЕДНОЙ ЗНАК
                BEQ     20⊙             ; ПЕРЕЙТИ, ЕСЛИ 0
                BMI     30⊙             ; КОНЕЦ, ЕСЛИ БАЙТ 200
                CALL    OCHRLP          ; ВЫВЕСТИ ОЧЕРЕДНОЙ ЗНАК
                BCC     10⊙             ; ОШИБКИ НЕТ =>
                                        ; ПОВТОРИТЬ
                BR      30⊙             ; ОШИБКА => ВЫХОД
20⊙:            MOVB    #CR, R0         ; ВЫВЕСТИ
                CALL    OCHRLP          ; CR
                BCS     30⊙             ; ВЫХОД ПО ОШИБКЕ
                MOVB    #LF, R0         ; ВЫВЕСТИ
                CALL    OCHRLP          ; LF
30⊙:            MOV     (SP)+, R1       ; ВОССТАНОВИТЬ РЕГИСТР
                RETURN                  ; ВЕРНУТЬ УПРАВЛЕНИЕ
                .END
```

Рассмотрим второй вариант подпрограммы OSTRLP, которая использует рекурсивное обращение к самой себе.

```
                .ENABL  LC
                .TITLE  OSTRLP – ПЕЧАТЬ СТРОКИ
                .IDENT  /V02.00/
                .GLOBL  OCHRLP
CR = 15
LF = 12
OSTRLP::
                MOV     R1,– (SP)               ; СОХРАНИТЬ РАБОЧИЙ РЕГИСТР
```

```
        MOV     R0, R1                  ; АДРЕС СТРОКИ
10⊙:    MOVB    (R1)+, R0               ; В R0 ОЧЕРЕДНОЙ ЗНАК
        BEQ     20⊙                     ; ОЧЕРЕДНОЙ ЗНАК – 0
        BMI     30⊙                     ; БАЙТ 200
        CALL    OCHRLP                  ; ВЫВЕСТИ ОЧЕРЕДНОЙ ЗНАК
        BCC     10⊙                     ; ОШИБКИ НЕТ
        BR      30⊙                     ; ОШИБКА ПРИ ВЫВОДЕ
20⊙:    MOV     #40⊙, R0                ; ВЫВЕСТИ
        CALL    OSTRLP                  ; УПРАВЛЯЮЩУЮ СТРОКУ
30⊙:    MOV     (SP)+  R1
        RETURN
40⊙:    .BYTE   CR, LF, 200             ; УПРАВЛЯЮЩАЯ СТРОКА
        .END
```

Рекурсивное обращение во втором варианте подпрограммы OSTRLP при условии окончания строки байтом 0 не приводит к «зацикливанию», так как при повторном вызове управляющая строка, которая выводится, завершается байтом 200. При этом управление передается на метку 30⊙ и третьего вызова OSTRLP не происходит. После возврата из повторного вызова управление снова попадает на метку 30⊙ и происходит нормальное завершение работы подпрограммы.

Если во время вызова подпрограммы OCHRLP устройство печати перешло в состояние «ошибка», подпрограмма возвращает управление с установленным битом C в PS. Подпрограмма OSTRLP при каждом вызове контролирует бит C и немедленно возвращает управление в вызвавшую ее программу (с установленным битом C).

Подпрограмма OCHRLP, осуществляющая вывод одного знака на печать, может иметь вид:

```
        .ENABL  LC
        .TITLE  OCHRLP – ПЕЧАТЬ ЗНАКА
        .IDENT  /V01.00/
;  +
;  ВЫЗОВ:  MOVB    #CHAR, R0           ; КОД ЗНАКА В R0
           CALL    OCHRLP              ; ВЫВОД ЗНАКА
;          BCS     ERROR               ; КОНТРОЛЬ
                                       ; ОШИБКИ
;  –
LPCSR   =  177514                      ; АДРЕС РЕГИСТРА СОСТОЯНИЯ
LPBUF   =  LPCSR+2                     ; БУФЕРНЫЙ РЕГИСТР
ERRBIT  =  100000                      ; БИТ ОШИБКИ
                                       ; В CSR
DONBIT  =  200                         ; БИТ ГОТОВНОСТИ В CSR
OCHRLP::
           BIT     #ERRBIT, @#LPCSR    ; ОШИБКА НА УСТРОЙСТВЕ?
           BNE     1⊙                  ; ДА
           BIT     #DONBIT, @#LPCSR    ; УСТРОЙСТВО ГОТОВО?
           BEQ     OCHRLP              ; НЕТ
           MOVB    R0, @#LPBUF         ; ВЫВЕСТИ ЗНАК
           TST     (PC)+               ; ОЧИСТИТЬ БИТ "C" И ВОЗВРАТ
1⊙:        SEC                         ; УСТАНОВИТЬ БИТ "C"
                                       ; (ОШИБКА)
           RETURN
           .END
```

Данная программа работает с внешним устройством без прерываний. Выяснение готовности устройства осуществляется в цикле, при этом процессор не выполняет никакой другой «полезной» работы.

Используя свойства инструкции TST и принимая во внимание тот факт, что опрашиваются в регистре состояния старшие разряды слова и байта соответственно, подпрограмму можно переписать в следующем виде:

```
OCHRLP::
        TST     @#LPCSR         ; ОШИБКА?
        BMI     1©              ; ДА
        TSTB    @#LPCSR         ; ГОТОВО?
        BPL     OCHRLP          ; НЕТ
        ...
```

Приведенный вариант программы короче предыдущего на 2 слова. На скорости выполнения проведенная оптимизация практически не отразится, поскольку темп вывода будет определяться техническими характеристиками (производительностью) устройства печати.

Алгоритм программы можно несколько изменить, если выполнять проверку готовности и ошибки одной командой:

```
OCHRLP::
        BIT     #ERRBIT!DONBIT,@#LPCSR  ; СОСТОЯНИЕ?
        BMI     1©              ; ОШИБКА
        BEQ     OCHRLP          ; НЕ ГОТОВО
```

Это позволяет сократить объем программы еще на одно слово.

Рассмотрим пример программирования терминала по готовности. Основными подпрограммами, которые при этом будут использоваться, являются ICHRTT (ввод одного знака с клавиатуры терминала) и OCHRTT (вывод одного знака на экран терминала).

```
        .ENABL  LC
        .TITLE  IOCHR – ВВОД - ВЫВОД ОДНОГО ЗНАКА НА TT
        .IDENT  /V01.00/
;  +
;  ВЫЗОВ:  CALL    ICHRTT          ; ВВЕСТИ ЗНАК С TT
;.         MOVB    R0,...          ; В R0
;          ...
;          MOVB    ...,R0          ; ЗНАК В R0
;          CALL    OCHRTT          ; ВЫВЕСТИ НА ЭКРАН
;  –
TTKCSR  =  177560                  ; РЕГИСТР СОСТОЯНИЯ КЛАВИАТУРЫ
TTKBUF  =  TTKCSR+2                ; БУФЕР КЛАВИАТУРЫ
TTPCSR  =  TTKCSR+4                ; РЕГИСТР СОСТОЯНИЯ ЭКРАНА
TTPBUF  =  TTKCSR+6                ; БУФЕР ЭКРАНА
```

```
;
ICHRTT::
        TSTB    @#TTKCSR            ; ЗНАК ВВЕДЁН?
        BPL     ICHRTT              ; НЕТ
        MOVB    @#TTKBUF, R0        ; ДА => ЗАНЕСТИ В R0
        BIC     #C177, R0           ; ОЧИСТИТЬ "ЛИШНИЕ" БИТЫ
        RETURN
OCHRTT::
        TSTB    @#TTPCSR            ; ЭКРАН ГОТОВ?
        BPL     OCHRTT              ; НЕТ
        MOVB    R0, @#TTPBUF        ; ДА => ВЫВЕСТИ ЗНАК
        RETURN
        .END
```

Приведенные программы не производят проверки терминала на «ошибку», поскольку не предусмотрены соответствующие разряды в регистрах состояния.

В программе ICHRTT после пересылки знака буферного регистра в R0 производится очистка всех разрядов, кроме младших семи, так как терминал работает с 7-разрядным кодом КОИ-7. Поскольку отдельные терминалы могут использовать дополнительные разряды с целью контроля, который данными программами не осуществляется, то очистка остальных разрядов является целесообразной.

В качестве более сложного примера работы с терминалом рассмотрим программу эхо-печати TTECHO. Под эхо-печатью будем понимать процесс вывода на экран знаков, вводимых с клавиатуры. Программа TTECHO будет использовать рассмотренные подпрограммы ICHRTT и OCHRTT. При вводе все управляющие знаки (имеющие код от 0 до 37_8 включительно) будут игнорироваться, за исключением <CR> (возврат каретки), который будет интерпретироваться как переход в начало следующей строки. По знаку CTRL/C (код 3), набираемому на клавиатуре комбинацией клавиш CTRL и C, программа завершает свою работу. На некоторых терминалах клавиша CTRL обозначается как УС. Знак DEL (забой) с кодом 177_8 будет служить для удаления предыдущего выведенного на экран знака, например

```
        .ENABL  LC
        .TITLE  TTECHO – ЭХО-ПЕЧАТЬ
        .IDENT  /V01.03/
; ИСПОЛЬЗУЕМЫЕ ПОДПРОГРАММЫ:
        .GLOBL  ICHRTT              ; ВВОД ЗНАКА
        .GLOBL  OCHRTT              ; ВЫВОД ЗНАКА
; +
; УПРАВЛЯЮЩИЕ ЗНАКИ:
; -
CTRLC   =  3                        ; ЗАКОНЧИТЬ ПРОГРАММУ
CR      =  15                       ; ПЕРЕЙТИ К НОВОЙ СТРОКЕ
BLANK   =  40                       ; ПРОБЕЛ – НАЧАЛО КОДОВ ПЕЧАТНЫХ
                                    ; СИМВОЛОВ
DEL     =  177                      ; УДАЛИТЬ ПРЕДЫДУЩИЙ ЗНАК
BS      =  10                       ; ВОЗВРАТ КУРСОРА НА ОДНУ ПОЗИЦИЮ
LF      =  12                       ; ЗНАК ПЕРЕВОДА НА НОВУЮ СТРОКУ
```

```
WIDTH   = 72                                    ; ШИРИНА ЭКРАНА В ЗНАКАХ
;  +
;  ЛОКАЛЬНЫЕ МАКРОКОМАНДЫ:
;  -
  .MACRO   TEST    CHAR, REL, LABEL
           CMPB    CHAR, R0
           B'REL   LABEL
  .ENDM
  .MACRO   TYPE    CH1, CH2, CH3, CH4, CH5
           .IRP    X, <CH1, CH2, CH3, CH4, CH5>
           .IIF    B, <X>
           .MEXIT
           MOVB    X, R0
           CALL    OCHRTT
           .ENDR
  .ENDM
;  +
;  ТЕКСТ ПРОГРАММЫ
;  -
TTECHO--::
           CLR     R5                           ; СЧЕТЧИК ЗНАКОВ В СТРОКЕ
1⊙:        CALL    ICHRTT                       ; ВВЕСТИ ЗНАК С КЛАВИАТУРЫ
           TEST    #CR, EQ, 2⊙                  ; ЗНАК CR?
           TEST    #DEL, EQ, 3⊙                 ; DEL?
           TEST    #CTRLC, EQ, 4⊙               ; CTRLC?
           TEST    #BLANK, GT, 1⊙               ; УПРАВЛЯЮЩИЙ ЗНАК?
                                                ; ИГНОРИРОВАТЬ
           CALL    OCHRTT                       ; ВЫВЕСТИ ЗНАК
           INC     R5                           ; УВЕЛИЧИТЬ СЧЕТЧИК
           CMP     #WIDTH, R5                   ; ГРАНИЦА ЭКРАНА?
           BGT     1⊙                           ; НЕТ
2⊙:        TYPE    #CR, #LF                     ; ПЕРЕЙТИ К СЛЕДУЮЩЕЙ СТРОКЕ
           BR      TTECHO                       ; И УСТАНОВИТЬ СЧЕТЧИК В 0
3⊙:        TST     R5                           ; НАЧАЛО СТРОКИ?
           BEQ     1⊙                           ; ДА => ИГНОРИРОВАТЬ ЗНАК DEL
           TYPE    #BS, #BLANK, #BS             ; ИНАЧЕ: СТЕРЕТЬ ЗНАК С ЭКРАНА
           DEC     R5                           ;        УМЕНЬШИТЬ СЧЕТЧИК И
           BR      1⊙                           ;        ПРОДОЛЖИТЬ РАБОТУ
4⊙:        HALT                                 ; ОСТАНОВИТЬ ПРОЦЕССОР
           BR      4⊙                           ; ОСТАНОВИТЬ ПРОЦЕССОР ЕЩЕ РАЗ
           .END    TTECHO                       ;
```

В программе для удобства описания алгоритма использовались макрокоманды TEST и TYPE. Макрокоманда TEST выполняет сравнение знака, находящегося в регистре R0 с кодом знака, указанного в качестве аргумента CHAR. В случае выполнения отношения REL, задаваемого вторым аргументом, управление передается на метку, указанную аргументом LABEL.

Макрокоманда TYPE выполняет вывод знаков (общим числом до пяти) на терминал, используя для этого вызов подпрограммы OCHRTT.

Программа TTECHO выполняет эхо-печать вводимых знаков до тех пор, пока длина введенной последовательности знаков не достигнет 72. В качестве счетчика знаков строки используется ре-

гистр R5. При достижении конца строки программа производит переход на новую строку. При этом счетчик устанавливается в 0.

Содержимое счетчика используется также при обработке знака DEL. Если счетчик равен нулю, т. е. курсор находится в начале строки (строка пуста), знак DEL игнорируется. Обработка знака DEL ведется в предположении, что в качестве видеотерминала используется устройство типа VT-340 (ВНР). Для удаления знака с экрана курсор перемещается на одну позицию влево под удаляемый знак (по коду BS), выводится знак «пробел», очищая текущую позицию курсора, и курсор снова возвращается на место исчезнувшего знака.

В качестве дополнительного примера работы с терминалом по готовности рассмотрим следующую подпрограмму:

```
TYPE:
        .ENABL
        .TITLE  TYPE    ; ПЕЧАТЬ СТРОКИ НА ТТ
        .IDENT  /V02.01/
; +
; ВЫЗОВ: MOV  #STR,R0
;        CALL TYPE
;        ...
; STR:    .ASCII /СТРОКА/<200>     ; ИЛИ
; STR:    .ASCIZ /СТРОКА/
; -
TTPCSR  = 177564
TTPBUF  = TTPCSR+2
CR      = 15
LF      = 12
TYPE::
10$:    TSTB    @R0             ; КОНЕЦ СТРОКИ?
        BMI     40$             ; ДА
        BEQ     30$             ; ДОПОЛНИТЬ СТРОКУ CR, LF
20$:    TSTB    @#TTPCSR        ; УСТРОЙСТВО ГОТОВО?
        BPL     20$             ; НЕТ
        MOVB    (R0)+,@#TTPBUF  ; ВЫВЕСТИ ЗНАК
        BR      10$             ; ПОВТОРИТЬ
30$:    MOV     #50$,R0         ; ВЫВЕСТИ ДОПОЛНИТЕЛЬНУЮ
        BR      10$             ; СТРОКУ
40$:    RETURN
50$:    .BYTE   CR,LF,200       ; ДОПОЛНЯЮЩАЯ СТРОКА
        .END
```

Данная подпрограмма выводит строку сообщения оператору на экран терминала. Адрес строки задается в нулевом регистре. Строка должна оформляться по правилам, аналогичным описанным для подпрограммы OSTRLP. Особенностью подпрограммы является использование перехода на начало подпрограммы при обнаружении нулевого байта для вывода управляющих знаков вместо рекурсивного вызова.

Программирование ввода-вывода для перфоленты во многом аналогично работе с устройствами, рассмотренными ранее. Основой для работы с перфолентой могут служить подпрограммы ввода и вывода одного байта.

```
        .ENABL LC
        .TITLE  IOBYTE                  ; В/В БАЙТА НА ПЕРФОЛЕНТЕ
;  +
;  ВЫЗОВ:  MOVB #BYTE, R0               ; ВЫВОДИМЫЙ БАЙТ В R0
;          CALL    OBYTPP              ; ВЫВЕСТИ БАЙТ
;          BCS     ERROR               ; ОШИБКА УСТРОЙСТВА?
;          ...
;          CALL    IBYTPP              ; ВВЕСТИ ОДИН БАЙТ
;          BCS     EOT                 ; КОНЕЦ ПЕРФОЛЕНТЫ?
;          MOVB    R0....              ; ПЕРЕСЛАТЬ НА ОБРАБОТКУ
;  -
PRCSR  = 177550                         ; АДРЕС РЕГИСТРА CSR
                                        ; ПЕРФОВВОДА
PRBUF  = PRCSR+2                        ; БУФЕР ПЕРФОВВОДА
PPCSR  = PRCSR+4                        ; CSR ВЫВОДА НА ПЛ
PPBUF  = PRCSR+6                        ; БУФЕР ВЫВОДА НА ПЛ
;
GOBIT  =        1
DONBIT=       200
ERRBIT=    100000
;.
        .ENABL LSB
IBYTPR::
        INC     @#PRCSR                 ; BIS #GOBIT, @#PRCSR
                                        ; СЧИТАТЬ БАЙТ В PRBUF С ПЛ
1⊙:     BIT     #ERRBIT!DONBIT,@#PRCSR  ; СОСТОЯНИЕ?
        BMI     3⊙                      ; ОШИБКА – КОНЕЦ ЛЕНТЫ
        BEQ     1⊙                      ; НЕ ГОТОВО
        MOVB    @#PRBUF, R0             ; ПЕРЕСЛАТЬ БАЙТ В R0
        RETURN
OBYTPP::
2⊙:     BIT     #ERRBIT!DONBIT,@#PPCSR  ; СОСТОЯНИЕ?
        BMI     3⊙                      ; ОШИБКА ВВОДА?
        BEQ     2⊙                      ; НЕ ГОТОВ
        MOVB    R0,@#PPBUF              ; ВЫВЕСТИ БАЙТ НА ПЛ
        RETURN
3⊙:     SEC                             : ВЫХОД ПО ОШИБКЕ
        RETURN
        .DSABL LSB
        .END
```

Отличительной особенностью подпрограммы считывания байта с перфоленты является необходимость использования специальной инструкции (INC) для «запуска» устройства, т. е. приведение в действие механизма продвижения ленты на одну позицию, определяемую по синхродорожке. Использование инструкции INC позволяет уменьшить объем программы по сравнению с инструкцией BIS, указанной в комментарии. В остальном программы работы с перфолентой совпадают по алгоритму с рассмотренными подпрограммами.

Дополнительное сокращение объема подпрограмм получено за счет использования одной точки выхода по ошибке для выхода из обеих подпрограмм (метка 3⊙). Для того чтобы на локальную метку можно было передать управление через символическую метку, блок локальных символов был начат с директивы .ENABL

LSB, а не с символической метки, как обычно. Этот блок завершается, как описано ранее, директивой .DSABL LSB.

Примером использования приведенных подпрограмм является программа дублирования перфоленты:

```
        .ENABL  LC
        .TITLE  PTCOPY                  ; ДУБЛИРОВАНИЕ ПЛ
        .IDENT  /V01.03/
;  +
;  ИСПОЛЬЗУЕМЫЕ ПОДПРОГРАММЫ
;  -
        .GLOBL  TYPE                    ; ВЫВОД СТРОКИ НА ТЕРМИНАЛ
        .GLOBL  OBYTPP                  ; ВЫВОД БАЙТА НА ПЕРФОЛЕНТУ
        .GLOBL  OCHRTT                  ; ВЫВОД ЗНАКА НА ТЕРМИНАЛ
        .GLOBL  OCHRLP                  ; ВЫВОД ЗНАКА НА ПЕЧАТЬ
        .GLOBL  IBYTPR                  ; ВВОД БАЙТА С ПЕРФОЛЕНТЫ
;  +
;  ЛОКАЛЬНАЯ МАКРОКОМАНДА
;  -
 .MACRO MES     TXT
        .PSECT  TEXTS
000     =.
        .ASCIZ %'TXT'%
        .PSECT
        MOV     #000, R0
        CALL    TYPE
 .ENDM  MES
;  +
;  ОПРЕДЕЛЕНИЕ КОНСТАНТ
;  -
SWR     = 177570                        ; РЕГИСТР ПЕРЕКЛЮЧАТЕЛЕЙ ПРОЦЕССОРА
TTECHO  =    1                          ; КОПИЯ НА ТЕРМИНАЛ
LPECHO  =    2                          ; КОПИЯ НА ПЕЧАТЬ
PTCOPY: MES     <ПОСТАВЬТЕ ПЕРФОЛЕНТУ И НАЖМИТЕ>
        MES     <КЛАВИШУ "ПРОД" НА ПУЛЬТЕ>
        MES     <ЕСЛИ ВЫ ХОТИТЕ ПОЛУЧИТЬ КОПИЮ:>
        MES     <НА ТЕРМИНАЛЕ – НАЖМИТЕ КЛЮЧ 1>
        MES     <НА ПЕЧАТИ – НАЖМИТЕ КЛЮЧ 2>
        HALT                            ; ОСТАНОВИТЬ ПРОЦЕССОР
100:    CALL    IBYTPR                  ; ВВЕСТИ БАЙТ С ПЛ
        BCS     EOT                     ; КОНЕЦ?
        BIT     #TTECHO, @#SWR          ; ЭТО НА ТТ:?
        BEQ     200                     ; НЕТ
        CALL    OCHRTT                  ; ДА
200:    BIT     #LPECHO, @#SWR          ; ЭХО НА LP:?
        BEQ     300                     ; НЕТ
        CALL    OCHRLP                  ; ДА
300:    CALL    OBYTPP                  ; ВЫВЕСТИ БАЙТ НА ПЛ
        BCC     100                     ; ОШИБКИ НЕТ
        MES     <•••ОШИБКА ВЫВОДА>
        BR      PTCOPY                  ; НАЧАТЬ СНАЧАЛА
EOT:    MES     <ДУБЛИРОВАНИЕ ОКОНЧЕНО•••>
        BR      PTCOPY                  ; ДЛЯ НОВОЙ ПЕРФОЛЕНТЫ
        .END    PTCOPY
```

После загрузки программа печатает на терминале сообщение и инструкцией HALT останавливает процессор. Для вывода сообщения оператору на терминал используется макрокоманда MES. Текст строки, указанной в качестве аргумента макрокоманды, размещается в программной секции TEXTS с помощью директивы .ASCIZ. После этого в макрокоманде производится возврат к не-

именованной программной секции, в которой осуществляется вызов подпрограммы TYPE с аргументом (адрес строки) в регистре R0.

Адрес начала строки определяется при помощи присваивания рабочей переменной (символу ⊙⊙⊙) значения счетчика адреса, начиная с которого в программной секции TEXTS размещается эта строка.

Прежде чем продолжить выполнение программы, оператору необходимо установить копируемую перфоленту в устройство считывания и привести в состояние готовности перфоратор. После этого необходимо нажать клавишу ПРОД на пульте процессора.

Основой программы является цикл: считать байт, вывести байт. Выход из этого цикла осуществляется по двум условиям: ошибка ввода, которая воспринимается программой как конец копируемой перфоленты и приводит к нормальному завершению цикла, и ошибка вывода, которая воспринимается как ошибка устройства. В обоих случаях после вывода соответствующего сообщения программа начинает свою работу с начала.

В основной цикл входят два оператора, выполняемые по условию: вывод введенного с перфоленты байта на терминал и на устройство печати. Обращение к соответствующим подпрограммам производится, если на пульте процессора нажаты клавиши 0 и 1. Для проверки, нажата ли клавиша, используется инструкция BIT, проверяющая установку соответствующего разряда в регистре переключателей процессора.

Работа с диском существенно отличается от работы с рассмотренными устройствами, поскольку диск является устройством с прямым доступом в оперативную память. Наличие прямого доступа позволяет процессору не принимать участия в передаче каждого слова (байта), так как этим занимается контроллер диска. Но для такой автоматической работы контроллеру требуется дополнительная информация: количество передаваемых слов, адрес памяти, начиная с которого нужно передавать информацию, адрес на диске и код выполняемой операции (чтение, запись и т. д.). Под адресом на диске подразумевается номер дисковода, номер дорожки, номер поверхности и номер сектора. Для адреса на диске отводится одно слово (16 разрядов).

Рассмотрим подпрограмму, выполняющую операции чтения-записи на диске.

```
        .ENABL  LC, LSB
        .TITLE  IODISK                  ; ЧТЕНИЕ-ЗАПИСЬ ДЛЯ ДИСКА
;  +
;  ПЕРЕД ВЫЗОВОМ:
;     R1 – АДРЕС НА ДИСКЕ
;     R2 – СЧЕТЧИК СЛОВ (ОТРИЦАТЕЛЬНЫЙ)
;     R3 – АДРЕС БУФЕРА В ПАМЯТИ
;  ОБРАЩЕНИЯ
;        CALL    DREAD               ; ЧТЕНИЕ С ДИСКА
;        CALL    DWRITE              ; ЗАПИСЬ НА ДИСК
;  РАБОЧИЕ РЕГИСТРЫ:
;     R0
```

```
; ЛОКАЛЬНЫЕ СИМВОЛЫ:
; -
DKDA    = 177412                        ; РЕГИСТР АДРЕСА НА ДИСКЕ
REABIT  =      4                        ; ОПЕРАЦИЯ ЧТЕНИЯ
WRIBIT  =      2                        ; ОПЕРАЦИЯ ЗАПИСИ
GOBIT   =      1                        ; ВЫПОЛНИТЬ ОПЕРАЦИЮ
;
DREAD::  MOV  #REABIT!GOBIT,-(SP)       ; ДЕЙСТВИЕ = ЧИТАТЬ
         BR   1⊙
;
DWRITE::
         MOV  #WRIBIT!GOBIT,-(SP)       ; ДЕЙСТВИЕ = ПИСАТЬ
;
1⊙:      MOV  #DKDA, R0                 ; R0 = РЕГИСТР АДРЕСА НА ДИСКЕ
         MOV  R1,@R0                    ; УСТАНОВИТЬ АДРЕС НА ДИСКЕ
         MOV  R3,-(R0)                  ; УСТАНОВИТЬ АДРЕС В ПАМЯТИ
         MOV  R2,-(R0)                  ; УСТАНОВИТЬ СЧЕТЧИК СЛОВ
         MOV  (SP)+,-(R0)               ; ВЫПОЛНИТЬ ОПЕРАЦИЮ
2⊙:      TSTB @R0                       ; ГОТОВО?
         BPL  2⊙                        ; НЕТ
         TST  @R0                       ; ДА, ЕСТЬ ОШИБКА?
         BPL  3⊙                        ; НЕТ
         SEC                            ; ДА - УСТАНОВИТЬ "С"
3⊙:      RETURN
         .DSABL LSB
         .END
```

Данная программа имеет две точки входа: для операций чтения и записи соответственно. Перед обращением к подпрограмме необходимо занести требуемые значения в регистры, как это описано в комментариях. Следует учесть, что при выполнении подпрограммы содержимое нулевого регистра будет изменено.

Регистр R2 должен содержать отрицательный счетчик слов. Например, для того чтобы выполнить операцию ввода-вывода одного блока (сектора), размер которого равен 256_{10} слов (как указывалось выше), можно использовать инструкцию

```
MOV #-256., R2 или MOV #  256., R2
                   NEG    R2
```

Приведенная подпрограмма при обращении к регистрам контроллера диска использует режим адресации с автоуменьшением, поскольку соответствующие регистры имеют последовательные адреса на общей шине.

Предложенная подпрограмма имеет ряд недостатков, которые затрудняют программисту работу с диском. Одним из них является необходимость работы с адресом на диске в терминах: номер диска, дорожка, поверхность, сектор. Второй недостаток — использование отрицательного счетчика слов как в операции чтения, так и в операции записи.

Для программиста удобнее представлять диск в виде последовательности пронумерованных блоков, имеющих фиксированный размер. Размер блока обычно выбирается равным размеру сектора на диске и составляет 256 слов.

Рассмотрим подпрограмму, позволяющую работать с диском в терминах «блоков». Эта программа имеет одну точку входа и различает операцию ввода и операцию вывода по знаку счетчика слов.

```
        .ENABL  LC
        .TITLE  DISK                    ; РАБОТА С ДИСКОМ ПО БЛОКАМ
;+
; ПЕРЕД ВЫЗОВОМ:
;   R0 – НОМЕР ДИСКОВОДА
;   R1 – НОМЕР БЛОКА
;   R2 – СЧЕТЧИК СЛОВ (>0 – ЧТЕНИЕ, <0 – ЗАПИСЬ)
;   R3 – АДРЕС БУФЕРА В ПАМЯТИ
; ОБРАЩЕНИЕ:
;       CALL    DISK
; ПОСЛЕ ВЫЗОВА:
;   R0 – ИСПОРЧЕН
;   R1 – АДРЕС НА ДИСКЕ
;   R2 – ОТРИЦАТЕЛЬНЫЙ СЧЕТЧИК СЛОВ
;   R3 – R5 – БЕЗ ИЗМЕНЕНИЙ
;
; ИСПОЛЬЗУЕМЫЕ ПОДПРОГРАММЫ:
;  –
        .GLOBL  DREAD                   ; ЧТЕНИЕ С ДИСКА
        .GLOBL  DWRITE                  ; ЗАПИСЬ НА ДИСК
;+
;–
NSEC  = 12                              ; ЧИСЛО СЕКТОРОВ НА ПОВЕРХНОСТИ
CONST = 20                              ; КОНСТАНТА ПЕРЕСЧЕТА
;
DISK::
        CLR     –(SP)                   ; СФОРМИРОВАТЬ В ВЕРШИНЕ СТЕКА
        ROR     R0                      ; НОМЕР ДИСКОВОДА
        ROR     @SP                     ; ИЗ РАЗРЯДОВ 0–2 РЕГИСТРА R0
        ROR     R0                      ; В РАЗРЯДЫ 13–15 СЛОВА
        ROR     @SP                     ; В ВЕРШИНЕ СТЕКА
        ROR     R0                      ; ПЕРЕНОС ВЫПОЛНЯЕТСЯ ЧЕРЕЗ БИТ "С"
        ROR     @SP                     ; В PS
        MOV     #NSEC, R0               ; НАЧАЛЬНОЕ ЗНАЧЕНИЕ НАКОПИТЕЛЯ
        BR      2⊙                      ; –> В ЦИКЛ ПЕРЕСЧЕТА
1⊙:     ADD     #CONST, R0              ;ПЕРЕЙТИ К СЛЕДУЮЩЕЙ ПОВЕРХНОСТИ
2⊙:     SUB     #NSEC, R1               ; ВЫЧЕСТЬ ДЛИНУ ПОВЕРХНОСТИ ИЗ ЧИСЛА
                                        ; БЛОКОВ
        BPL     1⊙                      ; КОНЕЦ ПЕРЕСЧЕТА?
        ADD     R0, R1                  ; ДА – ПОЛУЧИТЬ АДРЕС НА ДИСКЕ
        BIS     (SP)+, R1               ; ДОПОЛНИТЬ НОМЕРОМ ДИСКОВОДА

        TST     R2                      ; ОПЕРАЦИЯ ЗАПИСИ?
        BMI     3⊙                      ; ДА
        NEG     R2                      ; НЕТ, ИЗМЕНИТЬ ЗНАК СЧЕТЧИКА СЛОВ
        JMP     DREAD                   ; ПЕРЕЙТИ НА НУЖНУЮ
3⊙:     JMP     DWRITE                  ; ПОДПРОГРАММУ
        .END
```

В начале программы разряды 0—2 регистра R0, определяющие номер дисковода, переносятся в разряды 13—15, как это требуется форматом регистра адреса диска, и записываются в слове в вершине стека. В качестве второго варианта соответствующего фрагмента программы может быть использована приведенная последовательность инструкций:

```
ROR  R0
ROR  R0
ROR  R0
ROR  R0
BIC  #17777,R0
MOV  R0, —(SP)
```

Пересчет номера блока в адрес на диске заключается в получении номера сектора и номера поверхности дорожки. Это может быть выполнено делением номера блока на число секторов на поверхности дорожки. При этом частное от деления будет номером поверхности дорожки, а остаток — номером сектора.

Для пересчета номера блока в адрес на диске используется следующий алгоритм. Регистр R0 рассматривается как содержащий три поля: разряды 0—3 отведены для номера сектора; разряды 4—12 — для номера поверхности дорожки; разряды 13—15 — для номера дисковода.

Таким образом, инструкция ADD #CONST,R0 соответствует увеличению на 1 поля разрядов 4—12. Для определения требуемого номера поверхности дорожки проводится вычитание числа секторов на поверхности дорожки из заданного номера блока и увеличение на 1 значения в поле 2 регистра R0. Этот процесс продолжается до получения отрицательного значения в R1. Отрицательная величина, полученная в R1 при сложении с NSEC, дает остаток от деления номера блока на NSEC, т. е. номер сектора. В поле разрядов 4—12 содержится частное, т. е. номер поверхности дорожки.

После формирования значений в полях 0—3 и 4—12 инструкция BIS заносит в поле 13—15 номер дисковода. На этом определение адреса на диске завершается.

В заключительном фрагменте программы по знаку счетчика слов в регистре R2 производится передача управления на соответствующую подпрограмму. Перед переходом в подпрограмме чтения знак счетчика изменяется в соответствии с требованиями подпрограммы DREAD. Передача управления осуществляется по инструкции JMP, поскольку адрес возврата уже находится в стеке и никаких дополнительных действий не требуется.

8.3. РАБОТА С ПРЕРЫВАНИЯМИ

Работа по прерываниям с внешними устройствами, как правило, более сложная, чем работа по готовности.

В качестве примера рассмотрим программу копирования перфоленты по прерываниям:

```
        .ENABL  LC
        .TITLE  PTCOPY                  ; ДУБЛИРОВАНИЕ ПЕРФОЛЕНТЫ
        .IDENT  /V02.02/
+
```

```
;  МАКРОКОМАНДЫ ИЗ БИБЛИОТЕКИ
;  -
        .MCALL  MES                     ;  ВЫВОД СТРОКИ НА ТЕРМИНАЛ
;  +
;  ИСПОЛЬЗУЕМЫЕ ПОДПРОГРАММЫ:
;  -
        .GLOBL  TYPE                    ;  ДЛЯ МАКРОКОМАНДЫ
;  +
;  ЛОКАЛЬНЫЕ СИМВОЛЫ:
;  -
PRCSR  =  177550                        ;  АДРЕС РЕГИСТРА CSR П. ВВОДА
PRBUF = PRCSR + 2                       ;  БУФЕР ПЕРФОВВОДА
PPCSR = PRCSR + 4                       ;  CSR ВЫВОДА НА ПЛ
PPBUF = PRCSR + 6                       ;  БУФЕР ВЫВОДА НА ПЛ
INTBIT = 100                            ;  БИТ ПРЕРЫВАНИЯ В CSR
GOBIT = 1                               ;  БИТ ЗАПУСКА ВВОДА ПЛ
PR4    ≒ 200                            ;  ПРИОРИТЕТ ОБРАБОТКИ ПРЕРЫВАНИЯ
PRVEC = 70                              ;  ВЕКТОР ПРЕРЫВАНИЯ ВВОДА В ПЛ
PPVEC = PRVEC + 4                       ;  ВЕКТОР ПРЕРЫВАНИЯ ВЫВОДА НА ПЛ
        .ASECT
        .=PRVEC
        .WORD  INTPR                    ;  СТАТИЧЕСКАЯ УСТАНОВКА ВЕКТОРА
        .WORD  PR4                      ;  ПРЕРЫВАНИЯ PR
        .PSECT
START:
        MOV   #INTPP, @#PPVEC           ;  ДИНАМИЧЕСКАЯ УСТАНОВКА
        MOV   #PR4, @#PPVEC+2           ;  ВЕКТОРА ПРЕРЫВАНИЯ
LOOP:   MES   <ПОСТАВЬТЕ ПЕРФОЛЕНТУ И НАЖМИТЕ>
        MES   <КЛАВИШУ "ПРОД" НА ПУЛЬТЕ>
        HALT
1⊙:     BIS   #INTBIT ! GOBIT, @#PRCSR  ;  ЗАПУСТИТЬ ВВОД
        WAIT                            ;  ЖДАТЬ ПРЕРЫВАНИЯ
        TST   @#PRCSR                   ;  КОНЕЦ ПЛ?
        BMI   EOT                       ;  ДА
        MOVB  @#PRBUF, @#PPBUF          ;  ВЫВЕСТИ БАЙТ
        BIS   #INTBIT, @#PPCSR          ;  РАЗРЕШИТЬ ПРЕРЫВАНИЕ PP
        WAIT                            ;  ДОЖДАТЬСЯ ЕГО
        TST   @#PPCSR                   ,  ПРОВЕРИТЬ ОШИБКУ
        BPL   1⊙                        ;  НЕТ – ПРОДОЛЖИТЬ
        MES   <ОШИБКА ВЫВОДА••••>
        BR    LOOP                      ;  НАЧАТЬ СНАЧАЛА
EOT:    MES   <ДУБЛИРОВАНИЕ ОКОНЧЕНО••••>
        BR    LOOP                      ;  НАЧАТЬ СНАЧАЛА
;  +
;  ПОДПРОГРАММЫ ОБРАБОТКИ ПРЕРЫВАНИЙ.
;  -
INTPR:  BIC   #INTBIT, @#PRCSR          ;  ЗАПРЕТИТЬ ПРЕРЫВАНИЯ
        RTI                             ;  ВЫЙТИ ИЗ ПРЕРЫВАНИЯ
INTPP:  BIC   #INTBIT, @#PPCSR          ;  ЗАПРЕТИТЬ ПРЕРЫВАНИЯ
        RTI                             ;  ВЫЙТИ ИЗ ПРЕРЫВАНИЯ
        .END  START
```

В этой программе показано использование макрокоманды, находящейся в библиотеке макроопределений. Используемая макрокоманда описана в программе PTCOPY V01.03. Предполагается, что макрокоманда внесена в библиотеку, из которой она вызывается по директиве .MCALL. Макрокоманда использует внешнюю подпрограмму, поэтому точка входа в эту подпрограмму объявляется глобальным символом.

Программа демонстрирует два возможных способа установки вектора прерывания. Первый способ (статический): со-

держимое вектора прерывания определяется на этапе трансляции программы. При загрузке программы в память в ячейке вектора прерывания будут занесены соответствующие значения. Для такого определения всегда используется абсолютная программная секция.

Второй способ (динамический) определяет содержимое ячеек вектора прерываний в начале выполнения программ с помощью инструкции MOV. Этот способ целесообразно использовать при необходимости переопределять содержимое вектора во время работы программы.

При работе устройство запускается с одновременным разрешением прерывания и процессор переходит в состояние ожидания по инструкции WAIT. Завершение операции (нормальное или по ошибке) приводит к прерыванию, т. е. к передаче управления на соответствующую подпрограмму. В подпрограмме отменяется разрешение на прерывание («запрещаются» прерывания) и выполняется инструкция RTI. На этом обработка прерывания заканчивается, а управление передается на инструкцию, следующую за WAIT.

Общая схема программы совпадает с версией V01.03. Запускается устройство чтения, и программа ждет готовности (в данном случае по WAIT), затем анализируется наличие ошибки, т. е. конца ленты. Когда байт принят, он пересылается на вывод, с ожиданием готовности по WAIT и проверяется ошибка вывода.

Эта последовательность действий повторяется до обнаружения конца дублируемой ленты или ошибки вывода.

Используемая схема корректна и может применяться в различных случаях. Предпочтительнее способ, при котором работа с внешним устройством переносится в программу обработки прерывания. Продемонстрируем этот способ, изменив текст программы с метки 1⊙.

```
1⊙:     ...
        COM     @#PRSCR                 ; РАЗРЕШИТЬ ПРЕРЫВАНИЯ И НАЧАТЬ
                                        ; ВВОД С PR
2⊙:     BR      2⊙                      ; ЦИКЛ ОЖИДАНИЯ
; ПРОГРАММА ОБРАБОТКИ ПРЕРЫВАНИЙ
; —
INTPR:  CLRB    @#PRCSR                 ; ЗАПРЕТИТЬ ПРЕРЫВАНИЯ ОТ PR
        TST     @#PRCSR                 ; КОНЕЦ ЛЕНТЫ?
        BMI     EOT
        MOVB    @#PRBUF, @#PPBUF
        COM     @#PPCSR                 ; РАЗРЕШИТЬ ПРЕРЫВАНИЯ ОТ PP
        RTI
INTPP:  CLRB    @#PPCSR                 ; ЗАПРЕТИТЬ ПРЕРЫВАНИЯ ОТ PP
        TST     @#PPCSR                 ; ОШИБКА?
        BMI     ERR
        COM     @#PRCSR                 ; ПРОЧИТАТЬ С ПРЕРЫВАНИЕМ
        RTI
EOT:    MES     <ДУБЛИРОВАНИЕ ОКОНЧЕНО****>
        BR      FIN
ERR:    MES     <****ОШИБКА ВЫВОДА>
FIN:    CMP     (SP)+, (SP)+            ; ОЧИСТКА СТЕКА
        BR      LOOP
        .END    START
```

Поскольку регистр PRCSR имеет только два разряда (0 и 6), которые могут изменяться программой, команда COM в приведенном варианте выполняет действия, аналогичные инструкции BIS в предыдущем. Точно так же инструкция CLR очищает только разряд разрешения прерываний. При этом остальные разряды (7 и 15) не изменяются, и для их последующего анализа применяется инструкция TST.

При обнаружении конца ленты или ошибки вывода распечатывается соответствующее диагностическое сообщение и обработка прерывания заканчивается очисткой стека, в котором были заполнены адрес и состояние прерванной программы (два слова). Очистка производится инструкцией CMP (SP)+, (SP)+, вместо которой может быть использована инструкция ADD #4, SP.

После этого управление передается на начало программы, а не в цикл с меткой 2⊙, как это происходило при выходе из прерывания по RTI.

Указанный цикл выполняется процессором в течение всего времени, свободного от обработки прерываний. В это время процессор может выполнять любую «полезную» работу, например

```
        ...
2⊙:     TSTB    @#TTKCSR
        BPL     2⊙
3⊙:     BIT     #ERRBIT!DONBIT,@#LPCSR
        BMI     2⊙
        BPL     3⊙
        MOVB    @#TTKBUF,@#LPBUF
        BR      2⊙
        ...
```

В приведенном фрагменте по готовности осуществляется вывод на устройство печати знаков, введенных с клавиатуры терминала.

Перечисленные способы работы по прерываниям допускают различные модификации при работе с разными устройствами, в том числе и с диском. При этом не появляется принципиально новых элементов в программировании по сравнению с рассмотренными. Тем, кто заинтересуется возможностью выполнения рассмотренных примеров на ЭВМ, следует учесть несколько замечаний.

В программах не устанавливается начальное положение стека. Это может быть сделано в начале работы программы инструкцией

```
START: MOV  #START,SP  ;УСТАНОВИТЬ СТЕК
```

При трансляции текстов примеров макроассемблером под управлением одной из доступных операционных систем для получения нормального листинга программы следует обязательно использовать директиву .ENABL LC.

В противном случае листинг программы будет иметь несколько необычный вид: все используемые буквы кириллицы в комментариях и сообщениях будут преобразованы в соответствующие буквы латинского алфавита.

ЛИТЕРАТУРА

1. СМ ЭВМ: Комплексирование и применение/Г. А. Егоров, К. В. Песелев, В. В. Родионов и др.; Под ред. Н. Л. Прохорова. — М.: Финансы и статистика, 1986. — 304 с.
2. Кнут Д. Искусство программирования для ЭВМ. В 3-х т. — М.: Мир, 1976—1977. — Т. 1—3.
3. Пратт Т. Языки программирования: разработка и реализация. — М.: Мир, 1979. — 576 с.
4. Боярченков М. А., Кабалевский А. Н. Система аппаратных интерфейсов системы малых ЭВМ (СМ ЭВМ)//Технические средства мини-ЭВМ. Труды ИНЭУМ. — 1977. — Вып. 61. — С. 10—19.
5. Филин А. В., Солохин А. А. Организация обработки прерываний в системном интерфейсе «Общая шина» СМ ЭВМ//Технические средства мини-ЭВМ. Труды ИНЭУМ. — 1977. — Вып. 61. — С. 20—30.
6. Филинов Е. Н., Семик В. П. Программное обеспечение УВК СМ-3// Приборы и системы управления. — 1977. — № 10. — С. 15—17.
7. Операционная система СМ ЭВМ РАФОС/Валикова Л. И., Вигдорчик Г. В., Воробьев А. Ю., Лукин А. А.; Под общ. ред. Семика В. П.— М.: Финансы и статистика, 1984. — 207 с.
8. Беляков М. И. Система малых ЭВМ. Комплексы СМ-3 и СМ-4. Программное обеспечение (Операционная система ОСРВ): Отраслевой каталог.— М., 1983. — Вып. 3, 4, 5.
9. Диалоговая многотерминальная система для СМ ЭВМ/Семик В. П., Остапенко Г. П., Фридман А. Л., Горский В. Е — М.: Финансы и статистика, 1983. — 149 с.
10. Дисковая операционная система коллективного пользования/В. Д. Праченко, А. Г. Самборский, М. В. Чумаков; Под ред. Е. Н. Филинова. — М.: Финансы и статистика, 1985. — 207 с.
11. Инструментальная мобильная операционная система ИНМОС/Беляков М. И., Ливеровский А. Ю., Семик В. П., Шяудкулис В. И. — М.: Финансы и статистика, 1985. — 231 с.

ПРИЛОЖЕНИЯ

1. Технические характеристики моделей типа СМ-4

Общие

Система счисления — двоично-восьмеричная
Системный интерфейс — общая шина
Разрядность — 16
Количество регистров — 8
Число режимов адресации — 12
Система прерываний — приоритетная
Число приоритетных уровней: 4 — программных, 5 — внешних устройств

Индивидуальные

Характеристика	Модель				
	СМ1300	СМ1300.01	СМ-4	СМ1420	СМ1600
Адресуемый объем ОЗУ, Кслов	28	124	124	1920	1920
Диспетчер памяти	Нет	Есть	Есть	Есть	Есть
Операции с плавающей запятой	Програм-мно	FIS	FIS	FPP	FPP
Время выполнения операций, мкс:					
«регистр — регистр»	2	2	1,2	1	1
«регистр — память»	4,5	4,5	3,3	2,5	2,5
«память — память»	5,5	5,5	4,7	3,2	3,2
Обработка прерываний	20	20	10	8	8

2. Регистры внешних устройств

Адреса на ОШ	Назначение
777776	Слово состояния процессора
777774	Регистр — ограничитель стека
777656—777640	Регистры диспетчера памяти
777616—777600	Регистры диспетчера памяти
777570	Регистр переключателей пульта процессора
777566—777560	Регистры системного терминала

777556—777550	Перфоленточное устройство ввода-вывода
777546	Регистр состояния таймера
777516—777514	Устройство печати
777502—777500	Накопитель на кассетной магнитной ленте
777416—777400	Накопитель на кассетном магнитном диске
777172—777170	Накопитель на гибком магнитном диске
777166—777160	Устройство ввода с перфокарт
776736—776710	29-Мбайтовый пакет дисков
772544—772540	Регистры программируемого таймера
772532—772520	Накопитель на магнитной ленте
772356—772340	Регистры диспетчера памяти
772316—772300	Регистры диспетчера памяти

3. Векторы прерываний

Адрес/приоритет	Пояснения
004	Ошибка шины, обращение по нечетному адресу или к несуществующей ячейке
010	Зарезервированный код инструкции
014	Прерывание по инструкции BPT или биту слежения
020	Прерывание по инструкции IOT
024	Отказ питания
030	Прерывание по инструкции EMT
034	Прерывание по инструкции TRAP
060/4	Прерывание от клавиатуры системного терминала
064/4	Прерывание от экрана системного терминала
070/4	Устройство ввода с перфоленты
074/4	Устройство вывода на перфоленту
100/6	Прерывание от таймера (сетевого)
104/6	Прерывание от программируемого таймера
200/4	Устройство печати
220/5	Кассетный магнитный диск
224/5	Магнитная лента
230/6	Устройство ввода с перфокарт
250	Прерывание от диспетчера памяти
254/5	29-Мбайтовый пакет дисков
260/6	Кассетная магнитная лента
264/5	Накопитель на гибком магнитном диске

4. Набор знаков символьного кода КОИ-7

В таблице приводится полный набор знаков кода КОИ-7 и их наименований. Запись типа CTRL/A означает, что для ввода знака с клавиатуры терминала требуется нажатие клавиши A при нажатой клавише CTRL или УС.

Таблица

Восьмеричный код	Международное обозначение	Наименование и ввод с терминала
000	NUL	Пусто
001	SOH	Начало заголовка; CTRL/A
002	STX	Начало текста; CTRL/B
003	ETX	Конец текста; CTRL/C
004	EOT	Конец передачи; CTRL/D

Восьмеричный код	Международное обозначение	Наименование и ввод с терминала
005	ENQ	Кто там?; CTRL/E
006	ACK	Подтверждение; CTRL/F
007	BEL	Звонок; CTRL/G
010	BS	Возврат на шаг; CTRL/H
011	HT	Горизонтальная табуляция; CTRL/I
012	LF	Перевод строки; CTRL/J
013	VT	Вертикальная табуляция; CTRL/K
014	FF	Перевод формата; CTRL/L
015	CR	Возврат каретки; CTRL/M
016	SO	Выход; CTRL/N
017	SI	Вход; CTRL/O
020	DLE	Авторегистр 1; CTRL/P
021	DC1	Символ 1 управления устройством; CTRL/Q
022	DC2	Символ 2 управления устройством; CTRL/R
023	DC3	Символ 3 управления устройством; CTRL/S
024	DC4	Стоп; CTRL/T
025	NAK	Отрицание; CTRL/U
026	SYN	Синхронизация; CTRL/V
027	ETB	Конец блока; CTRL/W
030	CAN	Аннулирование; CTRL/X
031	EM	Конец носителя; CTRL/Y
032	SUB	Замена; CTRL/Z
033	ESC	Авторегистр 2; CTRL/Ш
034	FS	Разделитель файлов; CTRL/Э
035	GS	Разделитель групп; CTRL/Щ
036	RS	Разделитель записей; CTRL/Ч
037	US	Разделитель элементов; CTRL/—
040		Пробел
041	!	Восклицательный знак
042	"	Кавычки
043	#	Номер
044	⊙	Знак денежной единицы
045	%	Процент
046	&	Коммерческое «И»
047	'	Апостроф
050	(	Левая круглая скобка
051	)	Правая круглая скобка
052	*	Звездочка
053	+	Плюс
054	,	Запятая
055	—	Минус
056	.	Точка
057	/	Наклонная черта
060	0	Цифры
061	1	
062	2	
063	3	
064	4	
065	5	
066	6	
067	7	
070	8	
071	9	
072	:	Двоеточие

Восьмеричный код	Международное обозначение	Наименование и ввод с терминала
073	;	Точка с запятой
074	<	Меньше
075	=	Равно
076	>	Больше
077	?	Вопросительный знак
100	@	Коммерческое «ЭТ»
101	A	Буквы латинского алфавита
102	B	
103	C	
104	D	
105	E	
106	F	
107	G	
110	H	
111	I	
112	J	
113	K	
114	L	
115	M	
116	N	
117	O	
120	P	
121	Q	
122	R	
123	S	
124	T	
125	U	
126	V	
127	W	
130	X	
131	Y	
132	Z	
133	[	Левая скобка
134	\	Обратная косая черта
135	]	Правая скобка
136	↑	Стрелка вверх
137	←	
140	Ю	Буквы русского алфавита
141	А	
142	Б	
143	Ц	
144	Д	
145	Е	
146	Ф	
147	Г	
150	Х	
151	И	
152	Й	
153	К	
154	Л	
155	М	

Восьмеричный код	Международное обозначение	Наименование и ввод с терминала
156	Н	
157	О	
160	П	
161	Я	
162	Р	
163	С	
164	Т	
165	У	
166	Ж	
167	В	
170	Ь	
171	Ы	
172	З	
173	Ш	
174	Э	
175	Щ	
176	Ч	
177	DEL	Забой

5. Код RADIX-50

Символ	Восьмеричный эквивалент символьного кода КОИ-7	Эквивалент
Пробел	40	0
A—Z	101—132	1—32
⊙	44	33
.	56	34
Не используется		35
0—9	60—71	36—47

Приведенная ниже таблица позволяет переводить знаки символьного кода в эквивалентные коды RADIX-50. Например, строка X2B, записанная в коде RADIX-50, будет иметь следующее значение (арифметические действия выполняются в восьмеричной системе счисления):

X = 113000
2 = 002400
B = 000002
X2B = 115402

Знаком * отмечен неиспользующийся код.

Одиночный или первый знак		Второй знак		Третий знак	
Пробел	000000	Пробел	000000	Пробел	000000
A	003100	A	000050	A	000001
B	006200	B	000120	B	000002
C	011300	C	000170	C	000003
D	014400	D	000240	D	000004
E	017500	E	000310	E	000005

Продолжение

Одиночный или первый знак		Второй знак		Третий знак	
F	022600	F	000360	F	000006
G	025700	G	000430	G	000007
H	031000	H	000500	H	000010
I	034100	I	000550	I	000011
J	037200	J	000620	J	000012
K	042300	K	000670	K	000013
L	045400	L	000740	L	000014
M	050500	M	001010	M	000015
N	053600	N	001060	N	000016
O	056700	O	001130	O	000017
P	062000	P	001200	P	000020
Q	065100	Q	001250	Q	000021
R	070200	R	001320	R	000022
S	073300	S	001370	S	000023
T	076400	T	001440	T	000024
U	101500	U	001510	U	000025
V	104600	V	001560	V	000026
W	107700	W	001630	W	000027
X	113000	X	001700	X	000030
Y	116100	Y	001750	Y	000031
Z	121200	Z	002020	Z	000032
⊙	124300	⊙	002070	⊙	000033
.	127400	.	002140	.	000034
	132500*		002210*		000035*
0	135600	0	002260	0	000036
1	140700	1	002330	1	000037
2	144000	2	002400	2	000040
3	147100	3	002450	3	000041
4	152200	4	002520	4	000042
5	155300	5	002570	5	000043
6	160400	6	002640	6	000044
7	163500	7	002710	7	000045
8	166600	8	002760	8	000046
9	171700	9	003030	9	000047

6. Специальные знаки макроассемблера

Знак	Функция
<VT>	Ограничитель исходной строки
:	Ограничитель метки
=	Оператор прямого присваивания
%	Признак терма регистра
<HT>	Ограничитель отдельного элемента или поля
пробел	Ограничитель отдельного элемента или поля
#	Признак непосредственного режима адресации
@	Признак косвенной адресации
(	Начальный указатель регистра
)	Конечный указатель регистра
,	Разделитель полей операндов
;	Признак комментариев
+	Знак арифметического сложения или признак автоувеличения
—	Знак арифметического вычитания или признак автоуменьшения

*	Знак операции умножения
/	Знак операции деления
&	Знак логической операции «И»
!	Знак логической операции «ИЛИ»
#	Признак размещения двух знаков в символьном коде
'	Признак размещения одного знака в символьном коде
.	Счетчик адресов программы
<	Начальный указатель аргумента
>	Конечный указатель аргумента
↑	Знак однооперандной операции или признак аргумента
\	Признак символьного представления числового значения аргумента макрокоманды

7. Директивы макроассемблера

' — апостроф, за которым следует один знак; генерирует слово, содержащее 7-разрядное представление знака и нуль в старшем байте; этот знак используется так же, как указатель конкатенации аргументов в макрорасширении.

" — знак «кавычки», за которым следуют два знака; генерирует слово, содержащее 7-разрядное представление двух знаков в символьном коде; первый знак записывается в младшем байте, второй — в старшем.

↑BN — временное управление основанием счисления; число N должно обрабатываться как двоичное число.

↑C <EXPR> — временное управление форматом представления чисел; генерирует слово, содержащее значение выражения EXPR в обратном коде.

↑DN — временное управление основанием счисления; число N обрабатывается как десятичное.

↑FN — временное управление форматом представления чисел; формирует однословное представление числа с плавающей запятой.

↑ON — временное управление основанием счисления; число N должно обрабатываться как восьмеричное.

↑RSSS — преобразовать три знака SSS в код RADIX-50.

.ASCII/STRING/ — генерирует блок данных, содержащий эквивалент знаковой строки STRING в символьном коде; каждый знак размещается в одном байте.

.ASCIZ/STRING/ — генерирует блок данных, содержащий эквивалент знаковой строки STRING в символьном коде; строка заканчивается байтом, содержащим нуль; каждый знак размещается в одном байте.

.ASECT — начало или продолжение абсолютной программной секции.

.BLKB EXPR — резервирует блок памяти; длина блока в байтах равна значению выражения.

.BLKW EXPR — резервирует блок памяти; длина блока в словах равна значению выражения.

.BYTE EXPR1, EXPR2 — вычисляет и записывает в последовательно расположенные байты значения указанных выражений.

.CROSS SYM1, SYM2, ... — разрешение сбора информации об указанных символах SYM для таблицы перекрестных ссылок. Если аргументы в директиве не указаны, то разрешение касается всех символов программы.

.CSECT[NAME] — начало или продолжение наименованной или именованной перемещаемой секции; эта директива обеспечивает совместимость с другими трансляторами с языка Макро.

.DSABL ARG — запрещает выполнение функции макроассемблера, определяемой аргументом.

.ENABL ARG — разрешает выполнение функции макроассемблера, определяемой аргументом.

.END[EXPR] — указывает конец исходной программы; аргумент, если он указан, определяет адрес передачи управления при запуске программы.

.ENDC — указывает на конец блока условной трансляции.

.ENDM[NAME] — указывает на конец блока повторений, блока неопределенных повторений или макроопределения; имя NAME, если оно задано, должно быть идентично имени макрокоманды.

.ENDR — указывает на конец блока повторений; эта директива используется для совместимости с другими трансляторами с языка Макро.

.ERROR EXPR; STRING — вызывает вывод сообщения на втором проходе трансляции в файл листинга или на терминал, с которого введена командная строка транслятору; сообщение содержит значение выражения EXPR, если оно задано, и строку текста STRING.

.EVEN — обеспечивает четность счетчика адреса программы увеличением его на 1 в случае его нечетности.

.FLT2 ARG1, ARG2, ... — генерирует двухсловное представление чисел с плавающей запятой; если заданы несколько аргументов, они размещаются в последовательных двухсловных блоках памяти.

.FLT4 ARG1, ARG2, ... — генерирует четырехсловное представление числа с плавающей запятой; если заданы несколько аргументов, они размещаются в последовательных четырехсловных блоках памяти.

.GLOBL NAME1, NAME2, ... — определение символов NAME в качестве глобальных.

.IDENT/STRING/ — позволяет присвоить объектному модулю номер версии; номер версии — строка STRING в кодировке RADIX-50, заключенная в парные ограничители.

.IF CND, ARG1, ARG2,... — начало блока условной трансляции, тело блока транслируется только в том случае, если указанное условие CND выполняется при данных аргументах ARG.

.IFF — используется только внутри блока условной трансляции; указывает на начало группы строк программы, которая транслируется, если условие, указанное в начале блока, не выполняется.

.IFT — используется только внутри блока условной трансляции, указывает на начало группы строк программы, которая транслируется, если условие, заданное в начале блока, выполняется.

.IFTF — используется только внутри блока условной трансляции; указывает на начало группы строк программы, которая транслируется независимо от выполнения условия, указанного в начале блока.

.IIF CND, ARG, COMMAND — действует как блок условной трансляции, состоящий из одной строки; оператор COMMAND транслируется только в случае, если условие оказалось выполненным.

.INCLUDE STRING — указывает спецификацию файла, текст из которого включается в трансляцию в данном месте программы.

.IRP NAME, <ARG1, ARG2, ...> — указывает на начало блока неопределенных повторений, блок расширяется последовательно для каждого аргумента из списка, заключенного в угловые скобки.

.IRPC NAME,<STRING> — указывает на начало блока неопределенных повторений, в котором переменная NAME принимает последовательно значение каждого знака строки, заключенной в угловые скобки.

.LIBRARY STRING — добавляет указанную спецификацию файла к списку макробиблиотек, используемых при определении макрокоманд директивой .MCALL.

.LIMIT — резервирует два слова, в которые компоновщик запишет нижний и верхний адреса загрузочного модуля программы.

.LIST[ARG] — директива LIST без аргумента увеличивает содержимое внутреннего счетчика распечатки на 1; при наличии аргумента LIST не изменяет счетчик, а формирует листинг программы в соответствии с указанным аргументом.

.MACRO NAME, ARG1, ARG2, ... — указывает на начало макроопределения, содержит имя макрокоманды и список фиктивных (формальных) аргументов.

.MCALL ARG1, ARG2, ... — обеспечивает поиск и считывание макроопределений, указанных в директиве; поиск осуществляется сначала в пользователь-

ской библиотеке макроопределений, затем — в библиотеке системных макроопределений.

.MDELETE NAM1, NAM2, ... — удаляет определения указанных макрокоманд NAM из списков транслятора и освобождает занимаемую этими определениями память.

.MEXIT — вызывает прекращение генерации макрорасширения или расширения блока повторений.

.NARG NAME — присваивает указанному символу значение числа аргументов в вызове макрокоманды, расширяемой в текущий момент; может использоваться только в макроопределении.

.NCHR NAME <STRING> — присваивает указанному символу значение числа знаков в строке, заданной между ограничителями.

.NLIST[ARG] — директива без аргумента, уменьшает внутренний счетчик распечатки на 1; при наличии аргумента не создается часть листинга, указанная аргументом.

.NOCROSS SYM1, SYM2, ... — запрещение сбора информации об указанных символах SYM для таблицы перекрестных ссылок. Если аргументы в директиве не указаны, запрещение касается всех символов программы.

.ODD — обеспечивает нечетность значения счетчика адресов программы добавлением 1 в случае его четности.

.PACKED — упаковывает указанное десятичное число (до 31 десятичной цифры) по две цифры в байт.

.PAGE — по этой директиве начинается новая страница листинга.

.PRINT [EXPR]; STRING — обеспечивает вывод сообщения на втором проходе трансляции; сообщение содержит значение выражения, если оно указано в .PRINT, и текст строки STRING.

.PSECT[NAME], ARG1, ARG2,... — начинает или продолжает именованную или неименованную программную секцию, имеющую указанные аргументами атрибуты.

.RADIX N — заменяет текущее основание системы счисления на N; значения N : 2, 8 или 10.

.RAD50/STRING/ — генерирует блок данных, содержащий в коде RADIX-50 эквивалент знаковой строки, указанной между ограничителями.

.REM SIGN — включает следующие строки как комментарий до повторного появления знака SIGN в тексте.

.REPT EXPR — начинает блок повторений; число повторений тела блока указано в выражении EXPR.

.RESTORE — восстанавливает сохраненную предыдущей директивой .SAVE программную секцию. Восстановление производится с вершины стека сохраненных программных секций.

.SAVE — сохраняет имя и текущий счетчик ячеек указанной программной секции на вершине стека .PSECT.

.SBTTL STRING — определяет печать строки STRING как части заголовка страницы листинга; текстовая часть директивы заносится в оглавление листинга, который выводится на первом проходе трансляции.

.TITLE STRING — присваивает объектному модулю имя (первые 6 знаков указанной строки); заданная строка распечатывается на каждой странице листинга.

.WEAK SYM1, SYM2,... — определяет указанные символы SYM как ослабленные глобальные. Значение этих символов может задаваться в данном программном модуле или определяться при построении загрузочного модуля из других отдельно транслируемых модулей.

.WORD EXPR1, EXPR2,... — генерирует последовательно расположенные слова, содержащие значения указанных выражений.

8. Диагностика ошибок при трансляции

Код ошибки (флаг) печатается как первый символ в исходной строке листинга, которая содержит оператор с ошибкой, обнаруженной транслятором. Флаг ошибки идентифицирует тип обнаруженной ошибки.

Пример распечатки исходной строки, содержащей ошибку:

```
Q 125 003624 010500 MOV R5, R0, ABC
```

Посторонний (лишний) аргумент ABC в инструкции MOV вызывает появление флага ошибки Q. Ниже перечисляются флаги ошибок и возможные причины их появления.

Флаг A — ошибка трансляции. Этот флаг вызывается различными причинами, основными из которых являются следующие:

а) указано недопустимое значение аргумента для директив .RADIX, .LIST/.NLIST, .PSECT, .IF/.IIF, .ENABL/.DSABL, .MACRO;

б) опущен требуемый аргумент в директивах .TITLE, .IRP/.IRPC, .NARG/.NCHR/.NTIPE, .IF/.IIF;

в) аргумент содержит непарные ограничители или неправильная конструкция аргумента в директивах .ASCII/.ASCIZ/.RAD50/.IDENT, .NCHR;

г) общие ошибки в адресации аргумента, которые возникают, если:

в инструкции перехода превышен допустимый диапазон перехода, т. е. его значение выходит за границы от —128 до +127;

оператор прямого присваивания неверно изменяет текущий счетчик адресов, т. е. оператор вида «.=выражение» изменяет текущий счетчик адресов так, что его значение выходит за границу текущей программной секции;

оператор содержит недопустимое адресное выражение. Подобная ошибка возникает, если абсолютное адресное выражение содержит глобальный символ, перемещаемую величину или сложную перемещаемую величину, либо если относительное адресное выражение содержит глобальный символ или сложную перемещаемую величину. Особый случай этого типа ошибки возникает, если в директивах .BLKW/.BLKB/.REPT указано выражение, не являющееся абсолютным;

несколько выражений не разделены запятой, тогда следующий символ рассматривается как часть текущего выражения;

стек сохранения .PSECT переполнен при выполнении директивы .SAVE;

стек сохранения .PSECT оказался пуст при попытке выполнения макроассемблером директивы .RESTORE;

д) недопустимая ссылка вперед; эта ошибка возникает, если глобальный оператор присваивания /==/ содержит ссылку вперед, т. е. на символ, который будет определен в программе дальше, или в выражении, определяющем значение текущего счетчика адреса, содержится ссылка вперед.

Флаг B — ошибка границ адресации. Инструкции или слова данных транслируются по нечетному адресу памяти; текущий счетчик адреса корректируется прибавлением единицы.

Флаг D — ссылка на многократно определенную метку.

Флаг E — не обнаружена директива .END. Если Макро достигает конца входной программы и не встречает директивы .END, происходит ошибка с указанным флагом. Возникает также при переполнении стека транслятора. В этом случае в листинге программы в строку в точке возникновения ошибки ставится вопросительный знак (?).

Флаг I — обнаружен запрещенный знак. Запрещенные знаки, встретившиеся в исходном тексте программы, заменяются в листинге вопросительным знаком (?).

Флаг L — входная строка содержит более 132 знаков. Эта ошибка может появиться при замене в макрорасширении формальных аргументов фактическими.

Флаг M — многократное определение метки. Ошибка означает, что метка эквивалентна по первым шести знакам ранее встретившейся метке.

Флаг N — число содержит недопустимые в данной системе счисления цифры.

Флаг O — ошибочная операция. Неправильное использование директивы, например превышен допустимый уровень вложения блоков условной трансляции или сделана попытка использования макрокоманды, указанной в директиве .MCALL, макроопределение которой не обнаружено.

Флаг P — динамическая ошибка. Эта ошибка появляется, если значение метки меняется от одного прохода к другому, либо локальная метка многократно определена внутри блока локальных символов, либо блок локальных символов, определяемый директивой .ENABL LSB, переходит границу той программной

секции, в которой находится его начало. Данная ошибка появляется по директиве .ERROR.

Флаг Q — синтаксическая ошибка. Ошибка может быть вызвана недостатком или избытком аргументов для машинной инструкции.

Флаг R — ошибка использования регистра. Ошибка обычно связана с недопустимым использованием регистра или попыткой переопределения стандартного имени регистра без предварительного указания директивы .DSABL REG.

Флаг T — ошибка усечения. Используется число, занимающее более 16 разрядов слова, либо в директиве .BYTE или инструкциях EMT и TRAP используется выражение, значение которого занимает больше 8 разрядов.

Флаг U — неопределенный символ. При вычислении выражения встретился неопределенный символ, ему присваивается значение 0. В директиве .MCALL указано имя макрокоманды, которая не может быть найдена в макробиблиотеках. Оператор прямого присваивания содержит ссылку на символ, определение которого также содержит ссылку вперед; сделана ссылка на локальный символ, не существующий в текущем блоке локальных символов.

Флаг Z — ошибка машинной инструкции. Инструкция с заданными режимами адресации может неодинаково выполняться на различных типах процессоров СМ ЭВМ.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Практическое руководство

Геннадий Вениаминович Вигдорчик,
Александр Юрьевич Воробьев,
Виктор Дмитриевич Праченко

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АССЕМБЛЕРЕ ДЛЯ СМ ЭВМ

Зав. редакцией *И. Г. Дмитриева*
Редактор *Л. В. Речицкая*
Мл. редактор *Т. А. Студеникина*
Худож. редактор *С. Л. Витте*
Техн. редактор *Г. А. Полякова*
Корректоры *Г. А. Башарина* и *Е. А. Овешникова*
Обложка художника *А. Н. Жданова*

ИБ № 2004

Сдано в набор 20.11.86. Подписано в печать 29.05.87. А04421. Формат 60×90¹/₁₆. Бум. кн.-журн. Гарнитура «Литературная». Печать офсетная. Усл. п. л. 15,0. Усл. кр.-отт. 15,0. Уч.-изд. л. 15,52. Тираж 30 000 экз. Заказ 571. Цена 1 р. 10 к.

Издательство «Финансы и статистика». 101000, Москва, ул. Чернышевского, 7.

Типография им. Котлякова издательства «Финансы и статистика» Государственного комитета СССР по делам издательств, полиграфии и книжной торговли. 195273, Ленинград, ул. Руставели, 13.