МАКРО-АССЕМБЛЕР ДЛЯ СМ 1700

Г. П. Остапенко
Н. А. Толмачева
В. Е. Горский

МАКРО-АССЕМБЛЕР ДЛЯ СМ 1700

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1990

ББК 32.973
О-76

УДК 681.3.066

Рецензент: д-р физ.-мат. наук *С. В. Клименко*

Остапенко Г. П. и др.

О-76 Макроассемблер для СМ1700/Г. П. Остапенко, Н. А. Толмачева, В. Е. Горский. — М.: Финансы и статистика, 1990. — 239 с.: ил.

ISBN 5-279-00364-6.

Рассматриваются особенности языка Макроассемблер для 32-разрядных СМ ЭВМ, представление данных и форматы команд. Приводится описание процедур, необходимых для обработки и отладки программ, написанных на Макроассемблере. Благодаря использованию макроопределений и библиотек макросредств значительно упрощается и становится менее трудоемким процесс создания программного обеспечения.

Для разработчиков программного обеспечения СМ ЭВМ. Может использоваться как справочное пособие при обучении студентов и аспирантов.

$\text{О}\frac{2405000000-001}{010(01)-90}$ 116—90 **ББК 32.973**

ISBN 5-279-00364-6

ПРЕДИСЛОВИЕ

Машины серии СМ ЭВМ применяются в различных областях народного хозяйства. Широкий диапазон производительности моделей (от 8- до 32-разрядных), обилие периферийного оборудования и, самое главное, наличие значительного числа проблемно-ориентированных программных средств позволяют устанавливать эти ЭВМ как на местах возникновения и потребления данных, так и в крупных иерархических системах управления.

Данное пособие продолжает серию книг, посвященных описанию архитектуры и программного обеспечения 32-разрядной малой ЭВМ — СМ1700, которая с успехом применяется для решения широкого спектра пользовательских задач. В первой публикации [1] в основном обсуждались особенности структуры аппаратных средств СМ1700, во второй [2] — архитектура базовой операционной системы этой ЭВМ — многофункциональной операционной системы, поддерживающей виртуальную память (МОСВП).

В предлагаемой читателю книге рассматриваются основные возможности языка Макроассемблера СМ1700, а также способы его применения. Описание языка включает описание возможностей Макроассемблера для таких 16-разрядных моделей СМ ЭВМ, как СМ1420, СМ1600 и СМ1425. Это объясняется тем, что сохранение архитектурной и программной преемственности с перечисленными моделями являлось одной из главных задач при определении архитектуры СМ1700.

Язык Макроассемблер, или, как его иногда называют, Макро, представляет собой машинно-ориентированный язык программирования. Его основное назначение — использование архитектурных возможностей аппаратной машины [4], которые либо недоступны языкам программирования высокого уровня, либо снижают эффективность применения этих языков.

Макроассемблеру доступны практически все архитектурные возможности, заложенные в аппаратуру, и в то же время при

создании программ на этом языке используются сервисные средства операционной системы, включая развитые возможности самого транслятора. При этом Макроассемблер существенно повышает быстродействие программ (по сравнению с высокоуровневыми языками) и экономит оперативную память (в основном это относится к программированию работы внешних устройств).

Однако Макроассемблер не всегда является наиболее оптимальным средством программирования. При выборе языка программирования всегда следует учитывать не только предметную область, для которой будет использоваться программа (отсюда и возможные ограничения на быстродействие и объем памяти), но и срок ее разработки. Создать работающую и, главное, хорошо проверенную программу на языке высокого уровня можно намного быстрее, чем на Макроассемблере. Более того, в настоящее время широко используются трансляторы с языков программирования высокого уровня с оптимизацией по памяти. В связи с этим для того, чтобы создать достаточно объемную и действительно эффективную (по быстродействию и памяти) программу на языке Макроассемблера, помимо досконального знания архитектуры аппаратных средств, нужно обладать хорошим опытом программирования на этом языке, «выжимая» из него наиболее эффективные возможности.

Учитывая вышеизложенное, в книге, помимо подробных сведений о языке и многочисленных примеров, приведены краткие сведения об элементах архитектуры СМ1700, с которыми работает программист на Макроассемблере. Кроме того, для лучшего понимания материала приведен список терминов, который включает не только термины, используемые при описании Макроассемблера, но и термины из предыдущих публикаций [1, 2].

Необходимость подобного списка вызвана еще и тем, что, в отличие от предыдущих моделей СМ ЭВМ, на которых функционировало несколько операционных систем, МОСВП представляет собой «архитектурно-зависимую» систему, проектирование и разработка которой осуществлялась совместно с определением архитектуры аппаратных средств СМ1700. Это нашло свое отражение не только в том, что в МОСВП обеспечивается соответствующая аппаратная поддержка виртуальной органи-

зации памяти, но и аппаратно реализуются отдельные примитивы ОС (инструкции работы с очередями, обработки битовых полей переменной длины и т. п.). Кроме того, в набор программных интерфейсов СМ1700, с которыми работает программист на Макроассемблере, включены отдельные инструкции, являющиеся ключевыми как с точки зрения понимания архитектуры машины, так и с точки зрения внутренней организации и функционирования системы МОСВП.

В книге приведены и другие справочные материалы, которые в сжатом виде дают общее представление о Макроассемблере. В соответствующих таблицах даны ссылки на страницы, где обсуждаемые сведения (инструкции, режимы адресации, директивы Макроассемблера и т. п.) рассматриваются более подробно.

Система малых машин СМ ЭВМ постоянно развивается. Появляются новые программные продукты, разрабатываются модели с улучшенными показателями, чье развитие определяется не только новыми возможностями электронной промышленности, но и опытом конкретного применения. Практика использования вычислительных комплексов, построенных на базе машин СМ1700, выявила в основном только один их недостаток — в ряде случаев требуется бо́льшая производительность, чем та, которую предоставляет этот комплекс. Это «узкое» место в значительной мере преодолено с помощью 32-разрядных моделей СМ1702 и СМ1705, которые в архитектурном плане сохраняют преемственность с СМ1700, но обладают существенно большей производительностью.

Несмотря на отдельные архитектурные отличия (в частности, наличие в СМ1702 магистрального параллельного интерфейса (МПИ) вместо «общей шины» в СМ1700 и СМ1705), архитектура аппаратных средств, которые доступны программисту, создающему программы на Макроассемблере, одинакова для всех трех моделей. Исходя из этого содержание книги в полной мере относится к Макроассемблеру для СМ1702 и СМ1705, хотя упоминается только СМ1700.

От версии к версии происходит уточнение и развитие функциональных возможностей языка, а также вносятся изменения в компилятор, поэтому на момент выхода книги версия Мак-

роассемблера для СМ1702 или СМ1705 по отдельным позициям может отличаться от обсуждаемой в данной книге, однако эти изменения не будут носить принципиального характера и не повлияют на возможность применения ранее созданных пользовательских программ.

Книга рассчитана на читателей, знакомых с архитектурой моделей СМ ЭВМ и основами программирования. Специалисты, изучившие архитектуру СМ1700 [1], главу 1 могут пропустить. Для тех, кто владеет только начальными знаниями в области вычислительной техники, но хочет ознакомиться с основными возможностями Макроассемблера, можно порекомендовать работу [5].

Книга полезна для специалистов в области системного программирования и может использоваться как справочное пособие при обучении студентов и аспирантов.

Глава 1

АРХИТЕКТУРА СМ1700

Вычислительный комплекс (ВК) СМ1700 — очередная модель СМ ЭВМ, имеющая интерфейс «общая шина» [6]. На этот интерфейс выходят контроллеры внешних устройств, оперативная память и собственно процессор комплекса (рис. 1.1). Общая шина является магистралью, по которой передаются данные, адреса и управляющие сигналы. Процессор не различает обращение к оперативной памяти и регистрам контроллеров внешних устройств, в связи с чем в машинных инструкциях отсутствуют специальные инструкции ввода-вывода. Отличие оперативной памяти от регистров контроллеров заключается лишь в том, что диапазон адресов регистров фиксирован на общей шине и занимает самые старшие адреса.

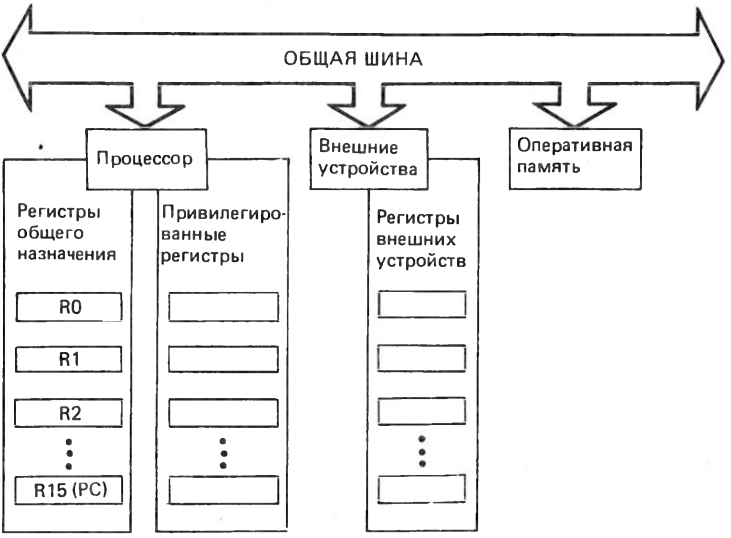


Рис. 1.1. Общая структурная схема СМ1700

На рис. 1.1 показаны только те структурные элементы СМ1700, с которыми имеет дело программист, работающий на Макроассемблере. Рассмотрим их более подробно, а также остановимся на некоторых вопросах организации работы основ-

ного процессора и контроллера оперативного запоминающего устройства (ОЗУ), которые полезно знать при работе на Макроассемблере.

В ряде случаев рассматриваемые нами аспекты функционирования этих устройств непосредственно не относятся к Макроассемблеру СМ1700. Однако понимание их внутренней организации и динамики работы полезно для разработки сложных программных комплексов на Макроассемблере.

1.1. ОСНОВНОЙ ПРОЦЕССОР

Основными элементами процессора, которые используются при программировании, являются слово состояния процессора и регистры различного назначения.

Длинное слово состояния процессора. Информация о текущем состоянии процессора и исполняемой в данный момент времени программы хранится в длинном 32-разрядном слове состояния процессора — PSL (рис. 1.2). Оно состоит из двух слов.

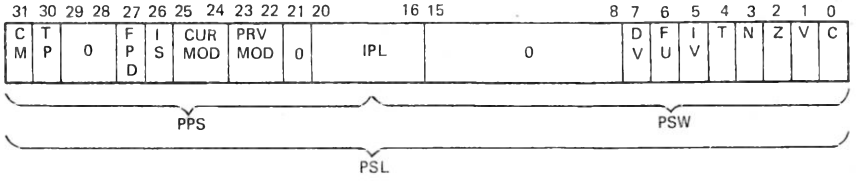


Рис. 1.2. Длинное слово состояния процессора

• привилегированного слова состояния процессора — PPS, которое содержит информацию о состоянии процессора (текущий и предыдущий режимы процессора, используется ли режим совместимости и т. д.). Эту информацию могут изменить только привилегированные системные программы;

• обычного слова состояния процессора — PSW, содержащего информацию о текущем состоянии исполняемой программы (коды условий, разрешение прерываний и т. д.).

Приведем значения разрядов слова состояния процессора.

Разряды 0—3 слова состояния представляют собой коды условий, которые содержат информацию о результате выполнения последней инструкции. При исполнении большинства инструкций они устанавливаются следующим образом:

C=1, если в результате выполнения инструкции произошел перенос или заимствование из старшего разряда результата.

V=1, если в результате выполнения инструкции произошло арифметическое переполнение;

Z=1, если результат выполнения инструкции равен нулю;
N=1, если результат выполнения инструкции отрицательный.

Разряды 4—7 являются разрядами слежения и определяются программистом:

T = 1 устанавливает разряд TP. Если разряд TP устанавливается в конце интерпретации инструкции, то перед выполнением следующей инструкции происходит прерывание;

IV=1 разрешает прерывание по целочисленному переполнению или по ошибке преобразования. Если разряд IV сброшен и произошло переполнение, прерывание не происходит, но при этом разряд V остается установленным;

FU=1 разрешает прерывание по нижней границе диапазона при выполнении операций с числами в формате с плавающей запятой;

DV=1 разрешает прерывание по десятичному переполнению.

Разряды 8—15 слова состояния процессора не используются и должны быть нулевыми.

Разряды 16—20 (IPL) определяют текущий приоритет процессора, который меняется в пределах 1—31. Процессор разрешает прерывания от внешних устройств, уровень приоритета которых больше текущего уровня процессора.

Разряд 21 не используется и должен быть нулевым.

Разряды 22—23 (PRV MOD) показывают предыдущий режим работы процессора. Они загружаются из поля текущего режима во время аппаратной обработки прерывания, а восстанавливаются при выполнении инструкции выхода из прерывания (REI).

Разряды 24—25 (CUR MOD) определяют текущий режим работы процессора. Возможны следующие режимы:

0 — режим ядра;
1 — режим управления;
2 — режим супервизора;
3 — режим пользователя.

Разряд 26 (IS) определяет стек, используемый в данный момент. Если IS=1, то процессор работает со стеком прерываний, в противном случае процессор работает со стеком, который определяется текущим режимом работы процессора.

Разряд 27 (FPD) определяет (если FPD=1), что первая часть инструкции выполнена и в случае исключительной ситуации выполнение инструкции не может быть начато с начала, а должно быть восстановлено с другой, зависящей от конкретной реализации точки. Разряд 27 может использоваться при программном моделировании нереализованных инструкций.

Разряды 28—29 не используются и должны быть нулевыми.

Разряд 30 (TP) — запрос трассировки. Если TP=1 в конце

интерпретации инструкции, то после ее выполнения будет инициировано прерывание.

Разряд 31 (СМ) — указатель режима совместимости. Когда СМ=1, процессор находится в режиме совместимости и выполняет инструкции из основного набора инструкций 16-разрядных СМ ЭВМ.

Регистры общего назначения. В процессоре СМ1700 имеется шестнадцать 32-разрядных регистров общего назначения (R0—R15). Эти регистры могут быть использованы как аккумуляторы, индексные регистры, указатели адресов, таблиц, списков и т. п., а также как указатели области памяти для временного хранения данных (стека). Конкретное использование регистров зависит от выбранного режима адресации.

Среди этих регистров следует особо выделить четыре регистра (R12—R15).

Регистр R15 используется в качестве счетчика инструкций. Он содержит адрес следующей исполняемой инструкции, что позволяет получить несколько дополнительных режимов адресации.

Регистр R14 употребляется как указатель стека. Каждый процесс имеет четыре стека, по одному для каждого режима работы процессора, определяемого разрядами 24—25 слова состояния процессора.

Регистр R13 является указателем кадра вызова. Для облегчения работы со стеком в СМ1700 при вызове процедуры создается кадр вызова, в котором обязательно сохраняются разряды 5—14 слова состояния процессора и регистры R15, R13 и R12. Кроме того, в кадре вызова могут сохраняться регистры R0 — R11. При выходе из процедуры слово состояния процессора и сохраненные регистры автоматически восстанавливаются.

Регистр R12 при вызове процедуры используется как указатель списка аргументов.

Этим регистрам присвоены следующие мнемонические обозначения:

AP — указатель списка аргументов (регистр R12);
FP — указатель кадра вызова (регистр R13);
SP — указатель стека (регистр R14);
PC — счетчик инструкций (регистр R15).

Специальное назначение регистров R12 — R14 не исключает возможность их использования для других целей.

Привилегированные регистры. В основном процессоре предусмотрено несколько десятков привилегированных регистров, доступ к которым, как правило, осуществляется в более привилегированных режимах, чем пользовательский. Наиболее важными из них являются:

• регистры управления и состояния основного процессора. Доступ к этим регистрам выполняется специальными инструкциями, требующими привилегии ядра;

• регистры, определяющие структуру виртуальной памяти, т. е. базовые регистры и регистры длин областей виртуальной памяти;

• регистры процесса и переключения контекста. Для процесса выделяется несколько регистров, которые загружаются и сохраняются во время операций загрузки контекста;

• регистры указателя стека. Обращение к указателю стека (SP), расположенному в области универсальных регистров, приведет к выбору одного из пяти возможных указателей стека — указателя стека режима пользователя, супервизора, управления, ядра или указателя стека прерываний;

• регистры часов времени года и программируемых часов с интервалом счета в 1 мкс.

1.2. РЕЖИМЫ РАБОТЫ ПРОЦЕССОРА

Процессор СМ1700 может работать в одном из четырех режимов, которые перечисляются ниже в порядке уменьшения привилегированности.

Режим ядра (режим 0). Он используется операционной системой МОСВП для управления процессами, управления виртуальной памятью, организации и выполнения операций ввода-вывода. В этом режиме поддерживаются и обрабатываются общесистемные структуры данных, отражающие состояние оперативной и виртуальной памяти, конфигурацию и параметры внешних устройств, параметры и состояние процессов. В этом же режиме работают системные обслуживающие программы, реализующие запросы на сервис от менее привилегированных режимов работы основного процессора. Они выполняют постановку заявок на ввод-вывод, изменяют состояние процесса, управляют виртуальным адресным пространством и т. п.

Режим управления (режим 1). Он служит для организации всех операций системы управления данными (СУД). СУД обеспечивает хранение и управление данными на дисковых и ленточных ВЗУ, а также доступ к устройствам терминального типа и печатающим устройствам. При этом СУД реализует обработку устройств с последовательной, относительной и индексной организацией.

Режим супервизора (режим 2). Он используется для работы интерпретатора с диалогового командного языка (DCL), представляющего собой интерфейс пользователя с системой. С помощью этого языка пользователь управляет распределением системных ресурсов, запускает на выполнение различные про-

граммы, отслеживает состояние системы и внешних устройств.

Режим пользователя (режим 3). Он предназначен для исполнения программ пользователя и различных сервисных средств МОСВП, используемых при разработке и отладке программных средств (редакторы, трансляторы, компоновщик, утилиты общего назначения и т. п.).

С помощью специальных инструкций обеспечивается стандартный механизм изменения режима работы процессора на более привилегированный, что позволяет менее привилегированным программам вызывать более привилегированные. При переходе к другому режиму работы предыдущий режим сохраняется в слове состояния PSL в поле предыдущего режима (разряды 24—25), позволяя таким образом более привилегированной программе определить уровень привилегированности вызвавшей программы.

1.3. СИСТЕМА ПРЕРЫВАНИЙ И ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

Система прерываний ВК СМ1700 организована в соответствии с многоуровневой системой приоритетов внешних устройств и обеспечивает эффективную работу основного процессора в режиме реального времени, разделения времени и пакетном режиме.

Процессор имеет 31 уровень приоритета прерывания (IPL), которые разделяются на 15 программных уровней (нумеруются от 1 до 15) и 16 аппаратных уровней (нумеруются от 16 до 31). Программы пользователя и программы системного обслуживания работают на уровне IPL0. Старшие номера уровней прерывания имеют более высокий приоритет, т. е. только запрос на уровне, более высоком, чем текущий IPL процессора, будет немедленно прерывать программу с низким приоритетом.

Уровни прерывания 1—15 существуют исключительно для программного обеспечения. Никакие устройства не могут запрашивать прерывания по этим уровням. Запрос программного прерывания будет сброшен аппаратурой сразу, как только произойдет прерывание.

Уровни прерывания 16—23 предназначены для использования устройствами и контроллерами, подключенными к общей шине. Уровни BR4 — BR7 для СМ1420 на общей шине соответствуют уровням прерываний 20—23 в СМ1700.

Уровни прерывания 24—31 отведены для инициализации и обработки безотлагательных прерываний, включая внутренние сетевые часы, серьезные ошибки и неисправность питания.

Процессор удовлетворяет запросы прерывания согласно приоритету. Только когда приоритет запроса на прерывание

выше, чем текущий IPL (разряды 16—20 длинного слова состояния процессора), процессор повышает свой уровень приоритета и обслуживает запрос прерывания. Процедура обслуживания прерывания инициируется на уровне IPL запроса прерывания и обычно не изменяет уровень приоритета, установленный процессором.

Если в процессе работы текущего процесса по какому-либо событию возникает необходимость выполнения определенного участка другого образа, внешнего по отношению к данному, то по соответствующему прерыванию процессор передает управление новому процессу, фиксируя это изменение в текущем выполненном процессе. В случае если событие, вызвавшее прерывание, относится к текущему выполненному процессу и инициирует работу другого образа в контексте текущего процесса, то оно называется исключительной ситуацией.

Большинство процедур обслуживания исключительных ситуаций выполняются на уровне IPL0 в ответ на события (например, нарушения), возникающие в программе. Процедуры обслуживания исключительной ситуации обычно программируются так, чтобы избежать других исключительных ситуаций, однако вложенные исключительные ситуации могут возникать.

Примерами исключительных ситуаций являются:

- «ловушка» (TRAP). Она создается в конце выполнения машинной инструкции, вызвавшей исключительную ситуацию. В связи с этим счетчик инструкций PC, сохраняемый в стеке, содержит адрес следующей инструкции, которая должна была бы нормально выполняться. Любая программа может разрешить или запретить некоторые из ловушек для какой-то конкретной инструкции;
- «сбой» (FAULT). Он возникает во время выполнения инструкции и оставляет регистры и память в корректном состоянии, т. е. при устранении условия, вызвавшего сбой, и последующем запуске инструкции она будет давать правильные результаты;
- «аварийное завершение» (ABORT). Оно возникает во время выполнения инструкции и оставляет регистры и память в неопределенном состоянии. В этом случае при последующем запуске инструкция не может быть правильно завершена.

В целом исключительные ситуации и прерывания очень похожи: когда одно из них возникает, значения состояния процессора (PSL) и программного счетчика (PC) загружаются в соответствующий стек. Однако имеются существенные различия:

- исключительная ситуация возникает при выполнении текущей инструкции, в то время как прерывание возникает асинхронно и может быть независимо от этой инструкции;

• исключительная ситуация обычно обрабатывается в контексте процесса, который привел к исключительной ситуации, в то время как прерывание обслуживается независимо от текущего выполняющего процесса;

• значение IPL процессора обычно не изменяется при возникновении исключительной ситуации, в то время как при прерывании значение IPL всегда повышается;

• процедуры обслуживания исключительных ситуаций обычно работают со стеком процессов, в то время как процедуры обслуживания прерываний, как правило, работают со стеками процессора;

• разрешенные исключительные ситуации всегда инициируются немедленно, независимо от значения IPL процессора, в то время как прерывания откладываются до тех пор, пока значение IPL процессора не станет ниже значения IPL запрашивающего прерывания;

• большинство исключительных ситуаций не может быть запрещено. Однако если ситуация запрещена, но все же возникает, ее инициализация не происходит даже после того, когда запрет отменяется. Это относится, например, к переполнению, которое всего лишь сигнализирует о нарушении посредством установки одного из разрядов кодов условий (V). Если же состояние прерывания возникает в тот момент, когда оно запрещено, или процессор имеет то же самое или более высокое значение IPL, то прерывание в конечном итоге будет удовлетворено, как только возникнет подходящая ситуация, если условие прерывания все еще существует;

• поле предыдущего режима в PSL по прерыванию всегда устанавливается в режим ядра, а по исключительной ситуации устанавливается в режим, существовавший на момент ее возникновения.

1.4. ОРГАНИЗАЦИЯ ПАМЯТИ

Минимально адресуемой единицей памяти в СМ1700 является байт (8 разрядов). Для адресации памяти (любого байта) используется 32 разряда, что позволяет адресовать более 4 Гбайт. Такое адресное пространство слишком большое, чтобы быть полностью представленным в реальной оперативной памяти, поэтому в архитектуру ВК СМ1700 включен диспетчер памяти, обеспечивающий (вместе с программной поддержкой МОСВП) виртуальную организацию памяти. Программисту предоставляется для программирования практически вся адресуемая виртуальная память, которая разбита на страницы в 512 слов. Часть этих страниц располагается в оперативной памяти, остальные страницы — на внешней дисковой памяти.

Перед началом исполнения образ процесса находится на диске. При инициализации часть страниц образа переносится в оперативную память, образуя так называемый рабочий набор страниц, и процесс начинает выполняться. При обращении по виртуальному адресу, соответствующая страница которого отсутствует в оперативной памяти, происходит аппаратное прерывание, вследствие чего операционная система выгружает одну из ранее используемых страниц процесса на диск, а на освободившееся место загружает страницу, к которой было произведено обращение.

Все виртуальное адресное пространство разделено на две части — область процесса и область системы (рис. 1.3).

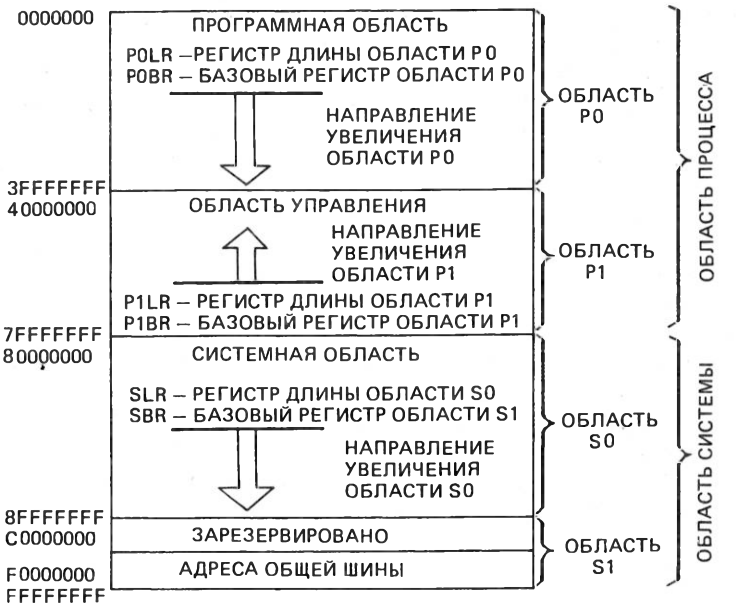


Рис. 1.3. Структура адресного пространства

Область процесса, занимающая младшую часть адресов, предназначена для всех процессов и делится в свою очередь на область программы — P0 и область управления — P1.

В области виртуальных адресов P0 располагается исполняемый образ задачи, т. е. инструкции и данные, необходимые для ее решения.

Область виртуальных адресов P1 используется для хранения управляющей информации. В частности, здесь находятся 4 стека, а также информация о распределении каналов и устройств ввода-вывода.

Пространство адресов системы состоит из области системы (S0) и резервной области (S1).

В адресном пространстве области S0 располагаются собственно операционная система МОСВП, а также блоки управления процессами (PCB), контексты и заголовки процессов, таблицы страниц и остальная информация, которая требуется операционной системе, включая управление виртуальной памятью. Область системы — это общая для всех процессов область.

Виртуальные адреса области S1 зарезервированы. В настоящий момент они не используются и, если какой-либо процесс вырабатывает виртуальный адрес, попадающий в данную область, то будет генерироваться ошибка.

Таким образом, адресное пространство процесса начинается с виртуального адреса 0 для всех процессов, а системная область имеет одно и то же значение виртуального адреса начала области для всех процессов в системе — 80 000 000 (в шестнадцатеричной системе счисления). Это означает, что все процессы получают одни и те же процедуры системного обслуживания.

Виртуальный адрес в СМ1700 состоит из двух полей (рис. 1.4): номера виртуальной страницы (разряды 9—31) и смещения в байтах внутри страницы (разряды 0—8).

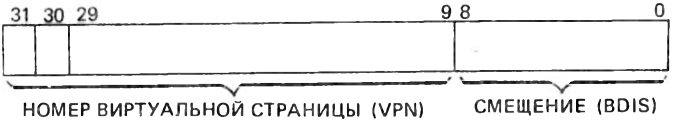


Рис. 1.4. Виртуальный адрес

Разряды 30 и 31 содержат область адресного пространства, на которую указывает виртуальный адрес.
Значения этих разрядов следующие:

00	область программы	(область P0)	пространство процесса(ов)
01	область управления	(область P1)	
10	область системы	(область S0)	пространство системы
11	резервная область	(область S0)	

Области программы (P0), управления (P1) и системы (S0) описываются с помощью соответствующей структуры данных —

таблицы страниц (PTE). Таблица страниц — это массив из 32-разрядных записей, называемых *элементами* (или *записями*) *таблицы страниц*. Каждая запись описывает одну страницу в системе. Общий формат записи таблицы страниц представлен на рис. 1.5.

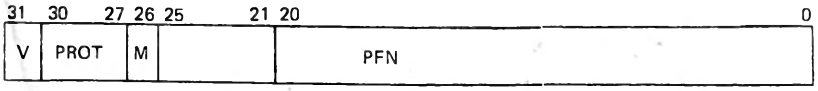


Рис. 1.5. Формат записи таблицы PTE:

разряды 0—20 — номер физической страницы (PFN), содержат информацию о размещении требуемой страницы (разряды 15—20 не используются); разряды 21—25 применяются операционной системой при отсутствии данной страницы в рабочем наборе; 26 — разряд модификации (M), показывает, производилась ли запись в данную страницу; разряды 27—30 — поле защиты (PROT), определяет необходимые привилегии для чтения или записи этой страницы; 31 — разряд достоверности (N), показывает, содержит или нет данный элемент таблицы страниц информацию, необходимую для отображения виртуального адреса в физический адрес, т. е. существует ли в оперативной памяти физический эквивалент данному виртуальному адресу

Каждая таблица страниц определяется двумя привилегированными регистрами в аппаратуре процессора СМ1700: базовым регистром адреса и регистром длины (см. рис. 1.3).

Системная таблица страниц (SPT) описывает страницы в виртуальной области системы (S0). Таблица SPT определяется с помощью базового регистра (SBR) и регистра длины системы (SLR). Регистр SBR содержит физический адрес начала таблицы SPT.

Таблицы страниц для каждого процесса размещаются в системной области адресного пространства. Базовые регистры области P0 и области P1 (P0BR, P1BR) содержат виртуальные адреса. Таблицы страниц P0 и P1 связаны с конкретным процессом, поэтому базовые регистры и регистры длины (P0LR, P1LR) изменяются с помощью процедуры смены (переключения) контекста, в отличие от системной таблицы страниц, которая разделяется всеми процессами, находящимися в системе.

Регистры длины SLR и P0LR указывают длину своих таблиц (в длинных словах), а регистр P1LR — величину неиспользуемой области таблицы страниц P1. Это решение связано с реализацией области управления процессом, где страницы выделяются в обратном направлении «сверху вниз».

В связи с тем что таблицы страниц областей P0 и P1 базируются относительно системной таблицы страниц, то они могут располагаться в физических страницах, не образующих непрерывной области.

Регистры длины служат для определения допустимости виртуального адреса, генерируемого некоторым исполняемым образом. Если образ вырабатывает виртуальный адрес, который находится вне диапазона разрешенных адресов, то адрес определяется как недопустимый. Для такого адреса не выполняется трансляция адреса в связи с тем, что ему нет соответствующего физического эквивалента, отображаемого с помощью таблицы страниц. Вследствие этого перед каждой процедурой трансляции адреса процессор использует соответствующий регистр длины для контроля на допустимость данного виртуального адреса, и если виртуальный номер страницы VPN выходит за диапазон адресов, указанный регистром длины, то генерируется ошибка.

Таким образом, при преобразовании системного адреса происходит одно обращение к памяти, а при преобразовании адреса пользователя — два. Причем эти обращения к памяти целиком относятся к накладным расходам и полезных функций не выполняют. Для уменьшения потерь времени, связанных с преобразованием адресов при часто повторяющихся обращениях к одним и тем же страницам, в архитектуре СМ1700 имеется буфер трансляции. Он состоит из 128 регистров, которые поровну поделены на две части: пользовательскую и системную. Пользовательская часть буфера трансляции очищается при смене процессов.

Преобразование адреса с помощью буфера трансляции происходит следующим образом. Старший разряд виртуального адреса определяет, какая часть буфера выбрана: пользовательская или системная. Далее разряды 9—14 виртуального адреса интерпретируются как номер регистра в буфере трансляции. После выбора регистра разряды 15—30 виртуального адреса сравниваются с соответствующими разрядами регистра буфера трансляции. При полном совмещении этих разрядов из регистра выбирается номер физической страницы, к которому добавляются разряды 0—8 виртуального адреса. После того как виртуальный адрес один раз будет преобразован по полному алгоритму и номер физической страницы вместе с разрядами 15—30 виртуального адреса записан в регистр буфера трансляции, дальнейшее преобразование адресов происходит с помощью аппаратуры буфера трансляции без дополнительных обращений к памяти.

1.5. СТЕКИ

Стек представляет собой непрерывный участок памяти, который используется для временного хранения данных (при обработке прерываний, вызове подпрограмм и т. п.). Занесение

и извлечение данных из стека организуется с помощью специального указателя на вершину стека в соответствии с правилом LIFO (последний пришел — первый вышел). Указатель стека всегда указывает на последний элемент, занесенный в стек.

Поскольку в любой момент времени процессор может работать либо с процессом в одном из четырех режимов (ядро, управление, супервизор, пользователь), либо обслуживать функционирование системы, то существуют пять стеков для каждого из отмеченных состояний и соответственно пять указателей. При переходе процессора из одного состояния в другое указатель стека текущего режима запоминается в стеке контекста и загружается информацией для нового состояния.

Введение отдельных стеков позволяет выполнять инструкции в более привилегированном режиме, сохраняя информацию в своем стеке без необходимости ее защиты от влияния другого уровня. При обслуживании событий, которые происходят асинхронно по отношению к выполнению процесса, процессор автоматически переключается на стек прерываний. В основном это касается прерываний от внешних устройств.

Стеки, используемые в режимах пользователя, супервизора и управления, могут быть нерезидентными. Стек ядра для текущего процесса и стек прерываний независимы от процесса и всегда резидентны.

В стек могут заноситься элементы различной длины. При этом, за исключением инструкции CALL, не делается попытки выравнивать стек. Однако рекомендуется при работе со стеками выравнивать их на границу длинного слова и размещать элементы в длинные слова (для этого используются специальные инструкции преобразования).

1.6. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

Организация ввода-вывода в ЭВМ СМ1700 в основном сохраняет преемственность с младшими 16-разрядными моделями СМ ЭВМ, т. е. внешние устройства подключаются к интерфейсу «общая шина», и обращение к регистрам этих устройств производится, как к обычным ячейкам памяти.

Область адресов общей шины отображается в физической оперативной памяти ЭВМ СМ1700 в диапазон адресов с FC000000 по FFFFFFFF. Старшие 8 Кбайт области адресов шины отводятся для регистров внешних устройств.

Векторы прерываний, так же как векторы обработки исключительных ситуаций, располагаются в блоке управления системой (SCB), доступ к которому осуществляется через регистр базы этого блока. Процессор, получив сигнал прерывания, использует регистр базы и значение вектора от устройства, вы-

давшего прерывание, для вычисления местоположения вектора прерывания. Вектор прерывания занимает длинное слово и содержит виртуальный адрес процедуры обслуживания. В момент прерывания текущее состояние регистра PC и длинного слова состояния процессора PSL сохраняется в стеке прерываний.

Обращение к внешним устройствам в ЭВМ СМ1700 осуществляется через адаптер общей шины, который в данном случае преобразует 24-разрядный физический адрес в 18-разрядный адрес общей шины. Перед тем как задать функцию ввода-вывода для внешнего устройства, имеющего прямой доступ к памяти, необходимо предварительно настроить определенным образом регистры адаптера общей шины. Это связано с тем, что внешние устройства с прямым доступом к памяти выдают на интерфейс «общая шина» 18-разрядные адреса в непрерывном диапазоне, а организация памяти в ЭВМ СМ1700 позволяет разбивать области ввода-вывода на страницы с произвольными физическими адресами. Адаптер общей шины (ОШ) представляет старшие 9 разрядов 18-разрядного адреса, выдаваемого устройством, как номер одного из 496 регистров преобразования адреса (рис. 1.6). Далее адаптер общей шины

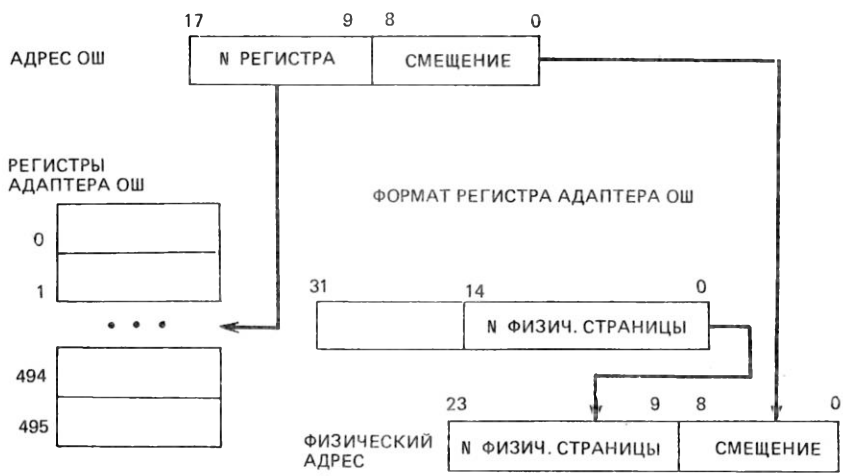


Рис. 1.6. Схема преобразования адреса общей шины

извлекает из выбранного регистра номер физической страницы области ввода-вывода и формирует из номера страницы и младших 9 разрядов адреса, выдаваемого устройством, физический 24-разрядный адрес ячейки оперативной памяти. Индикатором достоверности номера физической страницы служит старший разряд регистра адаптера общей шины.

Регистры преобразования адреса адаптера общей шины разделяются между отдельными внешними устройствами с прямым доступом к памяти. Каждому устройству в зависимости от объема операции ввода-вывода выделяется необходимое число указанных регистров для преобразования адресов на общей шине в адреса внутренней шины. Настройка регистров адаптера для определенного процесса, по существу, сводится к преобразованию виртуальных адресов страниц ввода-вывода процесса в физические адреса и записи полученных номеров физических страниц в выделенные регистры адаптера общей шины. Перед тем как внешнее устройство инициализировать для выполнения операции обмена данными, в регистре адреса внешнего устройства следует указать номер первого выделенного регистра адаптера и смещение в странице до области ввода-вывода. Во время обмена данными страницы области ввода-вывода должны быть заблокированы в памяти, чтобы обмен данными произошел в области ввода-вывода процесса, выдавшего запрос.

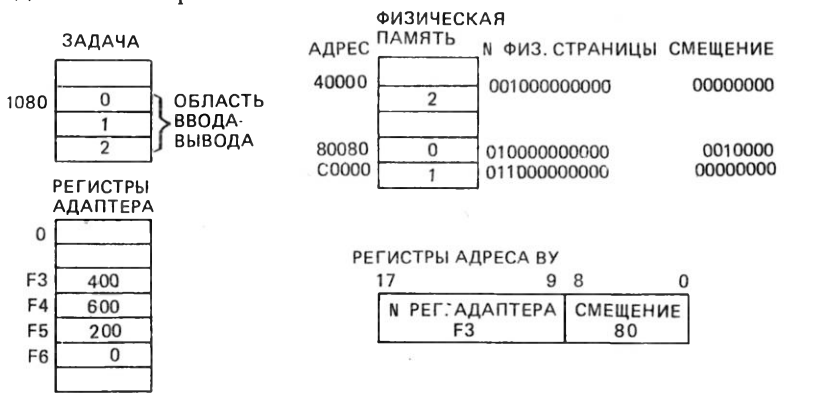


Рис. 1.7. Преобразование виртуального адреса в адрес общей шины

Из примера, представленного на рис. 1.7, видно, что страницы области ввода-вывода произвольно разбросаны в оперативной памяти. В соответствии с их положением формируется содержимое выделенных регистров адаптера общей шины, а в регистр адреса внешнего устройства записывается номер первого из выделенных регистров адаптера и смещение в странице.

В целях защиты памяти от устройства прямого доступа число выделяемых регистров адаптера на единицу больше, чем число страниц области ввода-вывода. В последний выделенный регистр записывается нулевое значение, чтобы в случае превышения запланированного объема ввода-вывода произошло прерывание и ввод-вывод завершился аварийно.

Глава 2

ПРЕДСТАВЛЕНИЕ ДАННЫХ И РЕЖИМЫ АДРЕСАЦИИ

2.1. ФОРМАТЫ И ТИПЫ ДАННЫХ

Адресация данных в СМ1700 осуществляется с точностью до байта, который является основной единицей информации. Кроме того, система инструкций СМ1700 обеспечивает работу с данными, образованными из нескольких байт или разрядов (бит). В качестве единицы информации (рис. 2.1) могут вы-

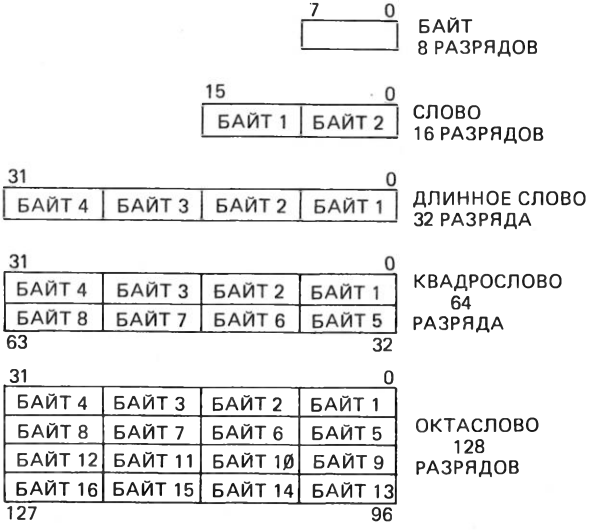


Рис. 2.1. Единицы информации СМ1700

ступать также слово (2 байта), длинное слово (4 байта), квадрослово (8 байт), октаслово (16 байт). Каждая из перечисленных единиц информации сформирована из последовательно расположенных байт и всегда адресуется так же, как младший байт в этой группе. Разряды нумеруются от младшего разряда 0, расположенного справа, до старшего разряда, расположенного слева, т. е. до 7, 15, 31, 63 и 127-го разряда соответственно для байта, слова, длинного слова, квадрослова, октаслова.

В зависимости от типа операции или соответствующего указания Макроассемблер интерпретирует различные форматы данных как арифметические или символьные типы данных, битовые поля переменной длины или очереди (рис. 2.2).

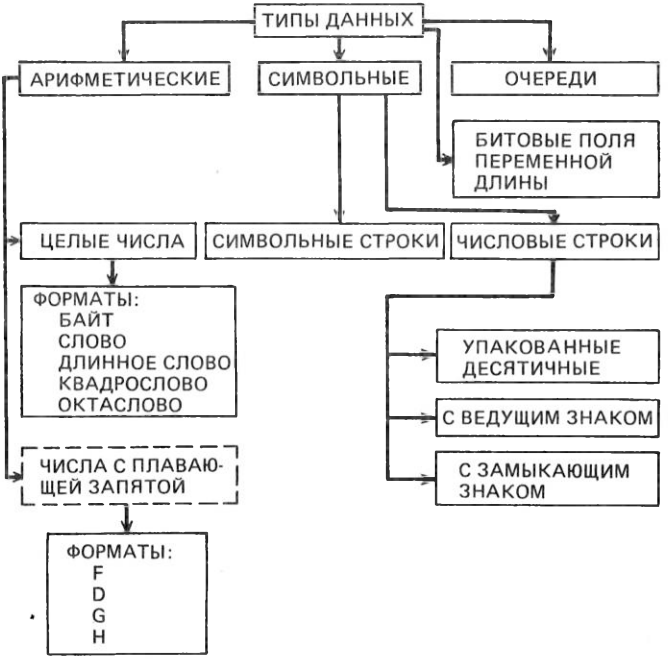


Рис. 2.2. Типы данных

Наличие нескольких типов данных облегчает разработку программ и позволяет писать их более компактно, так как для хранения одного и того же значения может быть использовано несколько вариантов. При этом в зависимости от выбранного способа представления данных транслятор выделит соответствующий объем памяти. С целью повышения гибкости и уменьшения сложности программирования в систему инструкций СМ1700 включена группа инструкций для преобразования одних типов данных в другие.

2.1.1. АРИФМЕТИЧЕСКИЕ ДАННЫЕ

Целые числа. Этот тип данных используется для хранения положительных и отрицательных целых чисел. Максимальное значение целого числа зависит от используемого формата (табл. 2.1).

Таблица 2.1

ДИАПАЗОН ПРЕДСТАВЛЕНИЯ ЦЕЛЫХ ЧИСЕЛ

Формат числа	Количество разрядов	Диапазон числа	
		со знаком	без знака
Байт	8	от -128 до $+127$	от 0 до 255
Слово	16	от -32768 до 32767	от 0 до 65535
Длинное слово	32	от -2^{31} до $+2^{31}-1$	от 0 до $2^{32}-1$
Квадрослово	64	от -2^{63} до $+2^{63}-1$	от 0 до $2^{64}-1$
Октаслово	128	от -2^{127} до $+2^{127}-1$	от 0 до $2^{128}-1$

Отрицательные числа представляются в СМ1700 в дополнительном коде, который образуется инвертированием числовых разрядов и добавлением 1 к младшему разряду.

Знак числа всегда содержится в последнем разряде (7— для байта, 25 — для слова и т. д.). Если значение знакового разряда 0, число положительное, если 1, — число отрицательное.

Макроассемблер имеет полный набор инструкций для сложения, вычитания, умножения, деления, формирования дополнительного кода, сдвига целых чисел. Однако следует отметить, что квадрослова и октаслова лишь частично используются для представления целых чисел и поддерживаются ограниченным числом инструкций.

Числа с плавающей запятой. Целые числа удобны для представления данных и решения задач во многих областях, однако их диапазона бывает недостаточно для решения сложных научно-технических задач, где необходимы числа с большими значениями. В этом случае используют числа с плавающей запятой. Диапазон и точность представления чисел с плавающей запятой зависят от применяемых форматов: F, D, G, H (табл. 2.2.).

Таблица 2.2

ДИАПАЗОН ПРЕДСТАВЛЕНИЯ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Формат числа с плавающей запятой	Диапазон представления чисел	Точность (количество десятичных цифр)
F	от $0.29*10^{-38}$ до $1.7*10^{38}$	7
D	от $0.29*10^{-38}$ до $1.7*10^{38}$	16
G	от $0.56*10^{-308}$ до $0.9*10^{308}$	15
H	от $0.84*10^{-4932}$ до $0.59*10^{4932}$	33

Число с плавающей запятой формата F. Оно состоит из четырех последовательно расположенных байтов с адресом младшего байта A и имеет нумерацию разрядов справа налево от 0-го до 31-го.

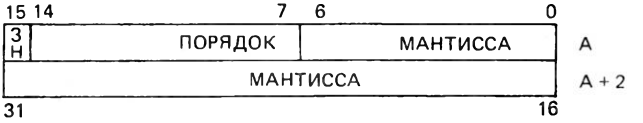


Рис. 2.3. Представление числа с плавающей запятой формата F

Формат, представленный на рис. 2.3, отображает число с 15-м разрядом в качестве знакового, порядком в разрядах с 14-го по 7-й и нормализованной 24-разрядной мантиссой в разрядах с 6-го по 0-й и с 31-го по 16-й. В мантиссе значение разрядов возрастает от 16-го до 31-го и от 0-го до 6-го. Восьмиразрядное поле порядка может иметь значение от 0 до 255. Число, равное нулю, отображается нулями в поле порядка и в знаковом разряде. Значение от 1 до 255 в поле порядка отображает двоичные порядки от —127 до +127.

Число с плавающей запятой формата D. Оно состоит из восьми последовательно расположенных байтов с адресом младшего байта A и имеет нумерацию разрядов справа налево от 0-го до 63-го.

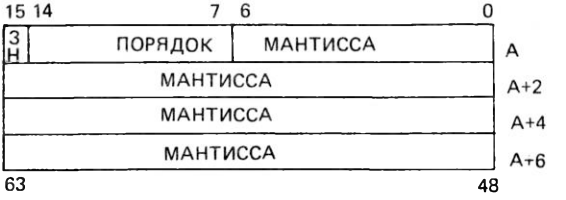


Рис. 2.4. Представление числа с плавающей запятой формата D

Формат, представленный на рис. 2.4, аналогичен предыдущему, за исключением дополнительных наименее значащих 32 разрядов мантиссы. В мантиссе разряды увеличивают свое значение от 48-го до 63-го, от 32-го до 47-го, от 16-го до 31-го

и от 0-го до 6-го разряда. Представление порядка то же самое, что и для числа с плавающей запятой формата F.

Число с плавающей запятой формата G. Оно состоит из восьми последовательно расположенных байтов с адресом младшего байта A и имеет нумерацию разрядов справа налево от 0-го до 63-го (рис. 2.5).

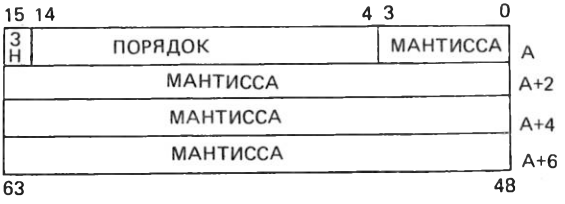


Рис. 2.5. Представление числа с плавающей запятой формата G

В этом формате представляется число со знаком в 15-м разряде, порядком — в разрядах с 14-го по 4-й и нормализованной 53-разрядной мантиссой в разрядах с 3-го по 0-й и с 63-го по 16-й. Старший разряд мантиссы не отображается. Внутри мантиссы значения разрядов возрастают от 48-го до 63-го, от 32-го до 47-го, от 16-го до 31-го и от 0-го по 3-й. 11-разрядное поле порядка может содержать значения от 0 до 2047. Нулевое значение числа формата G отображается нулями в поле порядка и нулем в знаковом разряде. Значения от 1 до 2047 в поле порядка соответствуют допустимым значениям двоичного порядка от —1023 до +1023.

В отличие от формата D, в котором используются также 64 разряда, диапазон чисел существенно увеличен за счет незначительного уменьшения точности (см. табл. 2.2).

Число с плавающей запятой формата H. Оно состоит из шестнадцати последовательно расположенных байтов с адресом младшего байта A и имеет нумерацию разрядов справа налево от 0-го до 127-го (рис. 2.6).

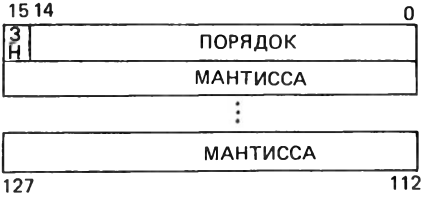


Рис. 2.6. Представление числа с плавающей запятой формата H

В этом формате представляется число со знаком в 15-м разряде, порядком — в разрядах с 14-го по 0-й и нормализованной 113-разрядной мантиссой — в разрядах от 16-го до 127-го. Старший разряд мантиссы не отображается. Внутри мантиссы значения разрядов возрастают от

112-го до 127-го, от 96-го до 111-го, от 80-го до 95-го, от 64-го до 79-го, от 48-го до 63-го, от 32-го до 47-го, от 16-го до 31-го. 15-разрядное поле порядка может содержать значения от 0 до 32767. Нулевое значение числа формата Н отображается нулевым значением поля порядка и нулем в знаковом разряде. Значения от 1 до 32767 соответствуют допустимым значениям двоичного порядка от —16383 до +16383.

2.1.2. СИМВОЛЬНЫЕ ДАННЫЕ

Символьная строка. Она представляет собой непрерывную последовательность байтов в памяти, содержимое которых интерпретируется в коде КОИ-8 (приложение 1).

Символьная строка определяется двумя атрибутами: адресом первого байта строки (A) и длиной строки (L), которая равна количеству байтов, занимаемых символами этой строки (рис. 2.7). Длина строки L меняется в диапазоне от 0 до 65535.

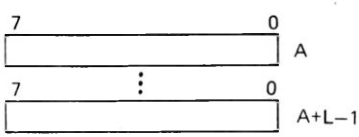


Рис. 2.7. Представление символьной строки

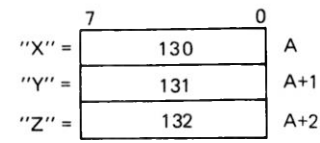


Рис. 2.8. Пример символьной строки:

130 — восьмеричный код символа «X»; 131 — восьмеричный код символа «Y»; 132 — восьмеричный код символа «Z»

Адрес строки определяет первый символ строки. Пример отображения символьной строки «XYZ» в памяти представлен на рис. 2.8.

Числовая строка. Этот тип данных используется для хранения чисел, представленных в виде последовательности десятичных цифр. Каждая числовая строка описывается двумя атрибутами: адресом первого байта памяти, отведенного для хранения строки (A), и длиной строки, выраженной в количестве символов (L). L может принимать значения в диапазоне от 0 до 31. Числовая строка с нулевой длиной имеет нулевое значение. Числовая строка в явном виде содержит знак числа «+» или «—». Существуют три типа числовых строк: чис-

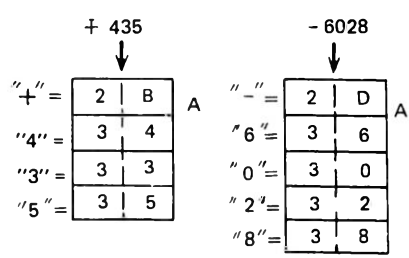


Рис. 2.9. Примеры числовых строк с ведущим знаком

ловая строка с ведущим знаком, числовая строка с замыкающим знаком и упакованная десятичная строка.

Числовая строка с ведущим знаком содержит знак числа в первом байте (рис. 2.9). Последующие байты содержат цифры в коде КОИ-8. Знак «+» имеет код 2B, а знак «—» — код 2D. Числовая строка с ведущим знаком занимает L+1 байт.

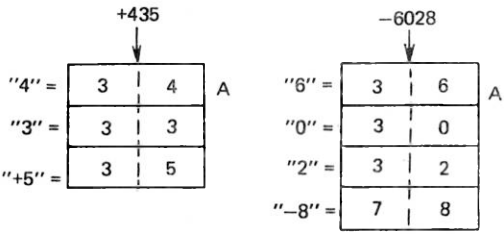


Рис. 2.10. Примеры числовых строк с замыкающим знаком

Числовая строка с замыкающим знаком содержит знак числа вместе с последней десятичной цифрой в последнем байте (рис. 2.10). Эта последняя цифра со знаком может быть закодирована в одном из двух форматов: зонном или перфоформате (табл. 2.3).

Таблица 2.3

ПРЕДСТАВЛЕНИЕ МЛАДШЕЙ ЗНАЧАЩЕЙ ЦИФРЫ И ЗНАКА В ЧИСЛОВОЙ СТРОКЕ С ЗАМЫКАЮЩИМ ЗНАКОМ

	Зонный формат		Перфоформат	
цифра	10СС	16СС	10СС	16СС
0	48	30	123	7B
1	49	31	65	41
2	50	32	66	42
3	51	33	67	43
4	52	34	68	44
5	53	35	69	45
6	54	36	70	46
7	55	37	71	47
8	56	38	72	48
9	57	39	73	49
—0	112	70	125	7D
—1	113	71	74	4A
—2	114	72	75	4B
—3	115	73	76	4C
—4	116	74	77	4D
—5	117	75	78	4E
—6	118	76	79	4F
—7	119	77	80	50
—8	120	78	81	51
—9	121	79	82	52

Примечание. 10СС — десятичная система счисления; 16СС — шестнадцатеричная система счисления.

Упакованная десятичная строка включает в себя вдвое больше чисел, что позволяет хранить их в меньшем объеме памяти по сравнению с другими числовыми строками. Основной единицей памяти для упакованных десятичных строк является полубайт длиной в 4 разряда (рис. 2.11). Полубайт может содер-

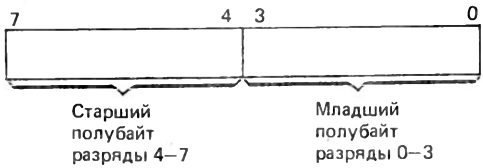


Рис. 2.11. Структура байта

жать в двоичном виде любое десятичное число от 0 до 15. Это означает, что в одном полубайте может быть представлена одна шестнадцатеричная цифра (от 0 до F). В упакованной десятичной строке каждый полубайт содержит одну десятичную цифру, за исключением младшего полубайта последнего байта, который содержит знак числа. Знак «+» кодируется шестнадцатеричной цифрой C, знак «—» — шестнадцатеричной цифрой D (рис. 2.12).

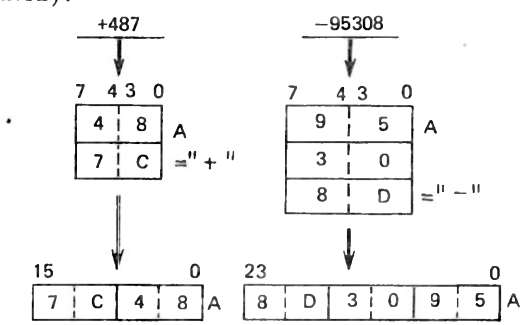


Рис. 2.12. Примеры упакованных десятичных строк

Таким образом, каждый полубайт упакованной десятичной строки должен содержать либо цифру либо знак числа. Для строки, состоящей из нечетного числа цифр, все просто. В случае, если строка содержит четное число цифр, старший полубайт первого байта должен содержать нуль (рис. 2.13). При трансляции этот нуль добавляется автоматически.

Следовательно, для хранения L десятичных цифр в формате упакованной десятичной строки потребуется L/2+1 байт памяти (L/2 округляется после деления).

Числовая строка в зависимости от использования может быть представлена в требуемом формате. Так, числовые строки с ведущим и замыкающим знаками используются в основном при вводе и выводе. Для этих типов числовых строк предусмотрены только операции преобразования в упакованный десятичный формат и обратно. Числовые строки с ведущим знаком предназначены для операций ввода-вывода с терминалов, когда знак числа печатается перед цифрами. Числовые строки с замыкающим знаком предназначены для операций ввода-вывода с перфокарт. Перфоратор пробивает соответствующую знаку позицию на перфокарте после последней цифры числа.

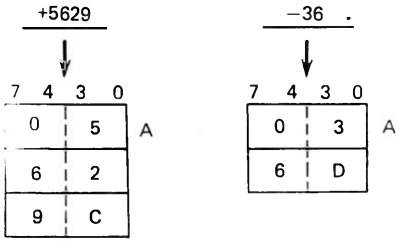


Рис. 2.13. Пример упакованной десятичной строки с четным количеством цифр

Упакованная десятичная строка часто используется компиляторами с таких «коммерческих» языков, как Кобол и ПЛ/1. При этом система инструкций СМ1700 обеспечивает арифметические операции с упакованными десятичными строками.

2.1.3. БИТОВОЕ ПОЛЕ ПЕРЕМЕННОЙ ДЛИНЫ

Этот тип данных обладает двумя особенностями:

- в отличие от других типов данных, имеющих размер в байтах, битовое поле переменной длины позволяет программисту определять любой размер данных от 0 до 32 разрядов;
- если данные всех остальных типов должны начинаться в памяти на границе байта, то битовые поля переменной длины могут начинаться с любого разряда памяти.

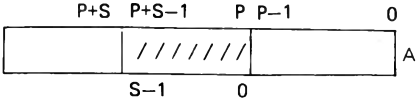


Рис. 2.14. Битовое поле переменной длины (на рисунке оно заштриховано)

Битовое поле переменной длины (рис. 2.14) определяется тремя атрибутами:

1) A—базовый адрес битового поля;

2) P — смещение между адресом A и адресом первого разряда битового поля. Смещение P может иметь отрицательное значение. Значение смещения должно быть в диапазоне от -2^{31} до $2^{31}-1$;

3) S — длина битового поля, выраженная в количестве занимаемых разрядов. Длина S может иметь значение в диапазоне от 0 до 32.

В соответствующих инструкциях, которые позволяют искать, проверять и модифицировать некоторое число разрядов как отдельное битовое поле, все три атрибута рассматриваются как операнды. Смещение хранится в длинном слове, длина — в байте. Базовый адрес может иметь косвенную адресацию. Как правило, этот тип данных используется для хранения и проверки информации о состоянии указателей («флагов»).

2.1.4. ОЧЕРЕДЬ

Очереди представляют собой тип данных, широко используемый для управления динамическими структурами в операционных системах, а также в задачах моделирования. Кроме того, очереди достаточно часто употребляются в современных языках высокого уровня, ориентированных на решение задач искусственного интеллекта.

В СМ1700 очередь представляет собой циклически двунаправленный список. Элемент очереди определяется адресом. Каждый элемент очереди связан с соседними элементами с помощью двух длинных слов. Первое из них является прямой связью, оно определяет адрес, по которому расположен последующий элемент. Второе длинное слово является обратной связью, оно задает адрес, по которому расположен предыдущий элемент.

Очередь определяется заголовком, который структурно идентичен паре длинных слов связи. Прямая связь заголовка указывает адрес элемента, который называется *началом очереди*, а обратная связь заголовка — адрес элемента, который называется *концом очереди*. Прямая связь конца очереди указывает на заголовок.

Очереди различаются по типу связи: *абсолютная* и *относительная*.

В абсолютных очередях в качестве связи используются абсолютные адреса. Первое длинное слово (с младшим адресом) представляет собой прямую связь: адрес последующего элемента очереди. Второе длинное слово (со старшим адресом) представляет собой обратную связь: адрес предыдущего элемента очереди.

В относительных очередях в качестве связи используются смещения элементов относительно друг друга. Первое длинное слово (с младшим адресом) является прямой связью — смещением последующего элемента относительно текущего. Второе длинное слово (со старшим адресом) является обратной связью — смещением предыдущего элемента очереди относительно текущего.

В Макроассемблере нет специальных директив, с помощью которых резервируются два длинных слова для заголовка очереди (как минимум). Это делает программист. Кроме того, он должен сам резервировать память для элементов очереди, которые помимо двух обязательных длинных слов для связи с другими элементами очереди могут содержать сколь угодно много другой информации об этом элементе. Однако соответствующие инструкции для работы с очередями СМ1700 обеспечивают необходимые операции с очередями, освобождая программиста от написания не сложных, но рутинных последовательностей традиционных инструкций. (В гл. 3 при описании инструкций для работы с очередями приведены поясняющие примеры.)

2.2. РЕЖИМЫ АДРЕСАЦИИ

При написании программы на Макроассемблере для обозначения операции и адресов операндов в инструкции используются специальные соглашения, мнемоника которых однозначно определяет имеющиеся в СМ1700 коды машинных операций и режимы адресации. Транслятор с Макроассемблера обрабатывает эти символические обозначения и генерирует соответствующий двоичный код.

В ранних моделях СМ ЭВМ каждая инструкция транслировалась в последовательность слов. Непременным условием корректности представления инструкции в памяти являлось то, что она всегда должна была начинаться с границы слова. В СМ1700 сняты ограничения на «словную» структуру машинной инструкции. В результате любая инструкция может начинаться с границы байта и необязательно должна иметь длину, кратную слову. Количество байтов, требуемых для размещения инструкции в памяти, определяется в основном теми режимами адресации, которые используются для получения адреса операнда.

Для лучшего понимания специфики отдельных режимов адресации рассмотрим формат инструкций СМ1700.

Формат подавляющего большинства инструкций СМ1700 включает поле кода операции (КОП) и несколько полей операндов, количество которых определяется соответствующим КОП. Для некоторых операций операнды могут отсутствовать. В этом случае инструкция состоит только из поля кода операции.

Поле КОП имеет самый младший адрес в инструкции и, как правило, занимает один байт, хотя есть отдельные инструкции, у которых это поле занимает два байта. Затем располагаются поля операндов, которых может быть от 1 до 6. Если в инструкции операнды отсутствуют, машинная инструкция состоит толь-

ко из поля КОП. Поле операнда может размещаться в одном или нескольких байтах, что зависит от режима адресации этого операнда (рис. 2.15).

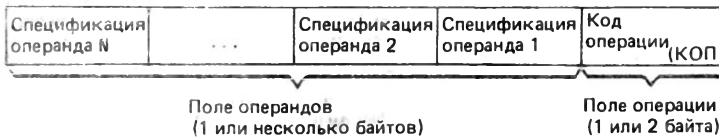


Рис. 2.15. Формат машинной инструкции

Спецификация операнда (т. е. номер используемого регистра и режим адресации) осуществляется достаточно большим числом вариантов, что соответствует разнообразию режимов адресации СМ1700. Эти режимы являются развитием режимов адресации 16-разрядных СМ ЭВМ и могут быть разделены на четыре группы:

- адресация через регистры общего назначения;
- адресация через счетчик инструкций (PC);
- адресация с индексацией;
- адресация в инструкциях относительного перехода.

Принадлежность к группе определяется номером заданного регистра, а также тем, как этот регистр используется для определения адреса операнда (режим адресации). Это нашло свое отражение и в структуре поля операнда.

Описывая режимы адресации (приложение 2) каждой группы, более подробно рассмотрим формат поля операнда. При этом будем использовать следующие обозначения регистров:
RN — регистр общего назначения;
RX — регистр общего назначения, применяемый как индексный.

В качестве RN и RX употребляются регистры общего назначения (R0 — R14), а также символические обозначения регистров R12, R13 и R14 — AP, FP,SP.

Прежде чем приступить к детальному обсуждению режимов адресации, рассмотрим возможные формы определения операндов, так как в некоторых режимах адресации операнд или его часть может быть выражением. Простейшие формы определения операндов — названия регистров, литералы, символические имена (например, переменных или областей памяти) и т. п. — не вызывают трудностей при их использовании в инструкции. Сложной конструкцией операнда является операнд-выражение. Более детально выражения рассматриваются в гл. 4. Здесь отметим только, что оценка выражений, значения которых используются для выражения адреса операнда, выполняется в процессе трансляции. При этом все арифметические и другие

операции, допустимые в выражениях, не приводят к изменению соответствующих областей памяти. Модификация памяти может быть осуществлена только на этапе выполнения программы.

2.2.1. РЕЖИМЫ АДРЕСАЦИИ ЧЕРЕЗ РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ

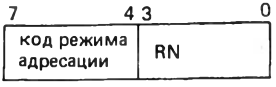
В этих режимах для указания адреса операнда используются 15 регистров общего назначения (R0 — R14) и восемь режимов адресации через эти регистры (табл. 2.4).

Таблица 2.4

РЕЖИМЫ АДРЕСАЦИИ ЧЕРЕЗ РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ

Режим	Код режима	Формат
Регистровая адресация	5	RN
Косвенно-регистровая адресация	6	(RN)
Адресация с автоувеличением	8	(RN)+
Косвенная адресация с автоувеличением	9	@(RN)+
Адресация с автоуменьшением	7	–(RN)
Адресация по смещению	10, 12, 14	смщ(RN)
Косвенная адресация по смещению	11, 13, 15	@ смщ(RN)
Литеральная адресация	0—3	S^литерал или #литерал

Формат поля операнда, в котором используются эти режимы, имеет несколько модификаций. Для первых пяти режимов формат поля общий, его длина 1 байт:



Код режима адресации приведен в табл. 2.4. Одного байта вполне достаточно для хранения номера регистра (от 0 до 15) и кода режима адресации (всего их 16). Другой информации для данных режимов адресации не требуется.

Для режимов со смещением, кроме номера регистра и кода режима, необходимо хранить дополнительную информацию, которая определяет смещение, в связи с чем для этих режимов поле операнда имеет длину более 1 байта.

Поле операнда для режима литеральной адресации, хотя и имеет длину 1 байт, однако в нем в явном виде отсутствует указание регистра. Формат поля операнда для этого режима также будет более подробно обсуждаться ниже.

Рассмотрим более детально каждый из режимов адресации через регистры общего назначения.

Регистровая адресация. В этом режиме операндом является содержимое регистра, к которому происходит обращение. Если операнд имеет длину, не превышающую размер регистра, то его получение из регистра не представляет сложности. Для отдельных единиц информации, имеющих длину более 32 разрядов, операнд формируется по-иному:

- в случае обработки данных в формате квадрослов, чисел с плавающей запятой форматов D и G, битовых полей переменной длины операндом является конкатенация содержимого регистра N и регистра N+1;
- в случае обработки данных в формате октаслов или чисел с плавающей запятой формата H операндом является конкатенация содержимого регистра N и регистров N+1, N+2, N+3.

Пример.

```
        CMPB    R0, R1    ; Сравнить младшие байты регистров R0 и R1.
        BEQL    LABEL     ; Если они совпадают, то перейти на метку
                          ; LABEL.
        CLRB    R0        ; Очистить младший байт регистра R0.
        INCB    R1        ; Добавить единицу к младшему байту регистра R1.
LABEL:  CLRQ    R2        ; Очистить регистры R2 и R3.
```

Косвенно-регистровая адресация. В этом режиме операндом является содержимое поля, адрес которого содержится в используемом регистре.

Пример.

```
        MOVAL   DATA1, R1   ; Получить адрес данного DATA1 в регистре R1.
        MOVAL   DATA2, R2   ; Получить адрес данного DATA2 в регистре R1.
        CMPL    (R1), (R2)  ; Сравнить значения, содержащиеся в полях DATA1 и DATA2.
        BEQL    LAB         ; Если равны, перейти на метку LAB.
        INCL    (R1)        ; Если нет, увеличить на 1 содержимое
                            ; поля DATA1.
LAB:    MOVL    (SP), R0    ; Переслать верхний элемент стека в регистр R0.
```

Адресация с автоувеличением. В этом режиме используемый регистр содержит адрес операнда. После выборки операнда производится увеличение содержимого регистра на 1, 2, 4, 8 или 16 соответственно для операндов в формате байта, слова, длинного слова, квадрослова, октаслова.

Режим с автоувеличением эффективен при работе с массивами данных одного формата, так как позволяет без дополнительных модификаций каждый раз иметь следующий элемент массива. Он может быть также использован с индексацией, однако индексный регистр не может быть тем же регистром, который указан в адресации с автоувеличением.

Пример.

```
MOVAL   FILE1, R0    ; Переслать адрес поля FILE1 в регистр
                     ; R0.
INCB    (R0)+        ; Добавить 1 к первому байту поля FILE1.
CLRB    (R0)+        ; Очистить второй байт поля FILE1.
MOVAL   FILE2, R1    ; Переслать адрес поля FILE2 в регистр
                     ; R1.
CLRQ    (R1)+        ; Очистить первое и второе длинное слово
                     ; поля FILE2.
INCL    (R1)+        ; Добавить 1 к третьему длинному слову
                     ; поля FILE2.
```

Косвенная адресация с автоувеличением. Используемый в данном режиме регистр содержит адрес поля, содержимое которого является адресом операнда. После выборки адреса операнда содержимое указанного регистра увеличивается на 4. При этом увеличение на 4 происходит вне зависимости от формата используемых в операции данных. В предыдущем режиме формат определял значение, на которое увеличивалось содержимое регистра.

Пример.

```
MOVAL   TABLE, R1    ; Получить адрес таблицы адресов в ре-
                     ; гистре R1.
INCL    @ (R1)+      ; Добавить 1 к длинному слову, адрес ко-
                     ; торого является первым в таблице ад-
                     ; ресов. Содержимое регистра R1 увели-
                     ; чивается на 4.
CLRB    @ (R1)+      ; Очистить байт, адрес которого является
                     ; вторым в таблице адресов, затем к содер-
                     ; жимому регистра R1 добавляется число
                     ; 4.
MOVL    R0, @ (R1)+  ; Переслать содержимое регистра R0 по
                     ; адресу, который является третьим в таб-
                     ; лице адресов, затем к содержимому ре-
                     ; гистра R1 добавляется число 4.
```

Адресация с автоуменьшением. В этом режиме перед выполнением операции содержимое используемого регистра уменьшается на размер, определяемый форматом данных операнда, после чего регистр содержит адрес операнда. Содержимое регистра уменьшается на 1, 2, 4, 8 и 16 соответственно для операндов в формате байта, слова, длинного слова, квадрослова и октаслова.

Пример.

```
INCB    — (R1) ; Из содержимого регистра R1 вычитается 1, после
               ; чего добавляется 1 к байту, адрес которого оп-
               ; ределяется новым содержимым регистра R1.
CLRL    — (R2) ; Вычесть 4 из содержимого регистра R2 и только
               ; после этого очистить длинное слово, адрес кото-
               ; рого содержится в этом регистре.
```

Адресация по смещению. В этом режиме адрес операнда определяется суммой содержимого используемого регистра и смещения. Смещение может быть выражением, значение которого хранится в памяти непосредственно в поле операнда. Оно может храниться в формате байта, слова и длинного слова со знаком (табл. 2.5). В соответствии с этим формат поля операнда может иметь три модификации:

Для указания длины смещения в Макроассемблере используется специальный указатель, который определяет количество байтов, необходимых для хранения данного смещения. Указатели приведены в табл. 2.5.

Таблица 2.5

ВОЗМОЖНЫЕ ВАРИАНТЫ СМЕЩЕНИЙ

Смещение	Код режима адресации		Обозначение указателя
	по смещению	косвенной по смещению	
Байт	10	11	B^
Слово (2 байта)	12	13	W^
Длинное слово (4 байта)	14	15	L^

Если выражение не имеет указателя длины смещения, а значение выражения известно, то транслятор с Макроассемблера сам определяет наименьшее количество байтов (1, 2 или 4), необходимых для хранения смещения. Если выражение не имеет указателя длины смещения, и значение выражения неизвестно, то для смещения по умолчанию резервируется одно слово (2 байта). Если действительное значение смещения не укладывается в отведенную память, выдается сообщение об ошибке.

Пример.

```
MOVAL   TAB—ARG, R0     ; Получить в регистре R0 адрес таб-
                        ; лицы значений аргументов.
CLRW    L^DSP(R0)       ; Очистить слово, адрес которого ра-
                        ; вен значению выражения DSP плюс
                        ; содержимое регистра R0. Смещение
                        ; хранится в длинном слове.
MOVL    RED(R0), R1     ; Переслать в регистр R1 длинное сло-
                        ; во, адрес которого равен адресу таб-
                        ; лицы TAB—ARG плюс значение сим-
                        ; волического имени RED. Смещение
                        ; хранится в слове, поскольку симво-
                        ; лическое имя RED не определено.
INCB    B^10(R0)        ; Добавить 1 к байту, адрес которого
                        ; равен адресу таблицы TAB—ARG,
                        ; увеличенному на 10, смещение хра-
                        ; нится в байте.
```

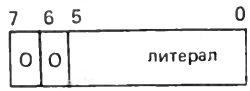
Косвенная адресация по смещению. Форматы поля операнда для этого режима совпадают с форматами предыдущего режима. Однако в отличие от предыдущего режима сумма содержимого используемого регистра и смещения определяет не операнд, а его адрес, по которому он выбирается. В соответствии с этим коды режимов адресации имеют другие значения. Так же, как и для режима адресации по смещению, возможно использование указателей длины смещения (см. табл. 2.5).

Пример.

```
MOVAL   FILE—PTR, R2    ; Получить в регистре R2 адрес поля
                        ; указателей FILE—PTR.
CLRL    @ W^MAN(R2)     ; Очистить длинное слово, адрес кото-
                        ; рого хранится по адресу FILE—PTR,
                        ; увеличенному на значение смещения
                        ; MAN. Для хранения смещения ис-
                        ; пользуется слово.
MOVL    @ B^MAP(R2), R3 ; Получить в регистре R3 длинное сло-
                        ; во, адрес которого хранится по ад-
                        ; ресу FILE—PTR, увеличенному на
                        ; значение смещения MAP. Для хра-
                        ; нения смещения используется байт.
TSTL    @ 32(R2)        ; Проверить длинное слово, адрес кото-
                        ; рого хранится по адресу FILE—PTR,
                        ; увеличенному на 32. Для хранения
                        ; смещения используется слово.
```

Литеральная адресация. В этом режиме нет адреса операнда и нет номера регистра, из которого его можно было выбрать. Значение литерала хранится в байте, в котором для предыдущих режимов адресации записывается номер используемого регистра и код режима адресации. Для этого случая формат поля операнда имеет вид:

Два старших разряда в байте должны быть нулевыми. Это означает, что для литерального режима код режима адресации может лежать в диапазоне от 0 до 3. Как только процессор распознает этот код, он интерпретирует первые шесть разрядов байта как литерал.



Литерал может быть выражением, целым числом или числом с плавающей запятой. Для хранения литерала используется только 6 разрядов, поэтому он может принимать ограниченный диапазон значений. В связи с этим целые числа должны быть в диапазоне от 0 до 63, а числа с плавающей запятой должны иметь мантиссу и порядок, занимающее не более чем по 3 разряда (табл. 2.6). При этом мантисса занимает разряды 0—2, а порядок — разряды 3—5.

Таблица 2.6

КОРОТКИЕ ЛИТЕРАЛЫ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Порядок	Мантисса							
	0	1	2	3	4	5	6	7
0	0.5	0.5625	0.625	0.6875	0.75	0.8125	0.875	0.9375
1	1.0	1.125	1.25	1.37	1.5	1.625	1.75	1.875
2	2.0	2.25	2.5	1.5	3.0	3.25	3.5	3.75
3	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5
4	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0
5	16.0	18.0	20.0	22.0	24.0	26.0	28.0	30.0
6	32.0	36.0	40.0	44.0	48.0	52.0	56.0	60.0
7	64.0	72.0	80.0	88.0	96.0	104.0	112.0	120.0

В программе на Макроассемблере признаком литерального режима является последовательная запись символов «S^», за которыми следует запись вида #литерал.

Пример.

```
S^#24      ; Целое число 24
```

Если используется формат без указателя режима литеральной адресации, т. е. символы «S^» отсутствуют, то в этом случае транслятор оценивает значение этого литерала, и если оно вписывается в диапазон чисел, которые можно использовать в литеральном режиме, то формируется инструкция, где в поле соответствующего операнда генерируется литеральный режим адресации. В противном случае генерируется непосредственный режим адресации.

Пример.

```
MOVB    #2, R0          ; Переслать 2 в младший байт регистра R0.
MOVL    S^#LITER, R1    ; Значение символического имени LITER
                        ; пересылается в регистр R1. Если оно
                        ; выходит из диапазона 0—63, то генерируется ошибка.
MOVF    #2.25, R4       ; В регистр R4 переслать число 2.25
                        ; в формате числа с плавающей запятой.
```

2.2.2. РЕЖИМЫ АДРЕСАЦИИ ЧЕРЕЗ СЧЕТЧИК ИНСТРУКЦИЙ

В этих режимах в качестве регистра общего назначения используется счетчик инструкций PC. Он всегда содержит адрес инструкции, предназначенной для выполнения. В процессе исполнения программы после выборки и интерпретации очередной инструкции счетчик увеличивается всякий раз на ее длину.

В ряде случаев значение счетчика инструкций можно использовать для вычисления адреса очередного операнда инструкции. Этот адрес определяется как текущее значение счетчика инструкций плюс (или минус) смещение, которое помещается в качестве операнда в машинную инструкцию. Величина смещения определяется во время трансляции как разность между адресом поля операнда и адресом следующей инструкции. Если адрес поля операнда больше текущего значения PC, то величина смещения положительная, если адрес поля операнда меньше адреса в PC, то величина смещения отрицательная.

Помимо смещения в инструкцию помещается информация о том, что в качестве регистра общего назначения следует использовать регистр R15 (PC), а также данные о порядке расположения и использования смещения (т. е. код режима адресации через PC). Во время выполнения инструкции величина смещения добавляется к адресу следующей инструкции и таким образом получается адрес операнда. Такая относительно сложная процедура определения адреса операнда используется, как правило, для экономии памяти. Указание абсолютного адреса требует 32 разряда. Если же взять длину смещения в одно слово (16 разрядов), то диапазон возможных адресуемых единиц памяти составит 32767 байт относительно текущего адреса. Это вполне приемлемо для написания программ большого объема и в то же время позволяет более экономно использовать память. Конкретный выигрыш памяти определяется величиной смещения.

Смещение иногда называют относительным адресом операнда, отсчитываемым от текущего значения счетчика инструкций.

Процедура получения действительного адреса операнда через смещение определяется режимом адресации.

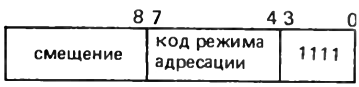
Существует пять режимов адресации через счетчик инструкций (табл. 2.7).

Таблица 2.7

РЕЖИМЫ АДРЕСАЦИИ ЧЕРЕЗ СЧЕТЧИК ИНСТРУКЦИЙ

Режим	Код режима адресации	Формат
Относительная адресация	A, C, E	адрес
Косвенная относительная адресация	B, D, F	@адрес
Абсолютная адресация	9	@#адрес
Непосредственная адресация	8	I⌒литерал или #литерал
Адресация общего вида	—	G⌒адрес

Поскольку помимо указания номера регистра (в данном случае R15) и режима адресации в поле операнда необходимо хранить смещение, формат операнда в этих случаях имеет вид:



В поле указания номера регистра (разряды 0—3) всегда будет стоять двоичное значение 1111, т. е. число 15 в десятичной системе счисления, что соответствует номеру регистра PC.

Смещение может иметь различную длину и будет определяться конкретным режимом адресации.

В зависимости от значения кода режима адресации (разряды 4—7) поле смещения будет интерпретироваться по-разному. Так, если для режимов относительной и косвенной относительной адресации оно рассматривается как относительный адрес, то для других режимов значение этого поля рассматривается по-иному.

Режим относительной адресации. В этом режиме Макроассемблер определяет относительный адрес как смещение относительно счетчика инструкций PC. Адресом может быть выражение, которому может предшествовать один из указателей длины смещения. Он определяет количество байтов, необходимых для хранения смещения. Обозначение и использование указателей длины смещения такие же, как при адресации по смещению через регистры общего назначения (см. табл. 2.5). В соответствии с указателем смещения его длина будет равна байту, слову или длинному слову. Если указатель смещения отсутствует, то длина смещения вычисляется транслятором ав-

томатически исходя из «расстояния» между текущей инструкцией и соответствующим полем операнда.

Пример.

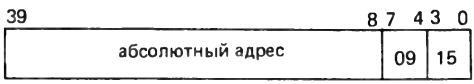
```
MOVL    W^<TAB+8>,·  ; Получить в регистре R0 длинное слово,
        R0           ; хранящееся по адресу TAB+8.
                     ; Для хранения смещения используется
                     ; слово.
MOVL    L^ POINT, R1 ; Получить в регистре R1 длинное слово,
                     ; хранящееся по адресу POINT.
                     ; Для хранения смещения используется
                     ; длинное слово.
MOVL    LABEL, R2    ; Получить в регистре R2 длинное слово,
                     ; хранящееся по адресу LABEL.
                     ; Длина смещения определяется Макроассемблером.
```

Режим косвенной относительной адресации. В отличие от предыдущего режима адрес, который формировался сложением смещения с адресом следующей инструкции, указывает не на операнд, а на поле, содержимое которого интерпретируется как адрес операнда. Другими словами, этот адрес представляет собой адрес операнда. Адресом может быть выражение, которому может предшествовать один из указателей длины смещения. Он определяет количество байтов, необходимых для хранения смещения. Указатели приведены в табл. 2.5.

Пример.

```
CLRB  @ L^ADA+8    ; Очистить байт, адрес которого хранится
                   ; по адресу ADA+8. Для хранения
                   ; смещения используется длинное слово.
TSTL  @ W^AIDA     ; Проверить длинное слово, адрес которого хранится по адресу AIDA. Для хранения смещения используется слово.
```

Режим абсолютной адресации. В данном режиме указанный в нем адрес является адресом операнда. Однако этот адрес интерпретируется транслятором как абсолютный виртуальный адрес (сравните с относительной адресацией, при которой адрес хранится в виде смещения относительно счетчика инструкций (PC). Так как транслятор при этом генерирует абсолютный адрес, то соответствующий адрес будет занимать 4 байта, а весь спецификатор операнда — 5 байт. Соответственно формат поля операнда будет иметь следующий вид:

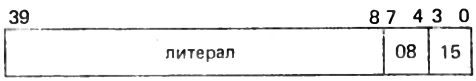


В данном случае поле смещения отводится для хранения абсолютного адреса операнда.

Пример.

```
CLRL    @#^X1000 ; Очистить содержимое длинного слова по
                 ; адресу 1000 (шестнадцатеричное).
MOVB    @#BIG, R0 ; Получить в регистре R0 байт с адресом
                  ; BIG. Адрес BIG хранится в абсолютной
                  ; форме, а не в виде смещения.
```

Режим непосредственной адресации. В этом режиме операндом является литерал, для хранения которого отводится длинное слово. Оно располагается в поле операнда непосредственно за кодом режима адресации. Для этого режима адресации формат поля операнда выглядит следующим образом:



В данном случае поле смещения используется для хранения литерала.

Пример.

```
MOVL    #1100, R0   ; Получить в регистре R0 значение 1100. Зна-
                    ; чение 1100 хранится в длинном слове.
MOVB    #DISK, R1   ; Значение символического имени DISK пере-
                    ; сылается в младший байт регистра R1.
MOVF    #0.3, R2    ; Получить в регистре R2 значение 0.3, ко-
                    ; торое хранится в формате с плавающей
                    ; запятой в четырех байтах (формат F). Оно
                    ; не может быть представлено в виде корот-
                    ; кого литерала.
CMPL    I^#4, R3    ; Сравнить 4 с содержимым регистра R3.
                    ; Число 4 хранится в длинном слове, по-
                    ; скольку операция I^указывает на непо-
                    ; средственную адресацию.
```

Режим непосредственной адресации может быть задан либо в формате I^#литерал, либо #литерал. Если используется первая форма, то транслятор отводит для хранения литерала длинное слово и генерирует код режима адресации 08. Когда используется формат #литерал, Макроассемблер выбирает, какую адресацию использовать: литеральную или непосредственную. Если литерал удовлетворяет диапазону короткого литерала, Макроассемблер использует литеральную адресацию, иначе — непосредственную.

Указателем непосредственной адресации служат последовательность двух символов I^, по которой и производится определение этого режима.

Режим адресации общего вида. Этот режим предназначается для написания позиционно-независимых программ, когда программист не знает, являются ли обрабатываемые адреса перемещаемыми или абсолютными. Если указан режим адресации общего вида, транслятор преобразует данный режим ад-

ресации либо к относительной адресации, либо к абсолютной. Если адрес является перемещаемым, транслятор преобразует адресацию общего вида в относительную. Если адрес является абсолютным, транслятор преобразует адресацию общего вида в абсолютную (в этом случае для хранения операнда требуется 5 байт).

При указании данного режима происходит его преобразование, поэтому он не имеет своего кода. В качестве адреса, так же как при относительной или абсолютной адресации, может использоваться выражение.

Пример.

```
TSTL   G^ADR; Проверяется длинное слово, помеченное меткой
              ; ADR.
              ; Если метка определена как абсолютная, то ис-
              ; пользуется абсолютная адресация, если метка оп-
              ; ределена как перемещаемая, то — относительная
              ; адресация.
```

2.2.3. РЕЖИМЫ АДРЕСАЦИИ С ИНДЕКСАЦИЕЙ

Индексирование — один из способов организации работы с массивами. При программировании любой массив обычно описывается посредством своей базы, т. е. адреса начала массива и размера каждого элемента массива. При этом доступ к любому элементу массива определяется как сумма адреса начала массива (базы) и смещения, которое получается произведением размера элемента на его индекс (если начальное значение индекса для первого элемента массива — нуль).

Для организации занесения или выборки элемента массива можно использовать любой из отмеченных выше режимов адресации, по отдельности определяя базовый адрес массива и смещения, которое задается индексом. Однако при таком подходе ухудшается наглядность программы и приходится выполнять достаточно много тривиальных операций, которые отвлекают от реализации основного алгоритма.

Режимы адресации с индексацией позволяют объединить процедуры получения базы и смещения (с последующим их сложением), что не только упрощает программирование работы с массивами, но и существенно ускоряет выполнение программы, так как при этом соответствующие операции сложения и умножения выполняются аппаратно.

Режим адресации с индексацией записывается следующим образом:

база[RX]

где база — любой режим адресации, по которому определяется базовый адрес, за исключением режимов регистровой адресации, непосредственной ад-

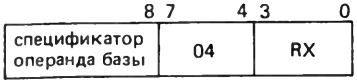
ресации, литеральной адресации, индексации и адресации в инструкциях относительного перехода;
[RX] — индексный регистр, обязательно заключаемый в квадратные скобки.

При обработке режима адресации с индексацией транслятор вначале определяет базовый адрес, расшифровывая соответствующий режим адресации базы. Затем к этому адресу добавляется произведение содержимого индексного регистра, умноженного на число байтов (размер), которое определяется из типа данных операнда. Полученная сумма является действительным адресом операнда.

Рассмотрим примеры применения режимов адресации с индексацией.

```
; Косвенно-регистровая адресация с идексацией
STEP=10
        MOVAB   SYS, R4         ; Получить в R4 адрес SYS.
        MOVL    #STEP, R1       ; Установить индексный регистр.
        CLRB    (R4) [R1]       ; Очистить байт, чей адрес равен ад-
                                ; ресу SYS плюс 10*1.
        CLRL    (R4) [R1]       ; Очистить длинное слово, адрес кото-
                                ; рого равен адресу SYS плюс 10*4
        CLRQ    (R4) [R1]       ; Очистить квадрослово, адрес которо-
                                ; го равен адресу SYS плюс 10*8.
        CLRO    (R4) [R1]       ; Очистить октаслово, адрес которого
                                ; равен адресу SYS плюс 10*16.
; Адресация с автоувеличением и индексацией
        CLRQ    (R4)+[R1]       ; Очистить квадрослово, адрес которого
                                ; равен адресу SYS плюс 10*8.
; Косвенная адресация с автоувеличением и индексацией
        MOVAB   LIST, R3        ; Получить в R3 адрес LIST.
        MOVL    #20, R2         ; Установить индексный регистр.
        CLRW    @(R3)+[R2]      ; Очистить слово, чей адрес равен 20*2
                                ; плюс адрес, хранящийся в LIST. R3
                                ; теперь содержит адрес SYS+8.
; Косвенная адресация по смещению с индексацией
        MOVAL   ADA, R8         ; Получить в R8 адрес ADA.
        MOVL    #80, R1         ; Установить индексный регистр.
        TSTF    @16(R8) [R1]    ; Проверить значение в формате с пла-
                                ; вающей запятой, чей адрес равен
                                ; 80*4 плюс адрес, хранящийся по ад-
                                ; ресу (ADA+16).
```

Поскольку режим с индексацией определяет два значения — базовый адрес и смещение, это нашло свое отражение и в формате операнда, который состоит из двух частей и выглядит следующим образом:



В правом полубайте первого байта записывается номер индексного регистра, затем следует код режима адресации 04,

который и определяет режим адресации с индексацией. Непосредственно за этим следует спецификатор операнда базы, по которому определяется базовый адрес. Он занимает столько байт, сколько необходимо для соответствующего режима адресации базы. На спецификатор базы распространяются правила его формирования при обычном использовании.

Сочетая индексацию с другими режимами адресации, можно образовать режимы, как показано ниже.

Режимы адресации с индексацией	Формат
Косвенно-регистровый с индексацией	(RN) [RX]
Индексация с автоувеличением	(RN)+[RX]
Косвенная адресация с автоувеличением и индексацией	@(RN)+[RX]
Индексация с автоуменьшением	—(RN) [RX]
Адресация по смещению с индексацией	смщ (RN) [RX]
Косвенная адресация по смещению с индексацией	@смщ (RN) [RX]
Относительная адресация с индексацией	адрес [RX]
Косвенная относительная адресация с индексацией	@ адрес [RX]
Абсолютная адресация с индексацией	@ # адрес [RX]
Адресация общего вида с индексацией	G ^ адрес [RX]

Примечания:

(RN) — любой регистр общего назначения: R0 ... R12, AP, FP, SP;

[RX] — любой регистр общего назначения: R0 ... R12, AP, FP, SP. При индексации с автоувеличением, при косвенной адресации с автоувеличением и индексацией и при индексации с автоуменьшением регистр RX не может быть одним и тем же регистром, что и RN;

смщ — выражение, определяющее смещение;

адрес — выражение, определяющее адрес.

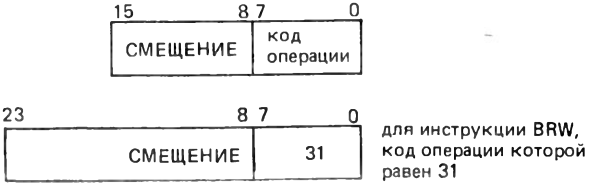
При индексации с автоувеличением, при косвенной адресации с автоувеличением и индексацией, а также при индексации с автоуменьшением регистр RX не может быть одним и тем же регистром, что и RN. В противном случае это может привести к непредсказуемым результатам.

Для каждого режима адресации с индексацией процесс обработки базового режима адресации и добавления индексного регистра одинаков.

2.2.4. АДРЕСАЦИЯ В ИНСТРУКЦИЯХ ПЕРЕХОДА

С точки зрения особенностей формирования операндов в инструкциях перехода наибольший интерес представляют инструкции условного перехода, а также безусловного перехода BRB и BRW. Для этих инструкций применяется режим адресации, аналогичный режиму адресации через счетчик инструкций с той лишь разницей, что смещение, которое складывается с те-

кущим значением счетчика инструкций и определяет адрес перехода, хранится в одном байте, непосредственно следующим за кодом операции. (Для инструкции BRW используется слово.) Формат рассматриваемых инструкций перехода имеет следующую структуру:



В структуре в явном виде не указывается ни счетчик инструкций, ни код режима адресации. Это все подразумевается по умолчанию и определяется по соответствующим кодам операций. Байт, который всегда содержал эту информацию, в данном случае содержит смещение, которое и определяет адрес перехода. Смещение в байте допускает адресацию на 127 байт вперед и на 128 байт назад, а смещение в слове — на 32767 байт вперед и на 32768 байт назад. При данном режиме адресации смещение считается относительно скорректированного значения счетчика инструкций PC, т. е. относительно байта, следующего за байтом или словом, содержащим значение смещения.

В качестве адреса перехода в рассматриваемых инструкциях может использоваться выражение. После его оценки оно усекается и помещается соответственно в байт или слово.

Пример.

```
CMPL    R0, R1    ; Сравнить содержимое регистров R0 и R1.
BEQL    TEST1     ; По равенству — переход на метку TEST1, смещение хранится в байте.
BRW     TEST2     ; Безусловный переход на метку TEST2.
                  ; Смещение хранится в слове.
```

Использование инструкций перехода позволяет значительно экономить память, поскольку для адресации требуемой точки перехода используется один байт. Однако их применение связано с некоторыми трудностями программирования. При первоначальном написании программы одного байта может быть достаточно для организации перехода. Программа при отладке и развитии может изменять свои размеры, и могут появляться ошибки, связанные с неверной адресацией полей, которые выходят за диапазон работы инструкции перехода.

Глава 3
СИСТЕМА ИНСТРУКЦИЙ

Базовый набор инструкций ЭВМ СМ1700 включает 122 типа инструкций, которые имеют 304 кода операции и 317 мнемонических обозначений (для удобства программирования некоторым одинаковым кодам присвоено несколько мнемоник). Многие инструкции непосредственно соответствуют операторам языков высокого уровня, а используемые в Макроассемблере мнемонические обозначения инструкций ассоциируются с наименованием широко распространенных функций. Большинство инструкций содержит естественное число операндов. Например, арифметические инструкции (ADD, SUB, MUL, DIV) имеют формы не только с двумя, но и с тремя операндами. Формат инструкции — переменной длины; он выравнивается по границе байта. Формат переменной длины делает инструкции компактными и позволяет расширять их набор. Код операции может занимать один или два байта. В зависимости от инструкции за кодом операции следует от 0 до 6 операндов.

По функциональному назначению инструкции делятся на десять основных групп:

- целочисленные арифметические и логические;
- адресные;
- для работы с битовыми полями переменной длины;
- перехода;
- вызова процедур;
- работы с очередями;
- работы с плавающей запятой;
- работы с символьными строками;
- работы с упакованными десятичными строками;
- специального назначения.

Состав групп во многом определяется теми типами данных, которые доступны в Макроассемблере для обработки. Кроме того, в состав инструкций включены инструкции, обеспечивающие эффективную организацию программирования. Ряд инструкций отражает отдельные архитектурные возможности СМ1700: такие, как работа с привилегированными регистрами процессора, работа с контекстом образа и т. д.

Для удобства изложения информация по каждой группе сведена в таблицу. Для каждой группы инструкций приведены примеры программирования. Мнемоника некоторых инструкций

при описании их форматов содержит список символов в квадратных скобках. Это означает, что данная инструкция имеет несколько модификаций в зависимости от типа данных, типа перехода, типа доступа, режима работы процессора и состояния отдельных разрядов. При программировании содержимое квадратных скобок заменяется на один из символов, которые имеют приведенные ниже значения.

Тип данных: B — байт; W — слово; L — длинное слово; Q — квадрослово; O — октаслово; F, D, G, H — число с плавающей запятой в соответствующем формате.

Режим процессора: K — ядро; E — управление; S — супервизор; U — пользователь.

Тип доступа: R — чтение; W — запись.

Состояние разряда: C — сброшен; S — установлен.

В форматах инструкций операнды имеют обозначение OPN, где N — целое число от 1 до 6. Если при описании инструкций обозначение операнда взято в скобки (OPN), то оно обозначает содержимое операнда (в данном случае операнда N).

При рассмотрении инструкций, как правило, не приводится описание установки кодов условий (в общем виде их значение описано в гл. 1 в подразделе «Основной процессор»), а также не дается информация по исключительным ситуациям, которые могут возникать при выполнении инструкции: это можно найти в документации.

Коды инструкций приводятся в шестнадцатеричной системе счисления.

3.1. ЦЕЛОЧИСЛЕННЫЕ АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ИНСТРУКЦИИ

Инструкции данной группы (табл. 3.1) выполняют традиционные действия и не требуют специальных пояснений. Отметим только, что в основном они работают с операндами длиной в байт, слово и длинное слово. Кроме того, для удобства программирования введены различные модификации инструкций (например, двух- или трехоперандные).

Пример 1.

```
; Использование целочисленных и
; логических инструкций MULL2,
; MULL3, ADDL2, DIVL3, INCL, MOVL.
; Фрагмент программы. Вычислить
; значение выражения (4*B*B+6)/2+1
; и результат поместить в поле COUNT.
; *Область данных*
B:          ..LONG   11246          ; B=11246
COUNT:       .LONG   0              ; Поле результата.
```

Таблица 3.1

ЦЕЛОЧИСЛЕННЫЕ АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ИНСТРУКЦИИ

Мнемоника	Код	Название и формат	Описание
ADAWI	58	Сложение выровненных слов с блокировкой памяти ADAWI ОП1, ОП2	Содержимое ОП1 и ОП2 складывается, результат запоминается по адресу ОП2. Блокировка памяти не допускает аналогичных операций со стороны других процессоров в мультипроцессорной системе
ADDB2 ADDW2 ADDL2	80 A0 C0	Сложение (двухоперандное) ADD [B, W, L] 2 ОП1, ОП2	Содержимое ОП1 и ОП2 складывается, результат запоминается по адресу ОП2
ADDB3 ADDW3 ADDL3	81 A1 C1	Сложение (трехоперандное) ADD [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП1 и ОП2 складывается, результат запоминается по адресу ОП3
ADWC	D8	Сложение с переносом ADWC ОП1, ОП2	Значение суммы двух операндов (ОП1, ОП2) складывается со значением кода условий C, содержащего значение переноса старшего значащего разряда. Результат запоминается по адресу ОП2
ASHL ASHQ	78 79	Арифметический сдвиг ASH [L, Q] ОП1, ОП2, ОП3	Содержимое ОП2 сдвигается на число разрядов, указанное в ОП1. Результат запоминается по адресу ОП3. При положительном ОП1 — сдвиг влево, при отрицательном — вправо
BICB2 BICW2 BICL2	8A AA CA	Очистка разрядов (двухоперандная) BIC [B, W, L] 2 ОП1, ОП2	Содержимое ОП2 логически умножается на инвертированное содержимое ОП1. Результат запоминается по адресу ОП2
BICB3 BICW3 BICL3	8B AB CB	Очистка разрядов (трехоперандная) BIC [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП2 логически умножается на инвертированное содержимое ОП1. Результат запоминается по адресу ОП3

Продолжение

Мнемоника	Код	Название и формат	Описание
BISB2 BISW2 BISL2	88 A8 C8	Установка разрядов (двухоперандная) BIS [B, W, L] 2 ОП1, ОП2	Содержимое ОП1 логически складывается с содержимым ОП2. Результат запоминается по адресу ОП2
BISB3 BISW3 BISL3	89 A9 C9	Установка разрядов (трехоперандная) BIS [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП1 логически складывается с содержимым ОП2. Результат запоминается по адресу ОП3
CLRB CLRW CLRL CLRQ CLRO	94 84 D4 7C 7CFD	Очистка CLR [B, W, L, Q, O] ОП1	По адресу ОП1 заносится нуль
CMPB CMPW CMPL	91 B1 D1	Сравнение CMP [B, W, L] ОП1, ОП2	Содержимое ОП1 сравнивается с содержимым ОП2. Результатом является соответствующая установка кодов условий, которые анализируются инструкциями переходов. N=1, если (ОП1) < < (ОП2) Z=1, если (ОП1) = = (ОП2) V=0 C=1, если (ОП1) меньше без знака (ОП2)
CVTBW CVTBL CVTWB CVTWL CVTLB CVTLW	99 98 33 32 F6 F7	Преобразование CVTB [W, L] ОП1, ОП2 CVTW [B, L] ОП1, ОП2 CVTL [B, W] ОП1, ОП2	Тип данных, содержащихся в ОП1, преобразуется в другой тип, который указан в инструкции. Результат запоминается в ОП2. При преобразовании некоторого типа данных в более длинный производится расширение знака. При преобразовании некоторого типа данных в более короткий производится усечение старших разрядов

Продолжение

Мнемоника	Код	Название и формат	Описание
DECB DECW DECL	97 B7 D7	Автоуменьшение DEC [B, W, L] ОП1	Из содержимого ОП1 вычитается единица. Результат запоминается в ОП1
DIVB2 DIVW2 DIVL2	86 A6 C6	Деление (двухоперандное) DIV [B, W, L] 2 ОП1, ОП2	Содержимое ОП2 делится на содержимое ОП1. Результат запоминается в ОП2
DIVB3 DIVW3 DIVL3	87 A7 C7	Деление (трехоперандное) DIV [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП2 делится на содержимое ОП1. Результат запоминается в ОП3
EDIV	7B	Расширенное деление EDIV ОП1, ОП2, ОП3, ОП4	Содержимое ОП2 делится на содержимое ОП1. Частное запоминается в ОП3, остаток — в ОП4
EMUL	7A	Расширенное умножение EMUL ОП1, ОП2, ОП3, ОП4	Содержимое ОП1 умножается на содержимое ОП2. Результат имеет двойную длину. Знак содержимого ОП3 расширяется до двойной длины, а затем прибавляется к результату. Окончательный результат запоминается в ОП4
INCB INCW INCL	96 B6 D6	Автоувеличение INC [B, W, L] ОП1	К содержимому ОП1 прибавляется единица. Результат запоминается в ОП1
MCOMB MCOMW MCOML	92 B2 D2	Пересылка в дополнительном коде MCOM [B, W, L] ОП1, ОП2	Дополнительный код содержимого ОП1 пересылается по адресу ОП2
MNEGB MNEGW MNEGL	8E AE CE	Пересылка со сменой знака MNEG [B, W, L] ОП1, ОП2	Содержимое ОП1 с противоположным знаком пересылается по адресу ОП2
MOVB MOVW MOVL MOVQ MOVO	90 B0 D0 CD 7DFD	Пересылка MOV [B, W, L, Q, O] ОП1, ОП2	Содержимое ОП1 пересылается в ОП2

Продолжение

Мнемоника	Код	Название и формат	Описание
MOVZBW MOVZBL MOVZWL	9B 9A 3C	Пересылка с дополнением нулями MOVZ [BW, BL, WL] ОП1, ОП2	При пересылке байта в слово (инструкция MOVZBW) в разряды 0—7 ОП2 заносится содержимое ОП1, а разряды 8—15 заполняются нулями. При пересылке байта в длинное слово (инструкция MOVZBL) в разряды 0—7 ОП2 заносится содержимое ОП1, а разряды 8—31 заполняются нулями. При пересылке слова в длинное слово (инструкция MOVZWL) в разряды 0—15 ОП2 заносится содержимое ОП1, а разряды 16—31 заполняются нулями
MULB2 MULW2 MULL2	84 A4 C4	Умножение (двухоперандное) MUL [B, W, L] 2 ОП1, ОП2	Содержимое ОП1 умножается на содержимое ОП2. Результат запоминается в ОП2
MULB3 MULW3 MULL3	85 A5 C3	Умножение (трехоперандное) MUL [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП1 умножается на содержимое ОП2. Результат запоминается в ОП3
PUSHL	DD	Запись в стек длинного слова PUSHL ОП1	Длинное слово, содержащееся по адресу ОП1, заносится в стек
ROTL	9C	Циклический сдвиг длинного слова ROTL ОП1, ОП2, ОП3	Содержимое ОП2 циклически сдвигается на число разрядов, указанное в ОП1. Результат запоминается в ОП3. При положительном ОП1 — сдвиг влево, при отрицательном — вправо
SBWC	D9	Вычитание с переносом SBWC ОП1, ОП2	Содержимое ОП1 и разряда С кода условий вычитаются из содержимого ОП2. Результат запоминается в ОП2

Продолжение

Мнемоника	Код	Название и формат	Описание
SUBB2 SUBW2 SUBL2	82 A2 C2	Вычитание (двухоперандное) SUB [B, W, L] ОП1, ОП2	Содержимое ОП1 вычитается из содержимого ОП2. Результат запоминается по адресу ОП2
SUBB3 SUBW3 SUBL3	83 A3 C3	Вычитание (трехоперандное) SUB [B, W, L] 3 ОП1, ОП2, ОП3	Содержимое ОП1 вычитается из содержимого ОП2. Результат запоминается по адресу ОП3
TSTB TSTW TSTL	95 B5 D5	Проверка TST [B, W, L] ОП1	В соответствии с содержимым ОП1 устанавливаются коды условий: N=1, если (ОП1) < 0; Z=1, если (ОП1)=0; V=0; C=0
XORB2 XORW2 XORL2	8C AC CC	Исключающее или (двухоперандное) XOR [B, W, L] 2 ОП1, ОП2	Результат логической операции «исключающее или» между содержимым ОП1 и ОП2 запоминается в ОП2
XORB3 XORW3 XORL3	8D AD CD	Исключающее или (трехоперандное) XOR [B, W, L] 3 ОП1, ОП2, ОП3	Результат логической операции «исключающее или» между содержимым ОП1 и ОП2 запоминается в ОП3

```
;   Область инструкций
        MULL3   B, B, R0        ; B*B в R0.
        MULL2   #4, R0          ; 4*B*B в R0.
        ADDL2   #6, R0          ; 4*B*B+6 в R0.
        DIVL3   #2, R0, R1      ; (4B*B+6)/2 в R1.
        INCL    R1              ; (4B*B+6)/2+1 в R1.
        MOVL    R1, COUNT       ; Запомнить результат в COUNT.
```

Пример 2.

```
; Сохранить содержимое регистра R2 в стеке.
        MOVL    R2, (—SP)       ; Вариант 1.
        PUSHL   R2              ; Вариант 2.
; Инструкции эквивалентны, но инструкция PUSHL занимает в памяти
; меньше места.
```

Пример 3.

```
; Использование логических инструкций BICB2, BISB2.
        BICB2   #3, R0  ; Очистить разряды 0 и 1 в R0.
        BISB2   #3, R1  ; Установить разряды 0 и 1 в R1.
```

3.2. АДРЕСНЫЕ ИНСТРУКЦИИ

В инструкциях этой группы (табл. 3.2) используются адреса операндов, а не их содержимое. Манипуляции с адресами можно осуществлять с помощью различных режимов адресации. Однако инструкции данной группы делают программу более «прозрачной» и позволяют сосредоточиться на реализации алгоритма задачи.

Таблица 3.2

АДРЕСНЫЕ ИНСТРУКЦИИ

Мнемоника	Код	Название и формат	Описание
MOVAB MOVAW MOVAL MOVAF MOVAQ MOVAD MOVAG MOVAH MOVAO	9E 3E DE DE 7E 7E 7E 7EFD 7EFD	Пересылка адреса	Адрес ОП1 пересылается по адресу ОП2. Контекст, по которому производится вычисление ОП1, определяется типом данных, указанным в инструкции
MOVA [B, W, L, F, Q, D, G, O, H] ОП1, ОП2			
PUSHAB PUSHAW PUSHAL PUSHAF PUSHAQ PUSHAD PUSHAG PUSHAO PUSHAH	9F 3F DF DF 7F 7F 7F 7FFD 7FFD	Запись адреса в стек	Адрес ОП1 заносится в стек. Контекст, по которому производится вычисление адреса ОП1, определяется типом данных, используемым в инструкции (например, PUSHAB — занесение в стек адреса байта)
PUSHA [B, W, L, F, Q, D, G, O, H] ОП1			

Пример 1.

```
; Использование адресной инструкции MOVAW.
; Установить адреса таблиц TABLE—1, TABLE—2 в R0, R1.
; *Область данных*
TABLE—1:. BLKW 20          ; TABLE—1 занимает 20 слов.
TABLE—2:. BLKW 200         ; TABLE—2 занимает 200 слов.
; *Область инструкций*
        MOVAW TABLE—1, R0  ; Переслать адрес таблицы
                           ; TABLE—1 в регистр R0.
        MOVAW TABLE—2, R1  ; Переслать адрес таблицы
                           ; TABLE—2 в регистр R1.
```

Пример 2.

```
; Использование адресных инструкций MOVAL, PUSHAL.
; *Область данных*
ARRAY—1:        . BLKL    30      ; Область массива ARRAY—1.
ARRAY—2:        . BLKL    36      ; Область массива ARRAY—2.
; *Область инструкций*
; Инструкция PUSHAL эквивалентна MOVAL,
; но занимает в памяти на 1 байт меньше.
        PUSHAL ARRAY—1          ; Адрес массива ARRAY—1 загружает-
                                ; ся в стек.
        MOVAL ARRAY—2, —(SP)    ; Адрес массива ARRAY—2 загружает-
                                ; ся в стек.
```

3.3. ИНСТРУКЦИИ ДЛЯ РАБОТЫ С БИТОВЫМИ ПОЛЯМИ ПЕРЕМЕННОЙ ДЛИНЫ

Операции над битовыми полями переменной длины можно производить с помощью логических инструкций, но использование инструкций, специально предназначенных для работы с этим типом данных, делает программирование более эффективным.

Инструкции данной группы (табл. 3.3) позволяют организовать гибкую работу при решении задач алгебры логики, операциях над множествами в информационных системах и т. д. Все инструкции выполняются с операндами, адресация осуществляется посредством базового адреса и смещения от этого адреса (позиции). Такая форма связана с тем, что адресации произвольной последовательности разрядов в байте требуется минимум 35 разрядов, тогда как виртуальный адрес любого байта имеет 32 разряда.

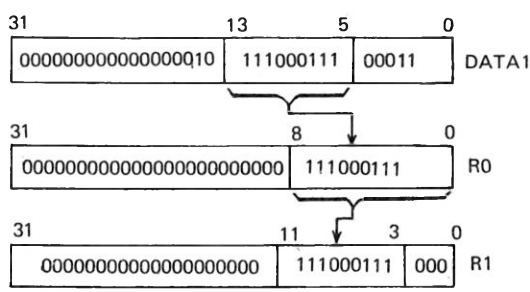


Рис. 3.1. Иллюстрация к примеру 1

Базовый адрес может относиться к регистру, как к любому байту в памяти. Полностью битовое поле определяется тремя операндами: базовым адресом, позицией и длиной поля. Пози-

Таблица 3.3

ИНСТРУКЦИИ ДЛЯ РАБОТЫ С БИТОВЫМИ ПОЛЯМИ ПЕРЕМЕННОЙ ДЛИНЫ

Мнемоника	Код	Название и формат	Описание
CMPV CMPZV CMP [V, ZV]	EC ED ОП1,	Сравнение битового поля с длинным словом ОП2, ОП3, ОП4	Содержимое поля, определяемого позицией (ОП1), длиной (ОП2), базовым адресом (ОП3), сравнивается с содержимым длинного слова (ОП4). При выполнении операции CMPV знак поля расширяется до длинного слова, при выполнении операции CMPZV содержимое поля дополняется нулями. Если поле имеет максимально возможную длину (32 разряда), то результат действия инструкций CMPV и CMPZV будет одинаков. В результате выполнения инструкции содержимое поля не изменяется, а коды условий устанавливаются в зависимости от значений поля и содержимого ОП4: N=1, (поле) < (ОП4); Z=1, (поле) = = (ОП4); V=0; C=1, (поле) > > (ОП4)
EXTV EXTZV EXT [V, ZV]	EE EF ОП1,	Пересылка битового поля в длинное слово ОП2, ОП3, ОП4	Содержимое поля, определяемого позицией (ОП1), длиной (ОП2), базовым адресом (ОП3), пересылается по адресу ОП4. При выполнении EXTV значение крайнего левого бита поля расширяется до длинного слова, а при выполнении EXTZV расширение до длинного слова обеспечивается нулями
FFS FFC FF [C, S]	EB EA ОП1,	Поиск первого разряда определенного состояния ОП2, ОП3, ОП4	Просматривается каждый разряд поля, определяемого позицией (ОП1), длиной (ОП2) и базовым адресом (ОП3), на наличие разряда, состояние которого соответствует указанному в инструкции (C — сброшен, S — установлен). Если такой разряд найден, то его номер заносится по адресу ОП4 и устанавливается Z=0, если не найден, то по адресу ОП4 заносится номер разряда, который находится на 1 разряд левее позиции поля и устанавливается Z=1
INSV INSV	F0 ОП1,	Пересылка из длинного слова в битовое поле ОП2, ОП3, ОП4	В поле, которое определяется позицией (ОП2), длиной (ОП3) и базовым адресом (ОП4), переносится содержимое младших разрядов длинного слова (ОП1). Количество пересылаемых разрядов определяется длиной поля (ОП3).

ция задается как целое число со знаком размером в длинное слово. Длина задается целым числом размером байт.

Ниже приведены примеры выделения, вставки битовых полей, поиска первого установленного, сброшенного разрядов (рис. 3.1—3.3).

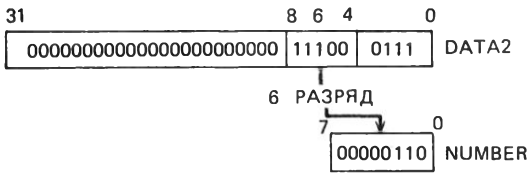


Рис. 3.2. Иллюстрация к примеру 2

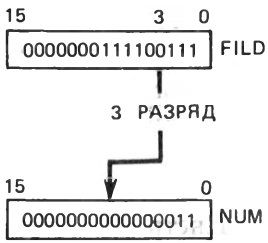


Рис. 3.3. Иллюстрация к примеру 3

Пример 1.

```
; В примере используются инструкции
; работы с битовыми полями EXTZV, INSV.
; *Область данных*
DATA1: . LONG    ^B1011100011100011    ; Двоичное значение DATA1
; *Область инструкций*
; Выделить поле (13 : 5) из DATA 1 в R0.
        EXTZV    #5, #9, DATA1, R0
; Вставить поле (11 : 3) из R0 в R1.
        INSV     R0, #3, #9, R1
```

Пример 2.

```
; В примере используется инструкция
; работы с битовыми полями FFS.
; *Область данных*
DATA2:   . LONG    ^B111000111    ; Двоичное значение DATA2.
NUMBER:. BYTE      0              ; Поле номера разряда.
; *Область инструкций*
; Найти в DATA2 в поле (8 : 4) номер первого установленного
; разряда и поместить его в NUMBER.
        FFS      #4, #5, DATA2, NUMBER ; Результат будет 6.
; Проверка: если разряд найден, то код условий Z=0.
        BNEQ     CONT1          ; Разряд найден, идти на метку CONT1.
        BRW      CONT2          ; Разряд не найден, идти на метку
                                ; CONT2.
```

Пример 3.

```
; В примере используется инструкция работы
; с битовыми полями FFC.
```

```
; *Область данных*
FILD:  . WORD   ^B111100111
NUM:   . WORD   0                ; Номер разряда.
; *Область инструкций*
; Найти в слове FILD в поле (0 : 6) номер
; первого сброшенного разряда и поместить его в NUM.
       FFC      #0, #7, FILD, NUM    ; Результат будет 3.
; Проверка: если разряд найден, то код условий Z=0.
       BNEQ     M1                   ; Разряд найден, идти на метку M1.
       BRB      M2                   ; Разряд не найден, идти на метку M2.
```

3.4. ИНСТРУКЦИИ ПЕРЕХОДА

Инструкции перехода (табл. 3.4 и 3.5) позволяют программировать ветвящиеся и циклические вычислительные процессы. В отличие от ранних моделей СМ ЭВМ в СМ1700 эти инструкции совмещены с другими, что обеспечивает более гибкое программирование логических задач, особенно при формировании циклов, управляемых счетчиков. Например, инструкция ACB эквивалентна оператору языков высокого уровня FOR... TO и позволяет программировать циклы, проверяя условия их окончания после завершения операции внутри тела цикла. С этой точки зрения она может быть интерпретирована следующим образом:

ACB предел, шаг, параметр—цикла, адрес—перехода

Аналогично интерпретируются инструкции AOB и SOB, которые можно достаточно часто использовать вместо инструкции ACB.

Таблица 3.4

ИНСТРУКЦИИ ПЕРЕХОДА

Мнемоника	Код	Название и формат	Описание
ACBB ACBW ACBL ACBF ACBD ACBG ACBH ACB [B, W, L, D, G, H] ОП1, ОП2, ОП3, ОП4	9D 3D F1 4F 4FFD 6F 6FFD	Сложение, сравнение и переход	Содержимое ОП2 и содержимое ОП3 складываются, сумма запоминается в ОП3. Затем содержимое ОП3 и ОП1 сравнивается, и если выполняется одно из двух условий (условие 1: (ОП2) $\geqslant 0$ и (ОП3) $\leqslant$ (ОП1); условие 2: (ОП2) < 0 и (ОП3) $\geqslant$ (ОП1)), то происходит переход по адресу, полученному путем прибавления к содержимому РС смещения, хранящегося в ОП4

Продолжение

Мнемоника	Код	Название и формат	Описание
AOBLEQ	F3	Прибавление единицы и переход, если меньше или равно AOBLEQ ОП1, ОП2, ОП3	К содержимому ОП2 прибавляется единица и результат запоминается в ОП2. Затем, если (ОП2) $\leqslant$ $\leqslant$ (ОП1), происходит переход по адресу, полученному путем прибавления к содержимому PC смещения, хранящегося в ОП3
AOBLSS	F2	Прибавление единицы и переход, если меньше AOBLEQ ОП1, ОП2, ОП3	К содержимому ОП2 прибавляется единица и результат запоминается в ОП2. Если (ОП2) $<$ (ОП1), то происходит переход по адресу, полученному путем прибавления к содержимому PC смещения, хранящегося в ОП3
BBS BBC	E0 E1	Переход по состоянию разряда BB [S, C] ОП1, ОП2, ОП3	Проверяется состояние разряда, позиция и адрес которого хранятся соответственно в ОП1 и ОП2. Если проверяемое состояние разряда соответствует указанному в инструкции (S — установлен, C — сброшен), то происходит переход по адресу, полученному путем прибавления к содержимому PC смещения, хранящегося в ОП3
BBSS BBCS BBSC BBCC	E2 E3 E4 E5	Переход по состоянию разряда и его модификация BB [S, C] [S, C] ОП1, ОП2, ОП3	Выполняется инструкция BB [S, C], а затем проверяемый разряд устанавливается в состояние, указанное в инструкции, независимо от того, произошел переход или нет
BBSSI BBCCI	E6 E7	Переход по состоянию разряда и его модификация с блокировкой памяти BB [SS, CC] I ОП1, ОП2, ОП3	Выполняется инструкция BBSS (если указана BBSSI) или BBCC (если указана BBCCI). Причем чтение проверяемого разряда и его установка производятся с блокировкой памяти, т. е. в это время другие процессоры или устройства не имеют к нему доступа
BLBS BLBC	E8 E9	Переход по состоянию младшего разряда BLB [S, C] ОП1, ОП2	Проверяется состояние нулевого разряда ОП1. Если это состояние соответствует указанному в инструкции, происходит переход по адресу, получаемому путем прибавления к содержимому PC смещения, хранящегося в ОП2

Продолжение

Мнемоника	Код	Название и формат	Описание
BRB BRW	11 31	Безусловный переход BR[B, W] ОП1	Происходит переход по адресу, получаемому путем прибавления к содержимому PC смещения, хранящегося в ОП1
CASEB CASEW CASEL	8F AF CF	Выбор перехода CASE [B, W, L] ОП1, ОП2, ОП3 смещение—1 смещение—2 . . . смещение—N	Разность ((ОП1) — (ОП2)) между значением переключателя (ОП1) и базы (ОП2) сравнивается с пределом (ОП3). Если результат меньше или равен 0, то к содержимому счетчика инструкций PC прибавляется то смещение, номер которого равен полученному результату. Предел (ОП3) используется для обнаружения ошибки при выходе переключателя за допустимый диапазон. Если полученный результат принимает недопустимое значение, то управление передается по адресу, следующему за последним смещением
JMP	17	Универсальный переход JMP ОП1	В счетчик инструкций PC заносится содержимое ОП1
JSB	16	Переход к подпрограмме JSB ОП1	Содержимое PC заносится в стек, затем в PC заносится содержимое ОП1
RSB	05	Возврат из подпрограммы RSB	В PC заносится длинное слово, выбранное из стека
SOBGEQ	F4	Вычитание единицы и переход, если больше или равно нулю SOBGEQ ОП1, ОП2	Из содержимого ОП1 вычитается единица. Если результат больше или равен 0, то к содержимому PC прибавляется смещение, хранящееся в ОП2, и по новому адресу происходит переход
SOBGTR	F5	Вычитание единицы и переход, если больше нуля SOBGTR ОП1, ОП2	Из содержимого ОП1 вычитается единица. Если результат больше нуля, то к содержимому PC прибавляется смещение, хранящееся в ОП2, и по новому адресу происходит переход

3.5. ИНСТРУКЦИИ ПЕРЕХОДА ПО КОДАМ УСЛОВИЙ

Среди инструкций перехода отдельно выделены инструкции перехода по кодам условий (табл. 3.5). Это выделение не принципиальное, а сделано лишь для наглядности изложения, так как семантика этих инструкций достаточно проста и не требует пояснений.

Таблица 3.5

ИНСТРУКЦИИ ПЕРЕХОДА ПО КОДАМ УСЛОВИЙ

Мнемоника	Код	Название	Значение кодов условий в PSW для перехода
BGTR	14	Переход, если больше	N=0 или Z=0
BLEQ	15	Переход, если меньше или равно	N=1 или Z=1
BNEQ	12	Переход, если не равно	Z=0
BNEQU	12	Переход, если не равно (без знака)	Z=0
BEQL	13	Переход, если равно	Z=1
BEQLU	13	Переход, если равно (без знака)	Z=1
BGEQ	18	Переход, если больше или равно	N=0
BLSS	19	Переход, если меньше	N=1
BGTRU	1A	Переход, если больше (без знака)	C=0 или Z=0
BLEQU	1B	Переход, если меньше или равно (без знака)	C=1 или Z=1
BVC	1C	Переход, если нет переполнения	V=0
BVS	1D	Переход, если есть переполнение	V=1
BGEQU	1E	Переход, если больше или равно (без знака)	C=0
BCC	1E	Переход, если нет переноса	C=0
BLSSU	1F	Переход, если меньше (без знака)	C=1
BCS	1F	Переход, если есть перенос	C=1

Наибольший эффект при использовании инструкций перехода можно получить, если передача управления производится на границу длинного слова. Кроме того, при организации переходов требуется определенная осторожность. Так, инструкции перехода по переполнению и переносу (BVS, BVC, BCS и BCC) целесообразно использовать для проверки наличия переполнения (в случае запрета внутренних прерываний по переполнению), для выполнения арифметических операций, когда используется повышенная точность, и т. п.

Инструкции беззнаковых переходов (BLSSU, BLEQU, BEQLU, BNEQU, BGEQU и BGTRU) целесообразно применять сразу после арифметических инструкций и инструкций для работы с битовыми полями переменной длины (т. е. для операндов, интерпретируемых как целые числа без знака), а также после инструкций для работы с символьными строками и после адресных инструкций.

Инструкции знаковых переходов (BLSS, BLEQ, BEQL, BNEQ, BGEQ и BGTR) лучше использовать после инструкций, в которых операнды интерпретируются как целые числа со знаком, а также после инструкций для работы с десятичными строками и числами с плавающей запятой.

Пример 1.

```
; В примере используются инструкции перехода
; BGEQ, SOBGTR, BRW.
; Фрагмент программы.
; Определение минимального элемента из
; списка целых чисел и их общей суммы.
; R0 содержит адрес списка,
; R1 содержит количество элементов списка.
; Минимальное число поместить в MIN, сумму в SUM.
; MIN и SUM имеют формат длинного слова.

; *Область данных*

MIN:    . LONG    0              ; Поле минимального числа.
SUM:    . LONG    0              ; Поле суммы.

; *Область инструкций*

        MOVL      (R0), MIN      ; Установить первое значение в MIN.
MINI:   CMPL      (R0), MIN      ; Сравнить очередной элемент с MIN.
        BGEQ      ADD            ; Если больше, перейти на суммиро-
                                 ; вание.
        MOVL      (R0), SUM      ; Если меньше, получить новое значе-
                                 ; ние MIN.
ADD:    ADDL2     (R0), SUM      ; Определение суммы.
        SOBGTR    R1, MIN        ; Если список не исчерпан, продолжить
                                 ; поиск MIN.
        BRW       CONTINUE       ; Переход на продолжение обработки.
```

Пример 2.

```
; Возможные варианты использования инструкции JMP.
        JMP     METKA       ; Происходит переход по адресу METKA.
        JMP     @ ADR       ; Происходит переход по адресу, со-
                            ; держащемуся в ADR.
        JMP     (R2)        ; Происходит переход по адресу, со-
                            ; держащемуся в R2.
        JMP     (R3)+       ; R3 содержит адрес, по которому после
                            ; его увеличения на 4 происходит пе-
                            ; реход.
```

Пример 3.

```
; Если разряд 2 в регистре R1 сброшен,
; то перейти на метку TASK.
; Вариант 1.
        BBC     #2, R1, TASK
; Вариант 2.
        BITB    #4, R1
        BEQL    TASK
; Варианты 1 и 2 эквивалентны, но вариант 1 короче.
```

3.6. ИНСТРУКЦИИ ВЫЗОВА ПРОЦЕДУР

В этот набор входят три инструкции (табл. 3.6), две из которых используются для вызова, третья — для возврата из процедуры. С помощью этих инструкций обеспечивается стандартная процедура вызова процедуры и возврата из нее, которые поддерживают обращения к процедурам из языков высокого уровня, а также все межмодульные взаимодействия в МОСВП. Вызов процедуры может быть осуществлен со списком параметров, которые располагаются в любом месте или в стеке. Однако в обоих случаях подразумевается, что по начальному адресу вызываемой процедуры расположено слово, называемое маской, за которой непосредственно следует процедура.

Маска определяет используемые в процедуре регистры, а также разрешения внутренних прерываний по переменной. При стандартном вызове процедуры регистры R0 и R1 могут не сохраняться, а задействованные в процедуре регистры (с R2 по R11) должны быть указаны в маске. Эта информация применяется для формирования в стеке структуры данных, называемой *кадром вызова*, или *кадром стека*. Такой кадр содержит значения регистров, часть слова состояния процессора, маску используемых регистров и т. п. Эта информация используется для сохранения состояния процессора и его восстановления при выходе из процедуры с помощью инструкции возврата.

Таблица 3.6

ИНСТРУКЦИИ ВЫЗОВА ПРОЦЕДУР

Мнемоника	Код	Название и формат	Описание
CALLG	FA	Вызов процедуры с обычным списком аргументов CALLG ОП1, ОП2	Вызывается процедура, имя которой указано в ОП2. ОП1 содержит адрес списка аргументов — последовательность длинных слов, где в первом длинном слове (счетчике) содержится количество следуемых за ним аргументов. Регистры R0, R1 не должны отмечаться во входной маске вызываемой процедуры
CALLS	FB	Вызов процедуры со списком аргументов в стеке CALLS ОП1, ОП2	Вызывается процедура, имя которой указано в ОП2, а количество используемых аргументов, предварительно записанных в стек, хранится в ОП1. Регистры R0, R1 не должны быть указаны во входной маске вызываемой процедуры
RET	04	Возврат из процедуры RET	Управление возвращается инструкции, следующей за инструкцией, вызвавшей обращение к процедуре

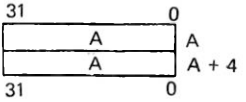
Пример.

```
; В примере используются инструкции работы с процедурами:
; CALLG, CALLS, RET.
; *Область данных*
  ARG—LIST:
        . LONG    3       ; Счетчик аргументов.
        . LONG            ; Первый аргумент.
        . LONG    200     ; Второй аргумент.
        . LONG    44      ; Третий аргумент.
; *Область инструкций*
START:
      ...                 ; Тело основной программы.
        CALLG     ARG—LIST, CAFE    ; Вызов процедуры CAFE
                                    ; с обычным списком
                                    ; аргументов ARG—LIST.
      ...                 ; Тело основной программы.
        PUSHL     #333    ; Записать в стек третий аргумент.
        PUSHL     #200    ; Записать в стек второй аргумент.
        PUSHL     #10     ; Записать в стек первый аргумент.
        CALLS     #3, TEA ; Вызов процедуры TEA со списком
                          ; аргументов в стеке.
      ...                 ; Тело основной программы.
```

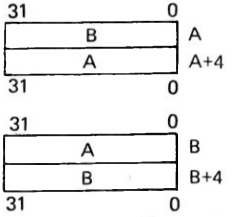
```
; Процедура        CAFE
CAFE:  . WORD     ^M<R2, R3>    ; Сохранить регистры R2, R3.
       ...                      ; Тело процедуры CAFE.
       RET                      ; Выход из процедуры CAFE.
; Процедура        TEA
TEA:   . WORD     ^M<R4, R5>    ; Сохранить регистры R4, R5
       ...
                                ; Тело процедуры TEA.
       RET                      ; Выход из процедуры TEA.
```

3.7. ИНСТРУКЦИИ РАБОТЫ С ОЧЕРЕДЯМИ

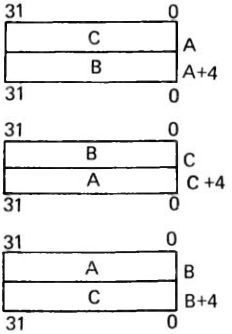
Эти инструкции (табл. 3.7) обеспечивают основные операции для работы с очередями: занесение и удаление элементов из очереди. Рассмотрим более подробно динамику их работы. Как уже отмечалось в гл. 2, пустая очередь определяется своим заголовком по адресу «А» следующим образом:



Если элемент по адресу «В» заносится в пустую очередь, то очередь будет иметь вид:



Если элемент по адресу «С» заносится в начало очереди, то очередь будет выглядеть следующим образом:



Операции с началом или концом очереди можно выполнять без каких-либо ограничений. Однако если операции осуществ-

Таблица 3.7

ИНСТРУКЦИИ РАБОТЫ С ОЧЕРЕДЯМИ

Мнемоника	Код	Название и формат	Описание
INSQHI	5C	Занесение элемента в начало очереди (с блокировкой памяти) INSQHI ОП1, ОП2	Элемент ОП1 заносится в начало очереди вслед за заголовком очереди ОП2
INSQTI	5D	Занесение элемента в конец очереди (с блокировкой памяти) INSQTI ОП1, ОП2	Элемент ОП1 заносится в конец очереди, заголовок которой определяется ОП2
INSQUE	0E	Занесение элемента в очередь INSQUE ОП1, ОП2	Элемент ОП1 заносится в очередь вслед за указанным элементом ОП2
REMQHI	5E	Удаление элемента из начала очереди (с блокировкой памяти) REMQHI ОП1, ОП2	Удаляется элемент из начала очереди, заголовок которой определяется в ОП1. Адрес удаленного элемента запоминается в ОП2
REMQTI	5F	Удаление элемента из конца очереди (с блокировкой памяти) REMQTI ОП1, ОП2	Удаляется элемент из конца очереди, заголовок которой определяется в ОП1. Адрес удаленного элемента запоминается в ОП2
REMQUE	0F	Удаление элемента из очереди REMQUE ОП1, ОП2	Из очереди удаляется элемент, определяемый ОП1. Адрес удаленного элемента заносится в ОП2

Примечание. Блокировка памяти производится для того, чтобы на время выполнения инструкции запретить другие операции с данной очередью.

ляются в произвольном месте очереди, следует соблюдать осторожность.

В случае если несколько процессов работают с одной очередью, то операции, которые выполняются в начале или в конце очереди, всегда будут верными, так как всегда сохраняется заголовок очереди, и соответствующие операции блокируют работу с этой очередью других процессов даже в мультипроцессорной системе. Операции в любом месте очереди зависят от конкретных имеющихся элементов и могут оказаться неверными, если другой процесс в то же время производит операции с этой очередью.

Пример.

```
; В примере используются инструкции
; работы с очередями REMQHI, REMQUE, INSQUE.
; *Область данных*
HEADER:                         ; Заголовок очереди.
        . BLKL    100
ADR—1:  . LONG    0             ; Рабочий адрес.
ADR—2:  . LONG    0             ; Рабочий адрес.
NEW:    . LONG    55            ; Новый элемент.

; *Область инструкций*
; Удалить элемент из начала очереди
; и записать его по адресу ADR—1.
        REMQHI @ HEADER, ADR—1
; Удалить элемент из конца очереди
; и записать его по адресу ADR—2.
        REMQUE @ HEADER, ADR—2
; Поставить новый элемент в начало очереди.
        INSQUE   NEW, HEADER
```

3.8. ИНСТРУКЦИИ ДЛЯ РАБОТЫ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Инструкции данной группы (табл. 3.8) работают с числами с плавающей запятой четырех форматов: два стандартных, которые использовались в ранних 16-разрядных моделях СМ ЭВМ (это формат F и D) и два расширенных формата (форматы G и H).

При работе с набором инструкций этой группы следует быть более внимательным, чем в случае использования других инструкций. Прежде всего в ряде случаев необходимо проверять входные операнды на допустимые типы. В частности, простым способом проверки является выполнение инструкции пересылки или тестирования входных операндов в формате с плавающей запятой.

Таблица 3.8

ИНСТРУКЦИИ ДЛЯ РАБОТЫ С ЧИСЛАМИ
С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Мнемоника	Код	Название и формат	Описание
ADDF2 ADDD2 ADDG2 ADDH2	40 60 40FD 60FD	Сложение чисел с плавающей запятой (двухоперандное) ADD [F, D, G, H] 2 ОП1, ОП2	Операнды ОП1 и ОП2 вкладываются, сумма запоминается по адресу второго операнда
ADDF3 ADDD3 ADDG3 ADDH3	41 61 41FD 61FD	Сложение чисел с плавающей запятой (трехоперандное) ADD [F, D, G, H] 3 ОП1, ОП2, ОП3	Операнды ОП1 и ОП2 складываются, сумма запоминается по адресу третьего операнда
CLRF CLRD CLRG CLRH	D4 7C 7CFD	Очистка числа с плавающей запятой CLR [F, D, G, H] ОП1	По адресу операнда ОП1 засылается 0
CMPF CMPD CMPG CMPH	51 71 51FD 71FD	Сравнение чисел с плавающей запятой CMP [F, D, G, H] ОП1, ОП2	Операнды ОП1 и ОП2 сравниваются. Результатом является установка кодов условий: N=1, если ОП1 < ОП2; Z=1, если ОП1=ОП2; V=0; C=0
CVTBF CVTBD CVTBG CVTBH	4C 6C 4CFD 6CFD	Преобразование байта в формат числа с плавающей запятой CVTB [F, D, G, H] ОП1, ОП2	Операнд ОП1 из формата байта преобразуется в формат числа с плавающей запятой. Результат запоминается в ОП2
CVTWF CVTWD CVTWG CVTWH	4D 6D 4DFD 6DFD	Преобразование слова в формат числа с плавающей запятой CVTW [F, D, G, H] ОП1, ОП2	Операнд ОП1 из формата слова преобразуется в формат числа с плавающей запятой. Результат запоминается в ОП2
CVTLF CVTLD CVTLG CVTLH	4E 6E 4EFD 6EFD	Преобразование длинного слова в формат числа с плавающей запятой CVTL [F, D, G, H] ОП1, ОП2	Операнд ОП1 из формата длинного слова преобразуется в формат числа с плавающей запятой. Результат запоминается в ОП2
CVTFB CVTDB CVTGB CVTHB	48 68 48FD 68FD	Преобразование числа с плавающей запятой в формат байта CVT [F, D, G, H] ОП1, ОП2	Операнд ОП1 из формата числа с плавающей запятой, который указан в инструкции, преобразуется в формат слова. Результат запоминается в ОП2
CVTFW CVTDW CVTGW CVTHW	49 69 49FD 69FD	Преобразование числа с плавающей запятой в формат слова CVT [F, D, G, H] W ОП1, ОП2	Операнд ОП1 из формата числа с плавающей запятой, который указан в инструкции, преобразуется в формат слова. Результат запоминается в ОП2

Мнемоника	Код	Название и формат	Описание
CVTFL CVTDL CVTGL CVTHL	4A 6A 4AFD 6AFD	Преобразование числа с плавающей запятой в формат длинного слова CVT [F, D, G, H] L ОП1, ОП2	Операнд ОП1 из формата числа с плавающей запятой преобразуется в формат длинного слова. Результат запоминается в ОП2
CVTFD CVTFG CVTFH	56 99FD 98FD	Преобразование числа с плавающей запятой формата F в другой формат с плавающей запятой CVTF [D, G, H] ОП1, ОП2	Число с плавающей запятой формата F (ОП1) преобразуется в число с плавающей запятой указанного формата (D, G или H). Результат запоминается в ОП2
CVTDF CVTDH	76 32	Преобразование числа с плавающей запятой формата D в другой формат с плавающей запятой CVTD [F, H] ОП1, ОП2	Число с плавающей запятой формата D (ОП1) преобразуется в число с плавающей запятой указанного формата (F или H). Результат запоминается в ОП2
CVTHF CVTHD CVTHG	F6FD F7FD 76FD	Преобразование числа с плавающей запятой формата H в другой формат с плавающей запятой CVTH [F, D, G] ОП1, ОП2	Число с плавающей запятой формата H (ОП1) преобразуется в число с плавающей запятой указанного формата (F, D или G). Результат запоминается в ОП2
CVTRFL CVTRDL CVTRGL CVTRHL	4B 6B 4BFD 6BFD	Преобразование округленных форматов с плавающей запятой в длинное слово CVTR [F, D, G, H] L ОП1, ОП2	Округленное число с плавающей запятой (ОП1) преобразуется в длинное слово. Результат запоминается в ОП2
DIVF2 DIVD2 DIVG2 DIVH2	46 66 46FD 66FD	Деление чисел с плавающей запятой (двухоперандное) DIV [F, D, G, H] 2 ОП1, ОП2	Делимое (ОП2) делится на делитель (ОП1) и частное запоминается в ОП2
DIDF3 DIVD3 DIVG3 DIVH3	47 67 47FD 67FD	Деление чисел с плавающей запятой (трехоперандное) DIV [F, D, G, H] 3 ОП1, ОП2, ОП3	Делимое (ОП2) делится на делитель (ОП1) и частное запоминается в ОП3
EMODF EMODD EMODG EMODH	54 74 54FD 74FD	Расширенное умножение чисел с плавающей запятой с выделением целой части EMOD [F, D, G, H] ОП1, ОП2, ОП3, ОП4, ОП5	К множителю (ОП1) присоединяется расширение (ОП2). Множимое (ОП3) умножается на расширенный множитель. Целая часть результата запоминается в ОП4, а округленная дробная часть запоминается в ОП5

Мнемоника	Код	Название и формат	Описание
MNEGF MNEGD MNEGG MNEGH	52 72 52FD 72FD	Пересылка числа с плавающей запятой со сменой знака MNEG [D, F, G, H] ОП1, ОП2	Содержимое ОП1, взятое с противоположным знаком, запоминается в ОП2
MOVF MOVD MOVG MOVH	50 70 50FD 70FD	Пересылка числа с плавающей запятой MOV [F, D, G, H] ОП1, ОП2	Содержимое ОП1 пересылается в ОП2
MULF2 MULD2 MULG2 MULH2	44 64 44FD 64FD	Умножение чисел с плавающей запятой (двухоперандное) MUL [F, D, G, H] 2 ОП1, ОП2	Множимое (ОП2) умножается на множитель (ОП1), произведение запоминается в ОП2
MULF3 MULD3 MULG3 MULH3	45 65 45FD 65FD	Умножение чисел с плавающей запятой (трехоперандное) MUL [F, D, G, H] 3 ОП1, ОП2, ОП3	Множимое (ОП2) умножается на множитель (ОП1), произведение запоминается в ОП3
POLYF POLYD POLYG POLYH	55 75 55FD 75FD	Вычисление полинома POLY [F, D, G, H] ОП1, ОП2, ОП3	Вычисляется полином методом Хорнера; ОП1 — аргумент; ОП2 — номер старшего коэффициента; ОП3 — адрес таблицы коэффициентов. Результат запоминается в регистрах R0—R1 для инструкций POLYD, POLYG и в регистрах R0—R3 для инструкции POLYH
SUBF2 SUBD2 SUBG2 SUBH2	42 62 42FD 72FD	Вычитание чисел с плавающей запятой (двухоперандное) SUB [F, D, G, H] 2 ОП1, ОП2	Вычитаемое (ОП1) вычитается из уменьшаемого (ОП2). Результат запоминается в ОП2
SUBF3 SUBD3 SUBG3 SUBH3	43 63 43FD 73FD	Вычитание чисел с плавающей запятой (трехоперандное) SUB [F, D, G, H] 2 ОП1, ОП2, ОП3	Вычитаемое (ОП1) вычитается из уменьшаемого (ОП2). Результат запоминается в ОП3
TSTF TSTD TSTG TSTH	53 73 53FD 73FD	Проверка TST [F, D, G, H] ОП1	Проверяется содержимое ОП1 и устанавливаются коды условий: N=1, если ОП1 < 0; Z=1, если ОП1=0; V=0; C=0

Для достижения большего быстродействия при работе с инструкциями данной группы накладываются некоторые ограничения на сочетание режимов адресации, используемых в одной инструкции. К таким сочетаниям относится логически противоречивое использование данных одновременно в качестве операнда с плавающей запятой и адреса.

При использовании операций над числами с плавающей запятой важное значение имеют вопросы точности, в частности округления. В СМ1700 используется округление по следующему алгоритму.

Прибавить 1 к разряду округления и выполнить распространение переноса (если он имеется). После округления может потребоваться повторная нормализация. Если это происходит, новый разряд округления будет равен 0. Соответствие между усеченным, округленным и истинным (бесконечной точности) результатами устанавливается таким образом:

1) если сохраняемый результат точен, то:
«округленная величина = усеченная величина = истинная величина»;

2) если сохраняемый результат не точен, то его величина:

- всегда меньше истинного результата при усечении;
- всегда меньше истинного результата при округлении, если бит округления равен 0;
- больше истинного результата при округлении, если бит округления равен 1.

Пример.

```
; В примере используются инструкции работы с
; плавающей запятой SUBF3, MNEGF, CVTREL.
; Вычисляется абсолютное значение
; разности между шестью парами чисел
; с плавающей запятой, которые хранятся
; в таблицах TABL1 и TABL2. Полученные
; значения разностей округляются и помещаются
; в таблицу TABL3.
; *Область данных*
TABL1: . FLOAT   99.9, 55.5, 30.22, 24.3, 22.16, 10.3
TABL2: . FLOAT   88.8, 33.3, 16.16, 21.5, 17.9, —15.5
TABL3: . BLKL    6

; *Область инструкций*
START:    . WORD   0              ; Слово входной маски.
          MOVL     #6, R0         ; Задать количество элементов таб-
                                  ; лицы.
          MOVAL    TABL1, R1      ; Адрес таблицы TABL1 поместить в
                                  ; регистр R1.
          MOVAL    TABL2, R2      ; Адрес таблицы TABL2 поместить в
                                  ; регистр R2.
          MOVAL    TABL3, R3      ; Адрес таблицы разностей TABL3 по-
                                  ; местить в регистр R3.
SUBFLOAT:
          SUBF3    (R1)+, (R2)+, R3 ; Найти разность пары чисел, ес-
          BGEQ     CONVERT        ; ли она не положительна, то
          MNEGF    (R3), (R3)     ; найти абсолютную величину.
```

```
CVTREL   (R3), (R3)        ; Округлить разность, если эле-
SOBGTR   R0, SUBFLOAT      ; менты таблиц не кончились, то
                           ; продолжить обработку.
⊙EXIT_S
. END    START
```

3.9. ИНСТРУКЦИИ РАБОТЫ С СИМВОЛЬНЫМИ СТРОКАМИ

При исполнении каждой из инструкций этой группы (табл. 3.9) используются регистры общего назначения от R0 до R5. Количество задействованных регистров определяется конкретной инструкцией и может быть R0 — R1, R0 — R3 или R0 — R5. В соответствующих регистрах создается управляющий блок, где хранится:

- информация об обрабатываемых строках — длина и адрес строк (формируется автоматически при наличии операции);
- состояние во время выполнения инструкции. В этот промежуток времени проверяется наличие прерывания. Если оно есть, инструкция может быть прервана с соответствующей отметкой в управляющем блоке;
- результат выполнения инструкции (например, R3 = адресу искомого байта в строке). После завершения инструкции регистры становятся программно доступны и могут быть использованы для тестирования.

Таким образом, при выполнении любой из инструкций данной группы содержимое отмеченных регистров меняется. Это следует помнить и стараться регистры с младшими номерами использовать на короткий срок для хранения промежуточных результатов.

Пример 1.

```
; Переслать строку, имеющую адрес STR1 и длину 12 байт,
; в строку с адресом STR2 и длиной 24 байта.
; В качестве заполнителя использовать пробел.
; *Область данных*

STR1:  . ASCII   /весна* *осень/
STR2:  . BLKB    24

; *Область инструкций*

       MOVC5    #128,@ STR1,  #^A/  /,  #24, @ STR2
```

Пример 2.

```
; В строке символов STR, длина которой 32,
; найти первый символ звездочка (*)
; и запомнить его адрес в A—STAR.
; *Область данных*
```

Таблица 3.9

ИНСТРУКЦИИ РАБОТЫ С СИМВОЛЬНЫМИ СТРОКАМИ

Мнемоника	Код	Название и формат	Описание
CMPC3	29	Сравнение строк (трехоперандное) CMPC3 ОП1, ОП2, ОП3	Строка 1, определенная длиной ОП1 и адресом ОП2, сравнивается со строкой 2, определенной длиной ОП1 и адресом ОП3. Сравнение продолжается до обнаружения несовпадения или проверки всех байтов в строках. Коды условий устанавливаются по результатам последнего сравнения байта строки 1 (бс1) с байтом строки 2 (бс2): N=1, если (бс1) < (бс2); Z=1, если (бс1) = (бс2); V=0; C=1, если (бс1) < (бс2)
CMPC5	2D	Сравнение строк (пятиоперандное) CMPC5 ОП1, ОП2, ОП3, ОП4, ОП5	Строка 1, определенная длиной ОП1 и адресом ОП2, сравнивается со строкой 2, определенной длиной ОП4 и адресом ОП5. Если строки имеют разную длину, то более короткая дополняется до длины другой строки посредством заполнителя ОП3. Результатом являются коды условий, устанавливаемые также, как и для инструкции CMPC3
LOCC	3A	Поиск символа LOCC ОП1, ОП2, ОП3	Производится поиск символа ОП1 в строке, определенной длиной ОП2 и адресом ОП3. Если символ найден, то регистр R0 содержит число оставшихся байтов в строке, регистр R1 содержит адрес найденного символа. Если символ не найден, то регистр R1 содержит адрес байта, следующего за строкой
MATCHC	39	Поиск подстроки символов MATCHC ОП1, ОП2, ОП3, ОП4	В строке, определенной длиной ОП3 и адресом ОП4, ищется подстрока, определенная длиной ОП1 и адресом ОП2. Если подстрока найдена, то регистр R0=0, регистр R1 содержит адрес байта, следующего за найденной подстрокой. Если подстрока не найдена, то регистр R0 содержит число байтов подстроки, а регистр R1 содержит ее адрес

Продолжение

Мнемоника	Код	Название и формат	Описание
MOVC3	28	Пересылка строки (трехоперандная) MOVC3 ОП1, ОП2, ОП3	Строка, определенная длиной ОП1 и адресом ОП2, полностью пересылается в поле начиная с адреса ОП3
MOVC5	2C	Пересылка строки (пятиоперандная) MOVC5 ОП1, ОП2, ОП3, ОП4, ОП5	Строка, определенная длиной ОП1 и адресом ОП2, пересылается по адресу ОП5 и приобретает длину, указанную в ОП4. Если длина ОП1 меньше длины ОП4, то старшие байты не пересылаются. Если длина одинаковая, то пересылаются все байты строки
MOVTC	2E	Пересылка перекодированной строки MOVTC ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Из строки 1, определенной длиной ОП1 и адресом ОП2, формируется посредством перекодировки строка 2, заданная длиной ОП5 и адресом ОП6. Перекодировка осуществляется через таблицу с адресом ОП4 длиной 256 байт. В соответствии с номером байта строки 1 из таблицы выбирается байт с таким же номером и пересылается в строку 2. Если длина строки 2 больше длины строки 1, то старшие байты замещаются заполнителем ОП3
MOVTUC	2F	Пересылка перекодированной строки до указанного символа MOVTUC ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Из строки 1, определяемой длиной ОП1 и адресом ОП2, формируется строка 2, задаваемая длиной ОП5 и адресом ОП6. Адрес таблицы перекодировки, размер которой 256 байт, содержится в ОП4. В соответствии с номером каждого байта строки 1 выбирается байт с таким же номером из таблицы перекодировки и пересылается в строку 2. Перекодировка продолжается до тех пор, пока не встретится байт, равный символу ОП3, или пока не закончится одна из строк

Продолжение

Мнемоника	Код	Название и формат	Описание
SCANC	2A	Сканирование символов в строке SCANC ОП1, ОП2, ОП3, ОП4	Байты строки, определяемой длиной ОП1 и адресом ОП2, последовательно используются как индексы таблицы, адрес которой указан в ОП3. Размер таблицы — 256 байт. Выбранный из таблицы байт логически умножается на маску ОП4. Операция продолжается до тех пор, пока не будет найден ненулевой результат (Z=0) или пока все байты строки не будут просмотрены (Z=1). После выполнения инструкций: регистр R0=числу оставшихся байтов в строке, если найден ненулевой результат. R1=адресу найденного байта, иначе — R1=адресу байта, следующего за строкой
SKPC	38	Пропуск символа SKPC ОП1, ОП2, ОП3	Символ, указанный в ОП1, сравнивается с символами строки, определяемой длиной ОП2 и адресом ОП3, до тех пор, пока будет обнаружено несовпадение (Z=0) или до окончания строки (Z=1). После выполнения инструкции: регистр R0=числу байтов, оставшихся в строке, если найден несовпадающий символ, иначе R0=0; регистр R1=адресу несовпадающего символа, иначе R1=адресу байта, следующего за строкой
SPANC	2B	Пропуск символов, предшествующих определенным SPANC ОП1, ОП2, ОП3, ОП4	Байты строки, определяемой длиной ОП1 и адресом ОП2, последовательно используются как индексы таблицы, адрес которой указан в ОП3. Размер таблицы — 256 байт. Выбранный из таблицы байт логически умножается на маску ОП4. Операция продолжается до тех пор, пока не получен нулевой результат умножения (Z=0) или до окончания строки (Z=1). После выполнения

Продолжение

Мнемоника	Код	Название и формат	Описание
			инструкции: регистр R0=числу байтов, оставшихся в строке, если обнаружен нулевой результат умножения, иначе регистр R0=0; регистр R1=адресу байта, вызвавшего нулевой результат умножения, иначе регистр R1=адресу байта, следующего за строкой
CRC	0B	Контроль на циклическую избыточность CRC ОП1, ОП2, ОП3, ОП4	Значения CRC для потока данных вычисляются с помощью начального значения полинома CRC. Поток данных определяется длиной строки ОП3 и ее адресом ОП4. Начальное значение CRC ОП2 обычно равно нулю. Таблица, адрес которой указан в ОП1, содержит специальный полином CRC. Размер таблицы — 16 длинных слов. После выполнения инструкции: регистр R0=конечное значение CRC; регистр R1=адресу байта, следующего за потоком данных

```
STR:      . ASCII   /# # # огонек * * * * семена * * * * * экран # # #/
A—STR:    . LONG    0   ; Поле адреса строки STR.
A—STAR:   . LONG    0   ; Поле адреса символа (*).

*Область инструкций*

          MOVAL     STR, A—STR          ; Запомнить адрес
                                        ; строки в поле A—STR.
          SKPC      #^A/*/, #32, @ A—STR ; Поиск символа (*).
          TSTL      R0                  ; Если регистр R0=0,
          BEQL      END                 ; то в строке нет симво-
                                        ; ла (*), тогда переход
                                        ; на метку END.
          MOVL      R1, A—STAR          ; Поместить адрес перво-
                                        ; го символа (*) в
                                        ; A—STAR.
```

3.10. ИНСТРУКЦИИ РАБОТЫ С УПАКОВАННЫМИ ДЕСЯТИЧНЫМИ СТРОКАМИ

Инструкции этой группы (табл. 3.10) так же, как и инструкции работы с символьными строками, используют регистры об-

Таблица 3.10

ИНСТРУКЦИИ РАБОТЫ С УПАКОВАННЫМИ ДЕСЯТИЧНЫМИ СТРОКАМИ

Мнемоника	Код	Название и формат	Описание
ADDP4	20	Сложение упакованных десятичных строк (четырехоперандное) ADDP4 ОП1, ОП2, ОП3, ОП4	Строка, определяемая длиной ОП1 и адресом ОП2, складывается со строкой, определяемой длиной ОП3 и адресом ОП4. Длина строки-результата запоминается в ОП3, а адрес — в ОП4
ADDP6	21	Сложение упакованных десятичных строк (шестиоперандное) ADDP6 ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Строка, определяемая длиной ОП1 и адресом ОП2, складывается со строкой, определяемой длиной ОП3 и адресом ОП4. Длина строки-результата запоминается в ОП5, а адрес — в ОП6
ASHP	F8	Арифметический сдвиг упакованных десятичных строк с округлением ASHP ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Строка, определяемая длиной ОП2 и адресом ОП3, сдвигается на число разрядов, указанное в счетчике ОП1. Положительное значение указывает на сдвиг влево (умножение), отрицательное — вправо (деление). Если строка сдвигается вправо, то старшая из цифр, которые теряются при сдвиге, будет округлена в соответствии с заданным коэффициентом округления ОП4. Длина строки-результата запоминается в ОП5, а адрес — в ОП6
CMPP3	35	Сравнение упакованных десятичных строк (трехоперандное) CMPP3 ОП1, ОП2, ОП3	Строка 1 (стр1), определяемая длиной ОП1 и адресом ОП2, сравнивается со строкой 2 (стр2), определяемая длиной ОП1 и адресом ОП3. В результате выполнения инструкции устанавливаются коды условий: N=1, если (стр1) < (стр2); Z=1, если (стр1) = (стр2); V=0; C=0
CMPP4		Сравнение упакованных десятичных строк (четырехоперандное)	Строка 1 (стр1), определяемая длиной ОП1 и адресом ОП2, сравнивается со

Продолжение

Мнемоника	Код	Название и формат	Описание
		CMPP4 ОП1, ОП2, ОП3, ОП4	строкой 2 (стр2), определяемой длиной ОП3 и адресом ОП4. В результате выполнения инструкций устанавливаются коды условий: N=1, если (стр1) < (стр2); Z=1, если (стр1) = (стр2); V=0; C=0
CVTLP	F9	Преобразование длинного слова в упакованную десятичную строку CVTPL ОП1, ОП2, ОП3	Содержимое длинного слова ОП1 преобразуется в строку, определяемую длиной ОП2 и адресом ОП3
CVTPL	36	Преобразование упакованной десятичной строки в длинное слово CVTPL ОП1, ОП2, ОП3	Строка, определяемая длиной ОП1 и адресом ОП2, преобразуется в длинное слово, которое запоминается по адресу ОП3
CVTPS	08	Преобразование упакованной десятичной строки в числовую с ведущим знаком CVTPS ОП1, ОП2, ОП3, ОП4	Строка, определяемая длиной ОП1 и адресом ОП2, преобразуется в числовую строку с выделенным ведущим знаком. Длина числовой строки-результата запоминается в ОП3, а адрес — в ОП4
CVTPT	24	Преобразование упакованной десятичной строки в числовую строку CVTPT ОП1, ОП2, ОП3, ОП4, ОП5	Строка, определяемая длиной ОП1 и адресом ОП2, преобразуется в числовую строку с использованием таблицы в 256 байт, адрес которой содержится в ОП3. Длина числовой строки-результата запоминается в ОП4, адрес — в ОП5
CVTSP	09	Преобразование числовой строки с ведущим знаком в упакованную десятичную строку CVTSP ОП1, ОП2, ОП3, ОП4	Числовая строка, определяемая длиной ОП1 и адресом ОП2, преобразуется в строку. Длина строки-результата запоминается в ОП3, адрес — в ОП4
CVTTP	26	Преобразование числовой строки в упакованную десятичную строку	Числовая строка, определяемая длиной ОП1 и адресом ОП2, преобразуется в строку с использованием таблицы в

Мнемоника	Код	Название и формат	Описание
		CVTTP ОП1, ОП2, ОП3, ОП4, ОП5	256 байт. Длина строки-результата запоминается в ОП3, адрес — в ОП5
DIVP	27	Деление упакованных десятичных строк DIVP ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Строка, определяемая длиной ОП3 и адресом ОП4, делится на строку, задаваемую длиной ОП1 и адресом ОП2. Длина строки-результата запоминается в ОП5, адрес — в ОП6
MOVP	34	Пересылка упакованных десятичных строк MOVP ОП1, ОП2, ОП3, ОП4	Строка, определяемая длиной ОП1 и адресом ОП2, пересылается в строку, задаваемую длиной ОП3 и адресом ОП4
MULP	25	Умножение упакованных десятичных строк MULP ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Строка, определяемая длиной ОП3 и адресом ОП4, умножается на строку, задаваемую длиной ОП1 и адресом ОП2. Длина строки-результата запоминается в ОП5, адрес — в ОП6
SUBP4	22	Вычитание упакованных десятичных строк (четырехоперандное) SUBP4 ОП1, ОП2, ОП3, ОП4	Из строки, определяемой длиной ОП3 и адресом ОП4, вычитается строка, задаваемая длиной ОП1 и адресом ОП2. Длина строки-результата запоминается в ОП3, а адрес — в ОП4
SUBP6	23	Вычитание упакованных десятичных строк (шестиоперандное) SUBP4 ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	Из строки, определяемой длиной ОП3 и адресом ОП4, вычитается строка, задаваемая длиной ОП1 и адресом ОП2. Длина строки-результата запоминается в ОП5, а адрес — в ОП6
EDITPC		Редактирование упакованных десятичных строк EDITPC ОП1, ОП2, ОП3, ОП4	Строка, определяемая длиной ОП1 и адресом ОП2, редактируется с помощью шаблона, адрес которого содержится в ОП3, в символьную строку. Символьная строка-результат запоминается по адресу, определенному в ОП4. (Подробно инструкция EDITPC описывается в конце данного раздела.)

щего назначения R0 — R5 с аналогичной целью. Однако результаты выполнения инструкций, получаемые в регистрах, интерпретируются по-иному в соответствии с конкретной операцией и используемыми типами данных. В табл. 3.10 описываются лишь основные данные по каждой инструкции и не рассматриваются вопросы, связанные с десятичным переполнением, нулевым результатом, и т. п.

Пример.

```
; В примере используются инструкции ADDP4, MULP, DIVP
; работы с десятичными упакованными строками.
; *Область данных*
ALPHA:  . PACKED   0212    ; Длина строки=4.
BETA:   . PACKED   14      ; Длина строки=2.
GAMMA:  . PACKED   —132    ; Длина строки=3.
SIGMA:  . PACKED   00396   ; Длина строки=5.
; *Область инструкций*
; ALPHA+GAMMA→  ALPHA
        ADDP4     #3, GAMMA, #4, ALPHA
; BETA*GAMMA→   ALPHA
        MULP      #2, BETA, #3, GAMMA, #4, ALPHA
; SIGMA/GAMMA→  BETA
        DIVP      #3, GAMMA, #5, SIGMA, #2, BETA
```

Инструкция редактирования EDITPC занимает особое место в группе инструкций работы с упакованными десятичными строками, что объясняется ее богатыми функциональными возможностями. Алгоритм, реализованный в инструкции EDITPC, напоминает работу оператора FORMAT языка Фортран. Эта инструкция может быть успешно использована при форматировании данных, предназначенных для вывода. Она позволяет подавлять незначащие нули, ставить знак перед первой значащей цифрой, вводить десятичную точку и т. д. Формат инструкции:

EDITPC ОП1, ОП2, ОП3, ОП4

где ОП1 — длина исходной строки;
ОП2 — адрес исходной строки;
ОП3 — адрес шаблона;
ОП4 — адрес строки-результата.

Описание.

Исходная упакованная десятичная строка редактируется с помощью операторов шаблона в символьную строку.

Шаблон состоит из последовательности операторов, которая всегда заканчивается оператором конца (E0⊙END). По выполняемым функциям операторы шаблона делятся на пять типов (табл. 3.11): вставки, пересылки, фиксирования, загрузки и управления. Каждый из операторов может иметь не более од-

Таблица 3.11

ОПЕРАТОРЫ ШАБЛОНА

Тип и мнемоника	Операнд	Описание
Вставить:		
EO⊙INSERT	Символ	Вставить заполнитель на место незначащей цифры
EO⊙STORE—SIGN	—	Вставить знак
EO⊙FILL	Счетчик	Вставить заполнитель
Переслать:		
EO⊙MOVE	Счетчик	Переслать цифру или заполнитель, если цифра незначащая
EO⊙FLOAT	Счетчик	Переслать цифру с распространением знака
EO⊙END—FLOAT	—	Распространение знака закончить
Фиксировать:		
EO⊙BLANK—ZERO	Длина	Если значение упакованной строки равно 0, то в последнем байте строки помещается заполнитель
EO⊙REPLACE—SIGN	Длина	Если значение упакованной строки равно 0, то в байт поместить заполнитель
Загрузить:		
EO⊙LOAD—FILL	Символ	Загрузить заполнитель ПЗП
EO⊙LOAD—SIGN	Символ	Загрузить знак в ПЗН
EO⊙LOAD—PLUS	Символ	Загрузить знак в ПЗН, если упакованная строка имеет положительный знак
EO⊙LOAD—MINUS	Символ	Загрузить знак в ПЗН, если упакованная строка имеет отрицательный знак
Управление:		
EO⊙SET—SIGNIF	—	Установить признак значимости
EO⊙CLEAR—SIGNIF	—	Очистить признак значимости
EO⊙ADJUST—INPUT	Длина	Установить длину символьной строки-результата
EO⊙END	—	Закончить редактирование

ного операнда. В качестве операнда могут быть использованы следующие величины:

- символ — один из символов кода КОИ-8;
- счетчик — целочисленное значение счетчика повторений в диапазоне от 1 до 15;
- длина — целочисленное значение длины строки в диапазоне от 1 до 255.

Инструкция EDITPC выполняется с помощью двух специальных полей, размещаемых в регистре общего назначения R2:

- поля символа-заполнителя (ПЗП), которое занимает разряды 0—7;
- поля символа знака (ПЗН), которое занимает разряды 8—15.

Пример.

```
; Пример шаблона, состоящего из 10 операторов
; редактирования.
MASKA:EO⊙ADJUST—INPUT  8   ; Установить длину строки-результата.
      EO⊙LOAD—FILL     *   ; Загрузить заполнитель (*) в ПЗП.
      EO⊙MOVE          3   ; Переслать 3 цифры.
      EO⊙INSERT        ,   ; Вставить запятую (,).
      EO⊙MOVE          3   ; Переслать 3 цифры.
      EO⊙SET—SIGNIF        ; «Обнулить» оставшиеся позиции.
      EO⊙INSERT        .   ; Вставить десятичную точку (.).
      EO⊙MOVE          2   ; Переслать 2 цифры.
      EO⊙BLANK—ZERO    2   ;
      EO⊙END               ; Конец редактирования.
```

Когда операторы шаблона MASKA будут выполнены в контексте инструкции EDITPC над десятичными упакованными строками, то полученные символьные строки будут иметь представленный ниже вид:

```
УПАКОВАННАЯ ДЕСЯТИЧНАЯ СТРОКА          СИМВОЛЬНАЯ СТРОКА-РЕЗУЛЬТАТ
.

8 7 6 5 4 3 2 1                        8 7 6,5 4 3.2 1
  7 6 5 4 3 2 1                        * 7 6,5 4 3.2 1
    6 5 4 3 2 1                        * * 6,5 4 3.2 1
      5 4 3 2 1                        * * * 5 4 3.2 1
        4 3 2 1                        * * * * 4 3.2 1
          3 2 1                        * * * * * 3.2 1
            2 1                        * * * * * * .2 1
              1                        * * * * * * .0 1
              0                        * * * * * * * *
```

3.11. ИНСТРУКЦИИ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ

Инструкции этой группы (табл. 3.12), как правило, относятся к категории привилегированных, т. е. требуют более высоких привилегий, чем обычный режим исполнения. Некоторые из них доступны обычным пользователям, например инструкции BISPSW, BICPSW, MOVPSL и т. д. Другие инструкции, например MFPR и MTPR, которые работают с одним из 30 привилегированных регистров процесса, требуют дополнительных привилегий.

Таблица 3.12

ИНСТРУКЦИИ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ

Мнемоника	Код	Название и формат	Описание
BICPSW	B9	Очистка разрядов слова состояния процессора BICPSW ОП1	Содержимое слова состояния процессора (PSW) логически умножается на инвертированное значение маски (ОП1). Результат заносится в PSW. Разряды 8—15 маски ОП1 должны быть равны 0
BISPSW	B8	Установка разрядов слова состояния процессора BICPSW ОП1	Содержимое слова состояния процессора (PSW) логически складывается с маской (ОП1). Результат заносится в PSW. Разряды 8—15 маски ОП1 должны быть равны 0
MOVPSL	DC	Пересылка длинного слова состояния процессора MOVPSL ОП1	Содержимое длинного слова состояния процессора (PSL) пересылается по адресу ОП1
BPT	03	Ловушка отладчика	Происходит прерывание в контрольной точке
CHMK CHME CHMS CHMU	BC BD BE BF	Изменение режима процесса CHM [K, E, S, U] ОП1	Изменяется режим процесса, код которого указан в ОП1. Тип режима указан символом в мнемонике инструкции: K — ядро; E — управление; S — супервизор; U — пользователь
HALT	00	Останов HALT	Если процесс выполняется в режиме ядра, то происходит останов процессора, в противном случае возникает ошибка
INDEX INDEX ОП1, ОП2, ОП3, ОП4, ОП5, ОП6	0A	Вычисление и проверка индекса на диапазон возможных значений	Содержимое ОП5 (входной индекс) складывается с содержимым ОП1. Полученная сумма умножается на содержимое ОП4 (размер) и заносится по адресу ОП6 (выходной индекс). Если содержимое ОП1 меньше ОП2 (нижняя граница индексного диапазона) или больше ОП3 (верхняя граница индексного диапазона), то происходит прерывание по нарушению индексного диапазона

Продолжение

Мнемоника	Код	Название и формат	Описание
LDPCTX	06	Загрузка контекста процесса LDPCTX	Загружается контекст процесса
SVPCTX	07	Сохранение контекста процесса SVPCTX	В режиме ядра сохраняется контекст процесса
MFPR	DB	Пересылка из привилегированного регистра процессора MFPR ОП1, ОП2	В режиме ядра содержимое привилегированного регистра, номер которого указан в ОП1, пересылается в ОП2
MTPR	DA	Пересылка в привилегированный регистр процессора MTPR ОП1, ОП2	В режиме ядра содержимое ОП1 пересылается в привилегированный регистр процессора, номер которого указан в ОП2
NOP	01	Отсутствие операции NOP	Не производится никакой операции
POPR	BA	Извлечение регистров из стека POPR ОП1	Содержимое регистров общего назначения, запомненных ранее в стеке, извлекается из стека и заносится в регистры, номера которых соответствуют разрядам, содержащим единицы в маске ОП1. Содержимое маски опрашивается от 0 до 14 разряда
PROBER PROBEW PROBE	0C 0D [R, W]	Проверка доступности операнда для чтения (записи) ОП1, ОП2, ОП3	В режиме, номер которого указан в ОП1, проверяется доступность чтения (PROBER) или записи (PROBEW) первого и последнего байтов области, определенной адресом (ОП2) и длиной ОП3. Промежуточные байты не проверяются. В случае доступности код условий Z=0, иначе Z=1
PUSHR	BB	Запись регистров в стек PUSHR ОП1	Содержимое регистров общего назначения, номера которых соответствуют установленным разрядам в маске ОП1, заносятся в стек в виде длинных слов. Содержимое маски опрашивается от 0 до 14-го разряда

Мнемоника	Код	Название и формат	Описание
REI	02	Возврат из прерывания или исключительной ситуации REI	Осуществляется возврат из прерывания или исключительной ситуации посредством извлечения из стека сохраненных PC и PSL
XFC	FC	Вызов специальной микропрограммной функции XFC	Расширяет систему инструкций за счет инструкций, определенных пользователем

Пример 1.

```
; Сохранить в стеке, а затем восстановить
; содержимое регистров R5, R6.
SAVE:     PUSHR    #^M<R5, R6>    ; Сохранить R5, R6.
            ...                   ; Инструкции, меняющие содер-
                                  ; жимое регистров R5, R6.
RESTORE:  POPR     #^M<R5, R6>    ; Восстановить R5, R6.
```

Пример 2.

```
; Сохранить длинное слово состояния процессора в PSL—SAVE.
; *Область данных*
PSL—SAVE:
          .LONG    0
; *Область инструкций*
SAVE:     MOVPSL   PSL—SAVE
```

Пример 3.

```
; Установить режим ядра для процедуры, специфицированной
; кодом 7.
          CHMK     #7
; Сохранить контекст процесса.
          SVPCTX
```

Глава 4
ОСНОВНЫЕ КОНСТРУКЦИИ МАКРОАССЕМБЛЕРА

Язык программирования Макроассемблер достаточно прочно вошел в состав программного обеспечения любой ЭВМ, заменив язык ассемблера. Основное отличие от этого языка заключается в расширенном использовании директив транслятора, а также макрокоманд, которые, если встречаются в исходном тексте программы, заменяются на последовательность машинных инструкций в соответствии с их макроопределениями.

По своей функциональной направленности макрокоманда напоминает обращение к подпрограмме, а макроопределение — ее описание с использованием языка «формальных» параметров. Однако если макроопределение всякий раз замещает макрокоманду при ее появлении в тексте программы, то подпрограмма присутствует в программе в одном экземпляре. Кроме того, обращение к подпрограмме и возврат к вызвавшей программе осуществляются на этапе выполнения программы и сопровождаются, как правило, реализацией вспомогательных операций (сохранение и восстановление регистров, передача параметров и т. д.), в то время как замена формальных параметров и возможная модификация текста макроопределения выполняются на этапе трансляции и не требуют дополнительных операций при использовании.

В Макроассемблере СМ1700 достаточно широко развиты оба подхода. Выбор того или иного определяется:

- достижением заданных показателей эффективности (память или быстродействие);
- обеспечением наглядности и удобства при сопровождении программы (особенно для комплексных программных средств);
- удобством программирования исходя из навыков или наклонностей при составлении программы.

Чаще всего при разработке сложных программных систем приходится учитывать все три возможных подхода и компромисс возможен лишь при умелом использовании возможностей языка.

В последующих глазах обсуждаются основные конструкции Макроассемблера, директивы и макрокоманды языка, а также особенности и способы их использования.

4.1. ФОРМАТ ИСХОДНОГО ПРЕДЛОЖЕНИЯ

Текст исходной программы состоит из последовательности исходных предложений (символьных строк). Макроассемблер последовательно обрабатывает эти строки, формируя при этом объектную программу и выполняя специальные действия этапа трансляции. Символьная строка содержит один оператор языка и может иметь длину до 132 символов. Размер строки за 80 символов не следует превышать, чтобы можно было бы отобразить одно предложение (вместе с двоичным кодом в памяти) на одной строке в файл листинга или на терминале.

Исходное продложение языка Макроассемблера, состоящее из четырех полей — метки, операции, операнда и комментария, имеет следующий формат:

```
метка: операция операнды  ; комментарий
```

Поле метки и поле комментария необязательны. Поле метки заканчивается двоеточием (:), а поле комментария начинается знаком «точка с запятой» (;).

Поле операции должно соответствовать формату инструкции, директивы или макрокоманды. Поле операнда определяет объект действия инструкции в соответствии с используемым режимом адресации операнда. Если операндов несколько, то они отделяются друг от друга запятой (,).

Поля предложения могут разделяться либо пробелом, либо знаком табуляции, но для лучшей наглядности рекомендуется форматировать поля знаком табуляции, как показано ниже.

Поле	Начальная колонка	Количество знаков табуляции для достижения колонки
Метки	1	0
Операции	9	1
Операнда	17	2
Комментария	41	5

Пример.

```
        CLRL    R0              ; Очистить регистр R0.
M1:     INCL    R0              ; Увеличить значение регистра R0 на 1.
M2:     ADDL3   (R3)+, R0, SYM  ; Получить значение в поле SYM.
        BLEQ    M1              ; Переход на метку M1, если результат меньше или равен 0.
        BRW     CONT            ; Безусловный переход на метку CONT.
```

Одно предложение может быть продолжено на несколько строк. При этом в качестве последнего значащего символа в конце строки, имеющей продолжение на следующей строке, следует указать знак дефиса «-».

Пример.

```
; Сохранить адрес таблицы адресов TABLE—ADDRESS—7
; по адресу TABLE—STORE
STORE:  MOVAL   TABLE—ADDRESS—7,—
                TABLE—STORE
```

Макроассемблер обрабатывает это предложение, как написанное в одну строку:

```
STORE:  MOVAL   TABLE—ADDRESS—7, TABLE—STORE
```

Предложение может быть продолжено на другой строке с любой точки. Однако постоянные символические имена и символические имена, определяемые пользователем, нельзя разделять. При необходимости их можно только полностью переносить на следующую строку.

Пустые строки, хотя и допустимы, не имеют в исходной программе значения, кроме средства форматирования распечатки программы.

Поле метки. Метка представляет собой определенное пользователем символическое имя, идентифицирующее адрес в программе. Метке присваивается значение, равное текущему значнию счетчика адресов для данного байта программной секции.

Метка может включать в себя произвольное число символов, но не более 31, и содержать любой алфавитно-цифровой символ, знак подчеркивания (—), знак денежной единицы (⊙) и точку (.). Символ, с которого начинается метка, не должен быть цифрой.

Метка ограничивается двоеточием (:) или двойным двоеточием (::). Двоеточие означает, что данная метка определена только для данного модуля (внутреннее символическое имя), а двойное двоеточие — что данная метка является глобальнс определенной (внешнее символическое имя), т. е. на нее можно ссылаться из другого отдельно оттранслированного модуля. При объединении этих модулей компоновщик определяет значение глобальной метки и передает его в другой модуль, в котором данная метка должна быть объявлена как глобальная с помощью директивы .GLOBL.

В поле метки может быть определена более чем одна метка. В этом случае для каждой из них вычисляется одно и то же значение счетчика адреса. Метки могут быть расположены на одной строке или на последовательных строках.

Пример.

```
; Вариант 1. Метки расположены на одной строке.
STEP: COMPARE:  CMPW    LIST1, LIST2
; Вариант 2. Метки расположены на двух
; последовательных строках.
```

```
STEP:
COMPARE:        CMPW    LIST1, LIST2
```

Предпочтительным является вариант 2, так как он не нарушает стандартного формата исходного предложения.

Метка, определенная в программе, не может быть переопределена далее в этой же программе. Если метка определена более одного раза, выдается сообщение об ошибке.

Если метка распространяется за колонку 7, предложение для лучшей наглядности следует перенести на следующую строку, чтобы поле операции могло начаться в колонке 9.

Пример.

```
M1:     MOVAL   TAB—ARG, R0     ; Получить в регистре R0 адрес поля
                                ; TAB—ARG.
COMPARE—ARG:
        CMPL    (R0), VALUE     ; Сравнить значения аргумента из табли-
                                ; цы TAB—ARG со значением VALUE.
        BGTR    M3              ; Если больше, переход на метку M3.
M2::    MOVL    (R0), MIN       ; Получить в поле MIN значение ар-
                                ; гумента.
        BRW     TEST1           ; Безусловный переход на метку TEST1.
M3::    MOVL    (R0), MAX       ; Получить в поле MAX значение ар-
                                ; гумента.
        BRW     TEST2           ; Безусловный переход на метку TEST2.
```

Поле операции. В поле операции определяется действие, которое должно быть выполнено. Поле операции может содержать мнемоническое обозначение инструкции, директиву транслятора или макрокоманду. Если операцией является мнемоническое обозначение инструкции, то Макроассемблер генерирует ее двоичный код и вычисляет адреса операндов. Если операцией является директива транслятора, то Макроассемблер выполняет определенные действия по управлению процессом трансляции исходной программы. Если операцией является макрокоманда, то Макроассемблер производит замену этой макрокоманды на соответствующую ей последовательность отдельных инструкций с последующей генерацией двоичного кода для каждой инструкции.

Поле операции может быть ограничено пробелом или знаком горизонтальной табуляции, однако рекомендуемым ограничителем является знак горизонтальной табуляции.

Поле операнда. Поле операнда может содержать операнды инструкции, аргументы директивы или макрокоманды.

Операнды инструкции в зависимости от используемых режимов адресации идентифицируют адреса ячеек памяти и/или регистры и определяют объекты, над которыми будут выполняться действия инструкции. Если в поле операнда указываются два или более операндов, то их следует разделить запятыми. Макроассемблер допускает использование пробела и

знака горизонтальной табуляции в качестве разделителей аргументов любой директивы, в которой не появляются выражения. В поле операнда могут использоваться выражения, которые должны быть разделены запятыми.

Аргументы директив транслятора должны отвечать требованиям соответствующих форматов, а аргументы макрокоманд — требованиям, определенным в соответствующих макроопределениях.

Поле операнда может ограничиваться знаком «точка с запятой» (;). С этого знака начинается поле комментария. Если в строке отсутствует комментарий, поле операнда заканчивает строку.

Поле комментария. Поле комментария содержит текст, поясняющий функцию данного предложения. Каждая строка программы может иметь комментарий. Он не оказывает воздействия на процесс трансляции или на выполнение программы, за исключением комментария в директивах .ERROR, .PRINT, .WARN, который выводится в виде сообщения во время трансляции.

Полю комментария должен предшествовать знак «точка с запятой» (;), а ограничивается оно концом строки.

Если комментарий не помещается на одной строке, его можно перенести на следующую строку, однако переносимой части должен предшествовать еще один знак «точка с запятой». На одной строке может располагаться лишь один комментарий.

Правильно составленный текст комментария должен передавать смысл, а не действие данного предложения.

Пример.

```
; Вариант 1. Комментарий передает действие исходного
; предложения:
        MOVAL   LIST—POINTER, R7    ; Переслать в регистр R7 адрес
                                    ; поля LIST—POINTER.
; Вариант 2. Комментарий передает смысл исходного
; предложения:
        MOVAL   LIST—POINTER, R7    ; Установить в регистре R7 адрес
                                    ; указателя списка элементов
                                    ; LIST—POINTER.
```

В приведенном примере вариант 2 является предпочтительным.

4.2. СИМВОЛЫ ЯЗЫКА

Символами языка Макроассемблера СМ1700 являются составные части операторов, из которых формируется программа. К ним относятся графические символы, символические имена, числа, унарные и бинарные операции и т. д.

Таблица 4.1

СПЕЦИАЛЬНЫЕ ЗНАКИ

Знак	Наименование	Назначение
_	Знак подчеркивания	Знак в символических именах
⊙	Знак денежной единицы	Знак в символических именах
.	Точка	Знак в символических именах, счетчик текущего адреса программы, десятичная точка
:	Двоеточие	Ограничитель метки
=	Знак равенства	Оператор прямого присваивания и ограничитель аргумента ключевого слова макрокоманды
<TAB>	Знак горизонтальной табуляции	Ограничитель (разделитель) поля предложения
<SP>	Пробел	Ограничитель поля предложения
#	Знак номера	Индикатор режима непосредственной адресации
@	Коммерческое «эт»	Индикатор режима косвенной адресации и операция арифметического сдвига
,	Запятая	Ограничитель полей, операндов. Разделитель символических аргументов в поле операнда
;	Точка с запятой	Индикатор поля комментария
+	Плюс	Индикатор режима адресации с автоувеличением, унарная операция «плюс» и операция арифметического суммирования
-	Минус или дефис	Индикатор режима адресации с автоуменьшением, унарная операция «минус», операция арифметического вычитания и индикатор переноса строки
*	Звездочка	Операция арифметического умножения
/	Дробная черта	Операция арифметического деления
&	Коммерческое «и»	Операция логического «И»
!	Восклицательный знак	Операция логического «ИЛИ»
\	Обратная дробная черта	Операция «исключающее ИЛИ» и индикатор символьного представления числовых аргументов макрокоманды
^	Стрелка вверх	Ограничитель унарных операций и аргументов макрокоманд
[ ]	Квадратные скобки	Индикаторы режима адресации с индексацией и коэффициента повторений
( )	Круглые скобки	Индикаторы регистровой косвенной адресации
< >	Угловые скобки	Ограничители аргументов и выражений
?	Знак вопроса	Индикатор автоматически создаваемых локальных меток в аргументах макрокоманд
'	Апостроф	Индикатор конкатенации аргументов макрокоманд
%	Знак процента	Строковые операции макрокоманд

4.2.1. ГРАФИЧЕСКИЕ СИМВОЛЫ

В исходных предложениях языка Макроассемблера употребляются следующие графические символы:

- строчные и прописные буквы латинского алфавита;
- строчные и прописные буквы русского алфавита;
- цифры от 0 до 9;
- специальные знаки, перечисленные в табл. 4.1.

Кодирование графических символов приведено в приложении 1.

4.2.2. ЧИСЛА

В Макроассемблере СМ1700 могут быть представлены целые числа, числа с плавающей запятой и десятичные числа в упакованном формате. Все числа в исходной программе интерпретируются в десятичной системе счисления, если перед числом не стоит операция управления основанием системы счисления.

Целые числа. Они могут употребляться в любом выражении, включая выражения в операндах и в операторах прямого присваивания. Эти числа имеют следующий формат:

snn

где s — необязательный знак числа; знак «плюс» (+) для положительных чисел или знак «минус» (—) для отрицательных. По умолчанию принимается знак «плюс»;

nn — последовательность цифр, допустимых в данном основании системы счисления.

Диапазоны представления целых чисел приведены в табл. 2.1.

Числа с плавающей запятой. Эти числа могут употребляться в директивах .F_FLOATING (.FLOAT), .D_FLOATING (.DOUBLE), .G_FLOATING и .H_FLOATING, а также в качестве операндов в инструкциях, выполняющих действия над числами с плавающей запятой. Число с плавающей запятой не может использоваться в выражениях или совместно с бинарной или унарной операцией, за исключением унарных операций формата с плавающей запятой (^F), плюс (+) и минус (—).

Число с плавающей запятой может быть специфицировано как с показателем степени, так и без него.

Форматы чисел с плавающей запятой без показателя степени:

snn
snn. nn
snn.

Формат числа с плавающей запятой с показателем степени:

snnElnn
snn. nnElnn
snn. Elnn

где s — знак числа;
nn — последовательность десятичных цифр в диапазоне от 0 до 9;
E — обозначение числа 10, которое имеет степень, указанную после символа «E»;
l — знак порядка («+» указывать необязательно).

Десятичная точка может появляться в любом месте справа от первой цифры. Число с плавающей запятой не может начинаться с десятичной точки, иначе Макроассемблер будет обрабатывать это число как символическое имя, определенное пользователем.

Числа с плавающей запятой могут быть одинарной точности (32 разряда), двойной точности (64 разряда) и расширенной точности (128 разрядов). Диапазоны представления чисел с плавающей запятой приведены в табл. 2.2.

Числа с плавающей запятой одинарной точности могут либо округляться (по умолчанию), либо усекаться. Это определяется директивами .ENABLE и .DISABLE с соответствующими аргументами. Числа с плавающей запятой двойной и расширенной точности всегда округляются.

Десятичные числа в упакованном формате. Эти числа могут употребляться только совместно с директивой .PACKED.

Они имеют следующий формат:

snn

где s — знак числа;
nn — последовательность десятичных цифр в диапазоне от 0 до 9, состоящая не более чем из 31 цифры.

Десятичное число в упакованном формате не может иметь в своем составе ни десятичной точки, ни показателя степени.

4.2.3. СИМВОЛИЧЕСКИЕ ИМЕНА

Символические имена, используемые в исходных программах на языке Макроассемблера, могут быть трех типов: постоянные символические имена, символические имена, определяемые пользователем, и имена макрокоманд.

Конкретное значение символического имени зависит от того, как оно используется в программе.

Символическое имя в поле операции может быть либо постоянным символическим именем, либо именем макрокоманды.

Символическое имя в поле операнда может быть либо символическим именем, определенным пользователем, либо именем регистра. Во всех других случаях выдается сообщение об ошибке.

Макроассемблер осуществляет поиск определения символического имени в следующем порядке:

1) среди заранее определенных имен макрокоманд;

2) среди символических имен, определенных пользователем;

3) среди постоянных символических имен (инструкций и директив);

4) в библиотеках макрокоманд.

Такой порядок поиска значения символического имени позволяет переопределять постоянное символическое имя с помощью имени макрокоманд.

Постоянные символические имена. К постоянным символическим именам относятся мнемонические обозначения инструкций, директив и имена регистров. Они не нуждаются в предварительном определении.

Имена регистров общего назначения также не требуют предварительного определения при указании режимов адресации и впоследствии не могут быть переопределены, т. е. ни одно из символических имен, определяемых пользователем, не может совпадать ни с одним из обозначений регистров. 16 регистров общего назначения могут быть определены в исходной программе только так, как представлено ниже.

Мнемоника	Назначение
R0	Регистр общего назначения 0
R1	Регистр общего назначения 1
. . .	. . . . .
R11	Регистр общего назначения 11
R12 или AP	Регистр общего назначения 12 или указатель аргумента. Если регистр R12 используется в качестве указателя аргумента, рекомендуется употреблять имя AP, если регистр R12 используется в качестве регистра общего назначения, рекомендуется употреблять имя R12
R13 или FP	Указатель кадра
R14 или SP	Указатель стека
R15 или PC	Счетчик инструкций

Символические имена, определяемые пользователем. Эти имена могут использоваться в качестве меток, а также встречаться в любом выражении. В первом случае значение символического имени будет определяться адресом первого байта инструкции, перед которым стоит метка. В программной секции, где встречается эта метка, ее переопределение запрещено, т. е. нельзя использовать одинаковые метки или определять символическое имя, аналогичное метке. Во втором случае символическому имени должно быть присвоено определенное значение с помощью оператора прямого присваивания.

Формирование символических имен, предварительно определяемых пользователем, должно удовлетворять следующим правилам:

• символические имена могут включать в себя алфавитно-цифровые символы, знак подчеркивания (_), знак денежной единицы (⊙) и точку (.). Любой другой графический символ является ограничителем символического имени;

• символическое имя не должно начинаться с цифры;

• символическое имя должно быть уникальным и может включать в себя не более 31 графического символа;

• знак денежной единицы (⊙) резервируется для имен, определяемых разработчиками операционной системы МОСВП;

• точка (.) не должна использоваться в глобальных символических именах, поскольку в других языках, таких, как Фортран, не допускается употребление точки в символических именах.

Этим же правилам подчиняются имена макрокоманд. Одно и то же имя может использоваться как символическое имя, определяемое пользователем, и как имя макрокоманды. Однако во избежание путаницы рекомендуется давать им различные имена.

Символические имена, определяемые пользователем, могут быть как глобальными (внешними), так и локальными (внутренними). Является ли символическое имя локальным или глобальным зависит от его использования в исходной программе.

На локальное символическое имя могут производиться ссылки только из того же модуля, в котором оно определено. Если одинаковые локальные символические имена определены в различных модулях, то они являются независимыми. Ссылка же на глобальное символическое имя допустима из любого другого модуля программы.

Однако символическое имя может быть объявлено глобальным с помощью одного из трех способов:

• путем использования двойного двоеточия (::) при определении метки;

• с помощью двух знаков равенства (==) в операторе прямого присваивания;

• посредством директив .GLOBAL, .ENTRY или .WEAK.

Если символическое имя определено в данном модуле и на него производится ссылка из этого же модуля, то Макроассемблер рассматривает эту ссылку как внутреннюю. Если же в данном модуле символическое имя не определено, то Макроассемблер рассматривает его как внешнее, в последнем случае этому символическому имени автоматически присваивается нулевое значение и оно считается неопределенным глобальным именем. С целью выявления ошибок, связанных с неправильным определением символических имен, целесообразно использовать директиву .DISABLE с соответствующими аргументами. Если она действует, то при трансляции отменяется автоматическое

объявление неопределенных символических имен неопределенными глобальными именами и все неопределенные имена в листинге программы будут соответствующим образом помечены. Для того чтобы все же выделить внешние ссылки, следует использовать директиву .EXTERNAL.

Использование глобальных символических имен существенно упрощает создание сложных программных продуктов, которые, как правило, разрабатываются по модульному принципу. Разбиение на отдельные модули с последующей автономной отладкой и тестированием предполагает использование механизма обращения за данными или же передачи управления в другие модули. В результате определения символического имени (метки или структуры данных) как глобального появляется возможность ссылаться на него из других модулей, где это имя не определяется, но описывается как внешнее.

На этапе компоновки устанавливается связь между автономно оттранслированными модулями путем определения всех неопределенных глобальных символических имен значениями определенных символических имен.

4.2.4. ЛОКАЛЬНЫЕ МЕТКИ

При трансляции исходной программы, написанной на Макроассемблере, транслятор организует и работает с несколькими информационными таблицами, включая таблицы символических имен. В таблицу символических имен заносятся все определенные пользователем символические имена, включая метки, а также их тип и значение. Там, где встречается ссылка на метку, из этой таблицы выбирается соответствующее значение и заносится в объектный модуль. Использование длинного слова (32 разряда) оправдано, если программа достаточно объемна, а метка и ссылка на нее удалены.

Однако, как правило, ссылка на метку и сама метка располагаются недалеко друг от друга в пределах обозримости листинга программы. В этом случае использование длинного слова представляется нецелесообразным для хранения значения ссылки. Чтобы сократить объем исполняемой программы, за счет использования меньшей разрядности для хранения ссылки, организуются локальные метки. Эти метки можно использовать только в пределах специально выделенных областей программы, которые называются блоками локальных меток.

Блок локальных меток может ограничиваться следующими предложениями:

- метками, определяемыми пользователем;
- директивой .PSECT;

• директивами .DISABLE и .ENABLE, с помощью которых блок локальных меток может быть расширен за пределы, обусловливаемые метками пользователя и директивой .PSECT.

Блок локальных меток, начавшийся с директивы .ENABLE LOCAL_BLOCK, должен завершаться одной из следующих директив:

• второй директивой .ENABLE LOCAL_BLOCK;

• директивой .DISABLE LOCAL_BLOCK, за которой следует метка, определяемая пользователем, или директива .PSECT. Обычно блок локальных меток ограничивается двумя метками, определяемыми пользователем.

Локальные метки имеют формат:

nn⊙

где nn — десятичное целое число в диапазоне от 1 до 65535.

Локальные метки могут использоваться так же, как и символические метки, определяемые пользователем, но со следующими отличиями:

• к локальным меткам можно ссылаться только из блока локальных меток исходной программы, в котором они появились;

• локальные метки могут повторно использоваться в другом блоке локальных меток исходной программы;

• локальные метки не появляются в таблице символических имен и, таким образом, не могут быть доступны для символического отладчика;

• локальные метки не могут употребляться совместно с директивой .END.

По соглашению локальные метки располагаются так же, как метки исходного предложения: с левой стороны исходного текста. Они могут появляться в программе в произвольном порядке, однако их рекомендуется располагать в каждом блоке локальных меток в порядке возрастания номеров.

Одинаковые локальные метки в пределах одного блока локальных меток программы запрещены. В разных блоках можно использовать одинаковые локальные метки.

Локальные метки удобно применять в качестве адресов перехода, если эти адреса указываются только в пределах данного блока локальных меток. Кроме того, их можно использовать, чтобы отличать адреса, ссылка к которым осуществляется в пределах небольшого блока локальных меток исходной программы от адресов, ссылка к которым осуществляется в пределах всего программного модуля. Недостатком локальных меток является то, что их числовые имена не позволяют показать их назначение в программе.

Локальные метки рекомендуется создавать в диапазоне от 1⊙ до 29999⊙, поскольку в диапазоне от 30000⊙ до 65535⊙

Макроассемблер автоматически создает локальные метки, используемые в макрокомандах.

Приведенный ниже пример иллюстрирует употребление локальных меток.

Пример.

```
LOG1:  MOVW    A, R0      ; Начинается блок локальных меток.
20⊙:   SUBW2   B, R0      ; Определяется локальная метка 20⊙.
       BGTR    20⊙        ; Условный переход на локальную метку 20⊙.
       INCW    R0         ;
LOG2:  MOVW    C, R1      ; Заканчивается предыдущий блок локальных
                          ; меток и начинается новый.
       MOVW    F, R2      ;
20⊙:   CMPW    R0, R1     ; Определяется локальная метка 20⊙.
       BGTR    40⊙        ; Условный переход на локальную метку 40⊙.
       ADDW2   E, R0      ;
       BRB     20⊙        ; Безусловный переход на локальную метку 20⊙.
40⊙:   MOVW    R2, EXT    ; Определяется локальная метка 40⊙.
       BRW     NEXT1      ; Безусловный переход на метку, определенную
                          ; пользователем.
       . ENABLE LOKAL—BLOCK    ; Начало блока локальных меток, ко-
                               ; торый не будет ограничен меткой,
                               ; определенной пользователем.
LAB1:  ADDL3   R0, R1, R3 ;
       CMPL    R2, R3     ;
       BRB     1⊙         ;
LAB2:  SUBL2   R2, R3     ; LAB2 не является началом очередного блока
                          ; локальных меток.
1⊙:    SUBL2   R2, R3     ; Определение локальной метки 1⊙.
       BGTR    2⊙         ; Условный переход на локальную метку 2⊙.
       INCL    R0         ;
       INCL    R1         ;
       BRB     NEXT2      ;
2⊙:    INCL    R1         ; Определение локальной метки 2⊙.
       DECL    R0         ;
       . DISABLE LOKAL—BLOCK   ; Завершение блока локальных меток.
NEXT2: MOVL    D, R4      ;
```

4.2.5. УНАРНЫЕ ОПЕРАЦИИ

Унарные операции модифицируют терм или выражение. Действие, которое должно быть выполнено, определяется знаком унарной операции.

После выполнения операции над термом (аргументом или операндом) он изменяет свое значение и может использоваться далее как самостоятельная величина или как элемент выражения. Если терм является элементом выражения, то после изменения своего значения он продолжает оставаться элементом этого выражения.

Выражение, над которым выполняется унарная операция, должно быть заключено в угловые скобки. В табл. 4.2 содержится краткая информация по унарным операциям.

Таблица 4.2

УНАРНЫЕ ОПЕРАЦИИ

Знак	Название	Пример	Описание
+	Плюс	+ A	Дает положительное значение A
—	Минус	— A	Дает отрицательное (в виде дополнения до двух) значение A
^B	Знак двоичного основания	^B11000111	Интерпретирует число 11000111 как двоичное
^D	Знак десятичного основания	^D127	Интерпретирует число 127 как десятичное
^O	Знак восьмеричного основания	^O34	Интерпретирует число 34 как восьмеричное
^X	Знак шестнадцатеричного основания	^X19B7	Интерпретирует число 19B7 как шестнадцатеричное
^A	Знак операции преобразования в КОИ-8	^A/ABC/	Графические символы, заключенные между парными ограничителями, преобразуются в последовательность байтов с кодами КОИ-8
^M	Знак операции регистровой маски	#^M<R3, R4, R5>	Определяет регистры R3, R4, R5 в регистровой маске
^F	Знак формата с плавающей запятой	^F3.0	Интерпретирует число 3.0 как число с плавающей запятой
^C	Знак дополнительного кода	^C24	Выполняет преобразование десятичного значения 24 в дополнительный код (дополнение до единицы)

С единичным термом или с выражением, заключенным в угловые скобки, можно использовать более одной унарной операции, например —+—A. Такое предложение эквивалентно следующему: —<+<—A>>.

Унарные операции «+» и «—», определяющие знак терма или выражения (по умолчанию значение считается положительным), являются достаточно традиционными. Рассмотрим более подробно операции, отличительной чертой которых является наличие признака унарной операции (знак «стрелка вверх» — «^»).

Операции управления основанием системы счисления. Эти операции определяются двумя символами, первый из которых символ ^, второй — один из четырех символов B, D, O, X, который указывает на основание системы счисления. С помощью этих операций значения термов и выражений могут быть представлены в двоичной, десятичной, восьмеричной, шестнадцате-

ричной системах счисления. Для каждого вида используются соответствующие графические символы.

В табл. 4.3 приведены графические символы, допустимые для каждого из рассматриваемых оснований системы счисления.

Таблица 4.3

ОПЕРАЦИИ УПРАВЛЕНИЯ ОСНОВАНИЕМ СИСТЕМЫ СЧИСЛЕНИЯ

Формат	Основание	Допустимые графические символы
^Bnn	Двоичное	0 и 1
^Dnn	Десятичное	от 0 до 9
^Onn	Восьмеричное	от 0 до 7
^Xnn	Шестнадцатеричное	от 0 до 9 и от A до F

Операции управления основанием системы счисления могут быть включены в любом месте исходной программы, где требуется соответствующее числовое значение. Они оказывают воздействие только на тот терм или выражение, которое следует непосредственно за указателем операции. Стрелка вверх (^) не может быть отделена от символов B, D, O и X, которые за ней следуют, но в целом операция управления основанием системы счисления может быть отделена пробелами или знаком табуляции от соответствующего терма или выражения. В результате выполнения операции данный терм или выражение примет значение в указанном основании системы счисления.

Пример.

```
. BYTE    ^B10011          ; Двоичное основание.
. WORD    ^D36             ; Десятичное основание.
. WORD    ^07767           ; Восьмеричное основание.
. LONG    ^XABF            ; Шестнадцатеричное основание.
. LONG    ^X<E2C4+3F+9>    ; Все числа в выражении имеют
                           ; шестнадцатеричное основание.
```

Как было отмечено выше, если особо не указано, то по умолчанию все числа интерпретируются как десятичные. В связи с этим операция управления десятичным основанием системы счисления (^D) становится необходимой только в выражении, перед которым стоит другая подобная операция. Так, в приведенном ниже примере число 77 интерпретировалось бы как восьмеричное число, если бы ему не предшествовала операция ^D.

Пример.

```
. WORD ^O<315+16+^D77>
```

Строковые операции. К ним относятся операция преобразования в КОИ-8 (^A) и операция регистровой маски (^M).

Операция ^A осуществляет преобразование последовательности печатных графических символов в соответствующую последовательность байтов с кодами КОИ-8 и сохраняет эти коды в программе, отводя на один символ один байт. Последовательность символов должна быть заключена в парные ограничители, в качестве которых может использоваться любой графический символ, за исключением пробела, горизонтальной табуляции и точки с запятой (;). Во избежание путаницы для этой цели не следует применять алфавитно-цифровые символы, хотя любой из них может быть ограничителем.

Максимальная длина строки, которая преобразуется операцией ^A, не должна иметь больше 16 символов. Кроме того, последовательность графических символов не должна превышать размер, характерный для инструкции, в поле операнда которой встретилась данная унарная операция. Если, например, операция ^A появляется в одном из операндов инструкции MOVQ (тип данного — квадрослово), то количество символов не должно превышать четырех.

Пример.

```
MOVW #^A/DG/, R0 ; Код символа D помещается в разряды 0:7,
                 ; а код символа G — в разряды 8:15 регистра R0.
MOVL #^A/GOLD/, R1 ; Поместить коды символов G, O, L, D в регистр R1. Код символа G помещается в самый младший байт (разряды 0:7), а код
                   ; символа D — в самый старший байт (разряды 24:31) регистра R1.
MOVL #^A/ZV/, R0 ; Коды символов Z и V помещаются в младшие байты регистра R0, два старших байта
                 ; регистра заполняются нулями.
```

Операция регистровой маски (^M). Эта операция формирует двухбайтовую регистровую маску, в которой каждый разряд соответствует номеру регистра общего назначения, и устанавливает соответствующий разряд для указанных регистров. Если задано несколько регистров, то их символические обозначения разделяются запятыми, а весь список регистров заключается в угловые скобки (например, ^M<R5, R8, R10>). В случае задания одного регистра его символическое обозначение следует непосредственно за операцией (например, ^MR4).

Как правило, операция регистровой маски используется для указания номеров регистров, которые необходимо сохранить при обращении к процедуре. В операции регистровой маски можно указать также спецификаторы разрешения арифметического прерывания (IV и DV).

В слове состояния процессора спецификатор IV устанавливает «ловушку» по переполнению целых чисел, а спецификатор DV — «ловушку» по переполнению десятичных чисел.

Если операция регистровой маски используется совместно с инструкциями POPR или PUSHR, то в ней можно указывать регистры с R0 по R11, R12 или AP, FP и SP, но нельзя задавать регистр PC и спецификаторы разрешения арифметического прерывания (IV и DV).

Когда операция регистровой маски используется совместно с директивами .ENTRY или .MASK, то в ней можно указывать регистры с R2 по R11 и спецификаторы разрешения арифметического прерывания (IV и DV), но нельзя задавать регистры R0, R1, FP, SP и PC.

В табл. 4.4 указаны разряды регистровой маски, которые соответствуют номеру регистра общего назначения процессора, и спецификаторы арифметического прерывания.

Таблица 4.4

НАЗНАЧЕНИЕ РАЗРЯДОВ МАСКИ

Регистр	Разрешение арифметического прерывания	Разряд маски
С R0 по R11	—	с 0 по 11
R12 или AP	—	12
FP	—	13
SP	IV	14
—	DV	15

Пример.

```
PUSHR #^M<R2, R3, R4>; Сохранить регистры R2, R3, R4.
POPR  #^M<R2, R3, R4>; Восстановить регистры R2, R3, R4.
. ENTRY PROG, ^M<R6, R7, DV>    ; Сохранить регистры R6,
                                ; R7 и установить «ловуш-
                                ; ку» переполнения десятич-
                                ; ных чисел.
```

Операции числового представления. К ним относятся операция формата с плавающей запятой (^F) и операция дополнения (^C).

Операция формата с плавающей запятой (^F) преобразует литерал в формате с плавающей запятой во внутреннее 4-байтовое представление. Далее это значение может быть использовано в любом выражении. При этом следует помнить, что обработку выражений, в которые включаются числа с плавающей запятой, Макроассемблер не производит.

Операция формата с плавающей запятой удобна при использовании чисел с плавающей запятой в инструкциях, воспринимающих только целые числа.

Пример.

```
MOVL  #^F5.6, R1 ; Инструкция MOVF более предпочтительна для
MOVF  #5.6, R1   ; пересылки числа с плавающей запятой.
```

Операция дополнения (^C). С помощью этой операции можно получить обратный код (дополнение до единицы) заданного значения. Им может быть терм или выражение. Если указывается выражение, оно заключается в угловые скобки.

Для получения обратного кода транслятор вычисляет значение терма или выражения в четырехбайтовом формате.

Пример.

```
. LONG  ^C^XFF ; Результатом является шестнадцатеричное чис-
               ; ло FFFFFF00.
. LONG  ^C25   ; Результатом является шестнадцатеричное чис-
               ; ло FFFFFFE6.
```

4.2.6. БИНАРНЫЕ ОПЕРАЦИИ

В отличие от унарных операций, которые «работают» с одним аргументом, бинарные операции указывают на действие, которое должно быть выполнено над двумя термами или выражениями. При этом выражения должны быть заключены в угловые скобки. В табл. 4.5 представлена краткая информация по бинарным операциям.

Таблица 4.5

БИНАРНЫЕ ОПЕРАЦИИ

Знак	Название	Пример	Описание
+	Плюс	A + B	Сложение
—	Минус	A — B	Вычитание
*	Звездочка	A * B	Умножение
/	Косая черта	A / B	Деление
@	Коммерческое «эт»	A @ B	Арифметический сдвиг
&	Коммерческое «И»	A & B	Логическое «И»
!	Восклицательный знак	A ! B	Логическое «ИЛИ»
\	Обратная косая черта	A \ B	Логическое «исключающее ИЛИ»

Все бинарные операции имеют равный приоритет. Порядок выполнения операций может быть изменен угловыми скобками. Термы и выражения, заключенные в угловые скобки, обрабатываются в первую очередь, а остальные операции выполняются по порядку, слева направо.

Пример.

```
        . LONG   1+2*3      ; Результат равен 9.
        . LONG   1+<2*3>    ; Результат равен 7.
```

Результатом выполнения всех бинарных операций является значение в четырехбайтовом формате. Если в качестве операндов используются одно- или двухбайтовые величины, то результат помещается в младшие байты (байт) четырехбайтового поля. Если усечение приводит к потере значимости, то Макроассемблер выдает сообщение об ошибке.

Рассмотрим на примерах действие некоторых бинарных операций.

Операция арифметического сдвига (@). Эта операция используется для выполнения сдвига арифметических величин влево (вправо). При этом первый аргумент сдвигается влево (вправо) на число разрядов, которое задается вторым аргументом.

Если второй аргумент положительный, то первый аргумент сдвигается влево и младшие разряды устанавливаются в нуль. Если второй аргумент отрицательный, первый аргумент сдвигается вправо. Соответствующие старшие разряды заполняются значением, присутствовавшим в старшем знаковом разряде до сдвига.

Пример.

```
        . BYTE   ^B111@3       ; Результат 111000 (двоичное).
        . BYTE   1@4           ; Результат 10000 (двоичное).
SD=2
        . LONG   1@SD          ; Результат 100 (двоичное).
        . LONG   ^B101@-SD     ; Результат 1.
        . LONG   ^X2244@-4     ; Результат 224 (шестнадцатеричное).
```

Логические операции. Эти операции являются традиционными практически для всех типов ЭВМ, поэтому для пояснения их действия приведем лишь примеры.

Операция логическое «И»:

```
M=^B10101
N=^B11110
        . LONG   M&N       ; Результат 10100 (двоичное).
```

Операция логическое «ИЛИ»:

```
M=^B10100
N=^B10101
K=^B11000

        . LONG   M!N       ; Результат 10101 (двоичное).
        . LONG   M!K       ; Результат 11100 (двоичное).
```

Операция логическое «исключающее ИЛИ»:

```
M=^B1010
N=^B1100
        . LONG   M\N       ; Результат 0110 (двоичное).
```

4.2.7. ОПЕРАТОР ПРЯМОГО ПРИСВАИВАНИЯ

С помощью этого оператора символическому имени присваивается определенное значение. Символическое имя, определенное с помощью оператора прямого присваивания, в программе может быть переопределено любое число раз. При первом определении оно заносится в таблицу символических имен пользователя и с каждой модификацией значения выполняется соответствующая корректировка в этой таблице.

Оператор прямого присваивания может использоваться следующим образом:

символ = выражение
символ = = выражение

где символ — символическое имя, определенное пользователем;
выражение — любое выражение, не содержащее неопределенных символических имен.

Формат с единственным знаком равенства (=) определяет локальное символическое имя, а формат с двумя знаками равенства (= =) — глобальное.

Пример.

```
VALUE= =4       ; Глобальное символическое имя VALUE имеет значение 4.
MIN=VALUE-3     ; MIN имеет значение 1.
MAX=VALUE@3     ; MAX имеет значение 20 (шестнадцатеричное).
MM=^X100/^X10   ; MM имеет значение 10 (шестнадцатеричное).
V=45+3          ; V имеет значение 48 (десятичное).
```

Использование операторов прямого присваивания подчиняется следующим синтаксическим правилам:

- символическое имя и выражение, определяющее его значение, должны разделяться знаком равенства или двойным знаком равенства. Пробелы, предшествующие или следующие за оператором прямого присваивания, не имеют значения при определении конечного значения;
- в одном операторе прямого присваивания может быть определено только одно символическое имя;
- за оператором прямого присваивания может следовать только поле комментария;
- символическое имя в операторе прямого присваивания размещается в поле метки;
- ссылки вперед на символические имена, значения которых определяются далее в программе, запрещены.

Пример.

```
AB=CD    ; Ссылка вперед запрещена.
CD=15
```

4.2.8. СЧЕТЧИК АДРЕСОВ ПРОГРАММЫ

Текущее значение счетчика адресов программы обозначается символом «точка» (.). Он всегда имеет значение, равное адресу текущего байта в программе.

Макроассемблер в начале трансляции и в начале каждой новой программной секции устанавливает значение счетчика, равное 0. Каждое исходное предложение языка Макроассемблера, в результате трансляции которого в объектном модуле распределяется память, приводит к увеличению значения текущего счетчика адресов программы на число распределенных байтов. Например, директива .LONG 0 увеличивает текущий счетчик адресов программы на 4.

Символ текущего значения счетчика адресов программы может использоваться в поле операнда инструкции и в поле операнда директивы Макроассемблера. Когда счетчик адресов программы встречается в поле операнда инструкции, то он принимает значение адреса данного операнда, а не значение адреса, с которого начинается данная инструкция. Если символ «точка» (.) используется в поле операнда директивы, то текущее значение счетчика равно адресу текущего байта.

Пример.

```
; В длинное слово с адресом MAP поместить его адрес

MAP:   . LONG   .       ; Вариант 1.
MAP:   . LONG   MAP     ; Вариант 2.

; Варианты 1 и 2 эквивалентны
```

Счетчик адресов программы может быть установлен на определенное значение с помощью специальной формы оператора прямого присваивания. При этом значение счетчика может быть увеличено либо уменьшено (такая установка значения счетчика адресов программы полезна при определении структур данных), что выполняется следующим образом:

.=выражение

Выражение не должно содержать неопределенных символических имен, ссылок вперед или символических имен, значение которых изменяется от одного прохода трансляции к другому.

Пример.

```
B=500
C=12
.=.+B+C     ; Увеличить значение счетчика на 512.
```

С помощью установки значения счетчика адресов программы не рекомендуется производить резервирование памяти; для этой цели следует использовать директивы .BLKX.

Пример.

```
.=.+40          ; Перевести счетчик вперед.
. BLKW 20       ; Использование директивы . BLKW
                ; предпочтительнее.
```

В зависимости от типа программной секции, в которой встречается символ счетчика адресов, значение счетчика будет относительным или абсолютным во всей секции. При трансляции относительной секции начальное значение счетчика всегда равно нулю. Если особо не указано, нулевое значение счетчика будет и для абсолютной секции. Однако при компоновке различных секций значение счетчика адресов в абсолютной секции будет отсчитываться от нуля (если не определено другое значение), тогда как в относительных секциях начальное значение будет устанавливаться с учетом длины предыдущих секций.

Когда ранее определенная программная секция продолжается в том же модуле, текущий счетчик адресов программы устанавливается на его последнее значение в данной программной секции.

4.3. ТЕРМЫ И ВЫРАЖЕНИЯ

Терм является элементом выражения и может быть числом, символом, текущим счетчиком адресов программы или строкой символов, перед которой стоит знак строковой операции, а также любым из перечисленных элементов, перед которым стоит знак унарной операции.

Макроассемблер вычисляет значения термов как значения длинных слов. Если в качестве терма употребляется неопределенный символ, то значение этого терма определяет компоновщик. Значение текущего счетчика адресов программы (.) равно значению счетчика адресов программы в начале текущего операнда.

Выражения представляют собой комбинации термов, объединенных бинарными операциями. Значения выражений вычисляются в формате длинных слов. Макроассемблер определяет значение выражения слева направо без учета приоритета операций. Однако для изменения порядка обработки выражения могут быть использованы угловые скобки (<>). Любая часть выражения, заключенная в угловые скобки, обрабатывается в первую очередь до получения некоторого значения, которое затем используется для дальнейших вычислений (например, выражения A*B+C и A*<B+C> различны).

Унарная операция считается составной частью терма, поэтому Макроассемблер выполняет действие, указанное данной операцией, до выполнения действия, указанного любой бинарной операцией. Угловые скобки можно использовать также для

распространения унарной операции на целое выражение, например $-<A+B>$.

Все выражения можно разделить на три категории: относительные, абсолютные и внешние (глобальные).

Выражение является *относительным*, если его значение фиксировано относительно начала программной секции, в которой встретилось это выражение. Так, текущее значение счетчика адресов программы является относительным выражением в перемещаемой программной секции. Другим относительным выражением в перемещаемой секции будет выражение, в котором термы содержат метки. Их значение изменяется в процессе компоновки.

Выражение считается *абсолютным*, если его значение определяется во время трансляции и является константой. Абсолютным будет выражение, все термы которого являются числами или представляют собой последовательности символьных кодов. Выражение, содержащее относительный терм, из которого вычитается другой относительный терм из той же самой программной секции, является абсолютным, поскольку такое выражение сводится к константе, определяемой во время трансляции. Абсолютным считается также выражение, в котором присутствует терм, содержащий метку из абсолютной секции, так как все метки в абсолютной секции имеют постоянное значение.

Выражение считается *внешним*, если в его состав входят один или несколько символических имен, определение которых отсутствует в текущей программной секции. Окончательное значение выражения определяется в процессе компоновки с другими программными секциями, в которых определяются эти символические имена.

Пример.

```
BETA=  5*60                    ; 5*60 абсолютное выражение.
       . BLKW   BETA+100       ; BETA+100 абсолютное выражение.
M:     . BLKW   BETA           ; M относительное выражение.
       ALPHA=M+BETA+200        ; ALPHA относительное выражение.
N:     . BLKW   N-M            ; N-M абсолютное выражение.
       . WORD   K-M            ; K-M абсолютное выражение.
K:     . BLKW   DELTA+2        ; Внешнее выражение, так как символическое имя DELTA не определено
                               ; в данной секции.
```

В большинстве директив и в операторах прямого присваивания могут употребляться выражения любого типа. Однако в некоторых случаях выражения могут включать в себя только такие символические имена, которые были предварительно определены в текущем модуле. Более того, они не могут включать в себя ни внешние символические имена, ни символические имена, определяемые в данном модуле позднее.

Глава 5

ОБЩИЕ ДИРЕКТИВЫ МАКРОАССЕМБЛЕРА

Общие директивы Макроассемблера обеспечивают программисту дополнительные, достаточно обширные возможности, которые реализуются транслятором с этого языка. В соответствии с выполняемыми ими функциями директивы Макроассемблера СМ1700 можно разделить на следующие группы:

- идентификации и управления листингом;
- вывода сообщений;
- управления функциями транслятора;
- управления памятью;
- управления счетчиком адресов программы;
- секционирования программы;
- определения входной точки программы;
- управления символическими именами;
- условной трансляции;
- перекрестных ссылок;
- генерирования инструкций;
- дополнительной компоновки.

В данной главе подробно описываются директивы всех групп, приводятся их форматы и примеры использования. В квадратных скобках указываются необязательные аргументы или варианты формирования директивы.

При описании директив в круглых скобках даются их возможные эквиваленты. Если формат директив в группе один и тот же, то он приводится только один раз. Директивы отдельной группы иногда рассматриваются не в алфавитном порядке, а исходя из их функционального назначения.

Состав директив достаточно обширен, в ряде случаев выполняемые ими функции дублируются. Отчасти это объясняется стремлением сохранить преемственность с Макроассемблером для более ранних моделей СМ ЭВМ.

5.1. ДИРЕКТИВЫ ИДЕНТИФИКАЦИИ И УПРАВЛЕНИЯ ЛИСТИНГОМ

К директивам данной группы относятся: .TITLE, .PAGE, .IDENT, .SHOW (.LIST), .SUBTITLE (.SBTTL), .NOSHOW

(.NLIST), .END. Они обеспечивают идентификацию программных модулей, а также организуют хорошо документированный листинг трансляции. Возможны гибкие варианты управления выдачей листинга.

.TITLE — директива заголовка. С помощью этой директивы объектному модулю присваивается имя, которое представляет собой строку символов, отличных от пробела. При формировании листинга трансляции программы это имя включается в его оглавление, выдаваемое на первой странице. Формат директивы:

. TITLE имя модуля ; комментарий

где имя модуля — идентификатор модуля размером от 1 до 31 символа; комментарий — строка символов КОИ-8 размером до 40 символов (лишние символы отбрасываются).

Имя модуля, указанное в директиве .TITLE, записывается в соответствующий объектный файл и выдается в карте загрузки при компоновке. Оно может использоваться при поиске требуемого модуля в библиотеке объектных модулей с помощью программы Библиотекарь.

Пример.

. TITLE TYPE ; Выдача сообщений.

Если директива .TITLE отсутствует в исходном тексте программы, то при ее трансляции Макроассемблер автоматически присваивает объектному модулю имя .MAIN. Если по ошибке в исходной программе присутствует более чем одна директива .TITLE, последняя из них дает имя объектному модулю.

При трансляции Макроассемблер игнорирует все символы табуляции и пробелы после имени директивы, пока не встретит отличный от них символ.

.IDENT — директива идентификации. Эта директива является средством дополнительной идентификации объектного модуля. В дополнение к имени, присвоенному объектному модулю с помощью директивы .TITLE, в директиве .IDENT может быть указана строка символов, которая распечатывается в заголовке файла листинга, а также заносится в объектный модуль. Формат директивы:

. IDENT строка

где строка — строка размером не более 31 символа. Ограничителями строки могут служить любые парные печатные символы, кроме левой угловой скобки (<) и точки с запятой (;). Ограничительный символ не должен использоваться в самой строке. Если ограничительные символы не составляют пару или представляют собой недопустимые знаки, в листинге трансляции выдается ошибка.

Обычно в директиве .IDENT указывается номер версии программы. При трансляции этот номер заносится в список гло-

бальных символических имен объектного модуля, а при компоновке указывается в карте загрузки. Если объектный модуль включается в библиотеку объектных модулей, то при распечатке каталога библиотеки он также будет выдан на печать.

Пример.

```
. IDENT          #4—6#    ; Номер версии и редакции
```

Строка символов «4—6» будет записана в объектный модуль.

Если в исходном тексте программы присутствует более чем одна директива .IDENT, идентификация объектного модуля осуществляется строкой символов из последней директивы.

.SUBTITLE — директива подзаголовка. Обычно в программном модуле имеются отдельные функционально законченные части, которые желательно выделить в листинге трансляции с целью бóльшей их наглядности. Это можно выполнить с помощью директивы .SUBTITLE, которая располагается в первой строке выделяемой части программы. Формат директивы:

```
. SUBTITLE комментарий
```

где комментарий — строка символов КОИ-8 размером от 1 до 40 символов (лишние символы отбрасываются). Эта строка распечатывается в качестве подзаголовка в листинге трансляции.

Встретив данную директиву, Макроассемблер определяет номер строки с директивой и запоминает текст, который записывается после директивы. При выдаче листинга программы, начиная со строки с директивой .SUBTITLE, этот текст печатается на каждой странице листинга на второй строке до конца модуля или до появления такой же директивы с другим текстом. Кроме того, из текстов всех директив .SUBTITLE модуля формируется и выводится на первой странице таблица оглавления листинга, в которой указываются диапазоны строк с текстами соответствующих директив.

Пример.

```
. SUBTITLE обработка прерывания
```

По данной директиве Макроассемблер распечатывает следующий текст в качестве подзаголовка в листинге трансляции:

Обработка прерывания

.END — директива логического конца модуля. Она определяет логический конец программного модуля. Встретив эту директиву, транслятор завершает свой текущий проход. Любой текст, который может появиться за данной директивой, транслятором игнорируется. Этот дополнительный текст не появится ни в файле листинга, ни в соответствующем объектном модуле. Формат директивы:

. END [имя]

где имя — символическое имя, определяющее адрес, с которого начинается выполнение программы.

Пример.

```
. ENTRY    START, 0    ; Маска входа.
. ...                  ; Основная программа.
. END      START       ; Конец программы.
                       ; Начало программы выполнено с метки START.
```

Если выполняемый образ создается из нескольких объектных модулей, то только один из них должен завершаться директивой .END с указанием адреса начала исполнения. Все остальные модули должны включать эту директиву без аргумента. В противном случае при компоновке объектных модулей будет выдана ошибка. Ошибкой будет являться и отсутствие этой директивы в исходном тексте программы.

Директива .END не может присутствовать в блоке условной трансляции, и в момент ее появления в незавершенном блоке будет выдано сообщение об ошибке.

Символическое имя, используемое в качестве аргумента директивы .END, должно быть указано в директиве .ENTRY, которая используется для организации правильного обращения к подпрограммам.

.PAGE — директива перевода страницы. При выводе листинга на печать перевод на новую страницу осуществляется автоматически после выдачи 58 строк. Директива .PAGE позволяет форматировать распечатку программы в произвольном виде, осуществляя переход на новую страницу в момент своего появления в исходном тексте программы. При этом номер страницы в листинге увеличивается на единицу. Сама директива не имеет аргументов и в листинге не распечатывается.

Директива .PAGE, входящая в состав макроопределения, игнорируется. Однако, если эта директива встречается при обработке макрорасширения, происходит переход на новую страницу с соответствующим увеличением на единицу номера страницы.

.SHOW и .NOSHOW — директивы управления листингом. Они определяют дополнительные возможности управления выдачей листинга исходного текста программы. Вместе с этими директивами может использоваться спецификация списка аргументов, которые конкретизируют требуемые действия.

Если эти директивы заданы со списком аргументов, то директива .SHOW включает в файл листинга строки определенных типов, а директива .NOSHOW их исключает. Директивы .SHOW и .NOSHOW управляют листингом исходного текста блоков условной трансляции, макрокоманд и блоков повторений.

Если эти директивы заданы без аргументов, то они изменяют значение счетчика листинга. В исходном состоянии значение счетчика равно нулю. При каждом задании директивы .SHOW его значение увеличивается на 1, а при каждом задании директивы .NOSHOW это значение уменьшается на 1.

Если значение счетчика является отрицательным, то распечатка листинга подавляется (если подавляемые строки не содержат ошибку). Наоборот, если значение счетчика листинга положительное, листинг формируется и выводится. Если значение счетчика листинга равно нулю, то строка либо распечатывается, либо подавляется, в зависимости от значений аргументов управления листингом. Форматы директивы:

. SHOW
. SHOW список аргументов
. NOSHOW
. NOSHOW список аргументов

где список аргументов — один или несколько аргументов, определение которых приведено в табл. 5.1.

Таблица 5.1

АРГУМЕНТЫ ДИРЕКТИВ .SHOW и .NOSHOW

Полная форма	Краткая форма	Значение по умолчанию	Функция
BINARY	MEB	Не печатать	Распечатывает расширения макрокоманд и расширения блоков повторения, которые генерируют двоичный код. Функция BINARY является частным случаем функции EXPANSIONS
CALLS	MS	Печатать	Распечатывает вызовы макрокоманд и спецификаторов блоков повторений
CONDITIONALS	CND	Печатать	Распечатывает блоки условной трансляции в случае, если не выполнены условия их трансляции
DEFINITIONS	MD	Печатать	Распечатывает определения макрокоманд и расширения блоков повторений
EXPANSIONS	ME	Не печатать	Распечатывает расширения макрокоманд и блоков повторений

Допустимо использование как полной, так и краткой формы указанных аргументов. Каждый из аргументов может быть указан отдельно или в сочетании с другими. Если указывается несколько аргументов, то они должны быть разделены пробелами, запятыми или знаками табуляции. Если какой-либо аргу-

мент не включен в директиву управления, то подразумевается его значение по умолчанию.

Пример.

Пусть задано макроопределение XYZ.

```
        . MACRO   XYZ              ; Определение
                  ...              ; макрокоманды XYZ.
        . SHOW                     ; Распечатать следующую строку.
Z=.
        . NOSHOW                   ; Не распечатывать остаток расшире-
                  ...              ; ния макрокоманды.
        . ENDM
        . NOSHOW EXPANSIONS        ; Не распечатывать расширение макро-
                                   ; команд.
        XYZ                        ; Вызов макрокоманды XYZ.
Z=.
```

Так как перед вызовом макрокоманды XYZ была указана директива .NOSHOW EXPANSIONS, то в листинге трансляции будет распечатана только одна строка: Z=.

5.2. ДИРЕКТИВЫ ВЫВОДА СООБЩЕНИЙ

С помощью директив .ERROR, .PRINT, .WARN пользователь может получать дополнительную информацию в процессе обработки транслятором исходного модуля. Соответствующие значения выдаются на терминал, если трансляция была инициирована с терминала. Эти директивы могут использоваться для вывода сообщений, указывающих на то, что макровызов содержит ошибку или недопустимый набор условий. Формат всех трех директив одинаков:

```
. ERROR   [выражение]   ; комментарий
. PRINT   [выражение]   ; комментарий
. WARN    [выражение]   ; комментарий
```

где выражение — выражение, значение которого выводится в процессе трансляции;
комментарий — дополнительный текст, который выводится в составе информационного сообщения вслед за значением выражения. Тексту комментария должна предшествовать точка с запятой (;).

Если значение выражения равно нулю или отсутствует, то оно в информационном сообщении отсутствует.

.ERROR — директива вывода сообщения об ошибках. Посредством директивы .ERROR Макроассемблер выводит во время трансляции сообщения об ошибках. Сообщения выводятся на терминал на распечатку листинга (если он задан).

Пример.

```
. IF GREATER ARG — 100
. ERROR 7                  ; недопустимый аргумент.
. ENDC
```

Если значение символического имени ARG больше или равно 100, то выводится следующее сообщение об ошибке:

MACRO-E-GENERR, GENERATED ERROR: 7 недопустимый аргумент.

При анализе листинга программы, содержащей директивы .ERROR, следует помнить, что строки, в которых встречается эта директива, в листинг не включаются.

.PRINT — директива вывода сообщений. Эта директива позволяет выводить информационные сообщения в процессе трансляции исходной программы. Сообщения состоят из значения некоторого выражения и текста комментария, которые указаны в директиве. Сообщение, выводимое по директиве .PRINT, не считается сообщением об ошибке или предупреждающим сообщением.

Пример.

```
. PRINT 4    ; аргумент принят.
```

При трансляции на терминал будет выдано сообщение:

MACRO-P-GENPRN, GENERATED PRINING: 4 аргумент принят.

Директива .PRINT по своему функциональному назначению идентична директиве .ERROR, за исключением того, что по этой директиве значение счетчика ошибок не увеличивается на единицу, как это происходит при обработке директивы .ERROR.

.WARN — директива вывода предупреждающего сообщения. По директиве .WARN во время трансляции выводится предупреждающее сообщение.

Пример.

```
. IF DEFINED     AREA
. IF GREATER     300 — EXP
. WARN           ; Копирование.
. ENDC
. ENDC
```

Если символическое имя AREA определено и значение символического имени EXP меньше или равно 300, то выводится следующее предупреждающее сообщение:

MACRO-W-GENWRN, GENERATED WARNING: Копирование.

Так как выражение в директиве. WARN отсутствовало, то в состав сообщения оно не попало.

Аналогично директиве .ERROR все строки в исходном тексте программы, содержащие директиву. WARN, в распечатке листинга будут отсутствовать.

5.3. ДИРЕКТИВЫ ДОПОЛНИТЕЛЬНЫХ ВОЗМОЖНОСТЕЙ

При помощи директив .ENABLE (.ENABL), .DEFAULT, .DISABLE (.DSABL) происходит управление функциями транслятора, выбор которых определяется задаваемыми аргументами.

.ENABLE — директива управления выбором функций. Наличие директивы .ENABLE разрешает транслятору Макроассемблера выполнять дополнительные функции. Директива .ENABLE и ее противоположность директива .DISABLE управляют следующими функциями транслятора:

- создание блоков локальных меток;
- объявление всех локальных символических имен доступными для отладчика и включение средств обратной трассировки;
- указание о том, что ссылки на неопределенные символические имена являются внешними ссылками;
- усечение или округление чисел с плавающей запятой одинарной точности;
- подавление распечатки символических имен, которые были определены, но на которые отсутствуют ссылки;
- указание о том, что все ссылки на счетчик инструкций PC являются абсолютными, а не относительными.

Формат директивы .ENABLE (так же, как и формат директивы .DISABLE) выглядит следующим образом:

. ENABLE список аргументов

где список аргументов — один или несколько символических аргументов; допустим как краткий, так и полный их формат.

Символические обозначения аргументов, а также выполняемые ими функции приведены в табл. 5.2. Если указывается несколько символических аргументов, то они должны быть разделены запятыми, пробелами или символами табуляции.

Пример.

```
. ENABLE GLOBAL          ; Все неопределенные символические
                         ; имена рассматриваются как внешние.
. ENABLE LOCAL—BLOCK     ; Завершается текущий блок локаль-
                         ; ных меток и начинается новый.
```

.DISABLE — директива запрещения выполнения функций. Задание этой директивы запрещает указанные в табл. 5.2 функции Макроассемблера. Более подробная информация содержится в описании директивы .ENABLE.

Так же как и для директивы .ENABLE, в этой директиве возможно указание нескольких символических аргументов, которые должны быть разделены запятыми, пробелами или символами табуляции.

.DEFAULT — директива управления значениями по умолчанию. Она определяет длину смещения по умолчанию при относительной и косвенной относительной адресациях. Формат инструкции, кроме ключевого слова DEFAULT, включает еще одно обязательное символьное слово DISPLACEMENT:

Таблица 5.2

СИМВОЛИЧЕСКИЕ АРГУМЕНТЫ ДИРЕКТИВ .ENABLE и .DISABLE

Формат		Значение по умолчанию	Выполняемая функция
полный	краткий		
ABSOLUTE	AMA	Запрещение	При разрешении этой функции все относительные адреса транслируются как абсолютные адреса (режим адресации PC)
DEBUG	DBG	Запрещение	При разрешении этой функции все локальные символические имена включаются в таблицу символических имен данного объектного модуля, которая используется отладчиком
GLOBAL	GBL	Разрешение	При разрешении этой функции все неопределенные символические имена рассматриваются как внешние. При запрещении этой функции любое неопределенное символическое имя, которое не указано в списке директивы .EXTERNAL, вызывает появление ошибки трансляции
LOCAL — BLOCK	LSB	Запрещение	При разрешении этой функции завершается текущий блок локальных меток и начинается новый. При запрещении этой функции текущий блок локальных меток завершается
SUPPRESSION	SUP	Запрещение	При разрешении этой функции все символические имена, которые определены, но к которым нет ссылок, не включаются в таблицу символических имен. При запрещении этой функции все определенные символические имена включаются в таблицу символических имен
TRACEBACK	TBK	Разрешение	При разрешении этой функции имена программных секций и их длины, имена модулей и имена программ включаются в объектный модуль для использования отладчиком; при запрещении этой функции Макроассемблер исключает указанную функцию и по любому локальному символу делает информацию недоступной для отладчика
TRUNCATION	FPT	Запрещение	При разрешении этой функции числа одинарной точности с плавающей запятой усекаются. При запрещении этой функции числа одинарной точности с плавающей запятой округляются. На числа в форматах D, G, H; директива .ENABLE FPT никакого воздействия не оказывает; эти числа всегда округляются

. DEFAULT DISPLACEMENT, ключевое слово

где ключевое слово — одно из трех ключевых слов (BYTE, WORD, LONG), указывающих длину смещения по умолчанию.

Пример.

```
. DEFAULT DISPLACEMENT, BYTE  ; Значением по умолчанию является байт.
MOVL   ABC, R2                ; Если не определена метка ABC, используется смещение в один байт.
. DEFAULT DISPLACEMENT, WORD  ; Значением по умолчанию является слово.
DECB @ DEF+6                  ; Если не определена метка DEF, используется смещение в одно слово.
```

Если в модуле не задана директива .DEFAULT, то длина смещения по умолчанию для косвенной относительной адресации и для относительной адресации равна одному длинному слову.

5.4. ДИРЕКТИВЫ УПРАВЛЕНИЯ ПАМЯТЬЮ

К директивам управления памятью относятся следующие: .ADDRESS, .OCTA, .ASCI [I, C, D, Z], .QUAD, .BYTE, .PACKED, .SIGNED_BYTE, .F_FLOATING (.FLOAT), .WORD, .D_FLOATING, .SIGNED_WORD, .G_FLOATING (.DOUBLE), .LONG, .H_FLOATING.

Помимо функций управления памятью, перечисленные директивы позволяют определять начальные значения, которые будут занесены в соответствующие ячейки оперативной памяти.

.ADDRESS — директива хранения адреса. По директиве .ADDRESS обеспечивается выделение длинного слова в оперативной памяти. При трансляции в него загружается адрес поля, символическое обозначение которого задается в качестве аргумента этой директивы. Возможно задание нескольких аргументов одновременно в виде списка адресов. В этом случае они должны быть отделены друг от друга запятыми. В качестве аргумента может выступать выражение, значение которого будет интерпретироваться как адрес.

Адрес поля можно задать с помощью директивы .LONG, однако для этой цели рекомендуется пользоваться директивой .ADDRESS, так как по этой директиве при хранении адресов данных или полей организуется дополнительная информация, необходимая на этапе компоновки модулей. Кроме того, использование директивы .ADDRESS в разделяемых образах порождает позиционно-независимый код. Формат директивы:

. ADDRESS список адресов

где список адресов — список символических имен или выражений, разделенных запятыми, которые интерпретируются Макроассемблером как адреса. Коэффициент повторений недопустим.

Пример.

```
LIST:   . ADDRESS PAGE, STRING, LETTER ; Список адресов трех полей.
PAGE:   BLKL      10
STRING: BLKW      15
LETTER: BLKB      20
```

.ASCI [I, C, D, Z] — директивы хранения строки символов в КОИ-8. Для определения строки символов в коде КОИ-8 в исходной программе можно использовать четыре директивы:
.ASCII — хранение строки в коде КОИ-8;
.ASCIC — хранение строки в коде КОИ-8 со счетчиком;
.ASCID — хранение строки в коде КОИ-8 с описателем строки;
.ASCIZ — хранение строки в коде КОИ-8 с завершающим нулем.

Формат всех четырех директив одинаков:

. ASCI [I, C, D, Z] строка

За каждой директивой следует строка графических символов, заключенная между одинаковыми ограничителями. Этими ограничителями могут быть любые символы кода КОИ-8, кроме пробела, символа табуляции, символа равенства (=), точки с запятой (;) и левой угловой скобки (<). Символ, используемый в качестве ограничителя, не должен появляться в самой строке. В качестве ограничителей могут употребляться алфавитно-цифровые символы, однако во избежание путаницы их употреблять не следует.

При обработке транслятором директив хранения строк происходит преобразование символов в восьмиразрядные значения кодов КОИ-8. При этом на один символ отводится один байт (кодирование символов приведено в приложении 1).

В строке может появляться любой символ, кроме символов нуля, возврата каретки и перевода формата. Однако эти символы могут быть представлены выражениями или символами пользователя, заключенными в угловые скобки вне ограничителей. При этом предварительно необходимо определить значения кода КОИ-8 для символов нуля, возврата каретки и перевода формата с помощью оператора прямого присваивания. В этом случае директивы хранения символов КОИ-8 запоминают восьмиразрядное двоичное значение, определенное оператором прямого присваивания. Используя выражение, в строку символов можно вставлять любые непечатные знаки.

Строки символов КОИ-8 могут продолжаться на несколько строк текста программы, однако последовательность символов, расположенная на одной строке текста программы, должна быть

ограничена с обоих концов, причем для каждой последовательности символов могут быть использованы различные пары ограничителей.

Если ограничительные знаки не составляют пару или используется запрещенный знак, то при трансляции в листинге на этом месте будет выдана ошибка.

Пример.

```
CR=13
LF=10
        . ASCIC ?HELLO!?  <CR><LF>! MY FRIEND!
        . ASCIZ /GO BACK!/
        . ASCIC @ список регистров@
        . ASCII  # привет, # <CR><LF>? будь здоров!?
```

.ASCII — директива хранения строки символов КОИ-8. Эта директива преобразует строку графических символов в последовательность байтов в КОИ-8. Каждый символ размещается в одном байте.

Пример.

```
        . ASCII   /состав!/                          ; Ограничитель/.
        . ASCII @  унарные операции +, —, ^ @        ; Ограничитель@.
```

.ASCIC — директива хранения строки символов КОИ-8 со счетчиком. Данная директива аналогична по своим функциям директиве .ASCII, за исключением того, что перед последовательностью байтов с восьмиразрядными кодами соответствующих символов включается дополнительный байт, в который записывается длина строки (в байтах). Эта длина соответствует количеству символов, указанных в директиве (по одному байту на символ), без учета дополнительного байта (счетчика).

Директива .ASCIC оказывается полезной при операциях пересылки (копирования) текста, поскольку счетчик указывает на длину текста, предназначенного для пересылки.

Пример.

```
CR=13                          ; Определение символа CR.
        . ASCIC /номер!/ <CR>  ; Эта строка с директивой .ASCIC эквива-
                               ; лентна комбинации следующих двух
                               ; строк.
        . BYTE  6              ; Счетчик, запоминающий один байт со
        . ASCII  /номер!/ <CR> ; значением 6, за которым следует строка
                               ; с директивой .ASCII.
```

.ASCID — директива хранения строки символов КОИ-8 с описателем. Данная директива выполняет ту же самую функцию, что и директива .ASCII, за исключением того, что перед последовательностью байтов с восьмиразрядными кодами соответствующих символов вставляется 32-разрядный описатель строки (рис. 5.1).

Как правило, описатель строки используется при вызове процедур.

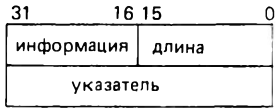


Рис. 5.1. Формат описателя строки:
в первых двух байтах описателя записывается длина строки; следующие два байта не используются (зарезервировано для будущих применений), в остальных — позиционно-независимый указатель на строку (4 байта)

Пример.

```
DATA—1: . ASCID    /строка—1/    ; Определение трех строк с описате-
DATA—2: . ASCID    /строка—2/    ; лями.
DATA—3: . ASCID    /строка—3/    ;
          PUSHAL   DATA—1        ; Поместить адреса описателей строк в
          PUSHAL   DATA—2        ; стек.
          PUSHAL   DATA—3        ;
          CALLS    #3, PROC      ; Вызов процедуры.
```

.ASCIZ — директива хранения строки символов КОИ-8 с завершающим нулем. Директива .ASCIZ выполняет ту же функцию, что и директива .ASCII, за исключением того, что при обработке транслятором директивы .ASCIZ в качестве последнего байта добавляется нулевой байт. Таким образом, при обработке строк, созданных при помощи директивы .ASCIZ, их длину и конец можно легко найти по нулевому байту.

Пример.

```
. ASCIZ ?устройство?          ; 10 символов в строке, данные зани-
                              ; мают 11 байт.
. ASCIZ /рабочая область/     ; 15 символов в строке, данные зани-
                              ; мают 16 байт.
```

.BYTE — директива хранения данных в виде байтов. Эта директива используется для задания последовательности данных, хранящихся в программе в формате байта. Аргументом директивы является список выражений, разделенных запятыми. По директиве

. BYTE выражение 1, выражение 2, ...

транслятор оценивает последовательно каждое выражение в формате длинного слова, затем значение каждого из них усекается до одного байта.

Значение первого усеченного выражения записывается по адресу директивы, в следующий байт записывается усеченное значение второго выражения и т. д. Так как запись осуществляется в байт, то значение любого выражения будет лежать в диапазоне от 0 до 255 для данных без знака и в диапазоне от —128 до +127 — для данных со знаком.

За каждым выражением может следовать необязательный коэффициент повторений, заключенный в квадратные скобки.

В этом случае аргумент директивы будет иметь следующий формат:

выражение 1 [выражение 2]

где выражение 1 — выражение, значение которого должно быть записано в байт;
[выражение 2] — коэффициент повторений, указывающий на число повторений запоминаемого значения.

Выражение, значение которого определяет коэффициент повторений, не должно включать в себя неопределенных символических имен и должно быть абсолютным. Квадратные скобки являются обязательной частью формата.

Пример.

```
. BYTE <20+20>*2              ; Запоминается значение 80.
. BYTE 15, 38—<12*2>, ^XA     ; Запоминаются 3 байта данных.
. BYTE 0                      ; Запоминаются 1 байт данных.
. BYTE 15, A+1 [8+2], ABC     ; Запоминается 12 байт данных.
```

При отсутствии аргумента в директиве. BYTE в программе резервируется поле длиной в 1 байт с нулевым значением. Аналогично нулевые байты появляются, если в поле операнда директивы подряд располагаются запятые, подразумевая разделение нулевых значений.

Пример.

```
. BYTE    5,, 8       ; Резервируются 3 байта со значениями 5,
                      ; 0 и 8.
```

.SIGNED__BYTE — директива хранения данных со знаком в виде байтов. Эта директива эквивалентна директиве .BYTE за исключением того, что она особо указывает на наличие знака у данных, запоминаемых в байте. Компоновщик может использовать данную информацию для проверки на условие переполнения.

Значение выражения, которое заносится в байт, должно лежать в диапазоне от —128 до +127. В противном случае при трансляции выдается диагностическое сообщение.

Пример.

```
. SIGNED—BYTE    ALPHA-DELTA     ; Данные
. SIGNED—BYTE    BETA [2*6]      ; хранятся
. SIGNED—BYTE    SIN, COS [4]    ; в виде байтов.
```

.WORD — директива хранения данных в виде слов. Директива по своему формату и функциям аналогична директиве .BYTE за исключением того, что она используется для задания данных, расположенных в последовательности слов. В соответствии с этим значение первого выражения из списка выражений записывается в слово по адресу самой директивы. Следующие значения заносятся в последовательно расположенные слова.

Каждое выражение обрабатывается сначала как выражение в формате длинного слова, затем оно усекается до слова. Зна-

чение выражения должно лежать в диапазоне от 0 до 65535 для данных без знака и в диапазоне от —32768 до +32767 для данных со знаком.

Пример 1.

```
. WORD      ^X3FF, SORT [3], ASIA, 25  ; Резервируется 6 слов.
```

Если в директиве отсутствует аргумент, транслятор резервирует слово с нулевым значением. Если в качестве аргумента присутствуют подряд повторяющиеся символы «,», то транслятор также резервирует слова с нулевым значением.

Пример 2.

```
. WORD,,,      ; Резервируется 4 слова с нулевым значением.
. WORD         ; Резервируется слово с нулевым значением.
```

.SIGNED_WORD — директива хранения данных со знаком в виде слов. Эта директива эквивалентна директиве .WORD, за исключением того, что она указывает на наличие знака у данных, запоминаемых в слове. Компоновщик может использовать данную информацию для проверки наличия условий переполнения. Директива .SIGNED_WORD оказывается полезной после инструкции выбора (CASE), при этом гарантируется, что смещение помещается в слове.

Значение выражения должно находиться в диапазоне от —32768 до +32767. В противном случае при трансляции выдается диагностическое сообщение.

Пример.

```
. SIGNED_WORD  LOC2—LOC1 ; Данные хранятся
. SIGNED_WORD  SORT[20]  ; в виде слов.
```

.LONG— директива хранения данных в виде длинных слов. Она используется для занесения в тело программы данных в виде последовательно расположенных длинных слов.

По своей функциональной направленности эта директива аналогична директиве .WORD, однако все операции по занесению значений выражений в память выполняются с длинными словами. Кроме того, каждое выражение в списке должно иметь значение, которое может быть записано в 32 разряда.

Пример.

```
DATA: . LONG  ^X4FFF,^A/MNK/ ; 2 длинных слова данных.
      . LONG  1@2            ; 100 (двоичное) хранится как длин-
                             ; ное слово.
      . LONG  1[64]          ; 64 данных в виде длинных слов, каж-
                             ; дое из которых хранит 1.
```

.OCTA — директива хранения данных в виде октаслов. Она используется при задании последовательности двоичных данных размером в 128 разрядов (16 байт). В отличие от предыдущих директив хранения данных (.BYTE, .WORD, .LONG) директива .OCTA может иметь только один аргумент в виде литерала или

символического имени. В соответствии с этим директива имеет два формата:

. OCTA символ
. OCTA литерал

где символ — символическое имя, определенное в программе. Оно хранится в октаслове в виде 32-разрядного значения с распространением знака;
литерал — любая постоянная величина. Может предшествовать один из операторов управления системой счисления ^B, ^O, ^D, ^X, который указывает соответственно на двоичное, восьмеричное, десятичное, шестнадцатеричное основание системы счисления. Текстовому значению предшествует оператор ^A, указывающий на строку символов в коде КОИ-8. По умолчанию принимается десятичная система счисления.

Пример.

```
.OCTA ^A/ARG=3. 1415926/ ; Каждый символ КОИ-8 хранится в
                         ; байте.
.OCTA ^B101010101010     ; Двоичное число хранится в окта-
                         ; слове
.OCTA ^X7FFFFF           ; Шестнадцатеричное число хранится
                         ; в октаслове.
```

.QUAD — директива хранения данных в виде квадрослов. Эта директива аналогична директиве .OCTA за исключением того, что она резервирует в программе квадрослово, их так же, как в директиве .OCTA, не может иметь в качестве аргумента выражение или список выражений.

Пример.

```
. QUAD ^B11111000001 ; Двоичное число хранится в квадрослове.
. QUAD ^044444       ; Восьмеричное число хранится в квадро-
                     ; слове.
. QUAD APRIL         ; APRIL имеет 32-разрядное значение с
                     ; распространением знака.
```

.F_FLOATING (.FLOAT) — директива хранения данных с плавающей запятой в формате F. По этой директиве осуществляется резервирование длинных слов, в которые заносятся данные с плавающей запятой формата F. Эти данные образуются как результат обработки соответствующих констант, записываемых в поле аргумента директивы. Так как директива работает с константами, то ее формат имеет вид:

. F_FLOATING список_литералов

где список_литералов — список констант в формате с плавающей константой не должны содержать унарных или бинарных операций, кроме унарного плюса и унарного минуса.

Пример.

```
. F_FLOATING 134. 5782, 74218. 34E20     ; Одинарная точность.
. F_FLOATING 134.2, 0. 1342E3, 1342E — 1 ; Во всех 3 случаях ге-
                                         ; нерируется число
                                         ; 134, 2.
. F_FLOATING —0. 75
. FLOAT      0, 25, 50
. F_FLOATING —0. 75, 1E38, —1. 0E — 37
```

При переводе заданной константы в число с плавающей запятой формата F может потребоваться выполнить операцию округления или усечения. Выбор соответствующей функции определяется с помощью директивы .ENABLE.

.D__FLOATING (DOUBLE) — директива хранения данных с плавающей запятой в формате D. Она используется для резервирования октаслов, в которые заносятся числа с плавающей запятой формата D. Формат директивы аналогичен формату директивы .F__FLOATING. Однако числа с плавающей запятой формата D всегда округляются (на них не оказывает действие директива .ENABLE). Кроме того, константам с плавающей запятой, входящим в список констант, не должна предшествовать операция формата с плавающей запятой (^F).

Пример.

```
. D__FLOATING     1000, 1. 0E3, 1. 0000000E—9      ; Список
. DOUBLE          3. 1415928, 1. 107153423828      ; констант.
. D__FLOATING     5, 10, 15, 0, 0. 5
```

.G__FLOATING — директива хранения данных с плавающей запятой в формате G. Эта директива аналогична директиве D__FLOATING за исключением того, что она обрабатывает константы и заносит в октаслова соответствующие числа с плавающей запятой формата G.

Пример.

```
. G__FLOATING 1. 75, 1. 0000E — 6, 2. 0E4    ; Список констант.
```

.H__FLOATING — директива хранения данных с плавающей запятой в формате H. Эта директива аналогична директиве D__FLOATING за исключением того, что она генерирует числа с плавающей запятой формата H и помещает их в квадрослова.

Пример.

```
. H__FLOATING 4814, 3. 0E16, 2. 00000E — 4    ; Список констант.
```

.PACKED — директива хранения в упакованном формате данных в виде десятичных строк. Она используется для задания десятичных чисел в упакованном формате, где две цифры занимают один байт. Представление десятичных чисел в упакованном формате в виде десятичных строк полезно при вычислениях, в которых требуется высокая точность. Для обработки данных этого типа предусмотрены специальные инструкции. Формат директивы:

```
. PACKED десятичная строка [, символ]
```

где десятичная строка — последовательность десятичных цифр, содержащая не более 31 цифры;
[, символ] — необязательное символическое имя, которому присваивается значение, равное количеству десятичных цифр в числе. При этом знак числа не учитывается в качестве цифры.

Пример.

```
. PACKED 657, AB        ; AB имеет значение 3.
. PACKED —2456, DF      ; DF имеет значение 4.
. PACKED 325
. PACKED +4160, C       ; C имеет значение 4.
```

5.5. ДИРЕКТИВЫ УПРАВЛЕНИЯ СЧЕТЧИКОМ АДРЕСОВ ПРОГРАММЫ

К директивам этой группы относятся следующие: .ALIGN, .EVEN, .BLK[A, B, D, F, G, H, L, O, Q, W], .ODD.

Эти директивы не только резервируют заданные области оперативной памяти, но и осуществляют настройку памяти на требуемую границу (байт, слово и т. д.).

.ALIGN — директива настройки счетчика адресов. В ряде случаев при написании программы на Макроассемблере требуется установить счетчик адресов на заданную границу памяти, так как его текущее значение определить затруднительно (например, если используется оператор прямого присваивания вида .=.+выражение). В этом случае с помощью директивы .ALIGN производится настройка счетчика адресов программы на определенную границу памяти. Настройка осуществляется увеличением текущего значения счетчика до ближайшей границы байта слова, длинного слова и т. д. Вид настройки определяется аргументом директивы, форматы которой:

```
. ALIGN целое [, выражение]
. ALIGN ключ [, выражение]
```

где целое — целое число в диапазоне от 0 до 9. Счетчик адресов программы настраивается на адрес, который равен значению числа два в степени данного целого числа;
ключ — одно из пяти ключевых слов, указывающих границу настройки. При этом счетчик адресов программы увеличивается и становится кратным соответствующему значению:

Ключевое слово	Размер в байтах
BYTE	2**0=1
WORD	2**1=2
LONG	2**2=4
QUAD	2**3=8
PAGE	2**9=512

Примечание. Символы ** означают степень числа;
выражение — выражение, значение которого заносится в каждый байт пропускаемой области памяти. Данное выражение не должно содержать никаких неопределенных символических имен и должно быть абсолютным.

Пример.

```
. ALIGN     BYTE, 1     ; Настроить на байт, заполнить 1.
. ALIGN     LONG, 0     ; Настроить на границу длинного слова, за-
                        ; полнить нулями.
. ALIGN     2, ^A/ /    ; Настроить на границу длинного слова, за-
                        ; полнить пробелами.
. ALIGN     PAGE        ; Настроить на границу страницы.
```

Если в момент обработки директивы текущее значение счетчика адресов программы уже находится на требуемой границе, то он не изменяется.

При использовании директивы .ALIGN следует помнить, что аналогичная настройка осуществляется при определении программной секции, в которой может встретиться эта директива. Настройка с помощью директивы имеет более низкий приоритет по сравнению с настройкой программной секции. Если они не совпадают, то при трансляции выдается ошибка.

Хотя большинство инструкций не требует настройки, кроме настройки адреса на границу байта, скорость выполнения программы будет выше, если для данных в формате слова, длинного слова и октаслова будет выполнена соответствующая настройка.

.BLK [A, B, D, F, G, H, L, O, Q, W] — директивы резервирования байтов памяти. В теле программы требуется резервировать определенную область памяти для хранения временных данных или для организации буферных пулов. Такое резервирование можно осуществить с помощью следующих десяти директив:

. BLKA — резервирует память для адресов (длинные слова);
. BLKB — резервирует память для данных в байтах;
. BLKW — резервирует память для данных в словах;
. BLKL — резервирует память для данных в длинных словах;
. BLKO — резервирует память для данных в октасловах;
. BLKQ — резервирует память для данных в квадрословах;
. BLKF — резервирует память для данных одинарной точности с плавающей запятой (длинные слова);
· BLKD — резервирует память для данных двойной точности с плавающей запятой (квадрослова);
· BLKG — резервирует память для данных формата G (квадрослова);
· BLKH — резервирует память для данных формата H (октаслова).

Каждая из директив резервирует память для данных соответствующего типа. Объем резервируемой памяти определяется аргументом директивы, которая имеет формат:

.BLK [A, B, D, F, G, H, L, O, Q, W] выражение

где выражение — величина, определяющая объем памяти, подлежащий ре-

зервированию. Все символические имена в этом выражении должны быть определены, а само выражение быть абсолютным. Если выражение опущено или его значение равно нулю, принимается значение по умолчанию равное 1.

Пример.

```
. BLKB    25        ; Область для 25 байт.
. BLKW    10        ; Область для 10 слов (20 байт).
. BLKL    10        ; Область для 10 длинных слов (320 байт).
. BLKO    0         ; Область для 1 октаслова (64 байта).
. BLKF    <3*2>     ; Область для 6 значений с плавающей запятой
                    ; формата F.
```

Рассматриваемые директивы производят такое же действие, как и оператор прямого присваивания .=. +выражение. Однако с точки зрения наглядности программы использование директив резервирования памяти является предпочтительным.

Резервирование памяти можно также осуществить, используя директивы . BYTE, . WORD и т. д., но, в отличие от них, директивы резервирования блоков памяти не определяют начальные значения байтов.

.EVEN — директива установки четности текущего счетчика адресов программы. Она обеспечивает установку счетчика адресов программы на четное значение. Это выполняется путем добавления к его значению единицы, если оно было нечетно. Если значение текущего счетчика адресов четное, то никаких действий не выполняется. Директива используется без аргументов и имеет следующий формат:

```
. EVEN
```

Появление любого аргумента в директиве приводит к появлению в листинге трансляции сообщения об ошибке.

.ODD — директива установки нечетности текущего счетчика адресов программы. Она выполняет функцию, обратную функции .EVEN, и обеспечивает установку счетчика адресов программы на нечетное значение. Это производится путем добавления единицы к текущему значению счетчика адресов программы, если это значение четно. Если оно нечетное, то никаких действий не выполняется. Директива используется без аргументов и имеет формат:

```
. ODD
```

5.6. ДИРЕКТИВЫ СЕКЦИОНИРОВАНИЯ ПРОГРАММЫ

Директивы .PSECT, .SAVE_PSECT (.SAVE) и .RESTORE_PSECT (.RESTORE) предоставляют в распоряжение программиста средства для организации модульного программирования, обеспечивая:

• разделение программы на отдельные модули-секции с последующей автономной или совместной трансляцией;

• определение атрибутов этих секций (например, секция данных или инструкций);

• организацию совместного использования секций несколькими процессами;

• защиту секции от несанкционированного использования или модификации другими программными модулями;

• организацию многомодульного программирования в рамках одной программы и др.

.PSECT — директива секционирования программы. С ее помощью определяются программная секция и ее атрибуты. Макроассемблер при обработке программы автоматически определяет две программные секции: абсолютную программную секцию и непоименованную программную секцию (именем считается пробел).

Определения символических имен, встречающиеся в программе до инструкций, данных или директивы .PSECT, помещаются в абсолютную программную секцию. Инструкции и данные, которые появляются до того, как определена первая поименованная программная секция, располагаются в неименованной программной секции. Директива .PSECT, не содержащая спецификации имени программной секции, относится к неименованной программной секции.

Имя программной секции определяется из первого аргумента директивы, которая имеет следующий формат:

. PSECT [имя, [список атрибутов]]

где имя — имя программной секции размером до 31 символа, среди которых может быть любой алфавитно-цифровой символ, символ подчеркивания (_) символ денежной единицы ($) и точка (.). При этом первый символ не может быть цифрой. Имена программных секций являются независимыми символическими именами по отношению к локальным, глобальным символическим именам и именам макрокоманд. Таким образом, одно и то же символическое имя может употребляться для обозначения локального символического имени и имени директивы;

список атрибутов — список, содержащий атрибуты программной секции и информацию о ее расположении.

Ниже перечислены атрибуты программной секции и их функции.

Имя атрибута	Функция
ABS	Признак абсолютной программной секции (неперемещаемой). Компоновщик присваивает данной секции абсолютный адрес. Программная секция с таким атрибутом может содержать лишь определения символических имен. (Сравните данный атрибут с его противоположностью — с атрибутом REL)
CON	Признак конкатенации. Программные секции с одинаковым именем и с одними и теми же атрибутами (включая атрибут CON) объединяются в одну программную секцию в порядке, в котором их обрабатывает компоновщик. Размещенное таким образом пространство виртуальных адресов равно сумме индивидуальных запросов на размещение

Имя атрибута	Функция
EXE	Признак выполняемости. Данная программная секция содержит инструкции. Атрибут EXE является средством отделения инструкций от данных, предназначенных только для чтения или для чтения и записи
GBL	Признак глобальной программной секции. Программные секции, имеющие одинаковые наименования и одни и те же атрибуты, включая атрибуты GLB и OVR, будут иметь одни и те же перемещаемые адреса в памяти
LCL	Признак локальной программной секции
LIB	Признак библиотечного сегмента. Он резервируется для будущего использования
NOEXE	Признак невыполняемости. Программная секция содержит только данные и не включает инструкций
NOPIC	Признак позиционной зависимости содержимого программной секции. Эта секция предназначена для фиксированного размещения в виртуальной памяти
NORD	Признак нечитаемости. Резервируется для дальнейшего использования
NOSHR	Признак запрета разделяемости. Программная секция резервируется для использования только инициирующим процессом во время выполнения
NOWRT	Признак запрета записи. Содержимое программной секции не может быть изменено во время ее выполнения
OVR	Признак перекрытия. Программные секции с одинаковым именем и с одними и теми же атрибутами, включая атрибут OVR, имеют в памяти один и тот же относительный базовый адрес. Размещенное пространство виртуальных адресов равно размещению, затребованному наибольшей из перекрывающихся программных секций
PIC	Признак позиционной независимости содержимого программной секции. Эта секция допускает перемещение, т. е. ей может быть назначена другая область памяти
RD	Признак читаемости. Он резервируется для дальнейшего использования
REL	Признак перемещаемой программной секции. Компоновщик назначает данной программной секции относительный базовый адрес. Программная секция может содержать инструкции или данные
SHR	Признак разделяемости программной секции. Программная секция допускает разделение между несколькими процессами во время ее выполнения. Данный атрибут назначается программным секциям, которые могут быть скомпонованы в разделяемый образ
USR	Признак пользовательского сегмента. Он резервируется для дальнейшего использования
VEC	Признак содержания вектора. Программная секция содержит вектор изменения режима, который указывает на привилегированный разделяемый образ. Атрибут SHR должен использоваться с атрибутом VEC
WRT	Признак записи. Содержимое программной секции во время выполнения может быть изменено

Когда Макроассемблер обрабатывает директиву .PSECT, в которой указана спецификация имени новой программной секции, он создает новую программную секцию, запоминает ее имя и атрибуты. Макроассемблер включает все данные и инструкции, которые следуют за директивой .PSECT, в эту программ-

ную секцию, до тех пор пока не встретится другая директива .PSECT. Пользователь может определить максимум 254 поименованные программные секции.

Перемещаемые программные секции начинаются с нулевого значения счетчика адресов программы. Если Макроассемблер встречает директиву .PSECT со спецификацией имени ранее определенной программной секции, он запоминает новые данные или инструкции после последней записи в ранее определенной программной секции. При этом счетчик адресов программы устанавливается на значение, которое имел счетчик адресов программы в конце ранее определенной программной секции. Программисту необязательно перечислять атрибуты программной секции при ее продолжении, однако любой указанный в перечислении атрибут должен быть тем же самым, который действовал в данной программной секции раньше. Продолжение программной секции не может содержать атрибуты, не согласующиеся с атрибутами, указанными в исходной директиве .PSECT.

Атрибуты, перечисленные в спецификации директивы .PSECT, обеспечивают только описание содержимого программной секции. Макроассемблер не проверяет, включает ли программная секция перечисленные атрибуты. Однако и Макроассемблер, и компоновщик проверяют, все ли программные секции с одинаковыми именами имеют одни и те же атрибуты. Если атрибуты программных секций не согласуются, выводится сообщение об ошибке.

Ниже приведены сведения об атрибутах программной секции, принятых по умолчанию, а также противоположно действующих атрибутах.

Атрибут по умолчанию	Противоположный атрибут
CON	OVR
EXE	NOEXE
LCL	GLB
NOPIC	PIC
NOSHR	SHR
RD	NORD
REL	ABS
WRT	NOWRT
NOVEC	VEC

Пример.

```
. PSECT   INSTR, EXE, LONG     ; Секция INSTR содержит только
                               ; инструкции с выравниванием на
                               ; границу длинного слова.
. PSECT   DATA, NOEXE, WORD    ; Секция DATA содержит только
                               ; данные с выравниванием на грани-
                               ; цу слова.
```

Ниже приведены атрибуты, принятые по умолчанию, в абсолютных и непоименованных программных секциях. (Имена данных программных секций содержат точки и пробелы.)

Имя программной секции	Перечень атрибутов
. ABS .	NOPIC, USR, CON, ABS, LCL, NOSHR, NOEXE, NORD, NOWRT, NOVEC, BYTE
. BLANK .	NOPIC, USR, CON, REL, LCL, NOSHR, NOEXE, RD, WRT, NOVEC, BYTE

В дополнение к атрибутам, приведенным ранее, в директиве .PSECT можно использовать ключевые слова (BYTE, WORD, LONG, QUAD, PAGE), обеспечивающие настройку программного модуля на определенную границу памяти. Их формат и назначение аналогичны ключевым словам директивы .ALIGN. Однако эта директива не может обеспечить выравнивания на границу по значению больше той, которая определена в директиве .PSECT.

По умолчанию директива .PSECT выполняет настройку программного модуля на границу байта.

Настройка на границу байта не вызывает действий транслятора, поскольку адрес любой выполняемой области памяти всегда равен целому числу байтов.

Выравнивание на заданные границы памяти целесообразно выполнять для повышения эффективности работы программного модуля, поскольку работа с данными, имеющими определенную длину и выравненными в соответствии с этой длиной, выполняется быстрее. При этом в директиве .PSECT следует указывать наибольший размер настройки адреса.

.SAVE_PSECT — директива сохранения программной секции. Эта директива сохраняет контекст текущей транслируемой программной секции в вершине стека контекста программной секции. Данный стек является внутренним стеком Макроассемблера и может содержать до 31 записи. Каждая запись в стеке включает в себя значение текущего счетчика адресов программы, а также максимальное значение, присвоенное счетчику в текущей программной секции. Если стек оказывается заполненным, когда встречается директива .SAVE_PSECT, то возникает ошибка. Формат директивы достаточно простой:

. SAVE_PSECT [LOCAL_BLOCK]

где [LOCAL_BLOCK] — необязательное ключевое слово, которое указывает на то, что вместе с контекстом программной секции должен быть сохранен и текущий блок локальных меток.

Пример.

Пусть задано макроопределение ERROR—MES.

```
. MACRO ERROR—MES, NEW—TEXT          ; Формирование сообщений об
                                     ; ошибках и таблицы указа-
                                     ; телей.
. IIF NOT—DEFINED MES—NUMBER MES—NUMBER=0
. SAVE—PSECT      LOCAL—BLOCK        ; Сохранить контекст и блок
                                     ; локальных меток.
        . PSECT   TEXT               ; Таблица сообщений об
                                     ; ошибках.
MES::   ASCIZ     /NEW—TEXT/
        . PSECT   POINTER            ; Таблица указателей.
        . ADRESS  MES
        . RESTORE—PSECT              ; Восстановить блок локаль-
                                     ; ных меток.
        PUSHL     # MES—NUMBER
        CALL      # 1, PRINT—MES      ; Печать сообщения об
                                     ; ошибке.
MES—NUMBER=MES—NUMBER+1
        . ENDM    ERROR—MES
```

В следующем фрагменте программы производится вызов макрокоманды.

```
        INCL      R4
        CLRL      R2
        BLBC      R1, 4⊙
        ERROR—MES <символ не найден>     ; ввести в таблицу
                                         ; сообщение «символ не най-
                                         ; ден»
4⊙:     RSB
```

Использование в данном примере директивы . SAVE—PSECT LOCAL—BLOCK означает, что локальная метка 4⊙ определена в том же самом блоке локальных меток, в котором выполняется ссылка к 4⊙. Если бы данная директива отсутствовала в макроопределении, то метка 4⊙ была бы не определена и было бы выдано сообщение об ошибке.

.RESTORE—PSECT — директива восстановления программной секции. Она восстанавливает программную секцию из вершины стека контекста программной секции, если она была сохранена с помощью директивы SAVE—PSECT. Этот стек является внутренним стеком Макроассемблера. Если данный стек оказывается пустым, когда задается директива .RESTORE—PSECT, то выводится сообщение об ошибке. При восстановлении программной секции по директиве .RESTORE—PSECT значение текущего счетчика адресов программы восстанавливается до значения, при котором данная программная секция была сохранена. Восстанавливается также блок локальных меток, если он был сохранен в программной секции. Директива используется без аргументов и имеет вид:

. RESTORE—PSECT

Директивы .RESTORE_PSECT и SAVE_PSECT особенно полезны в макрокомандах, в которых вводится определение программных секций.

Пример.

```
        . MACRO     DEF_SYMB                ; Определение символических
                                            ; имен и областей.
        . SAVE_PSECT                        ; Сохранение текущей секции.
        . PSECT     MACR1, NOWRT, WORD      ; Определение секции MACR1.
A=17
B=13
C=11
        . PSECT     MACR2, NOEXE, WORD      ; Определение секции MACR2.
        . BLKW      200
        . BLKB      100
        . RESTORE_PSECT                     ; Восстановление программной
                                            ; секции.
        . ENDM
```

В приведенном выше макроопределении сохраняется контекст текущей программной секции и вводится определение новой программной секции. Затем вновь восстанавливается сохраненная программная секция. Если бы макрокоманда не сохраняла и не восстанавливала контекст исходной программной секции каждый раз, когда вызывается эта макрокоманда, то исходная программная секция изменилась бы.

5.7. ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ВХОДНОЙ ТОЧКИ ПРОГРАММЫ

Директивы .ENTRY, .MASK, .TRANSFER определяют входную точку программного модуля и организуют его правильное оформление.

.ENTRY – директива определения входной точки программы. Эта директива определяет символическое имя точки входа в программу и маску сохранения регистров. Указанное символическое имя определяется как глобальное, при этом его значение равно значению адреса инструкции, следующей за директивой. Таким образом, выполнение программного модуля будет начато с этого адреса. Используемое в директиве символическое имя не должно далее появляться в программе в качестве метки. Входная точка может быть использована как адрес передачи управления программы.

Маска сохранения регистров используется для указания регистров, которые должны быть сохранены для вызова данной процедурой. Сохраненные регистры автоматически восстанавливаются, когда данная процедура возвращает управление в вызывающую программу. Формат директивы:

```
. ENTRY символ, выражение
```

где символ — символическое имя точки входа; выражение — маска сохранения регистров (2 байта) для точки входа. Данное выражение должно быть абсолютным и не должно содержать неопределенных символических имен.

Для установки разрядов в маске сохранения регистров удобно пользоваться оператором регистровой маски (^M).

Пример.

```
. ENTRY  SYM,^M<R2, R4, R7>  ; Процедура начинается
                             ; в данной точке.
                             ; R2, R4, R7 сохраняются
                             ; при выполнении инструкции
                             ; CALL.
```

Если выражение, указанное в директиве .ENTRY, имеет установленные разряды 0, 1, 12 и 13, то возникает ошибка трансляции. Данные разряды соответствуют регистрам R0, R1, AP и FP и используются операционной системой МОСВП.

Директиву .ENTRY целесообразно использовать для определения всех точек входа, включая адрес передачи управления программы, которая впоследствии будет вызываться инструкциями CALLS и CALLG. Программа, вход в которую осуществляется инструкциями BSB или JSB, не должна иметь маску сохранения регистров. Такая программа должна начинаться следующим образом:

```
START: первая инструкция
```

.MASK — директива маскирования. Она используется совместно с директивой .ENTRY: как только транслятор распознает директиву .MASK, он резервирует слово для маски сохранения регистров. Формат директивы:

```
. MASK символ [, выражение]
```

где символ — символическое имя, определенное в директиве . ENTRY;
выражение — выражение, значение которого определяет маску сохранения регистров.

Выражение может отсутствовать, в этом случае Макроассемблер указывает компоновщику скопировать в слово, зарезервированное по директиве .MASK, маску сохранения регистров, указанную в директиве .ENTRY. Если в директиве .MASK указан параметр «выражение», то Макроассемблер указывает компоновщику объединить в слове, зарезервированном по директиве .MASK, данное выражение и маску сохранения регистров, заданную в директиве .ENTRY. Компоновщик выполняет объединение масок согласно функции логического ИЛИ. Следовательно, в объединенную маску сохранения регистров включаются регистры, указываемые как в директиве .ENTRY, так и в директиве .MASK.

Пример использования директивы .MASK приведен при рассмотрении .TRANSFER.

.TRANSFER — директива передачи управления. Она определяет глобальное символическое имя как точку входа в разделяемый образ. После построения разделяемого образа компонов-

щик определяет данное символическое имя как значение счетчика адресов программы в точке директивы .TRANSFER, т. е. присваивает ему адрес первой инструкции, следующей за директивой.

После каждой директивы .TRANSFER должна появляться маска сохранения регистров (только для процедур) и инструкция перехода к первой инструкции программы.

Директива не требует памяти и не генерирует двоичный код. Она только формирует инструкции для компоновщика по определению символического имени при создании разделяемого образа. Формат директивы:

. TRANSFER символ

где символ — глобальное символическое имя, которое является входной точкой процедуры или программы.

Директива .TRANSFER может быть использована в процедурах, вызов которых осуществляется по инструкциям CALLS и CALLG. В таких случаях директива .TRANSFER используется совместно с директивами .ENTRY и .MASK. Переход к действующей программе должен выполняться по адресу точки входа плюс два. Добавление числа два к этому адресу нужно для того, чтобы пропустить два байта, относящиеся к маске сохранения регистров.

Пример.

```
. TRANSFER  ROUT—TEST
. MASK      ROUT—TEST, ^M<R4, R5>  ; Копирование входной
                                   ; маски по новому адресу.
BRW         ROUT—TEST+2            ; Переход к ROUT—TEST
  ...                              ; минуя входную маску.
. ENTRY     ROUT—TEST, ^M<R2, R3>  ; Точка входа и
  ...                              ; сохранение регистров
                                   ; R2 и R3.
RET
```

В данном примере по директиве .MASK происходит копирование входной маски программы в новый адрес передачи управления, указанный директивой .TRANSFER. Если данный фрагмент входит в состав разделяемого образа и осуществляется его вызов, то происходит сохранение регистров 2, 3, 4 и 5.

5.8. ДИРЕКТИВЫ УПРАВЛЕНИЯ СИМВОЛИЧЕСКИМИ ИМЕНАМИ

С помощью директив .DEBUG, .EXTERNAL (.EXTRN), .GLOBAL (.GLOBL) и .WEAK требуемые символические имена могут быть использованы вне программного модуля, где они определены, либо их определение обеспечивается в других модулях. Формат директив этой группы одинаков и состоит из имени директивы, за которым следует параметр — «список сим-

волов», т. е. список символических имен, разделенных запятыми.

.DEBUG — директива объявления символического имени отладки. Она определяет список символических имен, применяемых символическим отладчиком. В процессе интерактивной отладки эти имена могут использоваться для ссылки к соответствующим областям памяти или для проверки значений, которые были им присвоены.

Пример.

```
. DEBUG  PTR1, PTR2, DELTA  ; Эти символические имена
                            ; будут известны отладчику.
```

При трансляции символические имена, перечисленные в директиве .DEBUG, заносятся в таблицу символов соответствующего объектного модуля аналогично тому, как туда заносятся глобальные символы.

.EXTERNAL — директива объявления внешнего символического имени. Задание директивы .EXTERNAL указывает на то, что заданные в ней символические имена являются внешними, или, другими словами, эти символические имена определены в другом объектном модуле и не могут быть определены до этапа компоновки.

Пример.

```
.EXTERNAL  AB, CD, EF ; Внешние символические имена.
```

В ряде случаев задавать директиву .EXTERNAL нет необходимости. Так, если в директиве .ENABLE используется аргумент GLOBAL, то все неразрешенные ссылки будут помечены как глобальные и внешние. Однако если в модуле необходимо объявить символические имена, к которым производится обращение в текущем программном модуле, как внешние, то задавать директиву .EXTERNAL нужно обязательно. В противном случае, т. е. когда нет директивы .EXTERNAL и отсутствует аргумент GLOBAL в директиве .ENABLE, все неопределенные символические имена будут отмечены транслятором сообщением об ошибке.

.GLOBAL — директива объявления глобального символического имени. Эта директива показывает, что заданные в ней символические имена либо определены как глобальные в текущем модуле, либо определены как внешние в других модулях.

Пример.

```
. GLOBAL  SIN, COS  ; Эти символические имена
. GLOBAL  TAN       ; известны всем модулям,
                    ; участвующим в компоновке.
```

Директива .GLOBAL введена в Макроассемблер СМ1700 для обеспечения совместимости с Макроассемблером для ранних 16-разрядных моделей СМ ЭВМ. Глобальные определения рекомендуется вводить с помощью двойных символов двоеточия

и равенства, а внешние ссылки указывать в директиве .EXTERNAL.

.WEAK — директива подавления поиска символического имени. Она задает символические имена, которые определены как внешние в другом модуле либо как глобальные в текущем модуле. Она подавляет поиск соответствующих символических имен в любой библиотеке объектных модулей.

Если в директиве .WEAK указано символическое имя, не определенное в текущем модуле, то оно определяется как внешнее. При этом если компоновщик находит определение этого имени в другом модуле, то он его разрешает в соответствии с данным определением. Если компоновщик не находит внешнего определения, символическое имя приобретает нулевое значение. Компоновщик не сообщает об ошибке и не производит поиск указанного символического имени в библиотеке. Однако если модуль, извлеченный из библиотеки по другим причинам, содержит определение этого символического имени, компоновщик разрешает символическое имя в соответствии с использованием данного определения.

В случае если в директиве .WEAK указано символическое имя, которое определено в текущем модуле, то оно рассматривается как глобальное. Однако если данный модуль включен в библиотеку объектных модулей, то это символическое имя не заносится в таблицу символических имен библиотеки. При этом обращение к библиотеке во время компоновки с целью разрешения данного символического имени из другого модуля не влечет за собой его определения. Компоновщик его просто не находит.

Пример.

```
. WEAK JUNE, JULE, MAY—30
```

5.9. ДИРЕКТИВЫ И ПОДДИРЕКТИВЫ УСЛОВНОЙ ТРАНСЛЯЦИИ

К этой группе относятся директивы .IF, .IF—[TRUE, TRUE—FALSE], .ENDC и .IIF. При разработке программы часто формируется несколько взаимоисключающих секций, причем при выполнении программы используется только одна из них. Следовательно, в исполняемый модуль нет необходимости включать машинные коды неиспользуемых частей.

Директивы условной трансляции предоставляют программисту возможность включать или исключать в процессе трансляции отдельные части исходного текста программы, что позволяет получать несколько вариантов объектного кода программы. В зависимости от аргументов директив отдельные части текста,

не включаемые в текущий момент в трансляцию, могут присутствовать в листинге распечатки. Помимо улучшения наглядности и сопровождения программы это позволяет в ряде случаев проводить сравнительный анализ альтернативных вариантов исходного текста.

.IF — директива блока условной трансляции. *Блоком условной трансляции* называется последовательность исходных предложений программы, трансляция которых осуществляется только при соблюдении некоторого условия. Блок условной трансляции начинается директивой .IF, а заканчивается директивой .ENDC. Причем для каждой директивы .IF должна существовать соответствующая директива .ENDC. Если она появляется вне блока условной трансляции, то выдается сообщение об ошибке.

Директива .IF включает в себя правило проверки условия, которое должно быть выполнено, чтобы блок транслировался, и один или два аргумента, на которые действует это условие. Если проверка условия проходит успешно (т. е. результат — «истина»), то все предложения, заключенные между директивами .IF и .ENDC, транслируются. Если же проверка показывает, что условие не выполняется (результат — «ложь»), то эти предложения не транслируются. Исключения из данного правила могут быть, если используются поддирективы условной трансляции.

Формат директивы .IF, а также блока условной трансляции выглядит следующим образом:

```
. IF условие аргумент(ы)
блок условной трансляции
. ENDC
```

где условие — специальное условие, которое должно быть выполнено, чтобы данный блок условной трансляции был включен в трансляцию. В табл. 5.3 приведены условия, которые могут быть проверены директивами условной трансляции. Условие должно быть отделено от аргумента (аргументов) запятой, пробелом или символом табуляции;
аргумент(ы) — аргумент(ы) или выражение(я) для указанного правила проверки условия. Если аргументом является выражение, то оно не должно содержать неопределенных символических имен и должно быть абсолютным;
блок условной трансляции — последовательность предложений исходного текста программы, которая включается в трансляцию при выполнении условия, указанного в директиве.

Пример.

```
.IF EQ ABC=3  ; Если ABC=3, блок транслируется.
    ...       ; Блок условной трансляции.
.ENDC         ; Завершение блока условной трансляции.
```

Блоки условной трансляции могут быть вложенными друг в друга, т. е. в состав одного блока условной трансляции может входить другой блок условной трансляции. При этом вложенный блок условной трансляции будет транслироваться только

Таблица 5.3

УСЛОВИЯ, ДОПУСТИМЫЕ В ДИРЕКТИВАХ УСЛОВНОЙ ТРАНСЛЯЦИИ

Обозначение условия проверки		Обозначение альтернативного условия проверки		Тип аргумента	Количество аргументов	Условие, при котором блок транслируется (не транслируется)
Формат						
Полная форма	Краткая форма	Полная форма	Краткая форма			
EQUAL	EQ	NOT—EQUAL	NE	Выражение	1	Выражение=0 (или ≠0)
GREATER	GT	LESS—EQUAL	LE	Выражение	1	Выражение>0 (или ⩽0)
LESS—THAN	LT	GREATER—EQUAL	GE	Выражение	1	Выражение<0 (или ⩾0)
DEFINED	DF	NOT—DEFINED	NDF	Символическое имя		Символическое имя определено (или не определено)
BLANK	B	NOT—BLANK	NB	Макросредства	1	Аргумент пропущен (или не пропущен)
IDENTICAL	IDN	DIFFERENT	DIF	Макросредства	2	Аргументы идентичны (или различны)

Примечание. Условия BLANK, NOT—BLANK, IDENTICAL и DIFFERENT используются в определениях макрокоманд.

в том случае, если условия для внешнего и внутреннего блоков выполняются одновременно. Если условие для внешнего блока не выполняется, внутренние блоки игнорируются. Допустимая глубина вложения в Макроассемблере СМ1700 допускается до 31 уровня. Если осуществляется попытка превысить этот уровень, выдается сообщение об ошибке.

Пример.

```
; Пример директивы условной трансляции.
.IF EQUAL POINT+1    ; транслировать блок, если POINT+1=0
. . .
. ENDC
; вложенные директивы условной трансляции
. IF      DF ABC          ; Транслировать внешний блок, если симво-
 ...                      ; лическое имя ABC определено.
. IF      NDF BCD         ; Транслировать вложенный блок, если симво-
 ...                      ; лическое имя BCD
                          ; не определено, а символическое имя ABC
                          ; определено.
. ENDC
. ENDC
```

С помощью директив .SHOW и .NOSHOW можно включить (или исключить) в файл листинга трансляции программы блок условной трансляции, который в текущий момент не транслировался.

Трансляция по условию может также использоваться, когда требуется выделить отдельные блоки, расположенные в разных местах исходного текста программы. Вне зависимости от числа таких блоков принятие решения о включении их в трансляцию можно выполнить с помощью одного оператора прямого присваивания, определяющего значение какого-либо символического имени (флага разрешения трансляции).

.IF—[TRUE, FALSE, TRUE—FALSE]—поддирективы блока условной трансляции. Рассматриваемые ниже поддирективы используются только внутри блоков условной трансляции, ограниченных директивами .IF и .ENDC.

В языке Макроассемблера СМ1700 существуют три поддирективы блока условной трансляции:

Поддиректива	Функция
. IF—FALSE (.IFF)	Транслятор обрабатывает исходные предложения программы, следующие за директивой . IF—FALSE, если условие, проверенное при входе в блок условной трансляции, было ложным. Обработка продолжается до следующей поддирективы условной трансляции или до конца блока условной трансляции
. IF—TRUE (.IFT)	Транслятор обрабатывает исходные предложения программы, следующие за директивой .IF—TRUE, если условие, проверенное при входе в блок условной трансляции, было ложным. Обработка продолжается до следующей поддирективы условной трансляции или до конца блока условной трансляции
. IF—TRUE—FALSE (.IFTF)	Транслятор всегда обрабатывает исходные предложения программы, следующие за этой директивой, и включает их в объектный модуль до следующей поддирективы .IF—TRUE—FALSE условной трансляции или до конца блока условной трансляции. Эти исходные предложения включаются в программу независимо от того, является ли условие, проверенное при входе в блок условной трансляции, инстинным или ложным

Поддирективы условной трансляции используются без аргументов. Подразумеваемым аргументом поддиректив является результат проверки указанного условия при входе в блок условной трансляции. Они не обрабатываются во вложенных блоках условной трансляции, если условие внешнего блока не выполняется.

Блок условной трансляции, содержащий поддирективы условной трансляции, отличается от блока условной трансляции, не

содержащего их, тем, что если условие в директиве .IF не выполняется, то внутренний блок (блоки) условной трансляции не транслируется, тогда как поддирективы условной трансляции могут вызывать трансляцию блока.

Пример 1.

Предположим, что символическое имя SYM определено:

```
. IF DEFINED  SYM       ; Условие выполняется, поскольку SYM определено. Осуществляется трансляция блока условия.
   ...
. IF—FALSE              ; Поскольку условие предыдущей директивы .IF было истинным, следующие строки не транслируются.
   ...
. IF—TRUE               ; Трансляция возобновляется, так как условие проверки истинно: символическое имя SYM определено.
   ...
. IF—TRUE—FALSE         ; Следующие строки транслируются безусловно. Трансляция остатка блока условной трансляции.
   ...
. IF—TRUE               ; Трансляция остатка блока условной трансляции.
   ....
. ENDC
```

Пример 2.

Предположим, что символическое имя X определено, а символическое имя Y нет:

```
. IF DEFINED X          ; Условие истинно. Трансляция разрешена.
   ...                  ;
. IF DEFINED Y          ; Трансляция запрещается, так как символическое имя Y не определено.
   ...
. IF—FALSE              ; Трансляция возобновляется, так как условие при входе в блок было ложным.
   ...
. IF—TRUE               ; Трансляция запрещается. Символическое имя Y не определено.
   ...                  ; Следующие строки не транслируются.
. ENDC
. ENDC
```

Пример 3.

Предположим, что символическое имя A определено, а символическое имя B нет:

```
.IF DEFINED A           ; Условие истинно. Трансляция разрешена. Символическое имя A определено.
   ...
.IF—FALSE               ; Трансляция запрещается, так как условие при входе в блок было истинным.
   ...
. IF NOT—DEFINED B      ; Вложенная директива условной трансляции не обрабатывается, так как условие во внешнем блоке было ложным.
   ...
. ENDC
. ENDC
```

.ENDC — директива конца блока условной трансляции. Она завершает блок условной трансляции, начатый директивой .IF. Директива .ENDC используется без аргументов.

.IIF — директива непосредственной условной трансляции. Трансляцию по условию единственной команды можно выполнить с помощью директивы .IF. Однако в этом случае требуются три строки текста программы. Например, если нужно включить в трансляцию команду CLRL NAME и определен символ DMITRI, следует включить в программу следующие строки:

```
. IF        DF DMITRI
CLRL        NAME
. ENDC
```

Директива .IIF представляет собой средство для написания однострочного блока условной трансляции. Условие, которое подвергается проверке, и блок условной трансляции записываются полностью в той же самой строке, что и директива .IIF. При этом директива .ENDC не требуется. Формат директивы .IIF более сложный, чем директивы .IF:

. IIF условие [,] аргумент(ы), предложение

где условие — одно из допустимых условий, определенных в табл. 5.3 для директивы .IF. Данное условие должно отделяться от аргумента (аргументов) запятой, пробелом или символом табуляции, однако если первый аргумент является пробелом, то условие должно отделяться от аргумента (аргументов) запятой;
аргумент(ы) — аргумент, связанный с непосредственной директивой условной трансляции (см. табл. 5.3). Если аргументом является выражение, то оно не должно содержать никаких неопределенных символических имен и должно быть абсолютным. Аргумент (аргументы) должен отделяться от предложения запятой;
предложение — предложение, которое должно транслироваться, если условие выполняется.

С помощью директивы .IIF предыдущий пример будет иметь вид:

```
. IIF DEFINED      DMITRI,      CLRL NAME
```

Если символическое имя DMITRI определено в исходной программе, данная директива генерирует следующий код:

```
CLRL NAME
```

Примечание. Макроассемблер выводит сообщение об ошибке, если в директиве .IIF указывается:
- условие проверки, отличное от приведенных в табл. 5.3;
- недопустимый аргумент;
- пустой аргумент.

5.10. ДИРЕКТИВЫ ПЕРЕКРЕСТНЫХ ССЫЛОК .CROSS, .NOCROSS

.CROSS и .NOCROSS — директивы управления таблицей перекрестных ссылок. Таблица перекрестных ссылок в основном используется при отладке программного модуля и по умолчанию не создается. Таблица формируется только в том случае, если

в командной строке командного языка DCL для команды MACRO был задан квалификатор /CROSS_REFERENCE. Соответственно директивы .CROSS и .NOCROSS, которые могут управлять формированием таблицы, действуют только при задании этого квалификатора.

По умолчанию таблица перекрестных ссылок включает в себя определение каждого символического имени и все ссылки на него в модуле. С помощью директив .CROSS и .NOCROSS в таблицу могут заноситься только отдельные символические имена или подавляться занесение всех символических имен. Это определяется заданием директив соответствующих форматов:

. CROSS	. NOCROSS
. CROSS список имен	. NOCROSS список имен

где список имен — список разрешенных символических имен, разделенных запятыми.

Задание директивы .NOCROSS без списка символических имен запрещает формирование таблицы перекрестных ссылок для всех символических имен. Директива .CROSS без аргументов снова разрешает ее формирование. При этом любое определение символического имени или ссылка на него, появляющиеся в программе после директивы .NOCROSS без списка имен и до директивы .CROSS без списка имен, исключается из таблицы перекрестных ссылок.

Появление директивы .NOSROSS со списком имен подавляет формирование таблицы перекрестных ссылок для перечисленных в списке имен. Директива .CROSS со списком имен вновь разрешает формирование таблицы перекрестных ссылок для этих имен.

Пример 1.

```
        . NOCROSS               ; Прекращение формирования
                                ; таблицы перекрестных ссылок.
COPY1:  MOVW    ODG, OGP        ; Пересылка данных.
        . CROSS                 ; Продолжение формирования
                                ; таблицы перекрестных ссы-
                                ; лок.
```

Символическое имя COPYI и ссылки к символическим именам ODG и OGP не заносятся в таблицу перекрестных ссылок.

Пример 2.

```
        . NOCROSS ODG           ; Не включать имя ODG в
                                ; таблицу перекрестных ссы-
                                ; лок.
COPY2:  MOVW    ODG, OGP        ; Копирование данных.
        . CROSS   ODG           ; Продолжение формирования
                                ; таблицы перекрестных ссылок
                                ; для имени ODG.
```

Символическое имя COPY2 и ссылка к символическому имени OGP заносятся в таблицу перекрестных ссылок, а символическое имя ODG не заносится в эту таблицу.

При использовании директив .CROSS и .NOCROSS возможны ошибочные ситуации, которые могут возникать из-за попеременного использования этих директив со списком символических имен или без них. Так, директива .CROSS, заданная без списка имен, не вызывает продолжения формирования таблицы перекрестных ссылок для символов, указанных в списке символов директивы .NOCROSS.

Если формирование таблицы перекрестных ссылок запрещено для всех символов заданием директивы .NOCROSS, то появление директивы .CROSS со списком символических имен не разрешает занесение их в таблицу, до тех пор пока ее формирование не будет продолжено заданием директивы .CROSS без списка имен.

5.11. ДИРЕКТИВЫ ГЕНЕРИРОВАНИЯ ИНСТРУКЦИЙ

С помощью директив .OPDEF и .REF[1, 2, 4, 8, 16] можно определить новую инструкцию с требуемым кодом операции или изменить существующую для достижения наглядности листинга исходя из требований конкретного применения.

.OPDEF — директива определения кода операции. По этой директиве определяется символическое имя кода новой операции, который включается в таблицу кодов операций, определенных пользователем. Макроассемблер выполняет поиск заданного символического имени в этой таблице вместо поиска в таблице постоянных символических имен. С помощью директивы .OPDEF можно переопределить существующий код операции или создать новый. Формат директивы:

.OPDEF код выражение, список описателей операнда

где код — последовательность символов КОИ-8, определяющая имя кода операции. Данная последовательность может быть длиной до 31 символа и может содержать латинские буквы, цифры от 0 до 9, а также специальные знаки: знак подчеркивания (_), знак денежной единицы (⊙) и точку (·). Последовательность символов не может начинаться с цифры и не должна быть заключена в ограничители;

выражение — выражение, значение которого определяет код операции. Данное выражение не должно содержать никаких неопределенных символических имен и должно быть абсолютным. Оно может иметь значение в диапазоне от 0 до 65535 (шестнадцатеричное FFFF). Указанное выражение интерпретируется следующим образом:

Если 0<выражение<251, то выражение является однобайтным кодом операции; если выражение >255, то разряды выражения 0—7 являются первым байтом кода операции, а разряды 8—15 вторым байтом кода операции. Значения 252...255 не могут быть использованы, поскольку архитектура определяет их как начало двухбайтного кода операции.

Список описателей операнда — список, который указывает количество операндов и тип каждого из них. В списке допускается употребление до 16 описателей операнда. Описатели операндов перечислены в табл. 5.4.

Таблица 5.4

ОПИСАТЕЛИ ОПЕРАНДОВ

Тип доступа	Типы данных								
	B	W	L	Q	O	FF	FD	FG	FH
Адрес	AB	AW	AL	AQ	AO	AF	AD	AG	AH
Только чтение	RB	RW	RL	RQ	RO	RF	RD	RG	RH
Модификация	MB	MW	ML	MQ	MO	MF	MD	MG	MH
Только запись	WB	WW	WL	WQ	WO	WF	WD	WG	WH
Поле	VB	VW	VL	VG	VO	VF	VD	VG	VH
Переход	BB	BW	BL	BQ	BO	BF	BD	BG	BH

Описатель операнда имеет формат, в котором первый символ указывает на тип доступа к операнду, а второй — на тип данных операнда.

Пример.

```
. OPDEF MOVL3  ^XA9FF, RL, ML, WL      ; Определяется инструкция
                                       ; MOVL3, которая использует зарезервированный
                                       ; код операции FF.
. OPDEF DIVF2  ^X46, RF, MF            ; Переопределяются инструкции DIVF2 и MOVC5.
. OPDEF MOVC5  ^X2C, RW, AB, AB, RW, AB
. OPDEF CALL   ^X10, BB                ; Эквивалентно BSBB.
```

Наиболее целесообразно директиву .OPDEF использовать при создании «пользовательских» инструкций, которые выполняются по соответствующей микропрограмме, написанной самим пользователем. Данная директива позволяет программистам выполнять их микропрограммы в составе программ на языке Макроассемблера.

Кроме директивы .OPDEF, для переопределения кода операции можно использовать макрокоманду, поскольку поиск символического имени выполняется сначала в таблице имен макрокоманд, а затем в таблице кодов операций, определенных пользователем.

.REF [1, 2, 4, 8, 16] — директивы определения операнда. Макроассемблер имеет пять директив определения операндов, которые используются в макрокомандах для определения новых кодов операций:

Директива	Функция
. REF1	Генерирует операнд в виде байта
. REF2	Генерирует операнд в виде слова
. REF4	Генерирует операнд в виде длинного слова
. REF8	Генерирует операнд в виде квадрослова
. REF16	Генерирует операнд в виде октослова

Директивы .REF обеспечивают совместимость с Макроассемблером моделей более ранних версий. Следует помнить, что директива .OPDEF предоставляет более широкие функциональные возможности и она более проста в использовании по сравнению с директивами .REF, в связи с чем вместо директив .REF рекомендуется использовать директиву .OPDEF. Формат директивы:

. REF[1, 2, 4, 8, 16] операнд

где операнд — операнд в виде соответственно байта, слова, длинного слова, квадрослова или октаслова.

Пример.

```
. MACRO  MOVL3, A, B, C
. BYTE   ^XFF, ^XA9
. REF4   A                          ; Операнды представлены
. REF4   B                          ; в виде длинных слов.
. REF4   C
. ENDM   MOVL3
MOVL3    R0,@MN — 1, (R7)+[R10]
```

В данном примере используется директива .REF4, с помощью которой создается новая инструкция MOVL3. Для последней применяется зарезервированный код операции FF. В первом примере, приведенном для директивы . OPDEF, эта инструкция также определяется. Сравнение обоих вариантов показывает, что использование инструкции . OPDEF предпочтительнее.

5.12. ДИРЕКТИВА ДОПОЛНИТЕЛЬНОЙ КОМПОНОВКИ

С помощью директивы дополнительной компоновки .LINK программист может сформировать и в процессе трансляции передать компоновщику дополнительную информацию о последующей компоновке программного модуля. При его обработке совместно с другими модулями компоновщик будет использовать эту информацию только для работы с данным модулем.

Дополнительная информация, получаемая компоновщиком из директивы .LINK, позволяет не указывать соответствующие квалификаторы в команде LINK командного языка DCL. Более того, формат директивы практически такой же, как и формат команды:

. LINK "спец. файла" [/квалиф. [=(имя модуля [, ...])], ...]

где спец. файла — спецификация файла, представляющая ограниченную строку символов, которая определяет один или более входных файлов;
квалиф. — квалификатор файла, указывающий дополнительную информацию о файле.

В спецификации файла может быть задано несколько входных файлов:

. объектные файлы;

. библиотеки объектных модулей, которые необходимы для разрешения глобальных ссылок;

. разделяемые образы, которые должны быть включены в результирующий исполняемый модуль. Если указано несколько входных файлов, их имена

должны быть разделены запятыми. Знак «звездочка» (*) в спецификации файла недопустим. В любом случае вся строка символов, определяющая спецификацию файла, должна быть заключена в парные ограничители, среди которых может быть любая пара знаков, за исключением пробела, знака табуляции, знака равенства (=), точки с запятой (;), левой угловой скобки (<). (В строке, где приводится формат директивы, парные символы (") использованы как ограничители).

В качестве квалификаторов могут быть использованы:

/INCLUDE=(имя модуля[,...]) — определяет, что входной файл является объектной библиотекой или библиотекой разделяемого образа. Кроме того, специфицированные имена модулей являются обязательными для компоновщика. Если специфицируется более чем одно имя, то они разделяются запятыми и весь список имен заключается в круглые скобки. Имена модулей не могут содержать более 31 символа;

/LIBRARY — определяет, что соответствующий входной файл является библиотекой, в которой производится поиск модулей для разрешения неопределенных символов во входных файлах;

/SELECTIVE—SEARCH — указывает компоновщику о необходимости добавить к таблице символических имен из специфицированного файла только те глобальные символические имена, которые определены в файле и являются текущими неразрешенными. Если этот квалификатор не указан, то компоновщик включает все символические имена из специфицированного файла в таблицу глобальных символических имен;

/SHAREABLE — требует, чтобы компоновщик включил файл разделяемого образа.

Квалификатор может определяться не только полным, но и кратким форматом:

Полный формат	Краткий формат
/INCLUDE	/I
/LIBRARY	/L
/SELECTIVE—SEARCH	/SE
/SHAREABLE	/SH

Однако для удобства чтения, а также для совместимости с последующими версиями рекомендуется пользоваться полными форматами.

Пример.

```
. LINK "SYS⊙LIBRARY:MY—LIB" /INCLUDE=(MOD1, MOD2, MOD3)
```

При обработке данного предложения, содержащегося в исходном тексте какой-либо программы, Макроассемблер указывает компоновщику о необходимости скомпоновать файл с модулями MOD1, MOD2, MOD3 из библиотеки SYS⊙LIBRARY:MY—LIB.OLB (где SYS⊙LIBRARY — логическое имя для диска и каталога, в котором записан файл MY—LIB.OLB).

Это предложение эквивалентно компоновке файла с помощью команды LINK языка DCL:

```
⊙LINK MY—PROG, SYS⊙LIBRARY:MY—LIB/INCLUDE=(MOD1, MOD2,
MOD3)
```

Здесь MY—PROG определяет имя результирующего исполняемого образа.

Глава 6
МАКРОКОМАНДЫ

С помощью макрокоманд программист может заменить последовательность строк исходного текста программы одной строкой. Такая замена целесообразна, если последовательность достаточно часто встречается в тексте программы. В этом случае для соответствующей последовательности строк программы особым образом определяется имя (имя макрокоманды), которое записывается в программе вместо данных строк. Макроассемблер при обработке исходной программы заменяет это имя на ранее определенные строки программы (макроопределение).

Макроопределение напоминает описание подпрограммы и представляет собой описание некоторого текста (инструкций или данных), который включается в программу.

Если заменяемая последовательность строк при каждом ее появлении в тексте программы видоизменяется (например, изменяются названия обрабатываемых полей, регистров), то эти модификации учитываются путем указания для заменяемой последовательности формальных аргументов и задания соответствующих фактических аргументов.

Формальные аргументы используются только в пределах последовательности исходных строк и заменяются фактическими аргументами при вызове макрокоманды.

Вызов макрокоманды включает в себя ее имя, за которым могут следовать фактические аргументы. Макроассемблер производит замену строки, содержащей вызов макрокоманды на последовательность строк, которая находится в макроопределении данной макрокоманды. Далее Макроассемблер заменяет формальные аргументы, имеющиеся в макроопределении, на соответствующие фактические аргументы, указанные в макровызове. Этот процесс называется *расширением макрокоманды.*

По умолчанию расширение макрокоманды не распечатывается в листинге трансляции. Распечатка производится только в том случае, если указан аргумент EXPANSIONS в директиве .SHOW или квалификаторе /SHOW команды MACRO языка DCL. Во всех примерах, приведенных в данной главе, макрорасширение распечатывается, так как предполагается, что аргумент EXPANSIONS был указан.

В дальнейшем вместо терминов «определение макрокоманды», «вызов макрокоманды» и «расширение макрокоманды»

будут употребляться более удобные термины — *макроопределение*, *макровызов* и *макрорасширение*.

Основным назначением макросредств является расширение перечня машинных инструкций СМ1700 и представление транслятору дополнительных указаний на выполнение специфических действий. Кроме того, с помощью макрокоманд можно изменять функциональное назначение существующих инструкций в соответствии с требованиями конкретного приложения. В частности, система МОСВП включает набор системных макрокоманд, обеспечивающих обращение к соответствующим системным процедурам и таким образом расширяющих базовый набор инструкций.

6.1. ОПРЕДЕЛЕНИЕ И ВЫЗОВ МАКРОКОМАНД

Макрокомандой простейшей структуры является макрокоманда без аргументов. Макроопределение, соответствующее такой макрокоманде, имеет следующий формат:

```
. MACRO имя
тело макроопределения
. ENDM [имя]
```

где . MACRO — директива транслятору, указывающая на то, что дальше следует макроопределение;

имя — символическое имя макроопределения, которое является именем соответствующей макрокоманды. Это имя формируется в соответствии с правилами формирования символических имен в Макроассемблере. Определенное имя может встретиться в программе в качестве обычной метки. Учитывая алгоритм поиска по таблицам, это не приведет к ошибочной ситуации;

тело макроопределения — представляет собой фрагмент исходной программы, состоящий из последовательности инструкций или данных. Они включаются в программу вместо макрокоманды, имеющей такое же имя, как и для макроопределения;

. ENDM — директива транслятору, показывающая на конец макроопределения. За ним может быть приведено имя макроопределения. Если имя присутствует, то оно должно совпадать с именем макроопределения в директиве .MACRO; в противном случае будет выдана ошибка. Хотя указание имени не обязательно, рекомендуется это делать: для улучшения «читаемости» программы, а также для определения отдельных ошибочных ситуаций, особенно когда имеется несколько вложенных макроопределений.

Директива .ENDM не должна предваряться меткой. Хотя метка может быть у директивы .MACRO, однако такая конструкция нежелательна, особенно если используются вложенные макроопределения или блоки повторений.

С помощью комментария можно описать макроопределение, а также зафиксировать особенности его использования.

Пример.

```
. MACRO  DGOST  ; Макроопределение DGOST.
MOVA     C0, R0 ; Загрузка в регистры R0 и R1
MOVA     C1, R1 ; адресов полей C0 и C1.
. ENDM   DGOST  ; Конец макроопределения.
```

Макроопределения являются входными данными для транслятора и участвуют в формировании объектного кода программы. Встретив макроопределение, транслятор копирует его в специально отведенную область памяти, и оно хранится там до тех пор, пока не встретится соответствующая макрокоманда. После того как в исходном тексте программы все макрокоманды будут заменены на соответствующие макроопределения, транслятор их стирает, и они перестают существовать как некие «рабочие» единицы.

Для обращения к макроопределению его имя следует записать в строке программы так, как будто это мнемоника некоторой инструкции.

Пример.

```
    TSTW    FEB     ; В тело программы включено
    BGTR    10⊙     ; макроопределение DGOST.
    DGOST
10⊙: ...
```

После обработки этого фрагмента транслятором будет получен двоичный код программы, листинг которой выглядит следующим образом:

```
    TSTW    FEB     ; В тело программы включено
    BGTR    10⊙     ; макроопределение DGOST.
    MOVA    C0, R0
    MOVA    C1, R1
10⊙: ...
```

В более сложных макроопределениях иногда возникает необходимость досрочно завершить генерацию макрорасширения. Для указания логического конца макроопределения используется директива .MEXIT. Она бывает полезна при наличии директив условной трансляции и допустима в блоках повторений. Кроме того, при работе с вложенными макрокомандами она завершает текущее макроопределение аналогично директиве .ENDM (более детально все директивы обсуждаются в п. 6. 3).

6.2. АРГУМЕНТЫ МАКРОКОМАНД

Макрокоманды без аргументов достаточно редко используются при программировании. Более интенсивно они применяются при указании фактических аргументов. Вполне естественно, что макроопределение должно включать в себя после имени список формальных аргументов.

Поскольку макроопределение — это текст на Макроассемблере, формируемый на этапе трансляции, фактические аргументы, передаваемые макроопределению, не могут быть значениями соответствующих переменных. Они представляются лишь символическими именами и значениями своих адресов.

Если аргументы требуется задавать в директиве .MACRO, они записываются следующим образом:

. MACRO имя, список аргументов

где список аргументов — это аргументы макроопределения, отделенные друг от друга запятыми, пробелами или знаками табуляции. Обычно в качестве разделителя используется запятая. Список аргументов отделяется от имени, как правило, знаком табуляции, хотя можно использовать запятую или пробел.

В Макроассемблере не требуется, чтобы количество фактических аргументов точно соответствовало количеству формальных аргументов в макроопределении. Количество фактических аргументов в макровызове может быть равно или меньше количества формальных аргументов в макроопределении. В этом случае недостающие аргументы интерпретируются как пробелы. Но если количество фактических аргументов больше, чем количество формальных аргументов, лишние аргументы игнорируются, и Макроассемблер выдает сообщение об ошибке.

Обычно между формальными и фактическими аргументами установлено строгое позиционное соответствие, т. е. первый фактический аргумент в макровызове замещает первый формальный аргумент в макроопределении во всех точках, где он только появляется. Однако в ряде случаев (это рассматривается далее) такое строгое позиционное соответствие может не соблюдаться.

Рассмотрим пример макроопределения PAGE, в котором используются три формальных аргумента.

```
. MACRO  PAGE   ARG1, ARG2, ARG3
. LONG   ARG1   ; ARG1 первый аргумент.
. WORD   ARG3   ; ARG3 третий аргумент.
. BYTE   ARG2   ; ARG2 второй аргумент.
. ENDM   .PAGE
```

Следующие два примера иллюстрируют возможные макровызовы и соответствующие макрорасширения макроопределения PAGE.

Пример 1.

```
PAGE     A, B, C  ; Макровызов.
```

Макрорасширение:

```
. LONG   A        ; A первый аргумент.
. WORD   C        ; C третий аргумент.
. BYTE   B        ; B второй аргумент.
```

Пример 2.

```
PAGE X, X—Y, 70   ; Макровызов.
```

Макрорасширение:

```
. LONG   X        ; X первый аргумент.
. WORD   70       ; 70 третий аргумент.
. BYTE   X—Y      ; X—Y второй аргумент.
```

В следующем примере одно и то же символическое имя используется как метка и как имя макрокоманды:

```
FORT: MOVL  A, R0     ; Символичеокое имя FORT используется как метка.
      BEQL  FORT      ; Условный переход на метку FORT.
      ...
      FORT  PARAM, A, <CMP A, L>  ; Символическое имя; FORT
                      ; используется как имя макрокоманды.
```

В этом примере макрокоманда используется с тремя фактическими аргументами. Третий аргумент включает в себя разделительный символ (,), поэтому он заключен в угловые скобки.

Угловые скобки могут также применяться и в макроопределениях, ограничивая формальные аргументы, если они включают разделительные символы. Однако чаще всего они встречаются в макровызовах.

6.2.1. ЗНАЧЕНИЯ АРГУМЕНТОВ ПО УМОЛЧАНИЮ

В ряде случаев отсутствие фактического аргумента можно предусмотреть. В частности, как уже упоминалось, отсутствующий аргумент интерпретируется как пробел. Этим можно воспользоваться для стирания значения фактического аргумента, несмотря на выдаваемое сообщение об ошибке.

Более корректно отсутствие аргумента в макровызове можно предусмотреть указанием в макроопределении его значения по умолчанию.

Значением фактического аргумента по умолчанию является то значение, которое определено в самом макроопределении в списке аргументов. В директиве .MACRO оно указывается следующим образом:

имя_формального_аргумента = значение_по_умолчанию

Таким образом, если в макровызове отсутствует какой-либо аргумент, но соответствующий ему формальный аргумент определен по умолчанию, то значение по умолчанию принимается за значение фактического аргумента. Однако если этот фактический аргумент определен, то, несмотря на то, что есть значение соответствующего формального аргумента по умолчанию, в макроопределении будет использовано значение, заданное в макрокоманде. Пусть, например, определено макроопределение FINE, в котором указывается один аргумент со значением по умолчанию.

```
. MACRO  FINE    A, B = 10,C
. BYTE   A
. WORD   B
. QUAD   C
. ENDM   FINE
```

Тогда, если используется макрокоманда FINE с таким списком фактических аргументов:

```
FINE     18,, ABC; Второй аргумент не указан.
```

то получается следующее макрорасширение:

```
. BYTE    18
. WORD    10
. QUAD    ABC
```

Однако если указаны значения всех трех фактических аргументов:

```
FINE      18, 30, 40; Указаны все аргументы.
```

то получаем макрорасширение со значениями всех трех аргументов:

```
. BYTE    18
. WORD    30
. QUAD    40
```

Если фактический аргумент опущен, то даже если есть соответствующий формальный аргумент в макроопределении со значением по умолчанию, соответствующие разделители (например, запятые) должны присутствовать в списке фактических аргументов, отмечая отсутствие этого аргумента.

6.2.2. КЛЮЧЕВЫЕ АРГУМЕНТЫ

Позиционная зависимость формальных и фактических аргументов хорошо отслеживается, если макроопределение (макрокоманда) имеет незначительное число аргументов. В этом случае достаточно просто запомнить позицию аргумента и соответствующую ему функцию в макрорасширении. С увеличением числа аргументов становится все сложнее безошибочно определять в макрокоманде соответствующие позиции. Этой неприятной ситуации можно избежать, если использовать в макроопределении ключевые аргументы, которые являются позиционно-независимыми.

Ключевые аргументы позволяют указывать аргументы в макровызове в произвольном порядке, однако при этом в макровызове должны указываться те же самые имена формальных аргументов, которые присутствуют в макроопределении. Пусть, например, в макроопределении PAGE указаны три аргумента:

```
. MACRO   PAGE    A1, A2, A3
. LONG    A1
. WORD    A2
. BYTE    A3
. ENDM    PAGE
```

Тогда, указывая в макровызове ключевые аргументы

```
PAGE    A3=5, A2=17, A1=COS
```

получим следующее макрорасширение:

```
. LONG     COS
. WORD     17
. BYTE     5
```

При использовании ключевых аргументов нежелательно применение позиционно-зависимых. Смешение типов может привести к непредсказуемым результатам. Так, например, если в одной макрокоманде смешаны позиционно-зависимые и ключевые аргументы, то только позиционно-зависимые аргументы соответствуют по позиции формальным, а ключевые аргументы игнорируются.

6.2.3. АРГУМЕНТЫ В ВИДЕ СТРОКИ СИМВОЛОВ

Указание в качестве фактического аргумента символического имени метки, так же как использование конкретных числовых значений, не вызывает особых трудностей. В случае если аргументом является строка символов, то может возникнуть проблема использования ограничительных символов в этой строке.

Если фактический аргумент представляет собой строку символов, в состав которой входят такие символы, как например, запятая, пробел, знак табуляции, которые Макроассемблер воспринимает как разделители аргументов, то данная строка символов должна находиться между другими ограничителями, например парными угловыми скобками (<>). Однако при этом может возникнуть другая проблема, если в состав строки символов должны входить символы угловых скобок. В общем виде Макроассемблер интерпретирует в качестве ограничителей любые символы, следующие за символом «уголок вверх» (^). Таким образом, в случае, если угловые скобки должны использоваться в качестве элементов самой строки символов, программист может применить формат ограничителя с уголком вверх.

В случае если строка символов включает в себя точку с запятой, то эта строка символов также должна быть заключена в ограничители, в противном случае точка с запятой будет обозначать начало поля комментария.

Строка символов, заключенная между ограничителями, не может быть продолжена на следующую строку исходной программы.

Рассмотрим примеры аргументов макрокоманд, заключенных между ограничителями.

```
<январь, февраль, март — первый квартал>
<CLEAR: CLRW @(R2)+[R4]  ; очистка>
^%STRING IS FIRST<LAST> FOR AREA%
^?STACK POINTER <SP OR R14>?
```

В последних двух примерах уголок вверх указывает на то, что символ процента (%) и символ вопроса (?) являются ограничителями. Уголок вверх указывается только один раз с левым ограничителем.

Заключенный в ограничительные символы фрагмент текста рассматривается Макроассемблером как один фактический аргумент, который должен соответствовать определенному формальному аргументу. Если ограничители опущены, то транслятор разбивает фрагмент на отдельные поля, разделенные обычными ограничителями (запятые, пробел, знак табуляции) и каждое поле рассматривается как последовательное задание фактических аргументов. При этом транслятор пытается в соответствующем макроопределении найти соответствующие позиционные формальные параметры.

Рассмотрим пример макроопределения TRACT, у которого один формальный аргумент:

```
. MACRO   TRACT STRING
. ASCII   /STRING/
. ASCII   /STRING/
. ENDM    TRACT
```

Следующие два макровызова демонстрируют фактические аргументы, один из которых указан с ограничителями, другой — без ограничителей (в этом случае генерируется ошибка).

Макровызов 1:

```
TRACT     <SYSTEM>
```

Макрорасширение:

```
. ASCII   /SYSTEM/
. ASCII   /SYSTEM/
```

Макровызов 2:

```
TRACT     SYSTEM
%MACRO-E-TOOMNYARGS, TOO MANY ARGUMENTS IN MACRO CALL
```

При обработке второго макровызова Макроассемблер выдает сообщение: «в макровызове слишком много аргументов», так как интерпретирует его как макровызов, содержащий шесть фактических аргументов, разделенных пробелом, а не один аргумент с пробелами.

Использование двух символов ограничения строки (парные угловые скобки и символ «^»), помимо обычных ограничителей в Макроассемблере, дает возможность формировать практически любую строку символов в качестве фактического аргумента макрокоманды. Для передачи в строке символов числа с указанием операции управления основанием системы счисления необходимо, чтобы аргумент целиком был заключен между ог-

раничителями из парных угловых скобок, иначе Макроассемблер будет интерпретировать операцию управления основанием системы счисления как ограничитель.

Приведем примеры аргументов макрокоманд, которые заключены между ограничителями, поскольку они содержат операции управления основанием системы счисления:

```
<^B11001100>
<^XFFFF>
<^F2. 4>
```

6.2.4. АРГУМЕНТЫ ВЛОЖЕННЫХ МАКРОКОМАНД

Если макроопределение содержит вызов другой макрокоманды, то такая макрокоманда называется *вложенной*. Глубина вложения макрокоманд практически неограниченна. Появление вложенных макрокоманд в макроопределении влияет на правила задания аргументов, которые передаются вложенной макрокоманде через аргументы макроопределения. В этом случае передаваемые аргументы должны заключаться в ограничители на каждом уровне вложения. Так, если в рамках одного макроопределения вызывается макрокоманда и аргумент передается в виде строки символов, то строку необходимо заключить в угловые скобки так, чтобы во вторую макрокоманду полная строка передавалась как один аргумент.

В следующем примере макроопределение BOLT содержит вызов макрокоманды TRACT.

```
. MACRO  BOLT          ARG1, ARG2, STRING
. WORD   ARG1
. WORD   ARG2
TRACT    <STRING>  ; Аргумент заключен в ограничители.
. ENDM   BOLT
```

Здесь аргумент в макровызове TRACT заключен в угловые скобки, несмотря на то, что фактический аргумент не содержит ограничительных символов. Это делается по той причине, что фактический аргумент в макровызове TRACT одновременно является формальным аргументом в макроопределении BOLT и будет замещаться фактическим аргументом, который может содержать ограничительные символы.

В приведенном ниже примере показан вызов макрокоманды BOLT и соответствующее макрорасширение, из которого в свою очередь производится вызов макрокоманды TRACT.

```
BOLT     1945, 1988, <43 YEARS>     ; Макровызов.
. WORD   1945
. WORD   1988
TRACT    <43 YEARS>                 ; Вложенный макровызов.
. ASCII  /43 YEARS/
. ASCII  /43 YEARS/
```

Для передачи аргументов в виде строки символов во вложенных макрокомандах может применяться другой метод, который заключается в использовании вложенных ограничителей. В этом случае каждый уровень вложения может употреблять одну пару угловых скобок для передаваемого аргумента, но при этом макровызовы, входящие в состав макроопределения, не должны содержать ограничителей. Макроассемблер удаляет внешнюю пару ограничителей на каждом уровне вложения при расширении вложенных макрокоманд.

Описанный выше метод передачи аргументов с использованием вложенных ограничителей рекомендуется применять только тогда, когда программист четко знает глубину вложения макрокоманд, иначе могут возникнуть сложности в определении количества угловых скобок.

В следующем примере макроопределение BOLT содержит вызов макрокоманды TRACT, аргумент которого не заключен в угловые скобки.

```
. MACRO   BOLT    ARG1, ARG2, STRING
. WORD    ARG1
. WORD    ARG2
TRACT     STRING
. ENDM    BOLT
```

Ниже приведен пример вызова макрокоманды BOLT.

```
BOLT      1983, 1988, <<5 YEARS>>     ; Макровызов.
. WORD    1983
. WORD    1988
TRACT     <5 YEARS>                   ; Макровызов.
. ASCII   /5 YEARS/
. ASCII   /5 YEARS/
```

Здесь аргумент в макровызове TRACT не был заключен в ограничители, однако при вызове в макрорасширении он оказывается заключенным в ограничители, поскольку аргумент в виде строки символов в макровызове BOLT заключен в двойные вложенные ограничители.

Помимо вложенных макрокоманд Макроассемблер допускает использование вложенных макроопределений, т. е. когда одно макроопределение полностью включает в себя другое макроопределение. Внутреннее макроопределение не действует до тех пор, пока не будет вызвана соответствующая макрокоманда, например:

```
.MACRO    PART1   ARG1, ARG2          ; Внешнее
...                                   ; макроопределение.
.MACRO    PART2   ARG3, ARG4, ARG5    ; Внутреннее
...                                   ; макроопределение.
. ENDM
. ENDM
```

Из этого примера следует, что вызов макрокоманды PART2 не может быть осуществлен до вызова макрокоманды PART1, поскольку только при первом макрорасширении PART1 транслятор получит макроопределение для макрокоманды PART2.

6.2.5. КОНКАТЕНАЦИЯ АРГУМЕНТОВ

Для повышения гибкости в использовании макрокоманд их аргументы могут быть соединены либо с константами — строками символов, либо с другими аргументами.

Символ «апостроф» (') определяет операцию конкатенации строковых значений аргументов макроопределения в единый неделимый текст.

Апостроф может либо следовать за именем формального аргумента, либо предшествовать ему. В последнем случае при расширении макрокоманды текст до апострофа объединяется с фактическим аргументом. Если апостроф следует за именем формального аргумента, то при расширении макрокоманды фактический аргумент объединяется с текстом, следующим за апострофом. Например, пусть задано макроопределение FAMILY:

```
        . MACRO   FAMILY A1, A2, A3, A4, L
A1''A2: MOV'L     A3, A4
        . ASCII   /A1' — 1/
        . ASCII   /A2' — 2/
        . ENDM    FAMILY
```

Вызов макрокоманды FAMILY с аргументами:

```
        FAMILY    A, B, ONE, TWO, W
```

Макрорасширение:

```
AB:     MOVW      ONE, TWO
        . ASCII   /A — 1/
        . ASCII   /B — 2/
```

В первой строке макрорасширения произошло объединение текста MOV со значением фактического аргумента W, в результате чего тип пересылки содержимого из поля ONE в поле TWO определился как пересылка слова. Во второй и третьей строках сформировались текстовые аргументы директивы .ASCII путем объединения значения фактического аргумента и текста, который следовал за ним.

В обоих случаях символ ('), который предшествовал аргументу или следовал на ним, при макрорасширении удаляется, а формальный аргумент заменяется фактическим аргументом.

Для конкатенации двух формальных аргументов необходимо использовать два последовательных апострофа. С помощью

этого приема сформирована метка в первой строке предыдущего макрорасширения. Использование двух символов (′′) необходимо потому, что при каждом появлении этого символа в тексте он отбрасывается. В частности, рассмотрим пример того, как транслятор обрабатывает эти символы.

```
        .MACRO   CMND   INST, SIZE, NUM
CMN′ NUM′:
        INST′′SIZE     R′NUM
CMN′ NUM′Z:
        ENDM     CMND
```

Для конкатенации двух формальных аргументов INST и SIZE использованы два последовательных апострофа.

Если для данного макроопределения используется макровызов

```
        CMND   CLR, W, 2
```

то получим следующее макрорасширение:

```
CMN2:   CLRW   R2
CMN2Z:
```

Процедура формирования меток достаточно проста и состоит из конкатенации текста и значения фактического аргумента. Однако при обработке оператора макроопределения

```
        INST′′SIZE  R′ NUM
```

происходит следующая последовательность действий. При обработке транслятором символа (′) происходит оценка формального аргумента INST, который заменяется на фактическое значение и обнаруженный символ (′) отбрасывается. Сканирование возобновляется со следующего символа, которым опять является символ (′). Этот символ также отбрасывается, но заставляет провести оценку следующего формального аргумента, который после замены на значение фактического аргумента соединяется с предыдущим.

6.2.6. СИМВОЛЬНОЕ ПРЕДСТАВЛЕНИЕ ЧИСЛОВЫХ АРГУМЕНТОВ

Как было указано выше, если в качестве фактического аргумента в макрокоманде указано символическое имя, то в соответствующее макрорасширение передается само имя, а не его числовое значение. Однако если перед символическим именем в вызове макрокоманды используется обратная косая черта (\), то в макрорасширении может быть передано числовое значение аргумента в символьном виде. Числовое значение аргумента берется в текущем основании. Макроассемблер в этом случае передает в макрокоманду символы, представляющие десятичное

значение символического имени. Например, если символическое имя NUMBER имеет значение 5 и указан фактический аргумент \NUMBER, то в макрокоманду передается строка из одного символа 5, а не символическое имя NUMBER.

Особенно полезной передача числового значения символического имени оказывается при создании новых символических имен совместно с операцией конкатенации аргументов (/). Кроме того, это позволяет решить задачу, которая возникает при формировании последовательности инструкций, оперирующих с содержимым регистра, чей номер неизвестен до момента формирования соответствующего макрорасширения.

Рассмотрим макроопределение SPRING для передачи числовых значений символов.

```
. MACRO   SPRING, ARG1, ARG2=⌢?⌢M<>?
. ENTRY   DAY' ARG1, ARG2
. ENDM    SPRING
```

Ниже приведены возможные макровызовы и макрорасширения для макроопределения SPRING.

```
MARCH=8
          SPRING \MARCH                    ; Макровызов.
          Макрорасширение:
          . ENTRY DAY8, ⌢M<>
MARCH=MARCH+23
          SPRING \MARCH, ⌢?⌢M<R2, R3>?; Макровызов.
          Макрорасширение:
          . ENTRY DAY31, ⌢M < R2, R3>
```

6.2.7. АВТОМАТИЧЕСКИ СОЗДАВАЕМЫЕ ЛОКАЛЬНЫЕ МЕТКИ

При частом вызове одной и той же макрокоманды, макроопределение которой включает различные метки, может возникнуть проблема их уникального определения во всех макрорасширениях. В противном случае при каждом обращении к макрокоманде в макрорасширении будут сформированы одинаковые метки, что приведет к ошибочной ситуации. Формирование таких меток может быть выполнено с помощью передачи метки в качестве фактического аргумента или, например, с помощью операции конкатенации аргументов и передачи числовых значений в макрорасширение, как, например, это показано в следующем примере.

Пусть заданы два макроопределения: FIND и METKA.

```
. MACRO  FIND    A, B ; Макроопределение
                      ; FIND.
METKA    A, \B        ; Вложенная макрокоманда.
B=B+1
. ENDM   FIND
. MACRO  METKA   A, B ; Макроопределение
                      ; METKA.
```

```
A'' B:  . BLKW   \B
        . ENDM   METKA
```

Если в программе происходит обращение к макрокоманде,

```
        Z=1
        FIND     Y, Z
```

то она будет транслироваться в следующую строку:

```
Y1:     .BLKW    1
```

При повторном вызове этой макрокоманды получим:

```
Y2:     .BLKW    2
```

и т. д.

Однако, как уже отмечалось ранее, в программе целесообразно использовать по возможности локальные метки вместо символических. При этом следует помнить, что явное задание локальных меток в макроопределении также может привести к их многократному определению, если вызов макрокоманды осуществляется в пределах одного блока локальных меток.

Данная задача решается с помощью так называемых автоматически создаваемых локальных меток. Они представляют собой символические имена вида N⊙, где N — целое десятичное число в диапазоне от 30000 до 65535 (метки в диапазоне от 1⊙ до 29999⊙ используются программистом).

Каждый раз, когда Макроассемблер создает новую локальную метку, он производит увеличение числовой части имени этой метки на 1. Следовательно, в диапазоне от 30000⊙ до 65535⊙ не должно быть никаких определяемых пользователем локальных меток.

Чтобы определить автоматически создаваемую локальную метку при каждом обращении к макроопределению, необходимо ее поместить в конце списка формальных аргументов и указать символ «знак вопроса» (?) перед ней. Таким образом, если перед формальным аргументом стоит символ «знак вопроса» (?), то этот аргумент рассматривается транслятором при макрорасширении как метка. При расширении макрокоманды Макроассемблер создает новую локальную метку в том месте, где указан формальный аргумент, но только в том случае, если пропущен соответствующий фактический аргумент. Если соответствующий фактический аргумент указан, Макроассемблер производит его подстановку вместо формального аргумента.

Автоматически создаваемые локальные метки могут использоваться не более чем для 31 формального аргумента, указанного в директиве .MACRO. Кроме того, эти метки не могут сочетаться с позиционно-зависимыми фактическими аргументами и с фактическими аргументами, представленными ключевыми словами.

В следующем примере в макроопределении LOCAL—LAB определяются автоматически создаваемые локальные метки.

```
        . MACRO LOCAL—LAB  M, ?LAB
        TSTW    M
        BGTR    LAB
        MOVW    M, R0
LAB:    . ENDM  LOCAL—LAB
```

В последующих примерах макрорасширений приведенного выше макроопределения показывается, как используются локальные метки, определяемые пользователем, и автоматически создаваемые локальные метки.

```
        LOCAL—LAB R1        ; Макровызов с автоматически создава-
                            ; емой локальной меткой 30000⊙.
        TSTW      R1
        BGTR      30000⊙
        MOVW      R1, R0
30000⊙:
        ...
        LOCAL—LAB NEXT      ; Макровызов с автоматически создава-
                            ; емой локальной меткой 30001⊙.
        TSTW      NEXT
        BGTR      30001⊙
        MOVW      NEXT, R0
30001⊙:
        ...
        LCAL—LAB  OWN,20⊙   ; Макровызов с локальной меткой
                            ; пользователя 20⊙.
        TSTW      OWN
        BGTR      20⊙
        MOVW      OWN, R0
20⊙:
        ...
```

6.2.8. СТРОКОВЫЕ ОПЕРАЦИИ В МАКРООПРЕДЕЛЕНИЯХ

Макроассемблер СМ1 700 реализует три встроенные функции для обработки символьных строк, представленных в коде КОИ-8. Эти функции особенно полезны при программировании задач, ориентированных на обработку текстовых данных: они позволяют заменять целые последовательности «традиционных» операций. Такими операциями, которые применяются в макроопределениях и производят определенные действия над аргументами макрокоманд, являются: %LENGTH, %LOCATE, %EXTRACT.

При их использовании следует помнить, что они могут применяться только внутри макроопределений и в пределах блоков повторений. В качестве аргумента в каждой из трех функций может выступать либо аргумент макрокоманды, либо строка символов, ограниченная угловыми скобками или любыми одина-

ковыми символами, первому из которых предшествует символ (^).

Операция %LENGTH. С ее помощью определяется длина указанного аргумента макрокоманды. Формат операции:

%LENGTH(строка)

где строка — аргумент макрокоманды или строка символов, заключенная в ограничители.

Рассмотрим пример макроопределения, в котором используется строковая операция %LENGTH:

```
. MACRO  SIZE STR                    ; Макроопределение SIZE.
. IF GREATER 8 — % LENGTH(STR)       ; Если длина строки меньше
                                     ; 8, то вывести сообщение об
                                     ; ошибке.
. ERROR                              ; Строка STR мала.
. ENDC
. ENDM   SIZE
```

Рассмотрим два макровызова и соответствующие макрорасширения приведенного выше макроопределения.

Макровызов 1:

```
SIZE TAPE
```

Макрорасширение:

```
        IF GREATER  8—4
%MACRO-E-GENERR, GENERATED ERROR: Строка TAPE мала.
        . ENDC
```

Макровызов 2:

```
SIZE INTERRUPT
```

Макрорасширение:

```
. IF GREATER  8—9
. ERROR          ; Строка INTERRUPT мала.
. ENDC
```

При трансляции первого макровызова будет напечатано сообщение об ошибке, а при втором — нет.

Операция %LOCATE. Она определяет местонахождение фрагмента строки символов в проверяемой строке. Если операция %LOCATE нашла этот фрагмент, то она возвращает информацию о позиции первого символа совпадения в строке символов. Например, значение операции %LOCATE(<DE>, <ABCDEF>) равно трем, поскольку первая символьная позиция в строке обозначается нулем. Если операция %LOCATE не находит совпадения, она возвращает значение, равное длине указанной строки символов. Например, значение операции %LOCATE (<Z>, <ABCDEF>) равно шести. Формат операции %LOCATE:

%LOCATE(строка1,строка2[,символ])

где строка1 — строка символов, представляющая искомый фрагмент строки символов. Он может быть либо аргументом макрокоманды, либо строкой символов, заключенной в ограничители;
строка2 — проверяемая строка символов, т. е. строка, в которой ищется совпадение фрагмента. Данная строка символов может быть либо аргументом макрокоманды, либо строкой символов, заключенной в ограничители;
символ — необязательное символическое имя или десятичное число, указывающее позицию в проверяемой строке, с которой Макроассемблер должен начать поиск. Если этот параметр пропущен, Макроассемблер начинает поиск совпадения с нулевой позиции (начало строки символов). Данное символическое имя должно быть абсолютным, а число должно быть десятичным числом без знака. Недопустимо использование выражения и операции указания основания системы счисления.

В качестве примера рассмотрим следующее макроопределение:

```
. MACRO POSITION  STR, LN
. IF NOTEQUAL     %LOCATE(STR, <AABBCCDDEE> —LN
. ERROR                    ; * строка STR находится не в
                           ; указанной позиции *
. ENDC
. ENDM  POSITION
```

Макровызовы и соответствующие макрорасширения приведены ниже.

Макровызов 1:

```
POSITION BB,6
```

Макрорасширение:

```
.IF  NOTEQUAL  2—6
%MACRO-E- GENERR, GENERATED ERROR:  * строка BBB находится
                                   не в указанной позиции *
```

Макровызов 2:

```
POSITION  EE,8
```

Макрорасширение:

```
. IF  NOTEQUAL  8—8
. ERROR           ; * строка EE не находится в
                  ; указанной позиции *
.ENDC
```

Если среди аргументов функции %LOCATE присутствует необязательный параметр "символ", то поиск в проверяемой строке начинается с позиции, на которую указывает этот параметр. Например, значение функции %LOCATE (<бор>, <набор символов>, 3) равно 13, т. е. длине проверяемой строки, так как фрагмент <бор> начинается со второй позиции, а для поиска задана третья позиция.

Операция %EXTRACT. С ее помощью извлекается фрагмент текста из указанной строки символов, который начинается с определенной позиции и имеет заданную длину. Например, значе-

нием операции %EXTRACT (1, 4, <ABCDEF>) является строка BCDE, поскольку первый символ строки имеет нулевую позицию. Функция имеет следующий формат:

%EXTRACT(символ1, символ2, строка)

где символ1 — символическое имя или десятичное число, которое указывает начальную позицию фрагмента строки символов;
символ2 — символическое имя или десятичное число, которое указывает длину фрагмента строки символов.

Как для символ1, так и для символ2 символическое имя должно быть абсолютным и заранее определенным, а число должно быть десятичным числом без знака. Недопустимы выражения и операции указания основания системы счисления;
строка — аргумент макрокоманды или строка символов, заключенная в ограничители.

В качестве примера рассмотрим макроопределение EXTR.

```
. MACRO EXTR  ARG, LN
. IF EQUAL  LN-% LENGTH(ARG)
. WARN  ; *максимальная длина*
. ENDC
. BLKB   %EXTRACT(2, 2, ARG)
. ENDM  EXTR
```

Макровызовы и соответствующие макрорасширения приведены ниже.

```
EXTR     1988—AUGUST, 12    ; Макровызов 1.
```

Макрорасширение:

```
. IF EQUAL 12—11
. WARN  ; *максимальная длина*
. ENDC
. BLKB   88
EXTRA    1945—SEPTEMBER, 14  ; Макровызов 2.
```

Макрорасширение для второго макровызова:

```
        . IF EQUAL 14—14
%MACRO-W-GENWRN,  GENERATED WARNING:  *максимальная длина*
        . ENDC
        . BLKB  45
```

Если указанная символьная позиция больше или равна по значению длине строки символов, операция %EXTRACT определяет пустую строку (в строке нет символов). Если указана нулевая длина, операция %EXTRACT также определяет пустую строку.

6.3. ДИРЕКТИВЫ МАКРОКОМАНД

В Макроассемблере СМ1 700 имеется набор директив макрокоманд, используемых для организации макрокоманд и работы с аргументами макрокоманд. Эти директивы обеспечивают дополнительные возможности для выполнения различных функций трансляции.

Назначение	Имя
Директивы макроопределений	. MACRO . ENDM . MEXIT
Директивы библиотек макроопределений	. MCALL . LIBRARY
Директива удаления макрокоманд	. MDELETE
Директивы аргументов и макрокоманд	. NARG . NCHR . NTYPE
Директивы блока повторений	. REPEAT(. REPT) . ENDR
Директивы блоков неопределенных повторений	. IRP . IRPC

Примечание. Директива .REPT является эквивалентной формой директивы .REPEAT.

6.3.1. ДИРЕКТИВЫ МАКРООПРЕДЕЛЕНИЙ

С помощью директив .MACRO, .ENDM, .MEXIT формируются макроопределения. Все определенные пользователем макрокоманды должны предваряться соответствующими макроопределениями, которые либо записываются в этой же программе, либо включаются в библиотеки макроопределений. Каждое макроопределение должно быть правильно оформлено.

.MACRO — директива начала макроопределения. Она указывает на начало макроопределения. В ней определяются имя макроопределения, совпадающее с именем макрокоманды, по которой будет вызвано это макроопределение, а также список формальных аргументов. Если указанное имя совпадает с именем ранее определенного макроопределения, то прежнее макроопределение уничтожается и замещается новым. За директивой .MACRO следуют строки исходного текста, которые определяют тело макроопределения и впоследствии включаются в макрорасширение. Конец тела макроопределения указывается с помощью директивы .ENDM.

Имена макроопределений создаются в соответствии с правилами формирования символических имен и могут совпадать с символическими именами, определяемыми пользователем.

Когда Макроассемблер встречает директиву .MACRO, он добавляет имя макроопределения в таблицу имен макроопределений и запоминает исходный текст этого макроопределения (до соответствующей директивы .ENDM). Никаких других действий до вызова соответствующей макрокоманды транслятором не выполняется.

Символические имена из списка формальных аргументов связываются с данным именем макроопределения и их действие ограничено его пределами. По этой причине символические

имена, появляющиеся в списке формальных аргументов, могут также появляться в любом месте программы. Формат директивы:

```
. MACRO имя [список формальных аргументов]
тело макроопределения
. ENDM [имя]
```

где имя — символическое имя макроопределения, которое может быть длиной до 31 символа;
список формальных аргументов — любые разрешенные символические имена, разделенные запятыми. При макровызове эти символические имена замещаются фактическими аргументами;
тело макроопределения — исходный текст, который впоследствии включается в макрорасширение при вызове соответствующей макрокоманды.

Рассмотрим пример вложенного макроопределения, которое само себя переопределяет.

```
          . MACRO   REDEF
          . PSECT   MINMAX ABS, LONG
MIN=0
MAX=^XFFFF
          . PSECT   DATA NOEXE, LONG
LIST:     . BLKB    1000
          . BLKW    100
          . BLKL    10
          . MACRO   REDEF  ; Переопределить макроопределение.
          . ENDM    REDEF
          . ENDM    REDEF
```

Макровызовы и соответствующие макрорасширения приведены ниже.

```
          REDEF                              ; Макровызов 1.
```

Макрорасширение:

```
          . PSECT   MINMAX ABS, LONG
MIN=0
MAX=^XFFFF
          . PSECT   DATA NOEXE, LONG
LIST:     . BLKB    1000
          . BLKW    100
          . BLKL    10
          . MACRO   REDEF  ; Переопределить макро-
          . ENDM    REDEF  ; определение на пустое.
          REDEF            ; Макровызов 2.
```

Макрорасширение для второго макровызова отсутствует.

В данном примере макрокоманда, вызванная в первый раз, определяет некоторые символические имена и области хранения данных, а в следующий раз она переопределяет соответствующее макроопределение, поэтому при вызове данной макрокоманды во второй раз макрорасширение не содержит никакого исходного текста.

.ENDM — директива конца макроопределения. Она завершает макроопределение. Формат директивы:

. ENDM [имя]

где имя — имя макроопределения, которое является необязательным параметром, однако если оно указано, то оно должно совпадать с именем макроопределения соответствующей директивы . MACRO. Это имя особенно полезно указывать во вложенных макроопределениях для того, чтобы отследить нарушения правил вложения.

Директива .ENDM обязательно должна следовать за директивой .MACRO, если она встречается вне макроопределения, транслятор выдает сообщение об ошибке.

.MEXIT — директива выхода из макрокоманды. С ее помощью можно досрочно завершить макрорасширение до того, как встретится директива .ENDM. При этом завершение макрорасширения ничем не отличается от завершения по директиве .ENDM. Директива .MEXIT особенно полезна в макрорасширениях, где отдельные фрагменты формируются по условию, поскольку она позволяет обходить сложности вложенных условных директив и изменять направления трансляции.

Данная директива может быть использована также в теле блока повторений.

Директива .MEXIT используется без аргументов. Ее формат простой:

. MEXIT

Пример.

```
. MACRO DISK   CTRL, UNIT, COND
...
. IF  EQ  COND     ; Начало блока условной трансляции.
...
. MEXIT            ; Завершение макрорасширения и конец блока
. ENDC             ; условной трансляции.
...
. ENDM  DISK       ; Нормальный конец макрокоманды.
```

В данном примере, в случае если бы фактическое значение формального аргумента COND было равно нулю, блок условной трансляции был включен в макрорасширение и оно завершилось бы по директиве . MEXIT.

6.3.2. ДИРЕКТИВЫ БИБЛИОТЕК МАКРООПРЕДЕЛЕНИЙ

Прежде чем использовать макрокоманду, нужно быть уверенным в том, что соответствующее макроопределение существует и оно известно транслятору. При этом макроопределение может находиться в любом месте программы, однако оно должно появиться раньше, чем обращение к макрокоманде.

Нет необходимости всякий раз включать в текст программы неоднократно проверенные макроопределения. Целесообразнее создать из них на внешней памяти библиотеку и указывать транслятору только имена требуемых макроопределений. Такие макробиблиотеки должны быть созданы до трансляции про-

грамм, в которых используются соответствующие макрокоманды.

Директивы .MCALL и .LIBRARY обеспечивают транслятор необходимой информацией о работе с такими макробиблиотеками. Сами макробиблиотеки создаются и обслуживаются специальной системной программой Библиотекарь.

.MCALL — директива вызова макроопределений. В этой директиве указываются имена системных и/или определенных пользователем макрокоманд, чьи макроопределения требуются для трансляции текста исходной программы, но которые размещены в макробиблиотеках на диске.

Если требуемое макроопределение не найдено в указанных библиотеках, то выводится сообщение об ошибке. Библиотеки, в которых транслятор осуществляет поиск макроопределений, чьи имена отмечены в директиве .MCALL, должны описываться как входные файлы в команде на выполнение Макроассемблера. Формат директивы:

. MCALL список макрокоманд

где список макрокоманд — список макрокоманд, макроопределения которых должны быть вызваны из макробиблиотек в процессе трансляции. Имена макрокоманд должны быть разделены запятыми.

Пример.

```
. MCALL SIN, COS ; Макроопределения SIN и COS должны быть
                 ; вызваны из библиотеки.
```

Директива .MCALL с указанием имен макрокоманд, как правило, записывается в начале программы, поскольку указание транслятору о поиске в библиотеках должно появиться раньше первого обращения к любой из отмеченных макрокоманд. В особенности это правило относится к макрокомандам, которые переопределяют функции машинных инструкций, а их имена совпадают с символическими именами данных инструкций. Такое требование объясняется тем, что если Макроассемблер находит неизвестное символическое имя в поле инструкции, он автоматически производит поиск этого символического имени во всех библиотеках макрокоманд.

.LIBRARY — директива задания библиотеки макроопределений. Эта директива добавляет имя к списку имен библиотек макроопределений, который просматривается транслятором каждый раз, когда встречается директива .MCALL или неопределенный код операции. Библиотеки просматриваются в порядке, обратном тому, в котором они были указаны. Директива в качестве аргумента использует спецификацию файла. Формат директивы:

. LIBRARY имя макробиблиотеки

где имя макробиблиотеки — ограниченная строка символов, интерпретируемая

транслятором как спецификация файла, содержащего библиотеку макрокоманд.

Если программист опускает какие-то данные в спецификации файла библиотеки макрокоманд, то недостающие значения принимаются по умолчанию. Так, устройством по умолчанию всегда является пользовательский диск, каталогом — пользовательский каталог, а типом файла — MLB. (Более подробные сведения об этом приведены в гл. 7.)

Несмотря на возможность задания дополнительной библиотеки директивой .LIBRARY, предпочтительнее указывать имя библиотеки в команде MACRO языка DCL с помощью квалификатора /LIBRARY.

Пример.

```
. LIBRARY /DB1 : [JOB]ARTIC/
. LIBRARY ?DB1 : MYFILE. MLB?
. LIBRARY !WORK. MLB!
```

6.3.3. ДИРЕКТИВА УДАЛЕНИЯ МАКРОКОМАНДЫ

Директива .MDELETE удаляет макроопределения для указанных макрокоманд. Количество удаленных макрокоманд печатается в листинге трансляции в той же строке, в которой указана директива .MDELETE.

Директива .MDELETE полностью удаляет макроопределение, освобождая память, тогда как способ переопределения макрокоманды, приведенный в описании директивы .MACRO, память не освобождает. Формат директивы:

```
. MDELETE список макрокоманд
```

где список макрокоманд — список имен макрокоманд, макроопределения которых подлежат удалению. Имена макрокоманд должны быть разделены запятыми.

Пример.

```
. MDELETE ⊙BASE, SIN, COS, PAGE, LINE; Макроопределения
                                     ; пяти
                                     ; макрокоманд
                                     ;удаляются.
```

6.3.4. ДИРЕКТИВЫ АРГУМЕНТОВ МАКРОКОМАНД

Директивы данной группы .NARG, .NCHR и .NTYPE позволяют определить отдельные параметры макрокоманд, в основном относящиеся к их аргументам.

.NARG — директива определения количества аргументов. С ее помощью можно определить количество аргументов, заданных при вызове макрокоманды. Директива .NARG производит подсчет всех позиционно-зависимых аргументов, заданных в макро-

вызове, включая пустые аргументы (вместо каждого пустого аргумента указывается запятая). Аргументы в виде ключевых слов и аргументы, имеющие значение по умолчанию, директивой не рассматриваются. Формат директивы:

```
. NARG символ
```

где символ — символическое имя, которому присваивается значение, равное количеству аргументов в макровызове.

Рассмотрим макроопределение ACCOUNT

```
. MACRO ACCOUNT  A, B, C, D=ABC, E=XYZ
. NARG  CNT      ; Подсчет числа аргументов.
. WORD  CNT      ; Сохранить в CNT значение числа аргументов.
. ENDM  ACCOUNT
```

В первой строке соответствующего макрорасширения будет определено число аргументов, а во второй строке оно будет запомнено.

Далее приведены макровызовы и соответствующие макрорасширения.

Макровызов 1:

```
ACCOUNT MAR, MAY
```

Макрорасширение:

```
. NARG CNT
. WORD CNT
```

значение счетчика CNT равно 2.

Макровызов 2:

```
ACCOUNT MAR, APR=30, MAY=31
```

Макрорасширение:

```
. NARG CNT
. WORD CNT
```

значение счетчика CNT равно 1.

Макровызов 3:

```
ACCOUNT JUNE, JULE,,,
```

Макрорасширение:

```
. NARG CNT
. WORD CNT
```

значение счетчика CNT равно 5.

Директива .NARG может использоваться только внутри макроопределения. В противном случае при трансляции будет выдана ошибка.

.NCHR — директива определения количества символов. Она определяет количество символов в указанной строке символов. Эта директива может появляться в любом месте программы, написанной на языке Макроассемблера, и оказывается полезной

при вычислении длины аргументов макрокоманды. Формат директивы:

. NCHR символ,<строка>

где символ — символическое имя, которому присваивается значение, равное количеству символов в указанной строке символов;
<строка> — последовательность символов, ограниченная угловыми скобками или двумя одинаковыми символами, первому из которых предшествует символ «уголок вверх», второй вариант используется лишь тогда, когда указанная строка символов содержит разрешенные разделительные символы (запятая, пробел, символ табуляции) или точку с запятой.

Рассмотрим пример макроопределения COUNT:

```
. MACRO  COUNT  STRING
. NCHR   CNT, <STRING>
. WORD   CNT
. ASCIZ  /STRING/
. ENDM   COUNT
```

Макровызов:

```
COUNT  <SOVIET UNION>
```

Макрорасширение:

```
. NCHR   CNT, <SOVIET UNION>
. WORD   CNT
. ASCIZ  /SOVIET UNION/
```

значение счетчика CNT равно 12.

.NTYRE — директива определения режима адресации. Она определяет режим адресации для указанного фактического аргумента при формировании макрорасширения. Формат директивы:

. NTYPE символ, операнд

где символ — любое разрешенное символическое имя, которому при формировании макрорасширения присваивается 8- или 16-разрядное значение, указывающее на режим адресации фактического аргумента, который следует далее в качестве аргумента директивы;
операнд — любое разрешенное адресное выражение, которое используется вместе с кодом инструкции. Если «аргумент» не указан, то подразумевается нуль.

Значение символического имени устанавливается соответствующим образом для заданного режима адресации аргумента (режимы адресации подробно описаны в гл. 5). В большинстве случаев образуется 8-разрядное значение (1 байт). Разряды 0—3 указывают на регистр, который используется в данном режиме адресации, а разряды 4—7 указывают на режим адресации. Для обеспечения сжатой формы информации об адресации разряды 4—7 неточно воспроизводят числовое значение определенного режима адресации. В частности, литеральный режим обозначается нулем в разрядах 4—7 вместо значений 0—3. Режим 1 соответствует аргументу с непосредственной адресации,

режим 2 указывает на аргумент с абсолютной адресацией, а режим 3 — на аргумент с адресацией общего вида.

Для адресации с индексацией определяется 16-разрядное значение (2 байта). Старший байт — номер режима адресации для базового операнда, а младший байт содержит номер режима адресации первичного операнда (индексный регистр).

Для пояснения действий директивы .NTYPE рассмотрим следующий пример макроопределения.

```
; Макрокоманда REZERV используется для разервирования 10 слов в случае,
; если аргумент ARG использует нулевой регистр с непосредственной адреса-
; цией. В противном случае резервируется только одно слово.
    . MACRO  REZERV ARG
    . NTYPE  CNT, ARG
    . IF  EQ  CNT-16
    . BLKW   10
    . ENDC
    . BLKW   1
    . ENDM   REZERV
```

6.3.5. ДИРЕКТИВЫ БЛОКА ПОВТОРЕНИЙ

В ряде случаев возникает необходимость повторить несколько раз отдельный фрагмент текста программы. Для того чтобы каждый раз не записывать последовательность инструкций или данных, используют директивы .REPEAT, (.REPT) и .ENDR. Дублируемая несколько раз часть программы называется *блоком повторений.*

.REPEAT — директива начала блока повторений. По этой директиве выполняется дублирование блока повторений исходного текста программы указанное количество раз. Конец тела данных, используют директивы .REPEAT, (.REPT) и .ENDR. Формат директивы:

```
. REPEAT выражение
тело блока повторений
. ENDR
```

где выражение — выражение, значение которого определяет число повторений тела блока. Если значение выражения меньше или равно нулю, блок повторений не транслируется. Данное выражение не должно содержать никаких неопределенных символических имен и должно быть абсолютным;
тело блока повторений — фрагмент исходного текста, который должен быть повторен. Блок повторений может содержать макроопределения, блоки неопределенных повторений и другие блоки повторений. Кроме того, допустимо использование в блоке повторений директивы . MEXIT.

Рассмотрим пример макроопределения COPY, которое включает блок повторений.

```
. MACRO   COPY  STRING, NUM
. REPEAT  NUM
. ASCII   /STRING/
. ENDR
. WORD    0
. ENDM    COPY
```

Макровызов:

```
COPY      <ошибка в запросе!>, 2
```

Макрорасширение:

```
. ASCII   /ошибка в запросе!/
. ASCII   /ошибка в запросе!/
. WORD    0
```

.ENDR — директива конца блока повторений. Она указывает на конец блока повторений. Эта директива должна быть завершающей не только для блока повторений, но и для любой директивы блока неопределенных повторений (.IRP и .IRPC).

6.3.6. ДИРЕКТИВЫ БЛОКОВ НЕОПРЕДЕЛЕННЫХ ПОВТОРЕНИЙ

Блоки неопределенных повторений практически идентичны блокам повторений за исключением того, что дублирование фрагмента текста программы может сопровождаться его модификацией. Действие директив .IRP и .IRPC напоминает реализацию циклов, в которых при каждом обращении используется очередной параметр из списка значений параметра цикла.

Каждая из директив имеет один формальный аргумент, который при очередном повторении части программы принимает одно из значений, определенных в списке аргументов. Директивы отличаются лишь способом задания этих значений.

.IRP — директива блока неопределенных повторений. По этой директиве осуществляется последовательная замена формального аргумента последовательными фактическими аргументами, которые указаны в списке аргументов. Замена производится во время расширения тела блока неопределенных повторений.

Директива .IRP аналогична макроопределению, у которого есть только один формальный аргумент. При первом расширении тела блока повторений формальный аргумент принимает значение первого элемента из списка аргументов и осуществляется макрорасширение с этим значением. Затем формальный аргумент принимает значение второго элемента из списка аргументов, с которым выполняется очередное макрорасширение. Процедура продолжается до тех пор, пока не будет исчерпан весь список.

Директива .IRP может появляться внутри макроопределения, блока неопределенных повторений, блока повторений или независимо от них. Правила указания аргументов директивы .IRP те же самые, что и для указания аргументов макрокоманд. Формат директивы:

```
. IRP имя, <список аргументов>
блок неопределенных повторений
. ENDR
```

где имя — формальный аргумент, который последовательно замещается указанными фактическими аргументами из списка, заключенного в угловые скобки. Если не указано ни одного формального аргумента, Макроассемблер выводит сообщение об ошибке;
<список аргументов> — список фактических аргументов, заключенный в угловые скобки и используемый при расширении тела блока неопределенных повторений. Фактический аргумент может содержать один или несколько символов. Если указано несколько фактических аргументов, то они должны быть разделены допустимыми разделителями (запятая, пробел, символ табуляции). Если не указано ни одного фактического аргумента, то не производится никаких действий;
блок неопределенных повторений — блок исходного текста, который должен быть повторен один раз для каждого фактического аргумента из списка фактических аргументов. Этот блок может содержать макроопределения и блоки повторений. Кроме того, допустимо использование директивы . MEXIT в теле этого блока;
. ENDR указывает на конец блока неопределенных повторений.

Рассмотрим пример макроопределения ALGEBRA.

```
. MACRO  ALGEBRA FNC, A1, A2, A3, A4, A5
. NARG   CNT
. IRP    ARG, <A5, A4, A3, A2, A1>
. IIF    .NOT—BLANK, ARG, функция ARG
. ENDR
CALLS    # <CNT-1>, FNC
. ENDM   ALGEBRA
```

Макровызов:

```
ALGEBRA KEY, EXP, LIM, ABS, COS, SIN
```

Макрорасширение:

```
. NARG   CNT
. IRP    ARG, <SIN, COS, ABS, LIM, EXP>
. IIF    NOT—BLANK, ARG,  функция ARG
. ENDR
. IIF    NOT—BLANK , SIN  , функция SIN
. IIF    NOT—BLANK , COS  , функция COS
. IIF    NOT—BLANK , ABS  , функция ABS
. IIF    NOT—BLANK , LIM  , функция LIM
. IIF    NOT—BLANK , EXP  , функция EXP
CALLS    # <CNT—1>, KEY
```

В приведенном примере для подсчета числа аргументов используется директива . NARG, а для определения, не пропущен ли фактический аргумент, употребляется директива . IIF NOT—BLANK. Если фактический аргумент пропущен, то двоичный код не формируется.

.IRPC — директива начала блока неопределенных повторений с символами. Эта директива подобна директиве .IRP, за исключением того, что директива .IRPC в качестве списка фактических значений использует строку символов и допускает замену формального аргумента на один символ из этой строки, а не в виде аргумента. В каждом повторении тела блока неопределенных повторений формальный аргумент последовательно замещается следующим символом из указанной строки. На конец блока неопределенных повторений указывает директива .ENDR. Формат директивы:

```
. IRPC имя, <строка>
блок неопределенных повторений
. ENDR
```

где имя — формальный аргумент, который последовательно замещается указанными символами, заключенными в угловые скобки. Если не указано ни одного формального аргумента, Макроассемблер выводит сообщение об ошибке;
<строка> — последовательность символов, заключенная в угловые скобки, которая используется при расширении тела блока неопределенных повторений. Хотя угловые скобки нужны только в том случае, если строка символов содержит разделительные символы, угловые скобки рекомендуется писать всегда для повышения наглядности программы;
блок неопределенных повторений — фрагмент исходного текста, который должен быть повторен один раз для каждого символа из указанной строки. Этот блок может содержать макроопределения и блоки повторений. Допустимо использование директивы .MEXIT в теле этого блока.

Рассмотрим пример макроопределения.

```
. MACRO  MODE STRING
. NCHR   NUM, <BWL>
. IRPC   SIZE, <BWL>
. WARN   ; Адрес инструкции MOV'SIZE.
. ENDR
. ENDM   MODE
```

Макровызов:

```
MODE     <BWL>
```

Макрорасширение:

```
. NCHR   NUM, <BWL>
. IRPC   SIZE, <BWL>
. WARN   ; Адрес инструкции MOV'SIZE.
. ENDR
. WARN   ; Адрес инструкции MOVB.
. WARN   ; Адрес инструкции MOVW.
. WARN   ; Адрес инструкции MOVL.
```

Для подсчета числа символов в фактическом аргументе в данном примере используется директива .NCHR.

Глава 7

РАЗРАБОТКА ПРОГРАММ. ПРИМЕРЫ ПРОГРАММИРОВАНИЯ

В разработке программы на Макроассемблере СМ1700 можно выделить два этапа:

- создание (написание) самой программы на Макроассемблере, которая в виде последовательности инструкций и директив отображает требуемый алгоритм;
- подготовка этой программы к выполнению в среде МОСВП (ввод программы, трансляция и компоновка).

На первый взгляд кажется, что оба этапа мало зависят друг от друга и могут выполняться автономно. Однако на самом деле они тесно взаимосвязаны: любая программа должна быть правильно оформлена, чтобы транслятор Макроассемблера мог оттранслировать ее без ошибок (имеются в виду ошибки, связанные с неправильным заданием обязательных директив программы (.ENTRY, .END и т. п.). Кроме того, прежде чем программа будет оттранслирована, ее надо ввести в машину. Ввод, как правило, осуществляется с помощью сервисных средств МОСВП — одного из редакторов текста. При этом желательно, чтобы он обеспечивал форматирование строк программы, а также исправление синтаксических и отдельных семантических ошибок трансляции.

Если в программе используются, например, макрокоманды, то их макроопределения должны быть предварительно введены в состав соответствующей макробиблиотеки, что также осуществляется с помощью средств МОСВП. Помимо этого, МОСВП должна включать законченный набор средств по организации и работе не только с макробиблиотеками, но и с библиотеками объектных модулей, которые используются компоновщиком.

Ошибки исполнения (например, переполнение стеков для временного хранения данных, неверное задание размеров таблиц и т. п.) выявляются отладчиком, который подключается к пользовательской программе на этапе компоновки.

Таким образом, прежде чем получить готовую к исполнению программу, необходимо выполнить несколько операций, которые целиком определяются операционной системой МОСВП.

Представленная на рис. 7.1 схема разработки программ не претендует на полноту, а представляет собой лишь последова-

тельность основных действий по получению исполняемого образа. В ряде случаев могут присутствовать дополнительные действия, например проверка программы на тестовых данных, использование различного рода препроцессоров и т. д.

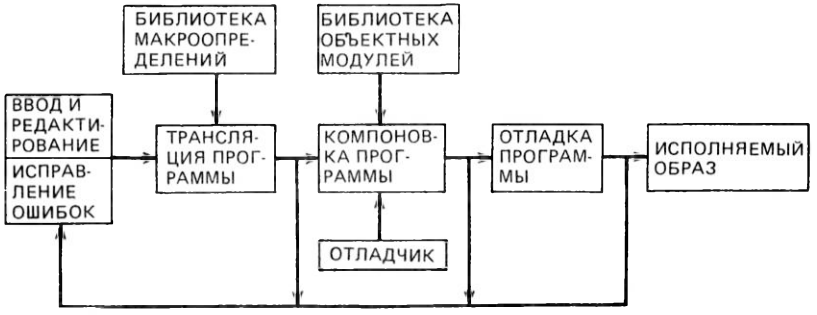


Рис. 7.1. Технологическая цепочка разработки программы

Программа, написанная на листе бумаги, еще не означает, что с ее помощью можно получить требуемый результат. Только выполнив определенные действия под управлением операционной системы, т. е. «пройдясь» по всей технологической цепочке, можно создать программу, готовую для обработки данных по заданному алгоритму. Наивно было бы предполагать, что программа не изменяется в результате работы на машине: выявляются ошибки в программе, изменяются ее первоначальный алгоритм, а также отдельные строки и фрагменты программы.

Таким образом, чтобы получить готовую к исполнению программу, необходимо не только хорошо владеть Макроассемблером, но и знать возможности операционной системы МОСВП и уметь грамотно ими пользоваться. С этой целью рассмотрим последовательность преобразования программы по получению исполняемого образа в операционной системе МОСВП, а также основные требования, которым должно удовлетворять оформление программы на этом языке. Приведенные ниже примеры помогут желающим в написании их собственных программ.

7.1. СТРУКТУРНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ

Программа, разработанная на Макроассемблере, будет успешно оттранслирована, скомпонована и выполнена в операционной системе МОСВП, если она написана с учетом определенных правил и соотношений между ее структурными элементами. Программа должна обязательно включать в себя:

- оператор входа;
- оператор выхода;
- операцию установки содержимого регистра R0.

Оператор входа служит для определения точки входа в программу. Программа может быть оттранслирована и при отсутствии этого оператора, однако при попытке ее выполнения будет выдано сообщение об ошибке. Оператор входа состоит из трех элементов:

- директивы .ENTRY;
- символического имени точки входа в программу;
- маски входа.

Так, в примере

```
. ENTRY INCOS, ^M<R5, R6, R7>
```

директива .ENTRY указывает, что данный оператор является точкой входа программы. Символическое имя INCOS представляет собой глобальное символическое имя, значение которого равно значению счетчика адресов программы оператора входа (т. е. директивы .ENTRY). Это имя также должно быть указано в соответствующей директиве .END, если программа является основной программой. Символическое имя точки входа используется в директивах .MASK, .TRANSFER, а также в других случаях, когда программист должен указать адрес точки входа. При этом оно не должно совпадать с любым другим именем, используемым в программе.

Маской входа для разобранного выше примера является ^M<R5, R6, R7>. Операция регистровой маски (^M) устанавливает соответствующие разряды для указанных в этой операции регистров и спецификаторов разрешения арифметического прерывания. Заданные в маске регистры сохраняются перед входом в программу. Маска входа используется для организации правильного обращения к подпрограммам. Если программа INCOS не вызывается другой программой, вместо операции маскирования может стоять нуль:

```
. ENTRY INCOS, 0
```

Оператор выхода указывает на логический конец программы и состоит из двух элементов: директивы .END и символического имени программы, т. е. адреса передачи управления.

Пример.

```
. END INCOS
```

Для успешного выполнения программы необходимо перед оператором выхода установить содержимое регистра R0, в который в особых случаях заносится значение, указывающее на состояние ошибки. Обычно в регистр R0 заносится значение системного параметра SS⊙_NORMAL, свидетельствующее о нормальном завершении программы. Это осуществляется с помощью одного из двух способов:

• перед оператором выхода в программу включается следующая последовательность инструкций

```
MOVL  #SS⊙— NORMAL, R0
RET
```

• перед оператором выхода вызывается системная программа ⊙EXIT указанием системной макрокоманды ⊙EXIT_S, которая используется без параметров.

Установка значения R0 является обязательной процедурой, даже если это значение не использовалось в теле программы.

Обычно в состав программы входят такие «малые» структурные единицы, как макрокоманды, блоки условной трансляции, блоки повторений. Для успешного выполнения программы эти единицы всегда должны содержать все необходимые элементы.

Помимо отмеченных выше замечаний, которые в основном касались корректного использования отдельных возможностей языка, существует целый ряд соглашений по оформлению и кодированию программ, следование которым в значительной мере отражает культуру программирования. Эти соглашения, помимо директив идентификации листинга, позволяют дополнительно структурировать исходный текст программ, увеличивая ее читаемость и облегчая ее дальнейшее сопровождение.

Программный модуль условно разбивается на несколько разделов. Количество и содержимое каждого раздела может произвольно меняться, но, как правило, каждая организация-изготовитель программных средств придерживается своих внутренних соглашений. Рассмотрим один из возможных вариантов оформления программного модуля [3].

Текст программного модуля разбивается как минимум на четыре раздела.

Вводная часть. В нее записывается общая информация о модуле: дата создания и имена разработчиков модуля; отметки в хронологическом порядке о всех корректировках модуля с указанием даты, имени разработчика и существа изменений. Здесь же приводится краткое описание функций модуля. Отдельные записи этой части можно выполнить, используя директивы .TITLE, .IDENT и т. п.

Раздел определений. Он, как правило, характеризует используемые внутренние и внешние символические имена и переменные, а также макрокоманды и, если необходимо, соответствующие макроопределения. В раздел включаются описание и определение внутренних символических имен и структур данных (тип, размер и т. п.), описание организации их размещения в памяти; описание глобальных символических имен и программных секций, используемых в модуле; перечень используемых библиотечных макрокоманд, указываемых в директиве

.MCALL, а также описание и определение внутренних макрокоманд. В этом же разделе приводится список применяемых регистров.

При формировании и определении символических имен также следует придерживаться определенных правил. Так, регистры внешних устройств должны иметь имена, совпадающие с их обозначениями в документации. Все локальные символические имена должны начинаться с одной буквы (например, с буквы А), глобальные — с другой (например, с буквы S).

Раздел спецификаций. Здесь в основном определяются входные и выходные данные, а также приводится подробное описание функций, выполняемых модулем. В случае необходимости записываются специфические особенности работы модуля: требования к используемой памяти, размеры стека, побочные эффекты, которые могут возникнуть в отдельных случаях, и т. п.

Раздел текста программы. Здесь предусмотрены отдельные области, которые необходимо снабжать подробными комментариями. По возможности следует выделять области данных и инструкций, размещая их в различных программных секциях. В свою очередь области данных целесообразно разделять на данные, доступные только для чтения, и данные, доступные для модификации, размещая их в отдельных программных секциях с соответствующими параметрами.

Подобное разбиение модуля на разделы требует от программиста дополнительных усилий и времени, однако это с избытком возмещается при поиске и исправлении возможных ошибок, а также при эксплуатации программного модуля.

Другим не менее важным аспектом, который следует учитывать при разработке программы, является следование соглашениям о связях между отдельными модулями и подпрограммами (правила вызова и возврата из модуля; корректное использование регистров, включая регистры для передачи параметров и результатов и т. п.). Все эти тонкости должны отмечаться в соответствующих разделах программного модуля и снабжаться подробными комментариями.

7.2. ПОДГОТОВКА ПРОГРАММ В СРЕДЕ МОСВП

Подготовка программ на Макроассемблере для последующего исполнения в среде МОСВП состоит из нескольких этапов. Они в основном определяются соглашениями, принятыми программистом, и средствами операционной системы по реализации технологии создания, а также сопровождения программных продуктов. Эти этапы одинаковы для всех систем программиро-

вания за исключением этапа, на котором используется конкретный компилятор (например, с языка Фортрана или Кобола).

На каждом этапе употребляются определенные команды операционной системы МОСВП, которые обеспечивают необходимый интерфейс пользователя с операционной системой и предоставляют обширные сервисные возможности по реализации требуемых действий. Вне зависимости от этапа все команды имеют одинаковый формат:

команда [/квалификатор(ы)] спец. файла [/квалификатор(ы)]

где команда — одна из набора команд диалогового командного языка (DCL) операционной системы МОСВП, характерная для текущего этапа;
квалификатор(ы) — квалификатор(ы) команды или файла, указывающий на специальные действия;
спец. файла — спецификация файла или список спецификаций файлов, которые определяют используемые файлы. В зависимости от команды спецификация указывает или входной файл, который должен быть обработан, или выходной, который будет получен в результате выполнения команды.

Более подробно описание команд приведено в [2]. Здесь же кратко остановимся на описании спецификации файла, которая не зависит от применяемых команд на разных этапах. Она имеет следующий формат:

устройство: [каталог] имя—файла. расширение; версия

где устройство — символическое наименование логического или физического устройства, на котором располагается входной или выходной файл. Если устройство идентифицируется как физическое устройство, его имя будет иметь вид:

DDCU

где DD — символическое имя физического устройства (LP — печатающее устройство, MM — магнитная лента и т. д.);
C — символическое обозначение номера контроллера, через которое подключено физическое устройство (A — нулевой, B — первый и т д.);
U — номер устройства на контроллере (как правило, от 0 до 7);
каталог — имя каталога на устройстве с каталоговой организацией (квадратные скобки обязательны);
имя файла — имя файла, которое может состоять из последовательности символов длиной не более 10;
расширение — описатель типа файла (до трех символов). В МОСВП можно использовать любую последовательность из трех символов, называемую расширением файла, для указания типа файла. Однако есть целый ряд расширений, которые используются по умолчанию. Например: OBJ — объектный файл, OLB — библиотечный файл, EXE — исполняемый образ и т. п.;
версия — номер версии файла, который обозначается десятичным числом. Он увеличивается всякий раз, как только создается новая версия файла (например, очередная редакция текстового файла).

Нет необходимости указывать все элементы спецификации файла, но имя файла должно указываться всегда. Если опускается другая часть спецификации, соответствующие значения устанавливаются по умолчанию. Ниже приведены элементы спецификации файла, используемые по умолчанию.

Элемент спецификации	Значение по умолчанию
Устройство	Подразумеваемые в данный момент по умолчанию тип устройства, контроллер и номер устройства пользователя.
Каталог	Текущий подразумеваемый каталог пользователя.
Тип	Зависит от использования: MAR — исходный файл; MLB — файл библиотеки макрокоманд; OBJ — объектный модуль; LIS — файл листинга; UPD — файл модификаций.
Версия	Ввод: наибольшая существующая версия Вывод: наибольшая существующая версия плюс 1.

Таким образом, используя унифицированный формат команды и спецификации необходимых файлов, можно достаточно просто переходить от одного этапа к другому.

Из рис. 7.1 видно, что процедура подготовки программы на Макроассемблере в среде МОСВП разбивается на четыре этапа.

Создание и/или исправление ошибок. Как правило, исходная программа формируется (вводится в ЭВМ) одним из редакторов текста, которые есть в МОСВП. Например, вызов универсального строчного редактора EDIT для работы с файлом DOST. MAR осуществляется командой

⊙EDIT/EDT DOST. MAR

Если программа вводится с терминала, целесообразно использовать экранный редактор текста, специально ориентированный на диалоговый «экранный» режим работы.

С помощью команд редактора пользователь выполняет действия по вводу, редактированию и форматированию исходного программного модуля. После завершения работы файл DOST. MAR представляет собой текстовый файл программы на Макроассемблере.

Трансляция. Перевод исходной программы из Макроассемблера в объектный код выполняется транслятором по команде

⊙MACRO [/квалификатор(ы)] DOST. MAR

где /квалификатор(ы) определяет дополнительные действия транслятора по обработке программы DOST. MAR.

Дополнительные функции транслятора можно осуществить посредством директив Макроассемблера, включаемых в исходный текст программы (см. гл. 4), и с помощью необязательных квалификаторов /LIBRARY, /LIST и /OBJECT (табл. 7. 1).

Так, по команде

⊙MACRO/LIST DOST. MAR

Таблица 7.1

КВАЛИФИКАТОРЫ КОМАНДЫ MACRO

Обозначение	По умолчанию	Назначение
/LIBRARY	Отсутствует	Указывает, что входной файл содержит библиотеку макрокоманд
/LIST (/NOLIST)	/NOLIST	Управляет созданием файла листинга. Если осуществляется трансляция в диалоговом режиме, файл листинга не создается. Однако, если выполняется команда MACRO в пакетном режиме, файл листинга создается
/OBJECT (/NOOBJECT)	/OBJECT	Управляет созданием объектного модуля

помимо объектного модуля будет сформирован файл листинга программы DOST. LIS.

Компоновка. Полученный после трансляции объектный файл не может быть использован для выполнения. Он должен быть предварительно обработан компоновщиком (редактором связей) по следующей команде:

⊙LINK DOST

В процессе компоновки (редактирования связей) объектный модуль объединяется с другими дополнительными программами и системными модулями. В результате создается файл DOST. EXE, называемый *образом задачи*.

Команда LINK использует по умолчанию тип входного файла OBJ, а для библиотек — OLB.

При обнаружении ошибки в процессе редактирования связей (например, присутствуют неопределенные глобальные символические имена) выходной файл не образуется, а выдается лишь подробная информация об обнаруженных ошибках. Если компоновка выполняется успешно, файл образа задачи может быть использован для выполнения.

Выполнение. Для исполнения образа задачи вводится команда

⊙RUN DOST

Команда RUN использует по умолчанию тип файла EXE. Если исходная программа не содержит логических ошибок, мешающих ее правильному выполнению (ошибки, которые операционная система не всегда может обнаружить), то выполнение образа задачи даст правильные результаты. В противном случае необходимо модифицировать исходную программу и вновь произвести трансляцию и ее компоновку. Такая итерационная про-

цедура может выполняться несколько раз. Чтобы не вводить каждый раз последовательности команд трансляции, компоновки и исполнения, можно создать специальный командный файл, который будет включать в себя необходимые команды. В результате необходимых модификаций программы, проделанных с помощью редактора текстов, выполняется этот командный файл.

Например, командный файл DOST. COM:

```
@MACRO/LIST DOST
@ON ERROR THEN EXIT
@LINK DOST
@ON ERROR THEN EXIT
@RUN DOST
```

может быть запущен на исполнение с помощью команды

```
@@DOST
```

В результате выполнения командного файла будет оттранслирован, скомпонован и выполнен программный модуль с именем DOST.

Большинство разрабатываемых на Макроассемблере программ являются системными, т. е. обеспечивающими необходимый сервис для пользователей, работающих в операционной системе. Несмотря на богатую систему инструкций СМ1 700, позволяющую создавать эффективные программы, такие программы обычно имеют сложные взаимосвязи.

В качестве системной программы МОСВП в примере 1 приведена программа планировщика процессов, которая начинает выполняться в ответ на прерывание с приоритетным уровнем программного прерывания, равным 3 (IPL3), и осуществляет контекстное переключение процессов.

Понять основные принципы построения программ на Макроассемблере можно и на более простых стандартных алгоритмах (примеры 2, 3).

Пример 1.

```
. TITLE ПЛАНИРОВЩИК
; Планировщик процессов — обслуживающая программа операционной си-
; стемы МОСВП, предназначенная для контекстного переключения процес-
; сов.
;
; * * Область определений * *
; Для определения переменных и структур используются системные макро-
; команды:
@DYNDEF          ; Определение типа структуры.
@IPLDEF          ; Определение приоритетного уровня прерывания.
@PCBDEF          ; Определение управляющего блока процесса (PCB).
@PHDDEF          ; Определение заголовка процесса (PHD).
@PRDEF           ; Определение регистров процессора.
@STATEDEF        ; Определение состояния процесса.
```

```
; SCH⊙RESCHED — перепланировщик.
; Эта программа начинает выполняться в ответ на программное прерывание
; с приоритетным уровнем 3 в режиме ядра.
; При входе в программу:
; IPL=3, MODE=0
;
; 00 (SP)=PC
; 04 (SP)=PSL

SCH⊙RESCHED::                                ; Обработка прерывания по пере-
                                             ; планированию процесса.
    SETIPL  #IPL⊙—SYNCH                      ; Синхронизация.
    SVPCTX                                   ; Сохранение контекста процесса.
    MOVL    W^SCH⊙GL—CURPCB, R1              ; Получение адреса текущего PCB.
    MOVZBL  PCB⊙B—PRI (R1), R2               ; Получение адреса текущего при-
                                             ; оритета.
    BBSS    R2, W^SCH⊙GL—COMQS, 10⊙          ; Определение очереди как
                                             ; непустой
10⊙: MOVW   #SCH⊙C—COM,—                     ; Установить в текущем PCB со-
            PCB⊙W—STATE(R1)                  ; стояние процесса как резидент-
                                             ; ное исполняемое.
    MOVAQ   W^SCH⊙AQ—COMT[R2], R3            ; Вычисление адреса очере-
                                             ; ди.
    INSQUE  (R1),@(R3)+                      ; Включение текущего PCB в ко-
                                             ; нец очереди.

; SCH⊙SCHED — планировщик нового процесса на выполнение.
; Эта программа выбирает для выполнения процесс, обладающий наивысшим
; приоритетом.
SCH⊙SCHED::                                  ; Планирование выполнения.
    SETIPL  #IPL⊙—SYNCH                      ; Синхронизация.
    FFS     #0, #32,—                        ; Определение наивысшего уров-
            W^SCH⊙GL—COMQS, R2               ; ня приоритета процесса, готово-
                                             ; го к выполнению.
    BEQL    SCH⊙IDLE                         ; Нет готовых к выполнению про-
                                             ; цессов?
    MOVAQ   W^SCH⊙AQ—COMH[R2], R3            ; Вычисление адреса начала
                                             ; очереди.
    REMQUE  (R3)+, R4                        ; Получение первого элемента оче-
                                             ; реди.
    BVS     QEMPTY                           ; Переход, если очередь оказалась
                                             ; пустой.
    BNEQ    20⊙                              ; Очередь не пустая.
    BBCC    R2, W^SCH⊙GL—COMQS, 20⊙          ; Установить признак пу-
                                             ; стой очереди.
20⊙:                                         ;
    CMPB    #DYN⊙C—PCB,—                     ; Если выбранный блок не PCB, то
            PCB⊙B—TYPE(R4)                   ; фатальная ошибка.
    BNEQ    QEMPTY                           ;
    MOVW    ⊙SCH⊙C—CUR,—                     ; Установить состояние для вы-
            PCB⊙W—STATE (R4);                ; бранного процесса как текущее
                                             ; выполняемое.
    MOVL    R4, W^SCH⊙GL—CURPCB              ; Отменить текущий PCB.
    CMPB    PCB⊙B—PRIB (R4),—                ; Сравнить текущий приоритет с
            PCB⊙B—PRI (R4)                   ; базовым.
    BEQL    30⊙                              ; Не менять текущий приоритет,
                                             ; если он равен базовому.
```

```
    BBC     #4, PCB⊙B—PRI (R4), 30⊙; Если процесс реального времени,
                                    ; то приоритет не меняется.
    INCB    PCB⊙B—PRI(R4)           ; Вернуться к базовому приори-
                                    ; тету.
30⊙:MOVB    PCB⊙B—PRI(R4), —        ; Установить глобальный приори-
            W^SCH⊙GB—PRI            ; тет.
    MTPR    PCB⊙L—PHYPCB (R4),—     ; Установить физический базовый
            #PR⊙—PCBB               ; адрес PCB.
    LDPCTX                          ; Загрузить контекст для выбран-
                                    ; ного процесса.
    REI                             ; Нормальное завершение (выход
                                    ; из прерывания).
```

Пример 2.

```
    . TITLE NAMES

; Программа NAMES осуществляет пересылку символов в коде КОИ-8 из
; одной строки в другую. Пробелы, встретившиеся в первой строке, во вто-
; рую переданы не будут.

; ** Область данных **

INPUT:                              ; Поле входной строки.
    . ASCII /   ЕЛЕНА*  НАТАЛЬЯ*  ОЛЬГА*  ТАТЬЯНА*   /
OUT:                                ; Поле выходной строки.
        . BLKB  35
DLINA:  . WORD  35                  ; DLINA содержит длину входной
                                    ; строки.
CURRENT:
        . LONG  0                   ; CURRENT содержит адрес
                                    ; INPUT.
CURRENT1:
        . LONG  0                   ; CURRENT1 содержит адрес сим-
                                    ; вола, отличного от пробела.
ADR—OUT: . LONG 0                   ; ADR—OUT содержит адрес OUT.

; ** Область инструкций **

        . ENTRY  NAMES, 0           ; Начало программы
        MOVAL  INPUT, CURRENT       ; В CURRENT переслать адрес
                                    ; INPUT.
        MOVAL  OUT, ADR—OUT         ; В ADR—OUT переслать адрес
                                    ; OUT.
BLANK:  SKPC   #^A/ /, DLINA, —     ; Поиск первого символа, отлич-
               @CURRENT             ; ного от пробела.
        TSTL   R0                   ; Проверка R0 на нуль.
        BEQL   END                  ; Если нуль, значит все символы,
                                    ; отличные от пробела, были пере-
                                    ; сланы в OUT.
        MOVL   R1, CURRENT1         ; Послать адрес символа, отлич-
                                    ; ного от пробела, в CURRENT1.
        MOVL   R0, DLINA            ; Послать количество оставшихся
                                    ; для обработки байтов в DLINA.
        LOCC   #^A/ /, R0, —        ; Определить, есть ли пробелы.
               @CURRENT1
```

```
GETLAST:
          SUB3    R0, DLINA, R6       ; Найти количество символов, от-
                                      ; личных от пробела, для пересыл-
                                      ; ки в выходную строку.
          MOVL    R0, DLINA           ; DLINA содержит количество ос-
                                      ; тавшихся для обработки симво-
                                      ; лов.
          MOVC3   R6,@CURRENT1, —     ; Переслать символы в OUT.
                  @ADR—OUT
          ADDL2   R6, ADR—OUT         ; ADR—OUT содержит следую-
                                      ; щий адрес OUT.
          MOVL    R1, CURRENT         ; CURRENT содержит следующий
                                      ; адрес INPUT.
          TSTW    DLINA               ; Проверка на конец INPUT.
          BNEQ    BLANK               ; Если не конец, идти на BLANK.
END:      ⊙EXIT—S                     ; Пересылка окончена,
          . END   NAMES               ; выход из программы.
```

Пример 3.

```
. TITLE PSL—ORDER

; Программа PSL—ORDER просматривает содержимое длинного слова со-
; стояния процессора (PSL) и размещает информацию из PSL по адресам,
; указанным в области данных.
; PSL содержит следующую информацию: коды условий (N, Z, V, C), флаги
; возможных «ловушек» (T, IV, FV, DV), уровень приоритета прерываний
; (IPL), флаг режима совместимости (CM).

  ** Область данных **
;
C:          . BYTE  0       ; Поле разряда переноса.
V:          . BYTE  0       ; Поле разряда переполнения.
Z:          . BYTE  0       ; Поле разряда нулевого результата.
N:          . BYTE  0       ; Поле разряда отрицательного резуль-
                            ; тата.
T:          . BYTE  0       ; Поле разряда смещения.
IV:         . BYTE  0       ; Поле разряда целочисленного перепол-
                            ; нения.
FU:         . BYTE  0       ; Поле разряда переполнения числа с
                            ; плавающей запятой.
DV:         . BYTE  0       ; Поле разряда десятичного переполне-
                            ; ния.
IPL:        . LONG  0       ; Поле разрядов уровня приоритета пре-
                            ; рывания.
CM:         . BYTE  0       ; Поле разряда режима совместимости.
PSL—STORE:                  ; Поле сохранения информации из
            . LONG  0       ; PSL.

; **Область инструкций**

          . ENTRY   START, ^M<R4, R5, R7>; Начало программы.
          MOVPSL    PSL—STORE       ; Сохранить PSL в PSL—STORE
          MOVAB     C, R5           ; Загрузить в R5 байт C.
          CLRL      R7              ; R7 указывает позицию извлекае-
                                    ; мого разряда.
```

```
BIT—EXTRACT:
        EXTZV   R7, #1, —           ; Извлечь разряд из PSL если он
                PSL—STORE, R4       ; не установлен, уйти на END.
        MOVB    R4, (R5)+           ; Поместить разряд в соответст-
                                    ; вующий байт.
END:                                ; Обработка конца цикла по изв-
                                    ; лечению байтов.
        AOBLSS  #8, R7, —           ; Проверить, все ли 8 байт обра-
                BIT—EXTRACT         ; ботаны.
        EXTV    #16, #5, —          ; Извлечь разряды IPL.
                PSL—STORE, IPL
        TSTL    PSL—STORE           ; Проверить бит CM.
        BGEQ    DONE                ; Если четный, уйти на DONE.
        MOVB    #1, CM              ; Установить CM-байт.
DONE:   ⊙EXIT—S                     ; Все разряды обработаны.
        . END   START               ; Выход из программы.
```

ЛИТЕРАТУРА

1. Васильев Г. П. и др. Малые ЭВМ высокой производительности: архитектура и программирование /Г. П. Васильев, Г. А. Егоров, В. С. Зонис, В. В. Родионов; Под ред. Н. Л. Прохорова. — М.: Радио и связь, 1989.

2. Остапенко Г. П. и др. Операционная система МОСВП для СМ1700 /А. В. Аксенов, А. А. Нестеров, С. Н. Суслов, В. И. Фролов. — М.: Финансы и статистика, 1988. — 183 с.

3. Вигдорчик Г. В., Воробьев А. Ю., Праченко В. Д. Основы программирования на ассемблере для СМ ЭВМ. — 2-е изд., перераб. и доп. — М.: Финансы и статистика, 1987. — 240 с.

4. Пратт Т. Языки программирования: разработка и реализация. — М.: Мир, 1979. — 574 с.

5. Сибеста Р. Структурное программирование языка ассемблера ЭВМ VAX—11. — М.: Мир, 1988. — 535 с.

6. Малые ЭВМ и их применение /Под ред. Б. Н. Наумова. — М.: Статистика, 1980. — 231 с.

7. СМ ЭВМ: комплексирование и применение /Под ред. Н. Л. Прохорова. — М.: Финансы и статистика, 1986. — 304 с.

b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	ПУС	АР1	Пробел	0	@	P	`	p				А	Р	а	р	
0	0	0	0	0	0	0	1	1	НЗ	СУ1	!	1	A	Q	a	q			Ё	Б	С	б	с	ё
0	0	0	0	0	0	1	0	2	НТ	СУ2	"	2	B	R	b	r				В	Т	в	т	
0	0	0	0	0	0	1	1	3	КТ	СУ3	#	3	C	S	c	s				Г	У	г	у	
0	0	0	0	0	1	0	0	4	КП	СТП	¤	4	D	T	d	t				Д	Ф	д	ф	
0	0	0	0	0	1	0	1	5	КТМ	НЕТ	%	5	E	U	e	u				Е	Х	е	х	
0	0	0	0	0	1	1	0	6	ДА	СИН	&	6	F	V	f	v				Ж	Ц	ж	ц	
0	0	0	0	0	1	1	1	7	ЗБ	КБ	'	7	G	W	g	w				З	Ч	з	ч	
0	0	0	0	1	0	0	0	8	ВШ	АН	(	8	H	X	h	x				И	Ш	и	ш	
0	0	0	0	1	0	0	1	9	ІТ	КН	)	9	I	Y	i	y				Й	Щ	й	щ	
0	0	0	0	1	0	1	0	10	ПС	ЗМ	*	:	J	Z	j	z				К	Ъ	к	ъ	
0	0	0	0	1	0	1	1	11	ВТ	АР2	+	;	K	[	k	{				Л	Ы	л	ы	
0	0	0	0	1	1	0	0	12	ПФ	РФ	,	<	L	/	l	|				М	Ь	м	ь	
0	0	0	0	1	1	0	1	13	ВК	РГ	–	=	M	]	m	}				Н	Э	н	э	
0	0	0	0	1	1	1	0	14	ВЫХ	РЗ	.	>	N	^	n	~				О	Ю	о	ю	
0	0	0	0	1	1	1	1	15	ВХ	РЭ	/	?	O	_	o	ЗБ				П	Я	п	я	

Примечание. Знак денежной единицы ¤ по технологическим причинам заменен в тексте знаком ⊙.

1. Таблица кодов КОИ-8 (см. с. 192)

2. Краткие сведения о режимах адресации

Систематизированные краткие сведения о режимах адресации Макроассемблера (см. таблицу) представляют собой справочный материал, нужный как для разработки программ, так и для быстрой оценки возможных вариантов написания конкретных инструкций.

РЕЖИМЫ АДРЕСАЦИИ

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможность индексации
Адресация регистров общего назначения	Регистровый	RN	5	Регистр содержит операнд	Нет
	Косвенно-регистровый	(RN)	6	Регистр содержит адрес операнда	Да
	Адресация с автоувеличением	(RN)+	8	Регистр содержит адрес операнда; процессор увеличивает содержимое регистра на размер типа данных операнда	Да
	Косвенная адресация с автоувеличением	@(RH)+	9	Регистр содержит адрес адреса операнда; процессор увеличивает содержимое регистра на 4	Да
	Адресация с автоуменьшением	—(RN)	7	Процессор уменьшает содержимое регистра на размер типа данных операнда, после чего регистр содержит адрес операнда	Да
	Адресация по смещению	смщ(RN) B^смщ(RN) W^смщ(RN) L^смщ(RN)	* A C E	Сумма содержимого регистра и смещения является адресом операнда; указатели B^, W^, L^ указывают соответственно на смещение в байтах, словах и длинных словах	Да

Продолжение

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможность индексации
	Косвенная адресация по смещению	@смщ(RN) @B^смщ(RN) @W^смщ(RN) @L^смщ(RN)	* B D F	Сумма содержимого регистра и смещения является адресом адреса операнда; указатели B^, W^, L^ указывают соответственно на смещение в байтах, словах и в длинных словах	Да
	Литеральная адресация	#литерал S^литерал	0—3	Указанный литерал является операндом; литерал хранится в формате короткого литерала	Нет
Адресация через счетчик инструкций PC	Относительная адресация	адрес B^адрес W^адрес L^адрес	* A C E	Указанный адрес является адресом операнда; указанный адрес хранится в виде смещения относительно PC; указатели B^, W^, L^ указывают на смещение соответственно в байтах, словах и длинных словах	Да
	Косвенная относительная адресация	@адрес @B^адрес @W^адрес @L^адрес	* B D F	Указанный адрес является адресом адреса операнда; указанный адрес хранится в виде смещения относительно PC; указатели B^, W^,	Да

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможность индексации
				L^ указывают на смещение соответственно в байтах, словах и длинных словах	
	Абсолютная адресация	@#адрес	9	Указанный адрес является адресом операнда; указанный адрес хранится как абсолютный действительный адрес, а не как смещение	Да
	Непосредственная адресация	#литерал I^#литерал	* 8	Указанный литерал является операндом; этот литерал хранится как байт, слово, длинное слово или как квадрослово	Нет
	Адресация общего вида	G^адрес	—	Указанный адрес является адресом операнда; если адрес определен как перемещаемый, компоновщик запоминает этот адрес в виде смещения относительно PC; если этот адрес определен как абсолютный действительный адрес, компоновщик запоминает его как абсолютное значение	Да
Индексация	Индексация	база [RX]	4	Параметр «база» определяет базовый адрес, а регистр определяет индекс; сумма базового адреса и произведения содержимого RX на размер типа данных операнда; «база» может оп-	Нет

Продолжение

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможность индексации
				ределять любой режим адресации, кроме прямого обращения к регистру, индексации непосредственной адресации литеральной адресации и адресации в инструкциях относительного перехода	
Адресация в инструкциях относительного перехода	Адресация в инструкциях относительного перехода	адрес	—	Указанный адрес является операндом; этот адрес хранится как смещение относительно PC; данный режим адресации может использоваться только в инструкциях относительного перехода	Нет

Примечание. В таблице используются следующие обозначения:

RN — любой регистр общего назначения с R0 по R14. Вместо RN может использоваться любой из регистров AP, FP и SP;

RX — любой регистр общего назначения с R0 по R12. Вместо RX может использоваться любой из регистров AP, FP и SP. RX не может быть тем же самым регистром, что и RN, который указан в параметре «база» для определенных базовых режимов;

смщ — выражение, определяющее смещение;

адрес — выражение, определяющее адрес;

база — один из допустимых режимов адресации, используемый для указания на операнд;

* — код данного вида режима адресации зависит от контекста программы и определяется операционной системой на этапе трансляции и компоновки.

3. Таблица кодов инструкций

В таблице содержатся постоянные символические имена, которые Макроассемблер интерпретирует как коды машинных инструкций, если они встречаются в поле операции. Коды инструкций приведены в шестнадцатеричной системе счисления.

Последняя графа таблицы указывает на страницу книги, где данная инструкция рассматривается более подробно.

Мнемоническое обозначение	Код	Выполняемое действие	Страница
ACBB	9D	Сложение, сравнение и переход для байт	59
ACBD	6F	Сложение, сравнение и переход для данных формата D	59
ACBF	4F	Сложение, сравнение и переход для данных формата F	59
ACBG	4FFD	Сложение, сравнение и переход для данных формата G	59
ACBH	6FFD	Сложение, сравнение и переход для данных формата H	59
ACBL	F1	Сложение, сравнение и переход для длинных слов	59
ACBW	3D	Сложение слов, сравнение и переход	59
ADAWI	58	Сложение выровненных слов с блокировкой памяти	50
ADDB2	80	Сложение байтов (2 операнда)	50
ADDB3	81	Сложение байтов (3 операнда)	50
ADDD2	60	Сложение данных формата D (2 операнда)	69
ADDD3	61	Сложение данных формата D (3 операнда)	69
ADDF2	40	Сложение данных формата F (2 операнда)	69
ADDF3	41	Сложение данных формата F (3 операнда)	69
ADDG2	40FD	Сложение данных формата G (2 операнда)	69
ADDG3	41FD	Сложение данных формата G (3 операнда)	69
ADDH2	60FD	Сложение данных формата H (2 операнда)	69
ADDH3	61FD	Сложение данных формата H (3 операнда)	69
ADDL2	C0	Сложение длинных слов (2 операнда)	50
ADDL3	C1	Сложение длинных слов (3 операнда)	50
ADDP4	20	Сложение упакованных десятичных строк (4 операнда)	78
ADDP6	21	Сложение упакованных десятичных строк (6 операндов)	78
ADDW2	A0	Сложение слов (2 операнда)	50
ADDW3	A1	Сложение слов (3 операнда)	50
ADWC	D8	Сложение с переносом	50
AOBLEQ	F3	Прибавление единицы и переход, если «меньше или равно»	60
AOBLSS	F2	Прибавление единицы и переход, если «меньше»	60
ASHL	78	Арифметический сдвиг длинного слова	50

Мнемоническое обозначение	Код	Выполняемое действие	Страница
ASHP	F8	Арифметический сдвиг с округлением упакованной десятичной строки	78
ASHQ	79	Арифметический сдвиг квадрослова	50
BBC	E1	Переход, если разряд равен 0	60
BBCC	E5	Переход, если разряд равен 0, присвоение ему значения 0	60
BBCCI	E7	Переход, если разряд равен 0, присвоение ему значения 0 с блокировкой	60
BBCS	E3	Переход, если разряд равен 0, присвоение ему значения 1	60
BBS	E0	Переход, если разряд равен 1	60
BBSC	E4	Переход, если разряд равен 1, присвоение ему значения 0	60
BBSS	E2	Переход, если разряд равен 1, присвоение ему значения 1	60
BBSSI	E6	Переход, если разряд равен1, присвоение ему значения 1 с блокировкой	60
BCC	1E	Переход, если нет переноса	60
BCS	1F	Переход, если есть перенос	60
BEQL	13	Переход, если «равно»	62
BEQLU	13	Переход, если «равно» (без знака)	62
BGEQ	18	Переход, если «больше или равно»	62
BGEQU	1E	Переход, если «больше или равно» (без знака)	62
BGTR	14	Переход, если «больше»	62
BGTRU	1A	Переход, если «больше» (без знака)	62
BICB2	8A	Очистка разрядов в байте (2 операнда)	50
BICB3	8B	Очистка разрядов в байте (3 операнда)	50
BICL2	CA	Очистка разрядов в длинном слове (2 операнда)	50
BICL3	CB	Очистка разрядов в длинном слове (3 операнда)	50
BICPSW	B9	Очистка разрядов в слове состояния процессора	84
BICW2	AA	Очистка разрядов в слове (2 операнда)	50
BICW3	AB	Очистка разрядов в слове (3 операнда)	50
BISB2	88	Установка разрядов в байте (2 операнда)	51
BISB3	89	Установка разрядов в байте (3 операнда)	51
BISL2	C8	Установка разрядов в длинном слове (2 операнда)	51
BISL3	C9	Установка разрядов в длинном слове (3 операнда)	51

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
BISPSW	B8	Установка разрядов слова состояния процессора	84
BISW2	A8	Установка разрядов в слове (2 операнда)	51
BISW3	A9	Установка разрядов в слове (3 операнда)	51
BITB	93	Проверка разрядов в байте	50
BITL	D3	Проверка разрядов в длинном слове	50
BITW	B3	Проверка разрядов в слове	50
BLBC	E9	Переход, если младший разряд равен 0	60
BLBS	E8	Переход, если младший разряд равен 1	60
BLEQ	15	Переход, если «меньше или равно»	62
BLEQU	1B	Переход, если «меньше или равно» (без знака)	62
BLSS	19	Переход, если «меньше»	62
BLSSU	1F	Переход, если «меньше» (без знака)	62
BNEQ	12	Переход, если «не равно»	62
BNEQU	12	Переход, если «не равно» (без знака)	62
BPT	03	Ловушка отладчика	84
BRB	11	Безусловный переход со смещением в байте	61
BRW	31	Безусловный переход со смещением в слове	61
BSBB	10	Переход на подпрограмму со смещением в байте	61
BSBW	30	Переход на подпрограмму со смещением в слове	61
BVC	1C	Переход, если нет переполнения	62
BVS	1D	Переход, если есть переполнение	62
CALLG	FA	Вызов процедуры с обычным списком аргумента	65
CALLS	FB	Вызов процедуры со списком аргументов в стеке	65
CASEB	8F	Выбор перехода со смещением в байте	61
CASEL	CF	Выбор перехода со смещением в длинном слове	61
CASEW	AF	Выбор перехода со смещением в слове	61
CHME	BD	Изменить режим на управление	84
CHMK	BC	Изменить режим на ядро	84
CHMS	BE	Изменить режим на супервизор	84
CHMU	BF	Изменить режим на пользовательский	84
CLRB	94	Очистка байта	51
CLRD	7C	Очистка данного формата D	69

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
CLRF	DF	Очистка данного формата F	69
CLRG	7C	Очистка данного формата G	69
CLRH	7CFD	Очистка данного формата H	69
CLRL	D4	Очистка длинного слова	51
CLRO	7CFD	Очистка октаслова	51
CLRQ	7C	Очистка квадрослова	51
CLRW	B4	Очистка слова	51
CMPB	91	Сравнение байтов	51
CMPC3	29	Сравнение строк (3 операнда)	74
CMPC5	2D	Сравнение строк (5 операндов)	74
CMPD	71	Сравнение данных формата D	69
CMPF	51	Сравнение данных формата F	69
CMPG	51FD	Сравнение данных формата G	69
CMPH	71FD	Сравнение данных формата H	69
CMPL	D1	Сравнение длинных слов	51
CMPP3	35	Сравнение упакованных десятичных строк (3 операнда)	78
CMPP4	37	Сравнение упакованных десятичных строк (4 операнда)	78
CMPV	EC	Сравнение поля (имеющего расширение знака) с длинным словом	57
CMPW	B1	Сравнение слов	51
CMPZV	ED	Сравнение поля (дополненного нулями) с длинным словом	57
CRC	0B	Контроль на циклическую избыточность	76
CVTBD	6C	Преобразование байта в формат D	69
CVTBF	4C	Преобразование байта в формат F	69
CVTBG	4CFD	Преобразование байта в формат G	69
CVTBH	6CFD	Преобразование байта в формат H	69
CVTBL	98	Преобразование байта в длинное слово	51
CVTBW	99	Преобразование байта в слово	51
CVTDB	68	Преобразование формата D в байт	69
CVTDF	76	Преобразование формата D в формат F	70
CVTDH	32FD	Преобразование формата D в формат H	70
CVTDL	6A	Преобразование формата D в длинное слово	70
CVTDW	69	Преобразование формата D в слово	69
CVTFB	48	Преобразование формата F в байт	69
CVTFD	56	Преобразование формата F в формат D	70
CVTFG	99FD	Преобразование формата F в формат G	70
CVTFL	4A	Преобразование формата F в длинное слово	70
CVTFW	49	Преобразование формата F в слово	69

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
CVTGB	48FD	Преобразование формата G в байт	69
CVTGF	33FD	Преобразование формата G в формат F	70
CVTGH	56FD	Преобразование формата G в формат H	70
CVTGL	4AFD	Преобразование формата G в длинное слово	70
CVTGW	49FD	Преобразование формата G в слово	69
CVTHB	68FD	Преобразование формата H в байт	69
CVTHD	F7FD	Преобразование формата H в формат D	70
CVTHF	F6FD	Преобразование формата H в формат F	70
CVTHG	76FD	Преобразование формата H в формат G	70
CVTHL	6AFD	Преобразование формата H в длинное слово	70
CVTHW	69FD	Преобразование формата H в слово	69
CVTLB	F6	Преобразование длинного слова в байт	51
CVTLD	6E	Преобразование длинного слова в формат D	69
CVTLF	4E	Преобразование длинного слова в формат F	69
CVTLG	4EFD	Преобразование длинного слова в формат G	69
CVTLH	6EFD	Преобразование длинного слова в формат H	69
CVTLP	F9	Преобразование длинного слова в упакованную десятичную строку	79
CVTLW	F7	Преобразование длинного слова в слово	51
CVTPL	36	Преобразование упакованной десятичной строки в длинное слово	79
CVTPS	08	Преобразование упакованной десятичной строки в числовую строку с ведущим знаком	79
CVTPT	24	Преобразование упакованной десятичной строки в числовую строку	79
CVTRDL	6B	Преобразование формата D в длинное слово (с округлением)	70
CVTRFL	4B	Преобразование формата F в длинное слово (с округлением)	70
CVTRGL	4BFG	Преобразование формата G в длинное слово (с округлением)	70
CVTRHL	6BFD	Преобразование формата H в длинное слово (с округлением)	70

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
CVTSP	09	Преобразование числовой строки с ведущим знаком в упакованную десятичную строку	79
CVTTP	26	Преобразование числовой строки в упакованную десятичную строку	79
CVTWB	33	Преобразование слова в байт	51
CVTWD	6D	Преобразование слова в формат D	69
CVTWF	4D	Преобразование слова в формат F	69
CVTWG	4DFD	Преобразование слова в формат G	69
CVTWH	6DFD	Преобразование слова в формат H	69
CVTWL	32	Преобразование слова в длинное слово	51
DECB	97	Уменьшение байта на 1	52
DECW	B7	Уменьшение слова на 1	52
DIVB2	86	Деление байтов (2 операнда)	52
DIVB3	87	Деление байтов (3 операнда)	52
DIVD2	66	Деление данных формата D (2 операнда)	70
DIVD3	67	Деление данных формата D (3 операнда)	70
DIVF2	46	Деление данных формата F (2 операнда)	70
DIVF3	47	Деление данных формата F (3 операнда)	70
DIVG2	46FD	Деление данных формата G (2 операнда)	70
DIVG3	47FD	Деление данных формата G (3 операнда)	70
DIVH2	66FD	Деление данных формата H (2 операнда)	70
DIVH3	67FD	Деление данных формата H (3 операнда)	70
DIVL2	C6	Деление длинных слов (2 операнда)	52
DIVL3	C7	Деление длинных слов (3 операнда)	52
DIVP	27	Деление упакованных десятичных строк	80
DIVW2	A6	Деление слов (2 операнда)	52
DIVW3	A7	Деление слов (3 операнда)	52
EDITPC	38	Редактирование упакованной десятичной строки	80
EDIV	7B	Расширенное деление	52
EMODD	74	Расширенное умножение с выделением целой части для данных формата D	70
EMODF	54	Расширенное умножение с выделением целой части для данных формата F	70

Мнемоническое обозначение	Код	Выполняемое действие	Страница
EMODG	54FD	Расширенное умножение с выделением целой части для данных формата G	70
EMODH	74FD	Расширенное умножение с выделением целой части для данных формата H	70
EMUL	7A	Расширенное умножение	52
EXTV	EE	Пересылка битового поля в длинное слово с расширением знака	57
EXTZV	EF	Пересылка битового поля в длинное слово с дополнением нулями	57
FFC	EB	Поиск первого разряда, равного нулю	57
FFS	EA	Поиск первого разряда, равного единице	57
HALT	00	Останов (в режиме ядра)	84
INCB	96	Увеличение байта на 1	52
INCL	D6	Увеличение длинного слова на 1	52
INCW	B6	Увеличение слова на 1	52
INDEX	0A	Вычисление и проверка индекса на диапазон возможных значений	84
INSQHI	5C	Занесение элемента в начало очереди с блокировкой	67
INSQTI	5D	Занесение элемента в конец очереди с блокировкой	67
INSQUE	0E	Занесение элемента в очередь	67
INSV	F0	Пересылка из длинного слова в битовое поле	57
JMP	17	Универсальный переход	61
JSB	16	Переход к подпрограмме	61
LDPCTX	06	Загрузка контекста процесса	85
LOCC	3A	Поиск символа	74
MATCHC	39	Поиск подстроки символов	74
MCOMB	92	Пересылка байта в дополнительном коде	52
MCOML	D2	Пересылка длинного слова в дополнительном коде	52
MCOMW	B2	Пересылка слова в дополнительном коде	52
MFPR	DB	Пересылка из привилегированного регистра процессора (в режиме ядра)	85
MNEGB	8E	Пересылка байта со сменой знака	52
MNEGD	72	Пересылка данного формата D со сменой знака	71
MNEGF	52	Пересылка данного формата F со сменой знака	71
MNEGG	52FD	Пересылка данного формата G со сменой знака	71
MNEGH	72FD	Пересылка данного формата H со сменой знака	71

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
MNEGL	CE	Пересылка длинного слова со сменой знака	52
MNEGW	AE	Пересылка слова со сменой знака	52
MOVAB	9E	Пересылка адреса байта	55
MOVAD	7E	Пересылка адреса данного формата D	55
MOVAF	DE	Пересылка адреса данного формата F	55
MOVAG	7E	Пересылка адреса данного формата G	55
MOVAH	7EFD	Пересылка адреса данного формата H	55
MOVAL	DE	Пересылка адреса длинного слова	55
MOVAO	7EFD	Пересылка адреса октаслова	55
MOVAQ	7E	Пересылка адреса квадрослова	55
MOVAW	3E	Пересылка адреса слова	55
MOVB	90	Пересылка байта	52
MOVC3	28	Пересылка строки (3 операнда)	75
MOVC5	2C	Пересылка строки (5 операндов)	75
MOVD	70	Пересылка данного формата D	71
MOVF	50	Пересылка данного формата F	71
MOVG	50FD	Пересылка данного формата G	71
MOVH	70FD	Пересылка данного формата H	71
MOVL	D0	Пересылка длинного слова	52
MOVO	7DFD	Пересылка октаслова	52
MOVP	34	Пересылка упакованной десятичной строки	80
MOVPSL	DC	Пересылка длинного слова состояния процессора	84
MOVQ	7D	Пересылка квадрослова	52
MOVTC	2E	Пересылка перекодированной строки	75
MOVTUC	2F	Пересылка перекодированной строки до указанного символа	75
MOVW	BO	Пересылка слова	52
MOVZBL	OA	Пересылка байта в длинное слово с дополнением нулями	53
MOVZBW	9B	Пересылка байта в слово с дополнением нулями	53
MOVZWL	3C	Пересылка слова в длинное слово с дополнением нулями	53
MTPR	DA	Пересылка в регистр процессора	85
MULB2	84	Умножение байтов (2 операнда)	53
MULB3	85	Умножение байтов (3 операнда)	53
MULD2	64	Умножение данных формата D (2 операнда)	71
MULD3	65	Умножение данных формата D (3 операнда)	71
MULF2	44	Умножение данных формата F (2 операнда)	71

Мнемоническое обозначение	Код	Выполняемое действие	Страница
MULF3	45	Умножение данных формата F (3 операнда)	71
MULG2	44FD	Умножение данных формата G (2 операнда)	71
MULG3	45FD	Умножение данных формата G (3 операнда)	71
MULH2	64FD	Умножение данных формата H (2 операнда)	71
MULH3	65FD	Умножение данных формата H (3 операнда)	71
MULL2	C4	Умножение длинных слов (2 операнда)	53
MULL3	C5	Умножение длинных слов (3 операнда)	53
MULP	25	Умножение упакованных десятичных строк	80
MULW2	A4	Умножение слов (2 операнда)	53
MULW3	A5	Умножение слов (3 операнда)	53
NOP	01	Нет операции	85
POLYD	75	Вычисление полинома для данных формата D	71
POLYF	55	Вычисление полинома для данных формата F	71
POLYG	55FD	Вычисление полинома для данных формата G	71
POLYH	75FD	Вычисление полинома для данных формата H	71
POPR	BA	Извлечение регистров из стека	85
PROBER	0C	Проверка доступности операнда для чтения	85
PROBEW	0D	Проверка доступности операнда для записи	85
PUSHAB	9F	Запись в стек адреса байта	55
PUSHAD	7F	Запись в стек адреса данного формата D	55
PUSHAF	DF	Запись в стек адреса данного формата F	55
PUSHAG	7F	Запись в стек адреса данного формата G	55
PUSHAH	7FFD	Запись в стек адреса данного формата H	55
PUSHAL	DF	Запись в стек адреса длинного слова	55
PUSHAO	7FFD	Запись в стек адреса октаслова	55
PUSHAQ	7F	Запись в стек адреса квадрослова	55
PUSHAW	3F	Запись в стек адреса слова	55
PUSHL	DD	Запись в стек длинного слова	53
PUSHR	BB	Запись регистров в стек	85
REI	02	Возврат из прерывания или исключительной ситуации	85

Мнемоническое обозначение	Код	Выполняемое действие	Страница
REMQHI	5E	Удаление элемента из начала очереди с блокировкой	67
REMQTI	5F	Удаление элемента из конца очереди с блокировкой	67
REMQUE	0F	Удаление элемента из очереди	67
RET	04	Возврат из процедуры	65
ROTL	9C	Циклический сдвиг длинного слова	53
RSB	05	Возврат из подпрограммы	61
SBWC	D9	Вычитание с переносом	53
SCANC	2A	Сканирование символов в строке	76
SKPC	3B	Пропуск символа	76
SOBGEQ	F4	Вычитание единицы и переход, если «больше или равно»	61
SOBGTR	F5	Вычитание единицы и переход, если «больше»	61
SPANC	2B	Пропуск символов, предшествующих определенным	76
SUBB2	82	Вычитание байтов (2 операнда)	54
SUBB3	83	Вычитание байтов (3 операнда)	54
SUBD2	62	Вычитание данных формата D (2 операнда)	71
SUBD3	63	Вычитание данных формата D (3 операнда)	71
SUBF2	42	Вычитание данных формата F (2 операнда)	71
SUBF3	43	Вычитание данных формата F (3 операнда)	71
SUBG2	42FD	Вычитание данных формата G (2 операнда)	71
SUBG3	43FD	Вычитание данных формата G (3 операнда)	71
SUBH2	62FD	Вычитание данных формата H (2 операнда)	71
SUBH3	63FD	Вычитание данных формата H (3 операнда)	71
SUBL2	C2	Вычитание длинных слов (2 операнда)	54
SUBL3	C3	Вычитание длинных слов (3 операнда)	54
SUBP4	22	Вычитание упакованных десятичных строк (4 операнда)	80
SUBP6	23	Вычитание упакованных десятичных строк (6 операндов)	80
SUBW2	A2	Вычитание слов (2 операнда)	54
SUBW3	A3	Вычитание слов (3 операнда)	54
SVPCTX	07	Сохранение контекста процесса (в режиме ядра)	85
TSTB	95	Проверка байта	54
TSTD	73	Проверка данного формата D	71
TSTF	53	Проверка данного формата F	71

Продолжение

Мнемоническое обозначение	Код	Выполняемое действие	Страница
TSTG	53FD	Проверка данного формата G	71
TSTH	73FD	Проверка данного формата H	71
TSTL	D5	Проверка длинного слова	54
TSTW	B5	Проверка слова	54
XFC	FC	Вызов специальной микропрограммной функции	85
XORB2	8C	Исключающее «или» байтов, участвуют 2 операнда	54
XORB3	8D	Исключающее «или» байтов, участвуют 3 операнда	54
XORL2	CC	Исключающее «или» длинных слов (2 операнда)	54
XORL3	CD	Исключающее «или» длинных слов (3 операнда)	54
XORW2	AC	Исключающее «или» слов (2 операнда)	54
XORW3	AD	Исключающее «или» слов (3 операнда)	54

4. Краткие сведения о языке Макроассемблер

В табл. 1 сведены как общие директивы транслятору, так и директивы макрокоманд. Элементы формата, заключенные в квадратные скобки, являются необязательными. В последнем столбце таблицы указана страница книги, где описание данной директивы приведено более подробно.

В табл. 2 приведены краткие сведения об операциях макрокоманд для обработки аргументов макрокоманды в виде символьных строк. Эти операции могут использоваться только в макроопределениях.

Таблица 1

ДИРЕКТИВЫ МАКРОАССЕМБЛЕРА (В АЛФАВИТНОМ ПОРЯДКЕ)

Формат	Выполняемое действие	Страница
.ADDRESS список адресов	Хранит последовательные длинные слова, содержащие адреса данных	119
.ALIGN ключ [, выражение]	Настраивает счетчик адресов на границу, указываемую параметром «ключ» (ключевое слово)	127
.ALIGN целое [, выражение]	Настраивает счетчик адресов на границу, указываемую числом 2 в степени значения параметра «целое» (целое число)	127
.ASCIC строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), перед которой следует счетный байт	121
.ASCID строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), перед которой следует описатель строки	121
.ASCII строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями)	121
.ASCIZ строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), за которой следует нулевой байт	122
.BLKA выражение	Резервирует длинные слова для адресов данных	128
.BLKB выражение	Резервирует байты под данные	128
.BLKD выражение	Резервирует квадрослова для хранения данных двойной точности с плавающей запятой в формате D	128
.BLKF выражение	Резервирует длинные слова для хранения данных с плавающей запятой одинарной точности в формате F	128
.BLKG выражение	Резервирует квадрослова для хранения данных с плавающей запятой в формате G	128
.BLKH выражение	Резервирует октаслова для хранения данных с плавающей запятой расширенной точности в формате H	128
.BLKL выражение	Резервирует длинные слова для хранения данных	128
.BLKO выражение	Резервирует октаслова для хранения данных	128
.BLKQ выражение	Резервирует квадрослова для хранения данных	128
.BLKW выражение	Резервирует слова для хранения данных	128
.BYTE список выражений	Генерирует последовательные байты данных, при этом каждый байт содержит значение, которое определяется значением выражения	122

Продолжение

Формат	Выполняемое действие	Страница
.CROSS	Разрешает формирование таблицы перекрестных ссылок для всех символических имен	144
.CROSS список символов	Таблица перекрестных ссылок формируется для указанных символических имен	144
.DEBUG список символов	Делает известными отладчику имена указанных символических имен	138
.DEFUALT смещение, ключ	Указывает длину смещения, принимаемую по умолчанию, для режимов относительной адресации	117
.D—FLOATING список литералов	Генерирует 8-байтное данное с плавающей запятой двойной точности формата D	126
.DISABLE список аргументов	Запрещает функцию (функции), указанную в списке аргументов	117
.DOUBLE список литералов	Эквивалентна директиве .D—FLOATING	126
.DSABL список аргументов	Эквивалентна директиве .DISABLE	117
.ENABL список аргументов	Эквивалентна директиве .ENABLE	117
.ENABLE список аргументов	Разрешает функцию (функции), указанную в списке аргументов	117
.END [имя]	Указывает на логический конец исходной программы, необязательный аргумент «имя» указывает на адрес точки входа	112
.ENDC	Указывает на конец блока условной трансляции	143
.ENDM [имя]	Указывает на конец макроопределения	169
.ENDR	Указывает на конец блока повторений	176
.ENTRY символ, выражение	Определяет входную точку программы	135
.ERROR [выражение] ; комментарий	Выводит определенное сообщение об ошибке	115
.EVEN	Гарантирует, что значение текущего счетчика адресов является четным (если значение нечетно, к нему добавляется единица)	129
.EXTERNAL список символов	Указывает на то, что символические имена, определенные в спецификации, являются внешними	138
.EXTRN список символов	Эквивалентна директиве .EXTERNAL	138
.F—FLOATING список литералов	Генерирует 4-байтные данные с плавающей запятой одинарной точности формата F	125

Формат	Выполняемое действие	Страница
.FLOAT список литералов	Эквивалентна директиве .F—FLOATING	125
.G—FLOATING список литералов	Генерирует 8-байтные данные с плавающей запятой в формате G	126
.GLOBAL список символов	Указывает на то, что символические имена, определенные в спецификации, являются глобальными	138
.GLOBL	Эквивалентна директиве .GLOBAL	138
.H—FLOATING список литералов	Генерирует 16-разрядные данные с плавающей запятой расширенной точности в формате H	126
.IDENT строка	Обеспечивает средство снабжения объектного модуля дополнительной информацией	111
.IF условие аргумент(ы)	Начинает блок условной трансляции, который состоит из исходного кода и включается в трансляцию только в том случае, если по отношению к указанному аргументу (аргументам) выполняется установленное условие	140
.IFF	Эквивалентна директиве .IF—FALSE	142
.IF—FALSE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит трансляции в том случае, если исходное условие ложно	142
.IFT	Эквивалентна директиве .IF—TRUE	142
.IFTF	Эквивалентна директиве .IF—TRUE—FALSE	142
.IF—TRUE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит трансляции в том случае, если исходное условие является истинным	142
.IF—TRUE—FALSE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит трансляции безусловно	142
.IIF условие [,] аргумент(ы) предложение	То же самое, что однострочный блок условной трансляции, когда условие проверяется для указанного аргумента. Операция подлежит трансляции только в том случае, если проверочное условие истинно	144
.IRP имя, <список аргументов>	Заменяет формальный аргумент последовательными фактическими аргументами, указанными в списке аргументов	176

Формат	Выполняемое действие	Страница
.IRPC имя, <строка>	Заменяет формальный аргумент последовательными одиночными символами, указанными в строке	178
.LIBRARY имя макробиблиотеки	Определяет библиотеку макрокоманд	171
.LIST [список аргументов]	Эквивалентна директиве .SHOW	113
.LINK	Включает дополнительные записи для компоновщика	148
.LONG список выражений	Генерирует последовательные слова данных. Каждое длинное слово содержит значение, определяемое указанным выражением	124
.MACRO имя [список формальных аргументов]	Начинает макроопределение	168
.MASK символ [, выражение]	Резервирует слово и копирует в него маску сохранения регистров	136
.MCALL список макрокоманд	Указывает системные и/или определяемые пользователем макрокоманды из библиотек, которые требуются для трансляции исходной программы	171
.MDELETE список макрокоманд	Удаляет из памяти макроопределения тех макрокоманд, которые указаны в списке	172
.MEXIT	Выход из расширения макрокоманды до завершения этой макрокоманды	170
.NARG символ	Определяет количество аргументов в текущем макровызове	172
.NCHR символ, <строка>	Определяет количество знаков в указанной строке	173
.NLIST [список аргументов]	Эквивалентна директиве .NOSHOW	113
.NOCROSS	Запрещает формирование таблицы перекрестных ссылок для указанных символических имен	144
.NOCROSS список символов	Запрещает формирование таблицы перекрестных ссылок для всех символических имен	144
.NOSHOW	Уменьшает значение счетчика уровня трансляции	113
.NOSHOW список аргументов	Управляет распечаткой листинга макрокоманд и блоков условной трансляции	113
.NTYPE символ, операнд	Может появляться только в рамках макроопределения, присваивает символу значение режима адресации указанного операнда	174
.OCTA литерал	Запоминает 16 байт данных	124
.OCTA символ	Запоминает 16 байт данных	124

Продолжение

Формат	Выполняемое действие	Страница
.ODD	Дает гарантию того, что значение текущего счетчика адресов является нечетным (если оно четное, то к нему добавляется единица)	129
.OPDEF код выражение, список описателей операнда	Определяет код операции и список его операндов	146
.PACKED десятичная строка [,символ]	Генерирует десятичные числа в упакованном формате, две цифры на байт	126
.PAGE	Вызывает перевод распечатки листинга трансляции на начало следующей страницы	113
.PRINT [выражение] ;комментарий	Выводит указанное сообщение	116
.PSECT [имя, [список атрибутов]]	Начинает или продолжает программную секцию, определяемую пользователем	130
.QUAD литерал	Запоминает 8 байт данных	125
.QUAD символ	Запоминает 8 байт данных	125
.REF1 операнд	Генерирует операнд байта	147
.REF2 операнд	Генерирует операнд слова	147
.REF4 операнд	Генерирует операнд длинного слова	147
.REF8 операнд	Генерирует операнд квадрослова	147
.REF16 операнд	Генерирует операнд октаслова	147
.RAPEAT выражение	Начинает блок повторений. Часть кода до следующей директивы .ENDR повторяется такое количество раз, которое определяется значением выражения	175
.REPT	Эквивалентна директиве .REPEAT	175
.RESTORE	Эквивалентна директиве .RESTORE—PSECT	134
.RESTORE—PSECT	Восстанавливает контекст программной секции из стека контекста программной секции	134
.SAVE [LOCAL—BLOCK]	Эквивалентна директиве .SAVE—PSECT	133
.SAVE—PSECT [LOCAL—BLOCK]	Сохраняет контекст текущей программной секции в стеке контекста программной секции	133
.SBTTL комментарий	Эквивалентна директиве .SUBTITLE	112
.SHOW	Дает приращение счетчику уровня листинга	113
.SHOW список аргументов	Управляет процессом распечатки макрокоманд и блоков условной трансляции	113

Продолжение

Формат	Выполняемое действие	Страница
. SIGNED—BYTE список выражений	Запоминает последовательные байты (8 разрядов), в которых хранятся данные со знаком	123
. SIGNED—WORD список выражений	Запоминает последовательные слова (16 разрядов), в которых хранятся данные со знаком	124
. SUBTITLE комментарий	Распечатывает указанную строку как часть заголовка страницы листинга трансляции. Кроме того, строки, указанные в каждой из директив . SUBTITLE, сводятся в таблицу содержания, помещаемую в начале листинга трансляции	112
. TITLE имя модуля комментарий	Принимает в качестве имени объектного модуля первые 15 символов указанного имени модуля, а также обусловливает появление его на каждой странице листинга трансляции	111
. TRANSFER символ	Указывает компоновщику переопределить значение глобального символа для использования в разделяемом образе	136
. WARN [выражение] ; комментарий	Выводит указанное предупреждающее сообщение	116
. WEAK список символов	Указывает на то, что каждый из перечисленных символов имеет атрибут подавления	139
. WORD список выражений	Генерирует последовательные слова данных. Каждое слово содержит значение, соответствующее указанному выражению	123

Таблица 2

СТРОКОВЫЕ ОПЕРАЦИИ МАКРОКОМАНД

Формат	Функция
% LENGTH (строка)	Определяет значение, равное длине строки символов
% LOCATE (строка1, строка2 [, символ])	Определяет местоположение фрагмента «строка1» в строке «строка2», начиная поиск с символа, положение которого определяется значением символа «символ»
% EXTRACT (символ1, символ2, строка)	Выделяет фрагмент строки «строка», который начинается со знаковой позиции, указываемой значением «символ1», и имеет длину, определяемую значением «символ2»

Абсолютное выражение (ABSOLUTE EXPRESSION) — выражение, значение которого определяется во время трансляции и остается постоянным (например, выражение, термы которого являются числами).

Аварийное завершение (ABORT) — исключительная ситуация, возникающая во время выполнения инструкции, в результате чего регистры и память могут оставаться в неопределенном состоянии, которое может препятствовать повторному выполнению инструкции.

Аварийный отказ (CRASH) — реакция системы на неустойчивое состояние, в частности на разрушение системы. Система перестает функционировать, на терминале пользователя нет индикации об аварийном отказе. *См.* **Зависание системы.**

Адресное пространство (ADDRES SPACE) — множество всех возможных адресов, доступных процессу. Виртуальное адресное пространство связано с множеством всех возможных виртуальных адресов. Физическое адресное пространство связано с множеством всех возможных адресов оперативной памяти.

Активатор образа (IMAGE ACTIVATOR) — системный процесс, который обеспечивает образу возможность выполняться в контексте вызвавшего его процесса. Создает структуры данных управления памятью, необходимые для отображения виртуальных страниц образа в физические страницы, а также организует структуры данных для выполнения обмена страниц.

Аппаратный блок управления процессом — аппаратный PCB (HARDWARE PROCESS CONTROL BLOCK) — известная процессору структура данных, которая содержит сохраненный аппаратный контекст в то время, когда процесс не выполняется. Аппаратный PCB процесса размещается в его заголовке.

Аппаратный контекст (HARDWARE CONTEXT) — значения, которые во время выполнения процесса содержатся в следующих регистрах процессора: счетчике инструкций (PC); длинном слове состояния процессора (PSL); 14 регистрах общего назначения (от R0 до R13); четырех регистрах процессора (P0BR, P0LR, P1BR, P1LR), описывающих виртуальное адресное пространство процесса; указателе стека (SP) для режима доступа, в котором работает процессор; содержимом, которое должно быть загружено в SP для каждого режима доступа, отличного от текущего режима. Во время выполнения процесса его аппаратный контекст постоянно корректируется процессором. Если процесс не выполняется, то аппаратный контекст этого процесса хранится в его аппаратном блоке управления процессом — аппаратном PCB.

Асинхронное системное прерывание (ASYNCHRONOUS SYSTEM TRAP (AST)) — программно-моделируемое прерывание с передачей управления программе, определенной пользователем. AST позволяет процессу пользователя получить сведения о происшедшем конкретном событии асинхронно по отношению к другим действиям. Если процесс определил программу AST для некоторого события, то, в случае его возникновения система прерывает выполнение этого процесса и исполняет соответствующую программу AST. При завершении программы AST система возобновляет выполнение процесса с того места, где он был прерван. Выполнение программы AST может быть задержано. *См.:* **Уровень ассинхронного системного прерывания.**

Ассемблер (ASSEMBLER) — языковый процессор, который транслирует исходную программу, содержащую директивы языка Ассемблера и мнемонические обозначения машинных инструкций, в объектный модуль.

Атрибуты программной секции (PROGRAMM SECTION ATTRIBUTE) — различные характеристики программной секции, указываемые с помощью аргументов директивы . PSECT.

База данных ввода-вывода (I/O DATABASE) — набор структур данных, который описывает запросы ввода-вывода, контроллеры, устройства, тома и драйверы устройств системы МОСВП. Примерами таких структур являются таблица точек входа драйвера, прологовая таблица драйвера, таблица данных устройства, блок управления устройством, блок запросов канала, пакет запроса ввода-вывода, блок данных прерывания.

Базовый адрес операнда (BASE OPERAND ADDRESS) — адрес базы некоторой таблицы или массива при адресации индексного типа.

Базовый приоритет (BASE PRIORITY) — приоритет, который система назначает процессу при его создании. Обычно базовый приоритет берется из файла авторизации пользователя. Планировщик никогда не устанавливает приоритет процесса ниже его базового приоритета. Базовый приоритет может быть изменен только администратором системы или самим процессом. Однако базовый приоритет выполняющегося процесса может быть изменен любым пользователем, имеющим привилегию ALTPRI.

Базовый регистр (BASE REGISTER) — регистр общего назначения, используемый для хранения адреса первого элемента в списке, таблице, массиве или другой структуре данных.

Базовый регистр блока управления системой (SYSTEM CONTROL BLOCK BASE REGISTER (SCBB)) — регистр процессора, содержащий базовый адрес блока управления системой.

Базовый регистр области управления (CONTROL REGION BASE REGISTER (P1BR)) — регистр процессора или его эквивалент в аппаратном блоке управления процессом, который содержит базовый виртуальный адрес таблицы страниц области управления процессом.

Базовый регистр области программы (PROGRAM REGION BASE REGISTER (P0BR)) — регистр процессора или его эквивалент в аппаратном блоке управления процессом, содержащий базовый виртуальный адрес того элемента таблицы страниц, который задает виртуальную страницу с номером 0 в области программы данного процесса.

Базовый регистр системы (SYSTEM BASE REGISTER (SBR)) — регистр процессора, содержащий физический адрес базы таблицы страниц системы.

Байт (BYTE) — восемь последовательно расположенных разрядов, начинающихся на границе адресуемого байта. Разряды нумеруются справа налево от 0 до 7. Разряд 0 — младший.

Балансный набор (BALANCE SET) — множество рабочих наборов процессов, размещенных в данный момент в оперативной памяти. Балансный набор обслуживается системным процессом, называемым планировщиком памяти.

Бинарная операция (BINARY OPERATION) — двуместная операция, т. е. операция, выполняемая над двумя операндами (например, операция сложения).

Битовое поле переменной длины (VARIABLE-LENGTH BIT FIELD) — последовательность от 0 до 32 бит (разрядов), размещаемая произвольно относительно границы байта. Битовое поле переменной длины определяется тремя атрибутами: базовым адресом поля, смещением между базовым адресом и первым разрядом поля и длиной поля, выраженной числом занимаемых разрядов.

Блок (BLOCK) — наименьшая логически адресуемая единица данных, которую устройство может передать за одну операцию ввода-вывода (512 последовательных байт для дисковых устройств); произвольное число непрерывных байтов, используемых для хранения логически связанных данных о состоянии, управлении и другой управляющей информации.

Блок данных прерывания (INTERRUPT DATA BLOCK (IDB)) — структура данных ввода-вывода, которая описывает характеристики отдельного контроллера и отмечает устройства, присоединенные к этому контроллеру.

Блок данных устройства (DEVICE DATA BLOCK (DDB)) — структура данных ввода-вывода, которая определяет общее имя устройства/контроллера и имя драйвера для набора устройств, присоединенных к одному контроллеру.

Блок дополнительных атрибутов (EXTENDED ATTRIBUTE BLOCK (XAB)) — пользовательская структура данных СУД, которая помимо атрибутов, определенных в блоке доступа к файлу (FAB), содержит такие дополнительные атрибуты, как типы границ (выровненный на цилиндре номер логического блока, номер виртуального блока) и информацию о защите файла.

Блок доступа к файлу (FILE ACCESS BLOCK (FAB)) — структура данных, описывающая конкретный файл СУД. Включает информацию, необходимую для таких операций над файлами, как OPEN (открыть), CLOSE (закрыть) или CREATE (создать).

Блок запросов канала (CHANNEL REQUEST BLOCK (CRB)) — структура данных ввода-вывода, которая описывает действия конкретного контроллера. Блок запросов канала некоторого контроллера содержит указатели на очередь ожидания драйверов, готовых обратиться к устройству через этот контроллер.

Блок начальной загрузки (BOOTSTRAP BLOCK) — блок на системном диске, содержащий программу загрузки операционной системы в оперативную память.

Блок повторений (REPEAT BLOCK) — совокупность исходных предложений, перед которой помещена директива .REPEAT и за которой следует директива .ENDR. При трансляции эта совокупность исходных предложений повторяется столько раз, сколько указывается аргументом директивы .REPEAT.

Блок состояния ввода-вывода (I/O STATUS BLOCK (IOSB) — структура данных, связанная с программой системного обслуживания очереди запросов ввода-вывода. Эта программа дополнительно возвращает в блок состояния ввода-вывода код состояния, число переданных байтов, а также информацию, зависящую от устройства и функции.

Блок управления адаптером (ADAPTER CONTROL BLOCK) — структура данных подсистемы ввода-вывода, описывающая адаптер общей шины.

Блок управления каналом (CHANNEL CONTROL BLOCK (CCB)) — структура данных ввода-вывода, созданная с помощью программы системного обслуживания «назначить канал ввода-вывода». Используется для описания того устройства, которому назначен канал.

Блок управления процессом (PROCESS CONTROL BLOCK (PCB)) — структура данных, используемая для хранения контекста процесса. Аппаратный блок PCB содержит аппаратный контекст, а программный блок PCB содержит программный контекст, в состав которого входит указатель на аппаратный PCB.

Блок управления системой (SYSTEM CONTROL BLOCK (SCB)) — структура данных в пространстве системы, которая содержит все известные системе вектора прерываний и исключительных ситуаций.

Блок управления устройством (UNIT CONTROL BLOCK (UCB)) — структура данных ввода-вывода, которая описывает характеристики и текущие действия устройства. Блок управления устройством содержит также форк-блок для драйвера устройств (он является обязательной частью форк-процесса драйвера). Кроме того, блок управления устройством предоставляет для драйвера динамическую область памяти.

Блок условной трансляции (CONDITIONAL ASSEMBLY BLOCK) — совокупность исходных предложений, которые транслируются только при выполнении определенного условия. Такой блок начинается директивой .IF и заканчивается директивой .ENDC.

Блокировка страницы в памяти (LOCKING A PAGE IN MEMORY) —

действие, в результате которого страница процесса становится недоступной для обмена страниц или обмена рабочих наборов процессов. Страница остается заблокированной в оперативной памяти до тех пор, пока операционная система ее не разблокирует.

Блокировка страницы в рабочем наборе (LOCKING A PAGE IN THE WORKING SET) — действие, в результате которого страница процесса становится недоступной для выгрузки из рабочего набора процесса. Эта страница может быть выгружена при обмене рабочих наборов процессов. Страница остается заблокированной в рабочем наборе до тех пор, пока она не будет разблокирована.

Блокировка страницы по вводу-выводу (I/O LOCKDOWN) — состояние страницы, при котором она не может быть выгружена из памяти ни программой обмена страниц, ни планировщиком памяти.

Буфер трансляции (TRANSLATION BUFFER) — внутренняя кэш-память процессора, содержащая результат трансляции последних использованных виртуальных адресов.

Ввод-вывод на уровне блоков (BLOCK I/O) — способ доступа к данным, при котором программа обрабатывает файл на уровне блоков (физические записи) в отличие от обработки логических записей. Допускается прямой доступ к блокам в файле без учета организации файла или формата записей.

Вектор (VECTOR) — вектор прерывания — это известная системе ячейка памяти, содержащая начальный адрес программы, которая должна выполняться при появлении данного прерывания или исключительной ситуации. Система определяет отдельные векторы для каждого контроллера устройств и для классов исключительных ситуаций. Каждый системный вектор представляет собой длинное слово. Для обработки исключительных ситуаций пользователи могут объявлять два вектора программных исключительных ситуаций (первичный и вторичный) для любого из четырех режимов доступа. Каждый вектор содержит адрес программы обработки исключительной ситуации.

Взаимосвязанные задачи (COOPERATING TASKS) — задачи, взаимодействующие друг с другом в режиме позадачной связи. В частности, взаимосвязанные задачи должны принять соглашение относительно того, как они будут передавать и принимать сообщения, чтобы обеспечить одну передачу информации на каждый прием, и какая задача будет разъединять линию.

Виртуальный адрес (VIRTUAL ADDRESS) — 32-битное целое число, определяющее место байта в виртуальном адресном пространстве. Аппаратура управления памятью преобразует виртуальный адрес в физический.

Виртуальное адресное пространство (VIRTUAL ADDRESS SPACE) — набор всех виртуальных адресов, которые может использовать образ, выполняющийся в контексте данного процесса, для определения месторасположения инструкции или данных. Виртуальное адресное пространство представляет собой линейный массив из 4294967296 (2**32) адресов байтов.

Виртуальная память (VIRTUAL MEMORY) — набор ячеек в оперативной памяти и на диске, к которым обращаются по виртуальным адресам. Размер виртуальной памяти в любой системе зависит от размера доступной оперативной памяти и размера дисковой памяти, используемой для расширения оперативной памяти.

Владелец (OWNER) — в контексте «система—владелец—группа—все», владелец — это конкретный пользователь (группы), которому принадлежит файл, глобальная секция, почтовый ящик или кластер флагов событий.

Внешнее (глобальное) выражение (EXTERNAL (GLOBAL) EXPRESSION) — выражение, содержащее внешнее (глобальное) символическое имя.

Внешняя (глобальная) метка (EXTERNAL (GLOBAL) LABEL) — последовательность символов, записанная в поле метки, ограниченном двумя символами «двоеточие» (: :).

Внешнее (глобальное) символическое имя (EXTERNAL SYMBOL, GLOBAL SYMBOL) — символическое имя, определенное пользователем с помощью оператора двойного присваивания (==), или внешняя метка. Обращение к внешнему символическому имени может быть выполнено из любого модуля программы.

Внутренняя метка (INTERNAL LABEL) — последовательность символов, записанная в поле метки, ограниченном одним символом «двоеточие» (:).

Внутреннее символическое имя (INTERNAL LABEL) — символическое имя, определенное пользователем с помощью оператора прямого присваивания (=), или внутренняя метка.

Вспомогательный управляющий процесс (ANCILLARY CONTROL PROCESS (ACP)) — интерфейсный процесс между пользовательской программой и драйвером ввода-вывода. ACP выполняет функции управления файлами и каталогами. Примерами ACP являются: ACP файловой структуры, ACP магнитной ленты и ACP сети.

Вторичный вектор (SECONDARY VECTOR) — ячейка, в которой хранится начальный адрес программы обработки исключительного состояния. Эта программа будет выполняться, если данное состояние имеет место и либо первичный вектор обнулен, либо заданная им программа обработки принимает решение не обрабатывать это состояние.

Вторичный пароль (SECONDARY PASSWORD) — пароль пользователя, который может быть запрошен во время входа в систему сразу после корректного ввода основного пароля.

Вход в систему (LOGGING IN) — опознавание пользователя операционной системой. При входе в систему пользователи печатают учетное имя и пароль в ответ на подсказки системы. Если имя и пароль соответствуют информации, имеющейся в системе в учетном файле, то пользователю разрешается работа в системе.

Вызов (CALL) — вызов и передача управления указанной программе.

Выход, завершение (EXIT) — действия, совершаемые как при нормальном, так и при ненормальном окончании выполнения образа. К таким действиям относится, например, отсоединение каналов ввода-вывода и открепление кластеров общих флагов событий. При этом вызываются определенные пользователем или системой программы обработки выхода.

Выход из системы (LOGGING OUT) — ввод команды LOGOUT языка DCL, которая информирует операционную систему о том, что пользователь завершил работу с ситемой на данном терминале.

Глобальная секция (GLOBAL SECTION) — структура данных (например, глобальная общая область для Фортрана) или разделяемая секция образа, потенциально доступная для всех процессов в системе. Доступ к ней защищается привилегией и/или номером группы в коде идентификации пользователя.

Группа (GROUP) — группа пользователей, которые, как правило, имеют право на доступ к каталогам друг друга и к файлам внутри этих каталогов; набор заданий (процессов и их подпроцессов) с правом доступа к общим флагам событий группы и к таблицам логических имен.

Дельта-время, интервал времени (DELTA TIME) — значение времени, выражающее смещение от текущей даты и времени. Дельта-время всегда выражается в системе отрицательным числом, абсолютное значение которого используется в качестве смещения от текущего времени.

Диспетчер заданий (JOB CONTROLLER) — системный процесс, который создает контекст процесса некоторого задания, запускает процесс, выполняющий образ LOGIN для этого задания, сопровождает учетный раздел задания, а также завершает процесс и его подпроцессы.

Диспетчер исключительных ситуаций (EXCEPTION DISPATCHER) — процедура операционной системы, которая ищет программу обработки со-

стояния по возникновению исключительной ситуации. Если программа обработки не найдена, то образ, вызвавший эту ситуацию, завершается.

Длинное слово (LONGWORD) — четыре смежных байта (32 бита), начинающихся на границе адресуемого байта. Биты нумеруются справа налево от 0 до 31. Адресом длинного слова является адрес байта, содержащего бит 0.

Длинное слово состояния процессора (PROCESSOR STATUS LONGWORD (PSL)) — регистр процессора, состоящий из привилегированного слова состояния процессора (PPS) и слова состояния процессора (PSW). Информация о привилегированном состоянии процессора включает текущий уровень приоритета прерываний, предыдущий режим доступа, текущий режим доступа, бит стека прерываний, бит подавления трассировки и бит режима совместимости.

Дозавершение ввода-вывода, довывод (I/O RUNDOWN) — функция операционной системы, осуществляющая окончание всех выполняемых операций ввода-вывода при завершении образа.

Драйвер (DRIVER) — программа, под управлением которой выполняется физический ввод-вывод.

Ждать (WAIT) — стать неактивным. Процесс входит в состояние ожидания, когда он приостанавливает себя, «спит» или объявляет, что ему надо ждать события, ресурса, взаимного исключения и т. д.

Зависание системы (HANGING) — крайне медленная реакция системы. Если при интерактивной работе кажется, что команды с терминала никуда не поступают или ничего не выполняют, то говорят, что имеет место зависание системы.

Заголовок процесса (PROCESS HEADER) — структура данных, которая содержит аппаратный блок PCB, учетный раздел, информацию о квотах, таблицу секций процесса, список рабочего набора и таблицы страниц, определяющие виртуальное размещение процесса.

Загрузка (BOOT) — занесение в оперативную память копии операционной системы с системного диска.

Задание (JOB) — учетная единица, эквивалентная процессу и его подпроцессам (если они есть) и всем подпроцессам, которые они создали. Задания делятся на пакетные и интерактивные. Например, когда пользователь входит в систему, диспетчер заданий создает интерактивное задание для обработки пользовательских запросов, а когда программа управления передает этому диспетчеру командный входной файл, диспетчер создает пакетное задание.

Задержанное эхо (DEFFERED ECHO) — эхо на терминал не выдается до тех пор, пока процесс не будет готов принять ранее введенные символы.

Запрос на ожидание прерывания (WAIT FOR INTERRUPT REQUEST) — запрос, выданный программой драйвера запуска ввода-вывода после того, как она активизирует устройство. Этот запрос приводит к приостанову форк-процесса драйвера до тех пор, пока устройство не даст прерывание или не вызовет тайм-аут.

Зарезервировать устройство (ALLOCATE A DEVICE) — устройство может быть зарезервировано для монопольного применения некоторым пользовательским процессом только в том случае, если оно не зарезервированно ни за каким другим процессом.

Защитное действие (EVASIVE ACTION) — действие МОСВП, выполняемое в ответ на предполагаемые попытки несанкционированного доступа. Например, если неавторизованный пользователь пытается войти в систему, то обычно ее защитное действие состоит в блокировке на ограниченное время всех дальнейших попыток его входа.

Значение аргумента по умолчанию (DEFAULT VALUE) — значение, присвоенное формальному аргументу в списке аргументов макроопределения с помощью оператора прямого присваивания. Если в вызове макрокоманды

не указан фактический аргумент, то будет использовано значение аргумента по умолчанию.

Идентификатор процесса (PROCESS IDENTIFICATION (PID)) — 32-битовое двоичное значение, которое однозначно определяет процесс. Каждый процесс имеет идентификатор и имя.

Имя образа (IMAGE NAME) — имя файла, в котором хранится образ.

Имя пользователя (USER NAME) — учетное имя, которое пользователь печатает на терминале при регистрации в системе.

Имя процесса (PROCESS NAME) — символьная строка, содержащая от 1 до 15 символов в коде КОИ-8, которую можно использовать для идентификации процессов, выполняющихся под одним и тем же номером группы.

Имя устройства (DEVICE NAME) — поле в спецификации файла, определяющее устройство, на котором находится файл.

Имя устройства загрузки (BOOT NAME) — имя устройства, используемого для загрузки программ.

Имя файла (FILE NAME) — поле длиной от 1 до 39 символов в спецификации файла, которое определяет имя файла.

Индексный режим адресации (INDEXED ADRESSING MODE) — режим адресации, в котором адрес операнда определяется путем прибавления к базовому адресу содержимого индексного регистра, умноженного на размер операнда. Базовый адрес определяется транслятором исходя из основного режима адресации, а размер операнда — это число байт в операнде, которое зависит от типа данных операнда. Названия режимов адресации, использующих индексный режим, образуются добавлением слова «индексный» или «с индексацией» к основному режиму адресации: косвенно-регистровый с индексацией, индексный с автоувеличением, косвенный индексный с автоувеличением (или абсолютный индексный), индексный с автоуменьшением, индексный со смещением и косвенный индексный со смещением.

Индексный режим с автоувеличением (AUTOINCREMENT INDEXED MODE) — индексный режим адресации, в котором спецификатор базового операнда использует адресацию режима с автоувеличением.

Индексный режим с автоуменьшением (AUTODECREMENT INDEXED MODE) — индексный режим адресации, при котором спецификатор базового операнда использует адресацию режима с автоуменьшением.

Индексный режим со смещением (DISPLACEMENT INDEXED MODE) — индексный режим адресации, в котором спецификатор базового операнда использует адресацию режима со смещением.

Интерпретатор команд (COMMAND INTERPRETER) — системная программа, обеспечивающая интерпретацию команд диалогового языка DCL. Она выполняется в режиме супервизора в контексте процесса с целью синтаксической проверки и анализа команд, вводимых пользователем с терминала или представленных в командном файле.

Информационный блок задания (JOB INFORMATION BLOCK (JIB)) — связанная с заданием структура данных, которая содержит квоты, собранные вместе от всех процессов задания.

Исключительная ситуация (EXCEPTION) — событие (отличное от прерывания, инструкций перехода или вызова), обнаруженное аппаратурой или программами, которое изменяет нормальный порядок выполнения инструкций. Исключительная ситуация возникает только в результате выполнения инструкции, в то время как прерывание вызывается внешним и независимым от выполняющейся инструкции действием. Существуют три типа аппаратных исключительных ситуаций: ловушки, сбои и аварийные завершения.

Исполняемый образ (EXECUTABLE IMAGE) — образ, который может быть выполнен в рамках процесса. Для этого образ считывается из некоторого файла.

Исходная программа (SOURSE PROGRAMM) — последовательность исходных предложений, написанных на языке программирования.

Исходное предложение (SOURSE STATEMENT) — строка символов, которой может быть одна из инструкций СМ1700, операция прямого присваивания, директива, макрокоманда, комментарий.

Исходный модуль (SOURSE MODULE) — файл (или конкатенация нескольких файлов), содержащий исходные предложения Макроассемблера до первой директивы . END включительно.

Кадр вызова (CALL FRAME) — *См.* **Кадр стека** (STACK FRAME).

Кадр стека (STACK FRAME) — стандартная структура данных, построенная в стеке во время вызова процедуры. Она начинается с ячейки, адрес которой содержится в регистре FP, и продолжается по направлению к младшим адресам; удаляется при возврате из процедуры. Кадр стека имеет и другое название — Кадр вызова (CALL FRAME).

Канал (CHANNEL) — логический канал, соединяющий процесс пользователя с физическим устройством. Процесс пользователя обращается к операционной системе назначить канал для связи с этим устройством.

Каталог (DIRECTORY) — файл, содержащий краткие характеристики файлов, хранящихся на диске или ленте. Каталог содержит имя, тип и номер версии каждого файла в данном наборе, а также уникальное число, которое определяет фактическое расположение файла и указывает на список его атрибутов. *См.* **Подкаталог.**

Квадрослово (QUADWORD) — четыре смежных слова (64 разряда), начинающиеся на границе адресуемого байта. Разряды нумеруются от 0 до 63 справа налево. Квадрослово определяется адресом слова, содержащего младший разряд (разряд 0).

Квалификатор (QUALIFIER) — часть командной строки, которая модифицирует или уточняет действие команды. Квалификатор, если он есть, следует за командой или параметром, к которому он относится.

Квота (QUOTA) — общее количество системного ресурса, например время процессора, которым периодически разрешено пользоваться некоторому процессу, или величина, определяющая объем пространства, выделенный файлу пользователя на общем томе. Квоты указываются системным администратором в файле авторизации пользователя.

Кластер (CLUSTER) — набор смежных блоков, который является основной единицей распределения пространства на дисковом томе с файловой структурой; набор страниц, загружаемый в память за одну операцию обмена страниц.

Кластер общих флагов событий (COMMON EVENT FLAG CLUSTER) — набор из 32 флагов событий, который позволяет взаимодействующим процессам посылать друг другу информацию о событиях. Кластеры общих флагов событий создаются по мере необходимости. Процесс может связываться не более чем с двумя кластерами общих флагов событий.

Кластер флагов событий (EVENT FLAG CLUSTER) — набор из 32 флагов событий, который используется для извещения о событиях. Для каждого процесса определены четыре кластера: два — локальных для процесса и два — общих. Восемь локальных флагов событий процесса зарезервированы для использования системой. *См.* **Кластер общих флагов событий.**

Код идентификации пользователя (USER IDENTIFICATION CODE (UIC) — средство идентификации, присваиваемое каждому пользователю, на основе которого определяются его привилегии (доступ на чтение и/или на запись, а для файлов — доступ на обработку и/или удаление и т. п.). Числовой код UIC состоит из двух разделенных запятыми и заключенных в квадратные скобки идентификаторов. Первый определяет номер группы, второй — номер пользователя в группе.

Код состояния (STATUS CODE) — значение длинного слова, которое указывает на успешное или ошибочное выполнение конкретной функции. Например, программы системного обслуживания при завершении своей работы возвращают код состояния в регистре R0.

Код функции ввода-вывода (I/O FUNCTION CODE) — 6-битное значение, заданное в программе системного обслуживания очереди запросов ввода-вывода. Описывает операцию ввода-вывода, которую необходимо выполнить.

Коды условий (CONDITION CODES) — четыре бита в слове состояния процессора, которые отображают результаты последней выполненной инструкции.

Командная процедура (COMMAND PROCEDURE) — файл, содержащий команды и данные, которые обрабатываются интерпретатором команд языка DCL, а не вводятся по отдельности с терминала. Средство автоматического выполнения команд языка DCL в операционной системе позволяет пользователям применять такие методы программирования, как циклы, счетчики, метки и подстановки символов с целью построения сложных последовательностей команд, которые могут быть изменены при взаимодействии с пользователем.

Командный уровень (COMMAND LEVEL) — определяет входной поток для интерпретатора команд диалогового языка DCL. Первоначальный входной поток всегда имеет нулевой командный уровень. Каждый последующий вызов некоторой командной процедуры с текущего командного уровня увеличивает командный уровень, а завершение командной процедуры уменьшает этот уровень.

Контекст (CONTEXT) — составная часть процесса, определяющая среду выполнения образа. *См.* **Контекст процесса, Аппаратный контекст** и **Программный контекст.**

Контекст процесса (PROCESS CONTEXT) — аппаратный и программный контексты процесса.

Косвенный индексный режим с автоувеличением (AUTOINCREMENT DEFERRRED INDEXED MODE) — индексный режим адресации, в котором спецификатор базового операнда использует адресацию косвенного режима с автоувеличением.

Косвенный индексный режим со смещением (DISPLACEMENT DEFEREND INDEXED MODE) — индексный режим адресации, в котором указанный базовый операнд использует адресацию косвенного режима со смещением.

Косвенный командный файл (INDIRECT COMMAND FILE) — *См.* **Командная процедура.**

Косвенный режим с автоувеличением (AUTOINCREMENT DEFERRED MODE) — режим адресации, при котором указанный регистр содержит адрес длинного слова с адресом фактического операнда. Содержимое регистра увеличивается на 4 (число байт в длинном слове). Если в качестве регистра используется PC, то этот режим называется абсолютным режимом.

Косвенный режим со смещением (DISPLACEMENT DEFEREND MODE) — режим адресации, при котором расширение спецификатора является смещением в байтах, словах или длинных словах. Смещение расширяется со знаком до 32 бит и добавляется к базовому адресу, извлекаемому из указанного в инструкции регистра. Результатом является адрес длинного слова, которое содержит адрес фактического операнда. Если используется режим адресации через PC, то в качестве базового адреса применяется скорректированное содержимое PC. Оно является адресом первого байта после расширения спецификатора.

Лимит (LIMIT) — размер или число элементов требуемых системных ресурсов (почтовые ящики, захваченные страницы, запросы ввода-вывода,

открытые файлы и т. д.), которые одновременно разрешается иметь выполняющемуся процессу. Лимиты задаются администратором системы в файле авторизации пользователя.

Личная секция (PRIVATE SECTION) — секция образа процесса, которая не разделяется между процессами.

Ловушка (TRAP) — состояние исключительной ситуации, которое имеет место в конце инструкции, вызвавшей эту ситуацию. Счетчик инструкций, сохраняемый в стеке, является адресом следующей инструкции, которая должна быть выполнена обычным образом.

Логическое имя (LOGICAL NAME) — имя, определенное пользователем, для любой части или для всей спецификации файла. Определения логических имен хранятся в одной из таблиц логических имен: процесса, задания, группы или системы.

Локальная метка (LOCAL LABEL) — метка, определяющая адрес внутри блока локальных меток. Формат имени локальной метки: N⊙, где N — любое целое десятичное число в диапазоне от 1 до 65535. Пользователю рекомендуется создавать локальные метки в диапазоне от 1⊙ до 29999⊙, так как диапазон от 30000⊙ до 65535⊙ используется транслятором для автоматической генерации локальных меток.

Локальность программы (PROGRAM LOCALITY) — характеристика программы, показывающая, насколько близко или далеко размещены в виртуальной памяти ячейки, к которым обращается программа за некоторый период времени.

Локальное символическое имя (LOCAL SYMBOL) — символическое имя, имеющее значение только в том модуле, в котором оно определено. Символические имена, не опознанные транслятором как глобальные, считаются локальными; символическое имя командного языка, которое доступно только на текущем командном уровне и на уровнях, вызванных с него.

Макрокоманда, макровызов (MACRO) — исходное предложение, в поле операции которого находится имя макрокоманды и, возможно, список фактических аргументов в поле операнда.

Маска входа (ENTRY MASK) — слово, последовательные номера битов которого задают сохраняемые и восстанавливаемые регистры при вызове подпрограммы или процедуры, использующей инструкции CALL и RET.

Малый процесс (SMALL PROCESS) — системный процесс, который в своем виртуальном адресном пространстве не содержит области управления и имеет сокращенный контекст. Примерами малых процессов являются процесс обмена рабочих наборов и пустой процесс. Малые процессы диспетчеризуются так же, как и пользовательские процессы, но до завершения своей работы они должны оставаться резидентными, т. е. не могут быть выгружены.

Метка (LABEL) — символическое имя, определенное пользователем, которое идентифицирует виртуальный адрес исходной программы. Различают метки внешние, внутренние, локальные.

Модификатор функции ввода-вывода (I/O FUNCTION MODIFIER) — 10-битное значение, указанное в программе системного обслуживания очереди запросов ввода-вывода. Модифицирует код функции ввода-вывода, например ввод с терминала без эхо-сопровождения.

Назначенный ресурс (DEDICATED) — системный ресурс (устройство ввода-вывода, образ или вся система), прикрепленный к одной прикладной или системной программе.

Назначить канал (ASSIGN A CHANNEL) — установить логическую связь между пользовательским процессом и устройством ввода-вывода.

Нарушение доступа (ACCESS VIOLATION) — попытка обратиться по адресу, который не отображен в виртуальной памяти или недоступен при текущем режиме доступа.

Непрерывная область (CONTIGUOUS AREA) — группа физически смежных блоков.

Номер виртуального блока (VIRTUAL BLOCK NUMBER (VBN)) — адрес блока на устройстве большой емкости. Первый блок файла всегда является виртуальным блоком 1.

Номер виртуальной страницы (VIRTUAL PAGE NUMBER (VPN)) — адрес страницы виртуальной памяти.

Номер логического блока (LOGICAL BLOCK NUMBER (LBN)) — логический номер блока на устройстве внешней памяти. Логические блоки, из которых составлен том, помечены последовательно, начиная с нулевого блока. *См.* **Номер физического блока** (PHYSICAL BLOCK NUMBER).

Номер физического блока (PHYSICAL BLOCK NUMBER) — физический адрес блока на устройстве большой емкости, который состоит из номера цилиндра, дорожки и номера сектора. *См.* **Номер логического блока** (LOGICAL BLOCK NUMBER), **Номер виртуального блока** (VIRTUAL BLOCK NUMBER).

Номер физической страницы (PAGE FRAME NUMBER (PFN)) хранящейся в оперативной памяти.

Область, регион (AREA) — зона индексного файла, поддерживаемая СУД, которая содержит произвольное число бакетов. В файле может быть от 1 до 255 областей. Пользователь может определить размещение и/или специфические размеры бакета для конкретных областей файла.

Область системы (SYSTEM REGION) — третья четверть виртуального адресного пространства или половина пространства системы с младшими адресами. Виртуальные адреса области системы разделяются между процессами. Примерами структур данных, отображаемых виртуальными адресами области системы, являются векторы входов в систему, блок SCB, таблица SPT и таблицы страниц процесса.

Область программы (PROGRAM REGION (P0)) — половина адресного пространства процесса с младшими адресами (область P0). Область программы содержит текущий, выполняемый в рамках процесса образ и другие программы пользователя, вызванные этим образом.

Область управления (CONTROL REGION (P1)) — часть адресного пространства процесса со старшими адресами (область P1), которая используется системой для управления процессом. Эта информация содержит, в частности, стек ядра, стек режима управления, стек супервизора, постоянные каналы ввода-вывода, векторы исключительных ситуаций и динамически используемые системные процедуры (такие, как процедуры интерпретатора команд). Как правило, стек пользователя также находится в области управления.

Образ (IMAGE) — программа, которая может быть выполнена на ЭВМ. Образ получается путем трансляции исходного текста программы в объектный модуль и обработки его компоновщиком. Имеются три типа образов: исполняемые, т. е. образы соответствующих процессов; разделяемые, т. е. образы, которые могут быть выполнены только путем обращения от исполняемого образа (при использовании в нескольких процессах в памяти содержится только одна копия разделяемого образа); системные, т. е. образы, которые выполняются только путем начальной загрузки.

Образ основного режима (NATIVE IMAGE) — образ, инструкции которого выполняются в основном режиме.

Обмен рабочих наборов (SWAPPING) — метод разделения ресурсов памяти между несколькими процессами за счет переписи всего рабочего набора во внешнюю память (выгрузка) и чтение другого рабочего набора в оперативную память (загрузка). Например, рабочий набор процесса может быть переписан во внешнюю память на время ожидания этим процессом завер-

шения ввода-вывода, выполняющегося на медленном устройстве. Он будет снова занесен в балансный набор после завершения ввода-вывода.

Обмен страниц (PAGING) — перенос затребованных страниц выполняющегося процесса из внешней памяти на диске в оперативную память. В МОСВП обмен страниц процесса выполняется в том случае, когда этот процесс обращается к большему количеству страниц, чем ему разрешено иметь в его рабочем наборе, или же когда процесс первый раз активизирует образ в памяти. При обращении процесса к странице, которой нет в рабочем наборе, происходит страничный отказ. Он приводит к тому, что системная программа обмена страниц считывает в память данную страницу, если она находится на диске (и другие связанные с ней страницы в зависимости от размера кластера при страничном отказе), заменяя по мере необходимости те страницы, отказ по которым произошел наиболее давно. Обмен страниц процесса производится только за счет ресурсов самого процесса.

Октаслово (OCTAWORD) — восемь смежных слов (128 разрядов), начинающихся на границе адресуемого байта. Разряды нумеруются справа налево от 0 до 127. Октаслово определяется адресом слова, содержащего младший разряд (разряд 0).

Оператор **присваивания** (ASSGMENT STATEMENT) — используется для присвоения определенного значения символическому имени. Символические имена могут определять синонимы для команд системы или могут использоваться в качестве переменных в командных процедурах.

Операция (OPERATION) — структурный элемент исходного предложения, определяющий действие, которое должно быть выполнено. Операцией может быть обозначение машинной инструкции, директива, макрокоманда.

Операция дополнения (COMPLEMENT OPERATION) — унарная операция, обозначаемая символами ^C, которая определяет дополнение указанного значения.

Операция КОИ-8 — унарная операция, обозначаемая как ^A, которая осуществляет преобразование последовательности графических символов в коды КОИ-8, отводя при этом на один символ один байт.

Операция регистровой маски (REGISTER MASK OPERATION) — унарная операция, обозначаемая как ^M, которая формирует регистровую маску для указанных регистров и спецификаторов арифметического прерывания. *См.* **Регистровая маска.**

Операция управления основанием системы счисления (NUMERIC CONTROL OPERATION) — унарная операция, обозначаемая двумя символами ^B, ^D, ^O или ^X, которая указывает на то, что значение терма или выражения должно обрабатываться в двоичной (^B), десятичной (^D), восьмеричной (^O) или шестнадцатеричной (^X) системе счисления.

Операция формата с плавающей запятой (FLOATING-POINT OPERATION) — унарная операция, обозначаемая как ^F, которая осуществляет преобразование указанного числа с плавающей запятой в его внутреннее (четырехбайтовое) представление.

Определение символического имени (SYMBOL DEFINITION) — исходное предложение, в котором символическому имени ставится в соответствие определенное значение. Если символическое имя определяется с помощью оператора прямого присваивания, то ему ставится в соответствие указанный элемент данных; если символическое имя является меткой, то ему ставится в соответствие значение счетчика текущего адреса.

Основной режим (NATIVE MODE) — режим работы процессора, в котором выполняются инструкции СМ1700.

Относительное выражение (RELATIVE EXPRESSION) — выражение, значение которого фиксировано относительно начала программной секции, в которой оно появилось.

Отображение номера физической страницы (PAGE FRAME NUMBER MAPPING (PEN MAPPING)) — отображение некоторой секции в одну или несколько страниц физической памяти или в пространство ввода-вывода.

Отсоединенный процесс (DETACHED PROCESS) — процесс, который не имеет владельца; процесс — предок дерева подпроцессов. Отсоединенный процесс создается диспетчером заданий при входе пользователя в систему, при инициализации пакетного задания или при запросе на соединение логического канала. Диспетчер задания не является владельцем создаваемых им процессов пользователей, поэтому данные процессы называются отсоединенными.

Очередь (QUEUE DATA TYPE) — тип данных, используемый для хранения списка элементов, подлежащих обработке. Каждый элемент очереди определяется своим адресом и связан с соседними элементами парой длинных слов. По типу используемой связи различают абсолютные и относительные очереди.

Очередь ввода-вывода (QUEUE INPUT/OUTPUT (QIO)) — организуется программой системного обслуживания очереди запросов ввода-вывода, которая управляет запросами ⊙QIO и ⊙QIOW. Готовит запрос ввода-вывода для обработки драйвером и выполняет независимую от устройств предобработку данного запроса.

Очередь процессов по состоянию (STATE QUEUE) — список процессов, находящихся в одинаковом состоянии обработки. Очереди процессов по состояниям используются планировщиком для выбора процессов на выполнение. Такие очереди включают процессы, ждущие общих флагов событий, приостановленные процессы и готовые к выполнению процессы.

Очередь спула (SPOOL QUEUE) — список файлов процессов, которые должны быть обработаны симбионтом. Например, очередь построчно-печатающего устройства — это список файлов, которые необходимо распечатать на таком устройстве.

Пакет запроса ввода-вывода (I/O REGUEST PACKET (IRP)) — структура данных ввода-вывода, описывающая отдельный запрос ввода-вывода. Такой пакет для каждого запроса ввода-вывода создается программой системного обслуживания очереди запросов ввода-вывода. МОСВП и соответствующий драйвер устройства используют информацию этого пакета для обработки запроса.

Пароль (PASSWORD) — символьная строка, используемая для защиты программ и данных пользователя, а также системы от несанкционированного доступа. Вводится пользователем при входе в систему. Существуют два вида паролей — системные и пользовательские.

Первичный вектор (PRIMARY VECTOR) — ячейка с начальным адресом программы обработки исключительного состояния, которая должна выполняться при его возникновении.

Переключение контекста (CONTEXT SWITCHING) — прерывание работы процесса и переключение на обработку другого процесса. Операционная система сохраняет аппаратный контекст прерванного процесса в его аппаратном блоке PCB; загружает в аппаратный контекст аппаратный PCB другого процесса и планирует этот процесс на выполнение.

Планировщик памяти (SWAPPER) — процесс, выполняющий общесистемное планирование памяти. Планировщик памяти переписывает модифицированные страницы во внешнюю память, создает прототип новых процессов, сжимает физический размер неактивных процессов, удаляет и заносит в балансные наборы процессы, ожидающие выполнения.

Подкаталог (SUBDIRECTORY) — каталог файлов, ссылка на который находится в каталоге более высокого уровня.

Подключение к вектору прерывания (CONNECT-TO-INTERRUPT) — функция, подключающая процесс к вектору прерывания устройства. Для подключения к вектору прерывания процесс должен создать карту простран-

ства ввода-вывода программы, содержащей этот вектор. *См.* **Отображение номера физической страницы.**

Подпрограмма (SUBROUTINE) — последовательность исходных предложений, содержащих инструкции СМ1700, выполнение которых заканчивается инструкцией RSB. Вызов подпрограмм осуществляется с помощью инструкций BSB, JSB.

Позиционно-независимая программа (POSITION-INDEPENDENT CODE) — программа, которая без модификации может выполнить свои функции правильно, независимо от того, где она размещена в виртуальном адресном пространстве (даже если ее местоположение изменено после компоновки).

Попытка незаконного доступа (BREAKIN ATTEMPT) — попытка неавторизованного пользователя получить доступ к системе.

Почтовый ящик (MAILBOX) — структура данных системы, которая трактуется как устройство, предназначенное для общей связи между процессами. Связь через почтовый ящик аналогична работе с любым устройством ввода-вывода. Отправитель записывает информацию в почтовый ящик, получатель читает ее из почтового ящика. Определены некоторые общесистемные почтовые ящики, информацию из которых читает регистратор ошибок и процесс сообщений оператору OPCOM.

Предварительный ввод (TYPE-AHEAD) — метод управления терминалом, при котором пользователь может ввести команды и данные, когда программа отрабатывает ранее введенную команду. Предварительно введенные команды не отображаются на терминале до тех пор, пока командный процессор не будет готов к их обработке. Они хранятся в буфере предварительного ввода.

Прерывание (INTERRUPT) — событие (отличное от исключительной ситуации, инструкций перехода, выбора или вызова), которое изменяет обычный порядок выполнения инструкций. Прерывания обычно являются внешними по отношению к процессу, выполняющемуся в тот момент, когда происходит прерывание. *См.* **Программное прерывание и срочное прерывание.**

Привилегии (PRIVILEGES) — дополнительные функциональные возможности, с помощью которых можно воздействовать на ресурсы системы и/или на ее целостность.

Привилегии образа (IMAGE PRIVILEGES) — привилегии, приписываемые образу при его установке. *См.* **Привилегии процесса.**

Привилегии пользователя (USER PRIVILEGES) — привилегии, предоставляемые пользователю администратором системы. *См.* **Привилегии процесса** (PROCESS PRIVILEGES).

Привилегии процесса (PROCESS PRIVILEGES) — привилегии, предоставляемые процессу системой. Эти привилегии являются комбинацией привилегий пользователя и привилегий соответствующего образа.

Приоритет (PRIORITY) — величина, приписанная процессу для определения его привилегий при получении ресурсов системы.

Приоритет процесса (PROCESS PRIORITY) — приоритет, приписанный процессу с целью его планирования, а также выделения ресурсов. Операционная система распознает 32 уровня приоритетов процессов, где 0 — это низший, а 31 — высшие уровни. Уровни от 16 до 31 используются для процессов реального времени. Система не модифицирует приоритет такого процесса (хотя это могут сделать администратор системы или сам процесс). Уровни от 0 до 15 используются для обычных процессов. Система может временно увеличить приоритет такого процесса, основываясь на его активности.

Приостанов (SUSPENSION) — состояние, в котором процесс не активен, но известен системе. Приостановленный процесс становится снова активным только в том случае, когда другой процесс просит операционную систему возобновить работу приостановленного процесса.

Пробудить (WAKE) — активизировать «спящий» процесс. Такой процесс может быть «разбужен» вызовом, запланированным на определенное время.

Программа запуска ввода-вывода (START I/O ROUTINE) — программа драйвера устройства, которая отвечает за получение необходимых ресурсов (например, канала данных контроллера) и за активизацию устройства.

Программа обработки исключительного состояния (CONDITION HANDLER) — при возникновении исключительной ситуации эта программа выполняет определенные действия по восстановлению системы с целью продолжения работы прерванного процесса. Программы обработки исключительных состояний выполняются в контексте процесса в режиме доступа программы, которая вызвала исключительную ситуацию.

Программа обмена страниц (PAGER) — набор процедур режима ядра, который выполняется при страничном отказе. Программа обмена страниц делает доступной в оперативной памяти ту страницу, по которой произошел страничный отказ, после чего образ может продолжить свою работу.

Программа обслуживания прерывания (INTERRUPT SERVICE ROUTINE (ISR)) — программа, выполняемая при прерывании.

Программа связи с оператором (MONITOR CONSOLE ROUTINE (MCR)) — дополнительный интерпретатор командного языка в системе МОСВП, обеспечивающий совместимость с ОСРВ.

Программная секция (PROGRAM SECTION (PSECT)) — часть программы с заданной защитой и набором атрибутов управления памятью. Программные секции, имеющие одинаковые атрибуты, собираются компоновщиком вместе и образуют секцию образа.

Программное прерывание (SOFTWARE INTERRUPT) — прерывание, генерируемое на уровнях приоритетов прерываний от 1 до 15; оно может быть запрошено только программой.

Программный блок управления процессом — программный PCB (SOFTWARE PROCESS CONTROL BLOCK (SOFTWARE PCB)) — структура данных, используемая для хранения программного контекста процесса. Операционная система определяет программный PCB для каждого процесса при его создании. Программный PCB включает следующую информацию о процессе: текущее состояние, адрес во внешней памяти процесса, выгруженного из основной памяти, уникальный идентификатор процесса, адрес заголовка процесса (который содержит аппаратный PCB). Программный PCB размещается в пространстве виртуальных адресов области системы. Он не участвует в обмене вместе с процессом.

Программный контекст (SOFTWARE CONTEXT) — описание процесса и выделенных ему ресурсов, поддерживаемое операционной системой. *См.* **Программный блок управления процессом.**

Продолжить, возобновить (RESUME) — активизировать приостановленный процесс. *См.* **Пробудить** (WAKE).

Просмотр остатка данных на диске (DISK SCAVENGING) — получение информации о данных, которые удалил (стер) их владелец на диске. Хотя при помощи обычных средств эта информация больше недоступна пользователю, тем не менее ее можно восстановить и использовать с помощью одного из методов просмотра данных.

Пространство ввода-вывода (I/O SPACE) — области в физическом адресном пространстве, которые содержат регистры конфигурации, команд и состояний, а также регистры данных. Такие области физически не смежны.

Пространство процесса (PROCESS SPACE) — адресное пространство, в котором размещаются инструкции и данные процесса. Оно делится на область программы и область управления.

Пространство системы (SYSTEM SPACE) — половина виртуального адресного пространства со старшими адресами.

Пространство физических адресов (PHYSICAL ADDRESS SPACE) — набор всех возможных 32-битных физических адресов, которые могут быть

использованы для обращения к ячейкам памяти (пространству памяти) или к регистрам устройств (пространству ввода-вывода).

Процедура (PROCEDURE) — последовательность исходных предложений, содержащих инструкции СМ1700, выполнение которых заканчивается инструкцией RET. Вызов процедуры осуществляется с помощью инструкций CALLS, CALLG.

Процедура входа в систему (LOGIN FILE) — командная процедура, выполняемая автоматически при входе пользователя в систему и в начале пакетного задания.

Процесс (PROCESS) — основная единица планирования и управления в системе МОСВП, состоящая из образа и контекста, в котором он выполняется. *См.* **Образ** (IMAGE) и **Контекст** (CONTEXT).

Процесс-владелец (OWNER PROCESS) — процесс или подпроцесс, который создал подпроцесс.

Процесс обмена рабочих наборов (WORKING SET SWAPPER) — системный процесс, который заносит рабочие наборы процессов в балансный набор и удаляет их из балансного набора.

Процесс-прототип (SHELL PROCESS) — заранее определенный процесс, который копируется инициатором заданий с целью создания минимального контекста, необходимого для объявления процесса.

Процесс сообщений оператору (OPERATOR COMMUNICATION MANAGER (OPCOM)) — системный процесс, получающий информацию об определенном событии и передающий соответствующее сообщение на терминал оператору. Процесс OPCOM всегда находится в активном состоянии.

Прямой ввод-вывод (DIRECT I/O) — операция ввода-вывода, во время выполнения которой передача данных происходит непосредственно из буфера процесса.

Псевдоустройство (PSEUDO DEVICE) — виртуальное устройство, рассматриваемое системой или пользователем как устройство ввода-вывода, хотя оно и не является некоторым отдельным физическим устройством.

Пустой процесс (NULL PROCESS) — малый системный процесс, который имеет самый низкий приоритет в системе (нулевой) и занимает целиком один класс приоритетов. Единственная функция пустого процесса заключается в использовании свободного времени процессора.

Рабочий набор (WORKING SET) — набор страниц в пространстве процесса, к которым выполняющийся процесс может обращаться без страничного отказа. Рабочий набор выполняющегося процесса должен быть размещен в основной памяти. Оставшиеся страницы этого процесса, если они имеются, находятся или в основной памяти, но не в рабочем наборе, или во внешней памяти.

Развернуть стек вызова (UNWIND THE CALL STACK) — удалить кадры вызова из стека путем обратного отслеживания вложенных вызовов процедур. При этом используется текущее содержимое регистра FP и содержимое регистров FP, хранимых в стеке каждого кадра вызова.

Разделение файла (FILE SHARING) — возможность одновременного доступа нескольких процессов к относительному или индексному файлу.

Разделяемый образ (SHAREABLE IMAGE) — образ, у которого разрешены все внутренние ссылки, однако для получения выполняемого образа он должен быть скомпонован с одним или несколькими объектными модулями. Разделяемый образ не может быть выполнен. Он может применяться для хранения библиотек или программ.

Размер кластера при страничном отказе (PAGE FAULT CLUSTER SIZE) — количество страниц, считываемых в память при страничном отказе.

Разрешение ловушки (TRAP ENABLE) — три бита в слове состояния процессора, которые управляют действиями процессора при некоторых арифметических исключительных ситуациях.

Расширение (EXTENSION) — размер области, выделяемой в конце файла всякий раз, когда последующая запись превышает зарезервированную длину файла.

Расширение имени файла (FILE NAME EXTENSION) — поле в спецификации файла, состоящее из символа точки «.», за которой следует от 0 до 39 символов определения типа. По соглашению это поле идентифицирует общий класс файлов, которые имеют одинаковые характеристики или применение. (Например, . OBJ — объектные файлы).

Регистр длины области программы (PROGRAM REGION LENGTH REGISTER (P0LR)) — регистр процессора, который содержит количество элементов в таблице страниц области программы процесса (P0).

Регистр длины области системы (SYSTEM LENGHT REGISTER (SLR)) — регистр процессора, содержащий размер таблицы страниц области системы в длинных словах.

Регистр длины области управления (CONTROL REGION LENGTH REGISTER (P1LR)) — регистр процессора, который содержит величину неиспользуемой области таблицы страниц области управления процессом (P1).

Регистр общего назначения (GENERAL REGISTER) — любой из шестнадцати 32-разрядных регистров, используемых в качестве первичных операндов инструкций основного режима. Общие регистры первичных операндов включают 12 регистров общего назначения, которые могут быть использованы как накопители, счетчики и указатели ячеек основной памяти, а также четыре специальных регистра: указатель кадра (FP), указатель аргумента (AP), указатель стека (SP), счетчик инструкций (PC).

Регистратор ошибок (ERROR LOGGER) — системный процесс, который освобождает буферы регистрации ошибок и записывает сообщения об ошибках в файл ошибок. Ошибки, регистрируемые системой, включают системные ошибки памяти, ошибки устройств и тайм-ауты устройств, а также прерывания с неправильными адресами векторов.

Регистровая маска (REGISTER SAVE MASK) — слово, в котором номера разрядов соответствуют номерам регистров общего назначения. По установленным разрядам этого слова определяется, содержимое каких регистров должно быть сохранено и впоследствии восстановлено перед вызовом процедуры.

Регистры возврата значений (VALUE RETURN REGISTERS) — общие регистры R0 и R1, используемые по соглашению для возврата значений функций. Вызванными процедурами эти регистры не сохраняются. Они доступны любой вызванной процедуре в качестве временных регистров. Все остальные регистры (R2, R3, ... R11, AP, FP, SP, PC) могут быть сохранены при вызовах процедур.

Реентерабельный код, реентерабельная программа (REENTRANT CODE) — программа, которая при выполнении никогда не модифицируется. Одну и ту же копию процедуры или программы, написанной в виде реентерабельного кода, могут разделять несколько пользователей.

Режим адресации (ADDRESSING MODE) — способ определения адреса операнда инструкции. Различают режимы адресации через регистры общего назначения, через счетчик инструкций (PC), а также адресацию в инструкциях перехода. Существует семь основных режимов и шесть индексных режимов адресации. Режимы адресации с индексацией используются в сочетании с другими режимами.

Режим доступа (ACCESS MODE) — один из четырех режимов доступа к процессору, в котором функционирует программное обеспечение: ядро, управление, супервизор, пользователь (в порядке убывания привилегий и защиты). Когда процессор находится в режиме ядра, исполняемая программа имеет полный контроль над системой и несет за нее ответственность. В любом другом режиме процессор не может выполнять привилегированные

инструкции. Текущий режим доступа определяется в длинном слове состояния процессора.

Режим исключительной ситуации по ошибке (FAILURE EXCEPTION-MODE) — режим выполнения, выбранный процессом, который дает указание системе объявить состояние исключительной ситуации, если в результате вызова программы системного обслуживания произойдет ошибка. Обычный для программ системного обслуживания режим заключается в возврате кода состояния ошибки, который должен быть проверен процессом.

Режим обмена (SWAP MODE) — состояние выполнения процесса, которое определяет возможность выгрузки балансного набора этого процесса. Если для процесса режим обмена запрещен, то рабочий набор процесса блокируется в балансном наборе.

Режим ожидания ресурса (RESOURCE WAIT MODE) — состояние процесса, когда выдав запрос на требуемый ресурс, он будет ждать, пока этот ресурс не станет доступным.

Режим перехода (BRANCH MODE) — режим адресации, при котором спецификатор операнда инструкции представляет собой смещение со знаком (размером в байт или слово). Оно добавляется к содержимому счетчика инструкций PC, представляющего собой адрес первого байта со смещением, и результат дает адрес перехода.

Режим пользователя (USER MODE) — наименее привилегированный режим доступа к процессору. В этом режиме работают пользовательские процессы и программы исполнительной библиотеки.

Режим с автоувеличением (AUTOINCREMENT MODE) — режим адресации, в котором содержимое специфицированного регистра используется как адрес операнда, а затем содержимое регистра увеличивается на размер этого операнда.

Режим с автоуменьшением (AUTODECREMENT MODE) — режим адресации, в котором содержимое выбранного регистра уменьшается, и результат используется как адрес фактического операнда для инструкции. Содержимое регистра уменьшается в соответствии с контекстом типа данных этого регистра: на 1 — для байта, на 2 — для слова, на 4 — для длинного слова и чисел с плавающей запятой формата F, на 8 — для квадрослова и чисел с плавающей запятой формата H.

Режим совместимости (COMPATIBILITY MODE) — режим работы, который позволяет центральному процессору СМ1700 выполнить непривилегированные инструкции 16-разрядных СМ ЭВМ с магистральной архитектурой. МОСВП обеспечивает режим совместимости, предоставляя для образа задачи ОСРВ среду функционирования этой операционной системы. Процедуры режима совместимости МОСВП перехватывают вызовы управляющей программы ОСРВ и преобразуют их в вызовы ядра МОСВП.

Режим со смещением (DISPLACEMENT MODE) — режим адресации, при котором расширение спецификатора является смещением в байтах, словах, длинных словах. Смещение расширяется со знаком до 32 бит и добавляется к базовому адресу, извлекаемому из указанного в инструкции регистра. В результате получается адрес фактического операнда. Если используется режим адресации через PC, то в качестве базового адреса используется скорректированное содержимое PC, которое является адресом первого байта после расширения спецификатора.

Режим супервизора (SUPERVISOR MODE) — третий наиболее привилегированный режим доступа к процессору. В режиме супервизора работает интерпретатор команд диалогового языка операционной системы.

Режим управления (EXECUTIVE MODE) — второй по привилегированности режим доступа к процессору. В этом режиме работает система управления данными (СУД) и ряд процедур системного обслуживания.

Режим ядра (KERNEL MODE) — самый привилегированный режим доступа к процессору. В режиме ядра работают наиболее привилегированные

программы операционной системы, например драйверы ввода-вывода и программа обмена страниц.

Ресурс (RESOURCE) — любой ресурс вычислительного комплекса, выделяемый для реализации задания: оперативная память, устройства ввода-вывода, время центрального процессора, наборы данных. Контроль за использованием ресурсов выполняется с помощью квот и лимитов.

Сбой (FAULT) — состояние аппаратной исключительной ситуации, которое возникает в середине инструкции и оставляет регистры и память в таком состоянии, что устранение сбоя и повторное выполнение инструкции даст правильный результат.

Сегмент ввода-вывода образа (IMAGE I/O SEGMENT) — часть области управления, которая содержит внутренние блоки доступа к файлу (IFAB) системы СУД, а также буферы ввода-вывода для образа, выполняющегося в данный момент в рамках некоторого процесса.

Сегмент ввода-вывода процесса (PROCESS I/O SEGMENT) — та часть области управления процесса, которая содержит блок внутреннего (для СУД) доступа к каждому открытому файлу и буфера ввода-вывода, включая буфер команд соответствующего интерпретатора и дескрипторы команд.

Секция образа (IMAGE SECTION (ISECT)) — группа программных секций с одинаковыми атрибутами (доступны только для чтения, доступны для чтения и записи, абсолютные, перемещаемые и т. д.). Секция образа является единицей выделения виртуальной памяти для образа.

Сжатие рабочего набора (SHRINK WORKING SET) — один из имеющихся у планировщика памяти способов получения страниц в оперативной памяти. Он сжимает размеры рабочих наборов выбранных процессов. *См.* **Обмен рабочих наборов** (SWAPPING).

Символьная строка (CHARACTER STRING) — непрерывная последовательность байтов, содержимое которых интерпретируется как графические символы. Символьная строка определяется двумя атрибутами: адресом и длиной. Адрес строки является адресом байта, содержащего первый символ строки. Последующие символы хранятся в байтах с возрастающими адресами. Длина представляет собой число символов в строке.

Система управления данными — СУД (RECORD MANAGEMENT SERVICES) — набор процедур операционной системы, осуществляющих обработку файлов и записей внутри файлов. СУД дает программам возможность работать на логическом уровне — уровне записей, а также выполнять операции на физическом уровне, т. е. осуществлять операции ввода-вывода на уровне блоков. Процедуры СУД выполняются в режиме управления.

Системная динамическая память (SYSTEM DYNAMIC MEMORY) — память, резервируемая для операционной системы. Из нее по мере необходимости выделяются блоки для временного хранения информации. Если, например, образ выдает запрос ввода-вывода, то для хранения пакета этого запроса используется системная динамическая память. Каждый процесс имеет лимит на размер системной динамической памяти, которая может быть выделена ему сразу.

Системный буферизованный ввод-вывод (SYSTEM BUFFERED I/O) — операция ввода-вывода, в которой вместо буфера, определяемого процессом, используется промежуточный буфер из системного буферного пула.

Системный виртуальный адрес (SYSTEM VIRTUAL ADDRESS (SVA)) — виртуальный адрес, определяющий ячейку в пространстве системы.

Системный образ (SYSTEM IMAGE) — образ, который считывается в оперативную память с диска при запуске системы.

Слово (WORD) — два смежных байта (16 бит), начинающихся на границе адресуемого байта. Биты нумеруются справа налево от 0 до 15. Слово определяется адресом байта, содержащего бит 0.

Слово состояния процессора (PROCESSOR STATUS WORD (PSW)) — младшее слово длинного слова состояния процессора, содержащее инфор-

мацию о текущем состоянии процессора для исполняемой программы: коды условий, разрешения арифметических прерываний и трассировки.

Слоты для заголовков процессов (PROCESS HEADER SLOTS) — часть адресного пространства системы, в котором хранятся заголовки процессов, входящих в балансный набор. Число слот в системе определяет число процессов, которые могут одновременно входить в балансный набор.

Событие (EVENT) — изменение состояния процесса или указание о некотором действии, касающегося отдельного процесса или взаимодействующих процессов; об этом сообщается планировщику, который изменяет порядок выполнения процессов. События могут быть синхронными по отношению к выполняющемуся процессу (запрос ожидания) или асинхронными (завершение ввода-вывода). Обмен рабочих наборов, запрос на пробуждение и страничный отказ являются событиями.

Сообщение прерывания (INTERRUPT MESSAGE) — сформированное пользователем сообщение, которое посылается помимо обычного обмена сообщениями данных при связи между задачами. Такое применение термина «прерывание» противоположно обычному, означающему программный или аппаратный механизм прерывания.

Спецификатор базового операнда (BASE OPERAND SPECIFIER) — регистр, используемый для вычисления базового адреса операнда таблицы или массива при адресации индексного типа.

Спецификация узла (NODE SPECIFICATION) — первое поле в спецификации файла, которое определяет местоположение вычислительной системы в сети.

Спецификация файла (FILE SPECIFICATION) — уникальное имя файла. Оно определяет узел сети ЭВМ, на котором располагаются файл, устройство, имя каталога, имя файла, тип файла и номер его версии.

Список прав по идентификаторам (RIGHTS LIST) — список процесса, включающий все идентификаторы, которыми он владеет.

Список управления доступом (ACCESS CONTROL LIST (ACL)) — определяет вид доступа к некоторому объекту. Список управления доступом можно создать для файлов, устройств и почтовых ящиков. Каждый список управления доступом состоит из одного или нескольких элементов, называемых элементами списка управления доступом.

Спул (SPOOLING) — метод использования дискового ВЗУ для буферизации данных, передаваемых между высокоскоростной памятью и низкоскоростными устройствами ввода-вывода.

«Спячка» (HIBERNATION) — состояние, в котором процесс неактивен, но известен системе с текущим состоянием. «Спящий» процесс становится активным вновь при выдаче запроса на его пробуждение. Процесс может планировать запрос на пробуждение перед переходом в «спячку», или же запрос на его пробуждение может выдать другой процесс. Кроме того, «спящий» процесс может стать активным на время, достаточное для обслуживания любого AST, которое он, возможно, получит, пока «спит».

Средства межпроцессных связей (INTERPROCESS COMMUNICATION FACILITY) — кластер общих флагов событий, почтовый ящик или глобальная секция, используемые для передачи информации между двумя или несколькими процессами.

Срочное прерывание (URGENT INTERRUPT) — прерывание, получаемое на уровнях приоритетов прерываний от 24 до 31. Такие прерывания могут генерироваться процессором только для интервальных часов, серьезных ошибок и сбоя питания.

Стек вызова (CALL STACK) — стековая структура, используемая при вызове процедуры. Каждый режим доступа имеет по одному стеку вызова для каждого контекста процесса, и один стек вызова в контексте обработки прерываний.

Стек прерывания (INTERRUPT STACK (IS)) — общесистемный стек, используемый при обслуживании прерываний. Процессор всегда находится или в контексте процесса, работая в одном из режимов — пользователя, управления, супервизора или ядра, или же в общесистемном контексте обслуживания прерываний, работая в режиме ядра. Это указывается битами стека прерываний и текущего режима в длинном слове состояния процессора. Контекст стека прерываний не переключается.

Страница (PAGE) — условная единица деления непрерывного пространства памяти на поля фиксированной длины (512 байт). Используется системой управления памятью как единица отображения и защиты.

Страничный отказ (PAGE FAULT) — исключительная ситуация, возникающая при обращении к странице, которая не входит в рабочий набор процесса.

Счетчик адресов программы (CURRENT LOCATION COUNTER) — обозначается символом «точка» (.), всегда имеет значение, равное адресу текущего байта. Транслятор устанавливает это значение равным нулю в начале трансляции исходной программы и в начале трансляции каждой новой программной секции. Значение счетчика адресов программы можно явно установить с помощью оператора прямого присваивания.

Счетчик инструкций (PROGRAM COUNTER (PC)) — регистр общего назначения 15 (R15). В начале выполнения инструкций PC обычно содержит адрес ячейки памяти, из которой процессор будет выбирать следующую выполняемую инструкцию.

Таблица глобальных символических имен (GLOBAL SYMBOL TABLE (GST)) — таблица символических имен, определенных в директивах . GLOBAL или . EXTERNAL, а также с помощью оператора двойного присваивания («==»). Компоновщик добавляет таблицу глобальных символических имен к выполняемым образам, которые предназначены для работы с символическим отладчиком, или добавляет ее ко всем разделяемым образам.

Таблица глобальных страниц (GLOBAL PAGE TABLE) — таблица страниц, содержащая элементы основной таблицы страниц для глобальных секций.

Таблица логических имен (LOGICAL NAME TABLE) — таблица, в которой содержатся определенные пользователем логические имена и соответствующие им эквивалентные имена или части спецификации файла

Таблица страниц системы (SYSTEM PAGE TABLE (SPT)) — структура данных, которая отображает виртуальные адреса области системы, включая адреса, используемые для обращения к таблицам страниц процессов. Таблица страниц системы содержит по одному элементу таблицы страниц (PTE) — для каждой страницы виртуальной памяти области системы. Физический базовый адрес таблицы SPT содержится в базовом регистре системы (SBR).

Таблицы страниц процесса (PROCESS PAGE TABLES) — таблицы страниц, используемые для описания виртуальной памяти процесса.

Текущий режим доступа (CURRENT ACCESS MODE) — режим доступа к процессору для выполняющегося в данной момент процесса. Зафиксирован в поле текущего режима в длинном слове состояния процессора.

Терм (TERM) — компонента выражения (например, число, символ, выражение и т. д.).

Тип доступа (ACCESS TYPE) — способ обращения процессора к операндам инструкции. Существуют следующие типы доступа: на чтение, на запись, на модификацию и переход; способ обращения процедуры к своим аргументам.

Тип доступа на модификацию (MODIFY ACCESS TYPE) — тип доступа, при котором во время выполнения инструкции или процедуры специфицированный операнд читается, возможно, модифицируется и записывается.

Тип файла (FILE TYPE) — *См.* **Расширение имени файла для конкретного процесса, конкретной группы или системы.**

Точка входа (ENTRY POINT) — ячейка, которую можно указать в качестве объекта вызова. Она может содержать маску входа и биты разрешения исключительных ситуаций, называемые маской точки входа.

Указатель аргумента (ARGUMENT POINTER (AP)) — регистр общего назначения 12 (R12). По соглашению, AP содержит адрес базы списка аргументов для процедур, инициированных с использованием инструкции CALL.

Указатель кадра (FRAME POINTER (FP)) — регистр общего назначения 13 (R13). По соглашению он содержит базовый адрес самого последнего кадра вызова в стеке.

Указатель стека (STACK POINTER (SP)) — регистр общего назначения 14 (R14 или SP). Указатель стека содержит адрес «вершины» (наименьший адрес) определенного процессором стека. При обращении к SP выбирается один из пяти возможных указателей стека — ядра, управления, супервизора, пользователя или прерываний, в зависимости от значения битов текущего режима и стека прерываний в длинном слове состояния процессора.

Указатель стека прерываний (INTERRUPT STACK POINTER (ISP)) — указатель общесистемного стека прерываний.

Унарная операция (UNARY OPERATION) — одноместная операция, т. е. операция, выполняемая над одним операндом (например, операция управления основанием системы счисления).

Упакованная десятичная строка (PACKED DECIMAL STRING) — непрерывная последовательность байтов (не более 16), в каждом из которых хранятся две десятичные цифры в коде КОИ-8. Каждая цифра занимает 1 полубайт (4 разряда). Знак числа, если он имеется, занимает младший полубайт последнего байта.

Управление доступом (ACCESS CONTROL) — проверка запросов на их корректность: контроль входа в систему или доступа к файлам. Имя пользователя и пароль — наиболее общие средства управления доступом.

Управление памятью (MEMORY MANAGEMENT) — функции вычислительной системы, обеспечивающие отображение виртуальных страниц на физические и их защиту (аппаратные компоненты), активатор образов и программа обмена страниц (компоненты операционной системы).

Управляющая программа совместимости (APPLICATION MIGRATION EXECUTE (AME)) — программа, позволяющая задачам, подготовленным в ОСРВ, выполняться под управлением МОСВП в режиме совместимости.

Управляющий ключ (CONTROL KEY) — символ клавиатуры, который вызывает управляющее действие. Управляющий ключ — это обычно комбинация символа УС и некоторого алфавитного символа (например, УС/Y).

Уровень асинхронного системного прерывания (ASYNCHRONOUS SYSTEM TRAP LEVEL (ASTLVL)) — значение, хранящееся во внутреннем регистре процессора (ASTLVL), которое определяет режим доступа, до которого задерживается выполнение программы AST. Прерывание не произойдет до тех пор, пока текущий режим доступа не понизится (увеличится в числовом значении) до значения большего или равного ASTLVL. Таким образом, AST-прерывание некоторого режима доступа не будет обслужено, пока процессор работает в более привилегированном режиме.

Уровень приоритета прерываний (INTERRUPT PRIORITY LEVEL (IPL)) — уровень прерывания, на котором генерируется аппаратное или программное прерывание. Существует 31 уровень возможных приоритетов прерываний: IPL0 является самым низким (на нем работает пустой процесс), IPL31 — самым высоким. Эти уровни разрешают конфликтные ситуации при поступлении запросов на обслуживание процессором.

Устройство загрузки (LOAD DEVICE) — устройство, на котором располагается дистрибутив при установке программного обеспечения.

Утилита (UTILITY) — программа, которая реализует набор связанных общецелевых функций. Таковы, например, обслуживающие программы для разработки программ (редактор, компоновщик), обслуживающие программы управления файлами (копирования файлов, преобразования формата файлов) или обслуживающие программы управления операциями (программа дисковых квот, диагностики).

Учет (ACCOUNTING) — фиксация происходящих в системе событий в специальном файле (завершение работы образа, завершение процесса, неудачная попытка пользователя войти в систему и др.). Впоследствии такие записи можно распечатать и получить информацию об имевших место событиях.

Учетная запись (ACCOUNT) — запись в файле авторизации пользователя (UAF), которую создает администратор системы для каждого пользователя системы. В эту запись включаются учетное имя и пароль пользователя, предоставляемые ему привилегии, имя каталога пользователя, его код UIC и другие данные, необходимые системе о данном пользователе.

Файл авторизации пользователя (USER AUTHORIZATION FILE) — файл, содержащий записи о каждом пользователе. Используется для получения доступа к системе. Каждая запись определяет учетное имя пользователя, пароль, код идентификации пользователя (UIC), квоты, лимиты и привилегии, приписанные отдельным лицам, использующим систему.

Файловая структура (FILE STRUCTURE) — способ размещения каталогов, подкаталогов, файлов и другой служебной информации на внешней памяти. Определяет физический метод доступа в соответствии с требуемой операцией поиска или обработки данных.

Фактический аргумент (ACTUAL ARGUMENT) — терм (или выражение), указываемый в вызове макрокоманды.

Физический адрес (PHYSICAL ADDRESS) — адрес, используемый аппаратурой для определения ячейки в оперативной памяти или на устройствах непосредственно адресуемой внешней памяти. Адрес оперативной памяти состоит из номера физической страницы и номера байта в странице. Физический адрес блока на диске состоит из номера цилиндра, дорожки и номера сектора.

Флаг события (EVENT FLAG) — бит в кластере флагов событий, который может быть установлен в 0 или 1 для указания того, что произошло связанное с ним событие. Флаги событий используются для синхронизации действий процесса или нескольких процессов.

Форк-блок (FORK BLOCK) — часть блока управления устройством, которая во время ожидания драйвером некоторого ресурса содержит контекст драйвера. Форк-блок драйвера, ожидающего ресурс процессора, помещается в форк-очередь.

Форк-диспетчер (FORK DISPATCHER) — программа МОСВП, обслуживающая прерывания. Она активизируется программным прерыванием на форк-уровне приоритета прерываний. Активизированный форк-диспетчер выбирает форк-процессы драйверов из форк-очереди до тех пор, пока не будет обслужена вся очередь для соответствующего уровня приоритета.

Форк-очередь (FORK QUEUE) — очередь форк-блоков драйверов, которая ждет своей активизации форк-диспетчером МОСВП на отдельном уровне приоритета прерываний.

Форк-процесс (FORK PROCESS) — процесс с минимальным контекстом, который выполняется при следующих ограничениях: он работает на повышенном уровне приоритета прерываний (IPL) и в виртуальном адресном пространстве системы; использует лишь регистры R0—R5 (другие регистры должны быть сохранены и восстановлены); ему разрешено обращаться только к статической памяти и модифицировать ее (эта память никогда не модифицируется программами более высокого уровня приоритета прерываний). МОСВП

использует программные прерывания и форк-процессы для синхронизации действий управляющей программы.

Форк-уровень драйвера (DRIVER FORK LEVEL) — уровень приоритетов прерываний, на которых выполняется форк-процесс драйвера (значение от 8 до 11). Каждый блок управления устройством (UCB) содержит форк-уровень драйвера этого устройства.

Формальный аргумент (FORMAL ARGUMENT) — терм (или выражение), указываемый в макроопределении. При макрорасширении заменяется на фактический аргумент.

Формат исходного предложения определяет правила интерпретации исходного предложения, которое состоит не более чем из четырех полей:

<поле метки>:<поле операции><поле операнда>;<поле комментария>

Число с плавающей запятой формата *F* (F—FLOATING POINT DATA) — число с плавающей запятой одинарной точности, длиной 4 байта, имеющее диапазон от $\pm 2.9 * 10 * * (-37)$ до $\pm 1.7 * 10 * * 38$ и точность примерно 7 десятичных цифр.

Число с плавающей запятой формата *D* (D FLOATING POINT DATA) — число с плавающей запятой двойной точности, длиной в 8 байт, имеющее диапазон от $\pm 2.9 * 10 * * 37$ до $\pm 1.7 * 10 * * 38$ и точность примерно 16 десятичных цифр.

Число с плавающей запятой формата *G* (G FLOATING POINT DATA) — число с плавающей запятой расширенного диапазона, длиной 8 байт, имеющее диапазон от $\pm 0.56 * 10 * * (-308)$ до $\pm 9 * 10 * * 308$ и точность примерно 15 десятичных цифр.

Число с плавающей запятой формата *H* (H—FLOATING POINT DATA) — число с плавающей запятой расширенного диапазона, состоящее из 16 байт, имеющее диапазон от $\pm 0.84 * 10 * * 4932$ до $\pm 0.59 * 10 * * 4932$ и точностью примерно 38 десятичных цифры.

Числовая строка (NUMERIC STRING) — непрерывная последовательность байтов, представляющая до 31 десятичной цифры (по одной в байте) и, возможно, знак. Числовая строка определяется двумя атрибутами: адресом первого байта строки и длиной строки, равной количеству символов в строке.

Чужой том (FOREIGN VOLUME) — любой том, отличный по файловой структуре от тома с файловой структурой системы МОСВП.

Экстент (EXTENT) — непрерывная область на диске, состоящая из нескольких смежных блоков и относящаяся к одному файлу.

Элемент таблицы страниц (PAGE TABLE ENTRY (PTE)) — структура данных, определяющая физическое размещение и состояние страницы виртуального адресного пространства. Если виртуальная страница находится в оперативной памяти, то PTE содержит номер физической страницы, необходимый для отображения виртуальной страницы в физическую. Если же страница не находится в оперативной памяти, то элемент таблицы страниц содержит информацию, необходимую для ее поиска во внешней памяти (на диске).

Язык Ассемблера (ASSEMBLER LANGUAGE) — машинно-ориентированный язык программирования.

Язык описания файлов (FILE DEFINITION LANGUAGE (FDL)) — специальный язык, используемый для написания спецификаций файлов данных, которые употребляются обслуживающими и библиотечными программами СУД для создания реальных файлов данных.

ОГЛАВЛЕНИЕ

Справочное издание

Остапенко Георгий Павлович
Толмачева Наталья Александровна
Горский Владимир Евгеньевич

Макроассемблер для СМ1700

Зав. редакцией *И. Г. Дмитриева*
Редактор *Л. А. Табакова*
Мл. редактор *А. Ю. Поляк*
Худож. редактор *С. Л. Витте*
Техн. редактор *Л. Г. Челышева*
Корректоры *Г. В. Хлопцева, М. А. Синяговская*
Обложка художника *Ф. Г. Миллера*

ИБ № 2545

Сдано в набор 24.06.89. Подписано в печать 14.09.89. А02023. Формат $60\times88^{1}/_{16}$. Бум. кн.-журн. офсетная. Гарнитура «Литературная». Печать офсетная. Усл. п. л. 14,7. Усл. кр.-отт. 14,7. Уч.-изд. л. 17,72. Тираж 25 000 экз. Заказ 540. Цена 1 р. 20 к.

Издательство «Финансы и статистика», 101000, Москва, ул. Чернышевского, 7.

Типография им. Котлякова издательства «Финансы и статистика» Государственного комитета СССР по печати. 195273, Ленинград, ул. Руставели, 13.

1р.20к.