Утвержден

26.00152-01 35 01-ЛУ

Операционная система МОС ВП

СИСТЕМА ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ МАКРОАССЕМБЛЕР

Описание языка

26.00152-01 35 01

Листов 302/4К

1987

Перв.примен.

26.00152-01

Литера

Операционная система МОС ВП

СИСТЕМА ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ МАКРОАССЕМБЛЕР

Описание языка

00152-01 35 01

Листов 298

1987

АННОТАЦИЯ

Данный документ предназначен для программистов, разрабатывающих программы на языке макроассемблер.

В документе дано описание языка макроассемблер. Для каждого элемента языка приведены формат и описание выполняемой функции. Документ состоит из шести разделов и трех приложений.

В разделе 1 приведены общие сведения о языке макроассемблер СМ 1700.

В разделе 2 описан формат исходных предложений.

В разделе 3 описаны элементы языка: набор знаков, числа, символы, локальные метки, термы и выражения, унарные и бинарные операции, операторы прямого присваивания, а также текущий счетчик адресов программы.

В разделе 4 содержится информация о режимах адресации.

В разделе 5 описаны общие директивы макроассемблера.

В разделе 6 содержится информация о макрокомандах и директивах макрокоманд.

В приложении 1 описаны коды инструкций.

В приложении 2 приводятся краткие сведения по языку макроассемблер.

Правила преобразования чисел из десятичной системы счисления в шестнадцатеричную и наоборот описаны в приложении 3.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Язык макроассемблер предназначен для разработки программ в операционной системе МОС ВП на ЭВМ СМ 1700.

Исходные программы, составленные на языке программирования макроассемблер, транслируются в об'ектный код, что выполняется программой макроассемблер, символическое имя которой MACRO. Программа MACRO создает об'ектный модуль и, если указано, файл листинга. Функции, выполняемые этой программой, описаны в [1]. В этом разделе описываются средства языка макроассемблер. Исходные программы, составленные на языке макроассемблер, состоят из последовательности исходных предложений (симаольных строк). Эти исходные предложения могут быть следующие:

- основные инструкции ЭВМ СМ 1700;
- операторы прямого присваивания;
- директивы макроассемблера.

С помощью инструкций производится обработка данных. Они обеспечивают такие функции, как суммирование, преобразование данных, а также передачу управления. В исходном предложении инструкции обычно сопровождаются операндами, в качестве которых могут выступать данные любого типа, необходимые для выполнения данной инструкции. Система инструкций ЭВМ СМ 1700 приведена в приложении 1, а в [2]. Она подробно описана.

С помощью операторов прямого присваивания символам присваиваются значения.

Директивы языка макроассемблер осуществляют управление процессом трансляции и обеспечивают средства для использования инструкций. Разделы 5 и 6 содержат подробное описание директив языка макроассемблер, а приложение 2 имеет краткие сведения по директивам. Существует два класса директив языка макроассемблер: общие директивы и директивы макрокоманд.

Общие директивы позволяют выполнять следующие действия:

- хранить данные или резервировать память для хранения данных;
- управлять размещением частей программы в памяти;
- определять способы доступа к секциям памяти, в которых будет храниться программа;
- определять точку входа в программу или в часть программы;
- определять способ, посредством которого будет производиться ссылка к символам;
- определять, какая часть программы должна транслироваться лишь при определенных условиях;
- управлять форматом и содержанием файла листинга;
- посылать информационные сообщения;
- управлять дополнительными возможностями, которые применяются для интерпретации исходной программы;
- определять новые коды инструкций.

Директивы макрокоманд используются для определения макрокоманд и блоков повторений. Они позволяют программисту выполнять следующие действия:

- повторять одинаковые или похожие последовательности исходных предложений, встречающиеся в программе;
- использовать строчные операции для обработки и проверки содержания исходных предложений.

Использование макрокоманд и блоков повторений помогает минимизировать количество ошибок, возникающих по вине программиста и ускорить процесс отладки программы.

2. ФОРМАТ ИСХОДНОГО ПРЕДЛОЖЕНИЯ ЯЗЫКА МАКРОАССЕМБЛЕР

Исходная программа состоит из последовательности исходных предложений, каждое из которых, одно за другим, программа макроассемблер интерпретирует и обрабатывает, формируя при этом об'ектный код или выполняя специальные действия этапа трансляции. Исходное предложение может занимать одну исходную строку, либо может быть расширено на несколько исходных строк. Каждая из исходных строк может содержать до 132 символов, однако не следует превышать размер строки за 80 символов, чтобы гарантировать размещение одной исходной строки (вместе с ее двоичным расширением) на одной строке в файле листинга.

Предложение языка макроассемблер может включать в себя до четырех полей:

1) поле метки, содержащее символическое определение адреса в программе;

2) поле операции, содержащее спецификацию действия, которое должно быть выполнено при данном предложении; в этом поле может располагаться обозначение инструкции, директивы или макровызова;

3) поле операнда, содержащее операнд (операнды) инструкции либо аргумент (аргументы) директивы, либо аргумент (аргументы) макрокоманды;

4) поле комментария, содержащее комментарий, поясняющий смысл данного предложения; это поле не оказывает влияния на выполнение программы.

Поле метки и поле комментария являются необязательными. Поле метки заканчивается двоеточием (:), а поле комментария начинается с точки с запятой (;). Поле операнда должно соответствовать формату инструкции, директивы или макрокоманды, которые определены в поле операции.

Хотя поля предложения могут разделяться либо пробелом, либо знаком табуляции, для лучшей наглядности рекомендуется форматировать предложение знаком табуляции. Знаки табуляции используются как показано в табл. 1.

Таблица 1

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

Пример 1.

```
        .TITLE  PROG1
        .ENTRY  BEGIN,^M<R1,R2>     Начало программы PROG1
         CLRL   R0
M1:      INCL   R0
M2:      ADDL3  (R3)+,R0,SYM         Получить значение SYM
         BLEQ   M1                  ; Переход на M1, если
                                     Результат меньше или равен 0
```

```
        BRW     CONT                    Безусловный переход на
                                        CONT
```

Одно предложение может быть продолжено на несколько строк. При этом в качестве последнего значащего символа перед полем комментария или в конце строки (если комментарий отсутствует) следует употребить дефис (-).

Пример 2.

```
SAVE:   MOVAL   VECTOR_ADDRESS_2,- ; Сохранить адрес
                                   ; загрузки
                VECTOR_2(R1)
```

Транслятор с языка макроассемблера обрабатывает приведенное предложение как эквивалентное следующему предложению:

```
SAVE: MOVAL VECTOR_ADDRESS_2,VECTOR_2(R1)   Сохранить ад-
                                            рес загрузки
```

Предложение может быть продолжено с любой точки. Однако символические имена, определяемые пользователем, и постоянные символы не могут быть продолжены на следующей строке. Если на другую строку переносится символическое имя и первым знаком на следующей строке является знак горизонтальной табуляции или пробел, то это символическое имя будет ограничено этим знаком.

Заметим, что если предложение появляется в макроопределении, то это предложение не может содержать более 1000 знаков.

Пустые строки, хотя и допустимы, не имеют в исходной программе какого-либо смысла, кроме как ограничителей пере-

несенной строки.

В подразделах 2.1 - 2.4 каждое из полей предложения описывается подробно.

2.1. Поле метки

Метка представляет собой определенный пользователем символ, идентифицирующий адрес в программе. Этому символу присваивается значение, равное текущему значению счетчика адресов для данного адреса программной секции, в которой эта метка появляется (за сведениями о программных секциях обращайтесь к [1]). Символическое имя, определяемое пользователем, может включать в себя до 31 графического символа и содержать любой алфавитно-цифровой знак и знак подчеркивания (_), а также знак денежной единицы (¤) и точку (.). Более подробно о правилах формирования символических имен, определяемых пользователем, описывается в подразделе 3.3.

Если предложение языка макроассемблер включает метку, то эта метка должна быть расположена в первом поле строки.

Метка ограничивается двоеточием (:) или двойным двоеточием (::). Двоеточие означает, что данная метка определена только для текущего модуля (внутренний символ). Двойное двоеточие означает, что данная метка является глобально определенной, то есть к данной метке может быть произведена ссылка из других об'ектных модулей.

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

выполняется определение этой метки, и, когда метка переопределяется.

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

Пример.

```
M1:    MOVAL     TAB_ARG, R0    Получить в R0 адрес TAB_ARG
COMPARE_ARG:
       CMPL      (R0), VALUE ;  Сравнить значения аргумента
                                из таблицы TAB_ARG со значе-
                                нием VALUE
       BGTR      M3             Если больше - переход на м3
M2::   MOVL      (R0), MIN      Получить в MIN значение ар-
                                гумента
       BRW       TEST1          Безусловный переход на TEST1
M3::   MOVL      (R0), MAX      Получить в MAX значение ар-
                                гумента
       BRW       TEST2          Безусловный переход на TEST2
```

В операторе прямого присваивания поле метки также используется для символа.

2.2. Поле операции

В поле операции определяется само действие, которое должно быть выполнено в данном предложении. В этом поле может содержать обозначение инструкции, директивы или макровызова.

Если операцией является инструкция, программа макроассемблер генерирует в об'ектном модуле двоичный код данной инструкции. Коды инструкций приведены в приложении 1, а система инструкций описана в [2]. Если операцией является директива, программа макроассемблер выполняет определенные управляющие действия или выполняет операции во время трансляции исходной программы. Директивы языка макроассемблер описаны в разделах 5 и 6. Если операцией является макровызов, программа макроассемблер производит расширение макрокоманды.

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

2.3. Поле операнда

Поле операнда может содержать операнды инструкций или аргументы директив или макровызовов.

Операнды инструкций идентифицируют адреса ячеек памяти или регистры, которые используются в машинной операции. Эти операнды определяют режим адресации для данной инструкции. Поле операнда для специальной инструкции должно содержать значение количества операндов, требуемых для данной инструкции. Описание инструкций и их операндов содержится в документе [2].

Аргументы директивы должны отвечать требованиям формата данной директивы. Аргументы макрокоманд должны отвечать требованиям, определенным в макроопределении. Описание

директивы .MACRO (см. раздел 6).

Если указываются два или более операндов, то их следует разделить запятыми. Язык макроассемблер допускает использование пробела и знака горизонтальной табуляции в качестве разделителей аргументов любой директивы, в которой не появляются выражения. Однако для разделения операндов инструкций и для директив, в которых в качестве аргументов появляются выражения, требуется запятая.

Поле операнда ограничивает точка с запятой. С этого знака начинается поле комментария. Если в строке отсутствует комментарий, поле операнда ограничивается концом этой строки.

2.4. Поле комментария

Поле комментария содержит текст, поясняющий функцию данного предложения. Каждая строка программы может иметь комментарий. Комментарий не оказывает воздействия на процесс трансляции или на выполнение программы, за исключением случаев выдачи сообщений во время трансляции директивами .ERROR, .PRINT, .WARN.

Полю комментария должен предшествовать знак точка с запятой (;), а ограничивается оно концом строки. Поле комментария может содержать любой печатный символ кода КОИ-8.

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 целой строке может располагаться лишь один коммен-

тарий.

Правильно составленный текст комментария передает смысл, а не действие данного предложения. Например, инструкция MOVAL BUF_PTR_I,R7 должна сопровождаться таким комментарием, как "установить указатель на первый буфер", а не таким, как "переслать адрес BUF_PTR_I в R7".

Пример.

```
        MOVAB   DATA, R3    ; Получить в R3 адрес DATA
        CMPL    (R3), R0      Сравнить значения DATA с R0
        BEQL    LABEL         Если равны - пропустить
        CLRL    (R3)          Очистить DATA
LABEL:  MOVL    (SP), R2      Скопировать верхний элемент
                              стека в R2
```

3. ЭЛЕМЕНТЫ ЯЗЫКА МАКРОАССЕМБЛЕР

Элементами языка макроассемблер являются составные части исходного предложения. В настоящем разделе описываются составные части исходного предложения языка макроассемблер. К этим составным частям относятся следующие элементы:

- набор графических символов;
- символы;
- локальные метки;
- термы и выражения;
- унарные и бинарные операции;
- операторы прямого присваивания;
- текущий счетчик адресов программы.

3.1. Набор графических символов

В исходных предложениях языка макроассемблер употребляются следующие графические символы:

- латинские буквы;
- буквы кириллицы;
- цифры от 0 до 9;
- знаки, перечисленные в табл. 2.

Таблица 2

Знаки, используемые в языке макроассемблер

Знак	Наименование знака	Назначение знака
_	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
	Знак горизонтальной табуляции	Ограничитель поля предложения
	Пробел	Ограничитель поля предложения
#	Знак номера	Индикатор режима непосредственной адресации
@	Коммерческое "эт"	Индикатор режима косвенной адресации и операция арифметического сдвига
,	Запятая	Разделитель полей, операндов

Продолжение табл. 2

Знак	Наименование знака	Назначение знака
	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 строк
*	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 числового преобразования в аргументах макрокоманд
^	Стрелка вверх	Ограничитель унарных операций и аргументов макрокоманд
[ ]	Квадратные	Индикаторы режима адресации с

Продолжение табл. 2

Знак	Наименование знака	Назначение знака
	скобки	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

В табл. 3 определяются разделительные знаки, применяемые в предложениях языка макроассемблера.

Таблица 3

Разделительные знаки, используемые в предложениях языка макроассемблер

Знак	Наименование знака	назначение знака
	Пробел или знак горизонтальной табуляции	Разделитель полей предложения. Пробелы в составе выражения игнорируются
,	Запятая	Разделитель между символическими аргументами в поле операнда. Множественные выражения в поле операнда должны быть разделены запятыми

3.2. Числа

В языке макроассемблер числа могут быть представлены как целые, с плавающей запятой и десятичные в упакованном формате.

3.2.1. Целые числа

Целые числа могут употребляться в любом выражении, включая выражения в операндах и в операторах прямого присваивания.

Формат

SNN,

где S - необязательный знак, знак плюс (+) для положительных чисел (умолчание) или знак минус (-) для отрицательных;

NN - последовательность цифр, допустимых при данном текущем основании.

Все встречающиеся в исходной программе целые числа интерпретируются программой макроассемблер как десятичные, если перед числом не стоит операция управления основанием системы счисления.

Целые числа должны находиться в диапазоне значений от -2 147 483 648 до 2 147 483 647 для чисел со знаком или в диапазоне от 0 до 4 294 967 295 для чисел без знака.

Отрицательным числам должен предшествовать знак минус, макроассемблер транслирует эти числа в дополнительный код (дополнение до двух). Для положительных чисел знак плюс является необязательным.

3.2.2. Числа с плавающей запятой

Числа с плавающей запятой 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 унарного плюса, унарного минуса и

унарной операции формата с плавающей запятой (^F).

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

Форматы

числа с плавающей запятой без показателя степени:

SNN

SNN.NN

SNN.

Числа с плавающей запятой с показателем степени:

SNNESNN

SNN.NNESNN

SNN.ESNN

где S - необязательный знак;

NN - последовательность десятичных цифр в диапазоне от 0 до 9.

Десятичная точка может появляться в любом месте справа от первой цифры. Отметим, что число с плавающей запятой не может начинаться с десятичной точки, поскольку в таком случае программа макроассемблер будет обрабатывать это число как символ, определенный пользователем.

Числа с плавающей запятой могут быть одинарной точности (32 разряда), двойной точности (64 разряда) и расширенной точности (128 разрядов). Степень точности составляет 7 цифр для чисел одинарной точности, 16 цифр для чисел двойной точности и 33 цифры для чисел расширенной точности.

Величина неотрицательного числа с плавающей запятой не может быть меньше, чем приблизительно 0.29e-38 и больше,

чем приблизительно 1.7e38.

Числа с плавающей запятой одинарной точности могут либо округляться (по умолчанию), либо усекаться. Должны ли числа с плавающей запятой округляться или усекаться определяется директивами .ENABLE и .DISABLE. Числа с плавающей запятой двойной точности и расширенной точности всегда округляются.

Внутренний формат представления чисел с плавающей запятой описывается в документе [2].

3.2.3. Десятичные числа в упакованном формате

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

Формат

SNN

где S - необязательный знак;

NN - последовательность, состоящая минимум из 1, максимум из 31 десятичных цифр в диапазоне от 0 до 9.

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

Более подробно о десятичных числах в упакованном формате описывается в документе [2].

3.3. Символы

В исходных программах, написанных на языке макроассемблер, допустимы символы следующих трех типов:

1) постоянные символы;

2) символы, определяемые пользователем;

3) имена макрокоманд.

3.3.1. Постоянные символы

К постоянным символам относятся мнемонические обозначения инструкций, обозначения директив языка макроассемблер и имена регистров. Мнемонические обозначения не нуждаются в предварительном определении того, как они будут определяться в поле операции исходного предложения языка макроассемблер. Не требуют предварительного определения и имена регистров при употреблении их для указания режимов адресации. Имена регистров не могут быть переопределены, то есть ни один из символов, определяемых пользователем, не может совпадать ни с одним из имен регистров, перечисленных в табл. 4. 16 регистров общего назначения, принадлежащих процессору могут быть определены в исходной программе как показано в табл. 4.

Таблица 4

Имя регистра	Назначение регистра
R0	Регистр общего назначения 0
R1	Регистр общего назначения 1
R2	Регистр общего назначения 2
.	
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
FR	Указатель кадра
SP	Указатель стека
PC	Счетчик инструкций

3.3.2. Символы, определяемые пользователем, и имена макрокоманд

Символы, определяемые пользователем, могут употребляться в качестве меток, или же с помощью оператора прямого присваивания им может быть присвоено определенное значение.

символы, определяемые пользователем, могут также употребляться в любом выражении.

Следующие правила действуют при создании символов, определяемых пользователем:

1) символы, определяемые пользователем, могут включать в себя алфавитно-цифровые знаки, знак подчеркивания (_), знак денежной единицы (¤) и точку (.). Любой другой знак является ограничением символа;

2) первым знаком символа не должна быть цифра;

3) символ должен быть уникальным и может включать в себя не более 31 знака;

4) знак денежной единицы (¤) резервируется для имен, определяемых разработчиком операционной системы МОС ВП. Это соглашение гарантирует, что определенное пользователем имя (в состав которого не входит знак денежной единицы) не войдет в конфликт с именем, определенным разработчиком (в состав которого входит знак денежной единицы);

5) точка (.) не должна употребляться ни в каких глобальных символических именах, поскольку в других языках, таких как фортран, не допускается употребление точки в символических именах.

Имена макрокоманд подчиняются тем же самым правилам и соглашениям, установленным для символов, определяемых пользователем. Символы, определяемые пользователем, и имена макрокоманд не вступают в конфликт, то есть одно и тоже имя может использоваться как символ, определяемый пользователем, и как макрокоманда. Однако, чтобы избежать путаницы,

символам, определяемым пользователем, и макрокомандам следует давать различные имена.

3.3.3. Определение значений символов

Конкретное значение символа зависит от того, как этот символ в программе используется. В языке макроассемблер применяются различные способы для определения значений символов в зависимости от того, находится ли этот символ в поле операции или в поле операнда.

Символ в поле операции может быть либо постоянным символом, либо именем макрокоманды. В языке макроассемблер поиск определения символа производится в следующем порядке:

1) среди заранее определяемых имен макрокоманд;

2) среди кодов операций, определенных пользователем;

3) среди постоянных символов (инструкций и директив);

4) в библиотеках макрокоманд.

Такой порядок поиска значения позволяет переопределять постоянный символ с помощью имени макрокоманды. Если символ, встреченный в поле операции, не является ни макрокомандой, ни постоянным символом, то выдается сообщение об ошибке.

Символ в поле операнда должен быть либо символом, определенным пользователем, либо именем регистра.

Символы, определяемые пользователем, могут быть как глобальными (внешними) символами, так и локальными (внутренними). Является ли символ локальным или глобальным зависит от его использования в исходной программе.

К локальному символу могут производиться ссылки только из того же модуля, в котором он определен. Если локальные символы с одинаковыми именами определены в различных модулях, то эти символы являются совершенно независимыми. К определению же глобального символа допустимы ссылки из любого модуля программы.

Обычно при определении символов, определяемых пользователем, программа макроассемблер обрабатывает их как все локальные. Однако с помощью одного из трех следующих способов определение символа может быть об'явлено глобальным:

1) путем использования двух двоеточий (::) при определении метки;

2) путем использования двух знаков равенства (==) в операторе прямого присваивания;

3) путем использования директив .GLOBAL, .ENTRY или .WEAK.

Если символ определен в данном модуле и из этого же модуля к нему производится ссылка, то программа макроассемблер рассматривает эту ссылку как внутреннюю. Если к символу производится ссылка из модуля, в котором данный символ не определен, то программа макроассемблер рассматривает такую ссылку как внешнюю (то есть символ определен в другом модуле). Для того, чтобы сделать недопустимыми ссылки к символам, не определенным в данном текущем модуле, может быть использована директива .DISABLE. В этом случае для определения того, что данная ссылка является внешней ссылкой, следует использовать директиву .EXTERNAL.

3.4. Локальные метки

Локальные метки используются для идентификации адресов в рамках блока локальных меток исходной программы.

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 символов 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 предложения - с левой стороны исходного текста. Хотя локальные метки могут появляться в программе в произвольном порядке, однако следует располагать их в каждом блоке локальных меток в порядке возрастания номеров.

Локальные метки удобно использовать в качестве адресов в инструкциях относительного перехода, если эти адреса указываются только в пределах данного блока локальных меток.

локальные метки можно использовать для различия адресов, ссылка к которым осуществляется в пределах небольшого блока локальных меток исходной программы и адресов, ссылка к которым осуществляется в пределах всего модуля. Недостатком локальных меток является то, что их числовые имена не позволяют показать их назначение в программе. Следовательно, локальные метки не следует использовать для пометки последовательностей предложений языка, не являющихся логически связанными. Вместо локальных меток следует использовать символы, определяемые пользователем.

Рекомендуется пользователям создавать локальные метки только в диапазоне от 1¤ до 29999¤, поскольку в диапазоне от 30000¤ до 65535¤ программа макроассемблер автоматически создает локальные метки, которые используются в макрокомандах.

Блок локальных меток, в котором действуют локальные метки, ограничивается следующими предложениями:

- метками, определяемыми пользователем;
- директивами .PSeCT;
- директивами .DISABLE и .ENABLE, с помощью которых блок локальных меток может быть расширен за пределы, обуславливаемые метками пользователя и директивами .PSECT.

Обычно блок локальных меток ограничивается двумя метками, определяемыми пользователем. Однако блок локальных меток начавшийся с директивы .ENABLE LOKAL_BLOCK завершается только следующими предложениями:

- второй директивой .ENABLE LOKAL_BLOCK;

- директивой .DISABLE LOKAL_BLOCk, за которой следует метка, определяемая пользователем, или директива .PSECT.

Хотя блоки локальных меток и могут быть расширены от одной программной секции до другой, рекомендуется, чтобы к локальным меткам одной программной секции не происходило ссылок из другой программной секции. Рекомендуется всегда для ограничения блоков локальных меток использовать символы, определяемые пользователем.

Локальные метки можно сохранить для будущих ссылок в тексте программной секции, в которой они были определены; описания директив .SAVE_PSECT [LOKAL_BLOCK] и .RESTORE_PSECT (см. раздел 5).

Приведенный 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
      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 локаль-
                                ную метку
40¤:    MOVW   R2, EXT          Определяется локальная метка
                              ; 40¤
        BRW    NEXT1            Безусловный переход на метку
                              ; определенную пользователем
        .ENABLE LOKAL_BLOCK     Начало блока локальных меток,
                                который не будет ограничен
                              ; меткой, определенной пользова-
                                телем
LAB1:   ADDL3  R0, R1, R3
        CMPL   R2, R3
        BRB    1¤
LAB2:   SUBL2  R2, R3           LAB2 не является началом оче-
                                редного блока локальных меток
1¤:     SUBL2  R2, R3           Определение локальной метки 1¤
        BGTR   2¤               Условный переход на локальную
                                метку
        INCL   R0
        INCL   R1
        BRB    NEXT2
2¤:     INCL   R1               Определение локальной метки
                                2¤
```

```
      DEKL  R0
      .DISABLE LOKAL_BLOCK; Завершение блока локальных
                           меток
NEXT2:.MOVL 0, R4
```

3.5. Термы и выражения

Термом может быть любой из следующих элементов:

- число;
- символ;
- текущий счетчик адресов программы;
- текстовая операция, за которой следует текст;
- любой из перечисленных элементов, перед которым стоит знак унарной операции.

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

Выражения представляют собой комбинации термов, об'единенных бинарными операциями (подраздел 3.7.); значения выражений вычисляются как значения длинных слов (четырехбайтные). Программа макроассемблер вычисляет 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 вычисления данного выражения в целом. Например, выражения a*B+c и a*<B+C> являются различными. Угловые скобки могут быть использованы также для распространения унарной операции на целое выражение, например, -<a+B>.

Следует заметить, что унарная операция считается составной частью терма. Следовательно, программа макроассемблер выполняет действие, указанное унарной операцией, до выполнения действия, указанного любой бинарной операцией.

Все выражения делятся на три категории:

1) относительные выражения;

2) абсолютные выражения;

3) внешние (глобальные) выражения.

Выражение является относительным, если его значение фиксировано относительно начала программной секции, в которой это выражение появилось. Текущее выражение сетчика адресов программы является относительным выражением в относительной программной секции.

Выражение является абсолютным, если его значение определяется во время трансляции и является константой. Абсолютным является выражение, все термы которого являются числами. 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.

Выражение является внешним, если в его состав входят один или несколько символов, неопределенных в текущем моду-

ле.

В большинстве макрокоманд могут употребляться выражения любого типа. Однако в некоторых случаях на выражения накладываются некоторые ограничения. Это такие случаи, когда выражения употребляются:

- в директиве выравнивания .ALIGN;
- в директивах распределения памяти .BLKX;
- в директивах блоков условной трансляции .IF и .IIF;
- в директиве блока повторений .REPEAT;
- в директиве определения кода инструкции .OPDEF;
- в директиве точки входа .ENTRY;
- в качестве коэффициентов повторений в директивах .BYTE, .LONG, .WORD, .SIGNED_BYTE, и SIGNeD_WORD;
- в операторах прямого присваивания.

Описания всех перечисленных директив, кроме директивы .REPEAT (см. раздел 5). Описание директивы .REPEAT (см. раздел 6). Выражения, которые употребляются совместно с этими директивами и в операторах прямого присваивания, могут включать в себя только такие символы, которые были предварительно определены в текущем модуле. Они не могут включать в себя ни внешние символы, ни символы, определяемые в данном текущем модуле позднее. Кроме того, выражения, употребляемые совместно с этими директивами, должны быть абсолютными. Выражения в операторах прямого присваивания могут быть относительными.

Рассмотрим пример, иллюстрирующий использование выражений.

Пример.

```
BETA = 5*60                 5*60 абсолютное выражение
       .BLKW  BETA+100      BETA+100 абсолютное выражение
M:     .BLKW  BETA        ; Относительное выражение
       ALPHA=M+BETA+200     ALPHA относительное выражение
N:     .BLKW  N-M           N-м абсолютное выражение
       .WORD  K-M           K-M абсолютное выражение
K:     .BLKW  DELTA+2       Внешние выражение, т.к.
                            Символ DELTA является внешним
```

3.6. Унарные операции

Унарные операции модифицируют терм или выражение и указывают на операцию, которая должна быть выполнена над этим термом или выражением. Выражение, модифицированное унарной операцией, должно быть заключено в угловые скобки. Унарные операции могут использоваться, чтобы показать является ли данный терм или выражение положительным или отрицательным (если унарный плюс или минус не указаны, по умолчанию значение является положительным). В табл. 5 собрана краткая информация по унарным операциям.

Таблица 5

Унарные операции

Знак	Наименование	Пример	Выполняемая операция
+	Знак плюс	+A	Дает положительное значение a
-	Знак 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	^XFGF9	Интерпретирует число FGF9 как шестнадцатеричное
^A	Знак операции КОИ-8	^A/ABC/	Порождает строку кодов КОИ-8; знаки, заключенные между

Продолжение табл. 5

Знак	Наименование	Пример	Выполняемая операция
			парными ограничителями, преобразуются в коды 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, допустимо использование более чем одной унарной операции. Например, - + - a. Такое предложение эквивалентно следующему: - < + < - a > >

3.6.1. Операции управления основанием системы счисления

Программа макроассемблер воспринимает термы и выражения, значения которых представлены в четырех различных основаниях системы счисления:

1) в двоичном основании;

2) в десятичном основании;

3) в восьмеричном основании;

4) в шестнадцатеричном основании.

Выражения, значения которых модифицируются операцией управления основанием системы счисления, должны быть заключены в угловые скобки.

Форматы

^BNN

^DNN

^ONN

^XNN

где NN - последовательность знаков, являющихся допустимыми для определяемого основания системы счисления.

В табл. 6 приведены знаки, допустимые для каждого из рассматриваемых оснований системы счисления.

Таблица 6

Формат	Основание	Допустимые знаки
^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 счисления могут быть включены в любом месте исходной программы, где допустимо числовое значение. Операция управления основанием счисления оказывает воздействие только на тот терм или выражение, которое следует непосредственно за ним. Результатом его влияния является то, что данный терм или выражение обрабатываются в указанном основании счисления.

Пример 1.

```
.BYTE   ^B11001             Двоичное основание
.WORD   ^D28                Десятичное основание
.WORD   ^O6666            ; Восьмеричное основание
.LONG   ^XFFF               Шестнадцатеричное основание
.LONG   ^X<F1C2+2C+10>    ; Все числа в выражении имеют
                            шестнадцатеричное основание
```

Стрелка вверх (^) не может быть отделена от символов B, D, O и X, которые за ней следуют, но в целом операция управления основанием счисления может быть отделена от терма или выражения, которые обрабатываются в указанном основании счисления, пробелами или знаками табуляции.

Операция десятичного основания счисления (^D), которая действует по умолчанию, становится необходимой только в выражении, перед которым стоит какая-то другая операция управления основанием счисления. В примере 2 число 66 интерпретировалось бы как восьмеричное число, если бы ему не предшествовала операция ^D.

Пример 2.

```
.WORD ^O<511 + 16 + ^D66>
```

Правила шестнадцатерично-десятичного преобразования чисел описываются в приложении 3.

3.6.2. Текстовые операции

Текстовыми операциями являются операция КОИ-8 (^A) и операция регистровой маски (^M).

3.6.2.1. Операция ^A

Операция ^A осуществляет преобразование последовательности печатных графических символов в восьмибайтные коды КОИ-8, соответствующие этим символам, и сохраняет эти коды в программе, отводя на один символ один байт. Последовательность символов должна быть заключена в парные ограничители.

В качестве ограничителей может использоваться любой печатный символ, за исключением пробела, символа горизонтальной табуляции и точки с запятой (;). Хотя могут быть использованы и алфавитно-цифровые символы в качестве огра-

ничителей, но для избежания путаницы с этой целью не следует использовать алфавитно-цифровые символы

Формат

^Астрока

где строка - ограниченная последовательность символов кода КОИ-8, размером от 1 до 16 символов.

Ограниченная последовательность символов кода КОИ-8 не должна превышать размер, характерный для операнда данного типа данных. Если, например, операция ^a появляется в одном из операндов инструкции MOVW (тип данного - слово), то ограниченная последовательность символов не должна содержать более двух символов.

Пример.

```
MOVW   #^A/NT/,R0      N помещается в младший байт,
                       T помещается в старший байт слова
MOVL   #^A/GOLD/,R0 ; Поместить G, O, L, D в регистр R0
                       G помещается в младший байт,
                       D помещается в старший байт
                       длинного слова
MOVL   #^A/AB/, R0     A, B помещаются в младшие байты
                       длинного слова, два старших байта
                       заполняются нулями
```

3.6.2.2. Операция регистровой маски ^M

Операция регистровой маски осуществляет преобразование обозначения регистра или списка обозначений регистров, заключенного в угловые скобки, в 1- или 2-х байтную регистро-

вую маску. Эта регистровая маска используется директивами .ENTRY и .MASK.

Форматы

^Mрегистр

^M<список регистров>

где регистр - обозначение одного из регистров или спецификатора разрешения арифметического прерывания (IV или DV);

Список регистров - список обозначений регистров и/или спецификаторов разрешения арифметического прерывания, разделенных запятыми.

Операция регистровой маски осуществляет установку бита в регистровой маске для каждого указанного обозначения регистра и разрешения арифметического прерывания. В табл. 7 указаны биты регистровой маски, соответствующие тому или иному обозначению регистра и спецификатора арифметического прерывания.

Таблица 7

Обозначение регистра	Разрешение арифметического прерывания	Биты
С R0 по R11	-	с 0 по 11
R12 или AP	-	12
FP	-	13
SP	IV	14
-	DV	15

Когда операция регистровой маски используется совместно с инструкциями POPR или PUSHR, с ней можно указывать регистры с R0 по R11, R12 или aP, FP и SP. В этом случае нельзя указывать регистр PC и спецификаторы разрешения арифметического прерывания (IV и DV).

Когда операция регистровой маски используется совместно с директивами .ENTRY или .MASK, с ней можно указывать регистры с R2 по R11 и спецификаторы разрешения арифметического прерывания (IV и DV), но нельзя указывать регистры R0, R1, FP, SP и PC. IV устанавливает ловушку по переполнению целых чисел, а DV устанавливает ловушку по переполнению десятичных чисел.

Более подробную информацию о регистровых масках и спецификаторах разрешения арифметического прерывания можно получить в документе [2].

Пример.

```
PUSHR   #^M<R0,R1,R2>          ; Сохранить регистры R0, R1,R2
POPR    #^M<R0,R1,R2>            Восстановить регистры
                                 R0, R1, R2
.ENTRY PROG, ^M<R3,R5,IV>        Сохранить регистры R3, R5 и
                               ; установить ловушку переполнения целых чисел
```

3.6.3. Операции числового представления

К операциям числового представления относятся операция формата с плавающей запятой (^F) и операция дополнения (^C). Порядок применения этих двух операций числового

представления поясняется в подпунктах 3.6.3.1, 3.6.3.2.

3.6.3.1. Операция формата с плавающей запятой

Операция формата с плавающей запятой (^F) воспринимает число с плавающей запятой и преобразует его во внутреннее представление (четырехбайтное значение). Это значение может быть использовано в любом выражении. Программа макроассемблер не производит обработки выражений с плавающей запятой.

Формат

^Flитерал

где литерал - число с плавающей запятой.

Операция формата с плавающей запятой оказывается полезной операцией, поскольку обеспечивает возможность употребления чисел с плавающей запятой в инструкциях, воспринимающих только целые числа.

Пример.

```
MOVL  #^F5.6,R1     Инструкция MOVF является предпочтитель-
                    ной для пересылки числа с плавающей
MOVF  #5.6,R1       запятой
```

3.6.3.2. Операция дополнения

Операция дополнения (^C) задает дополнение до единицы (обратный код) указанного значения.

Формат

^Стерм

где терм - любой терм или выражение. Если указывается выражение, оно должно быть заключено в угловые скобки.

До генерирования дополнения программа макроассемблер вычисляет значение терма или выражения в четырехбайтном формате.

Пример.

.LONG	^C^XFF	Результатом является шестнадцатеричное число FFFFFF00
.LONG	^C25	Результатом является шестнадцатеричное число FFFFFFE6

3.7. Бинарные операции

В отличие от унарных операций, 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. 8 представлена краткая информация по бинарным операциям.

Таблица 8

Бинарные операции

Знак	Наименование	Пример	Выполняемое действие
+	Плюс	a + в	Сложение
-	Минус	a - в	Вычитание
*	Звездочка	a * в	Умножение
/	Косая черта	a / в	Деление
@	Знак коммерческое "эт"	a @ в	Арифметический сдвиг
&	Коммерческое "и"	a & в	Логическое "и"
!	Восклицательный знак	a ! в	Логическое "или"
\	Обратная косая черта	a \ в	Логическое исключающее "или"

Все бинарные операции имеют равный приоритет. Термы и выражения могут быть сгруппированы для обработки посредством использования угловых скобок. Термы и выражения, заключенные в угловые скобки обрабатываются в первую очередь, а остальные операции выполняются по порядку, слева направо.

Пример.

```
.LONG   1 + 2 * 3          Равно 9
.LONG   1 + <2 * 3>        Равно 7
```

Отметим, что в результате выполнения всех бинарных операций получается значение в четырехбайтном формате. Если в качестве операндов используются одно- или двухбайтные величины, то результат помещается в младшие байты (байт) четырехбайтной области. Если усечение приводит к потере значимости, программа макроассемблер выдает сообщение об ошибке.

3.7.1. Операция арифметического сдвига

Операция арифметического сдвига (@) используется для выполнения сдвига влево или вправо арифметических величин. Первый аргумент сдвигается влево или вправо на число разрядов, которое задается вторым аргументом. Если второй аргумент является положительным, то первый аргумент сдвигается влево; если второй аргумент является отрицательным, первый аргумент сдвигается вправо. Если первый аргумент сдвигается влево, младшие разряды устанавливаются в нуль; если же первый аргумент сдвигается вправо, то старшие разряды заполняются значением, присутствовавшем в старшем знаковом разряде до сдвига (знаковый бит).

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 (шестнадцате-
```

ричное)

3.7.2. Операция логическое "И"

Операция логическое "и" (&) задает функцию логическое "И" между двумя операндами.

Пример.

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

3.7.3. Операция логическое "или"

Операция логическое "или" (!) задает функцию логическое "или" между двумя операндами.

Пример.

```
M = ^B10100
N = ^B10101
K = ^B11000
        .LONG   M!N       ; Результат 10101 (двоичное)
        .LONG   M!K         Результат 11100 (двоичное)
```

3.7.4. Операция логическое исключающее "или"

Операция логическое исключающее "или" (\) задает функцию логическое исключающее "или" между двумя операндами.

Пример.

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

3.8. Операторы прямого присваивания

С помощью оператора прямого присваивания символу присваивается определенное значение. В отличие от символа, который употребляется в качестве метки, символ, определенный с помощью оператора прямого присваивания, может быть переопределен любое число раз.

Форматы

Символ = выражение

Символ == выражение

где символ - символ, определенный пользователем;

выражение - любое выражение, не содержащее неопределенных символов.

Посредством формата с единственным знаком равенства (=) определяется локальный символ, а с помощью формата с двумя знаками равенства (==) определяется глобальный символ.

Операторы прямого присваивания подчиняются следующим трем синтаксическим правилам:

1) символ и выражение, определяющее значение символа, должны разделяться знаком равенства или двойным знаком равенства. Пробелы, предшествующие или следующие за оператором прямого присваивания, не имеют значения при определе-

нии конечного значения;

2) в одном операторе прямого присваивания может быть определен только один символ;

3) за оператором прямого присваивания может следовать только поле комментария.

Кроме того, символ в операторе прямого присваивания размещается в поле метки.

Пример.

```
VALUE == 4                  Глобальный символ VALUE имеет
                            значение 4
MIN = VALUE - 3             MIN имеет значение 1
MAX = VALUE @ 3           ; MAX имеет значение 20 (шестнад-
                            цатеричное)
MM = ^X100/^X10             MM имеет значение 10 (шестнадца-
                            теричное)
V = 45 + 3                  V имеет значение 48 (десятичное)
```

3.9. Счетчик адресов программы

Символ точка (.), которым обозначается текущий счетчик адресов программы, всегда имеет значение, равное адресу текущего байта. Программа макроассемблер в начале трансляции и в начале каждой новой программной секции устанавливает значение этого счетчика равное 0.

Каждое исходное предложение языка макроассемблер, в результате трансляции которого в об'ектном модуле распределяется память, приводит к увеличению значения текущего счетчика адресов программы на число распределенных байтов.

например, директива .LONG 0 приводит к увеличению текущего счетчика адресов программы на 4. Однако оператор прямого присваивания, за исключением его особой формы, не приводит к увеличению значения текущего счетчика адресов программы, поскольку в результате его трансляции не происходит распределения памяти.

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. Явная установка значения счетчика адресов программы часто оказывается полезной при определении структур данных. Посредством явной установки значения счетчика адресов программы не следует производить резервирование памяти, для этой цели следует использовать директивы .BLKX.

Формат

.=выражение

где выражение - выражение, которое не содержит неопределенных символов.

В перемещаемой программной секции указанное выражение должно быть перемещаемым, то есть оно должно относиться к адресу в текущей программной секции. Оно может относиться к текущему счетчику инструкций.

Пример.

```
.=.+40                  Перевести счетчик инструкций
                        вперед
```

Когда ранее определенная программная секция продол-

жается в том же текущем модуле, текущий счетчик инструкций устанавливается на последнее значение текущего счетчика инструкций в данной программной секции.

Когда счетчик инструкций встречается в поле операнда инструкции, текущий счетчик инструкций принимает значение адреса данного операнда, а не значение адреса, с которого начинается данная инструкция. По этой причине текущий счетчик инструкций обычно не используется в качестве элемента определителя операнда.

4. РЕЖИМЫ АДРЕСАЦИИ

В данном разделе собрана информация по режимам адресации ЭВМ СМ 1700, а также приведены примеры предложений языка макроассемблер, в которых используются описываемые режимы адресации. Полная информация по режимам адресации представлена в документе [2].

Всего существует четыре типа режимов адресации:

1) адресация регистров общего назначения;

2) адресация программного счетчика;

3) индексация;

4) адресация в инструкциях относительного перехода.

Хотя режим индексации по сути является одним из режимов адресации регистров общего назначения, этот режим рассматривается как отдельный тип адресации, поскольку может быть использован только в комбинации с другими типами режимов адресации.

Краткая информация по режимам адресации приведена в приложении 2.

4.1. Режимы адресации регистров общего назначения

В режимах адресации регистров общего назначения используются регистры с R0 по R12, указатель аргумента ap (то же самое, что и R12), указатель кадра FP и указатель стека SP.

Существует восемь режимов адресации регистров общего назначения:

1) регистровый режим;

2) косвенно-регистровый режим;

3) режим адресации с автоувеличением;

4) режим косвенной адресации с автоувеличением;

5) режим адресации с автоуменьшением;

6) режим адресации по смещению;

7) режим косвенной адресации по смещению;

8) режим литеральной адресации.

4.1.1. Регистровый режим

При регистровом режиме операндом является содержимое регистра, к которому происходит обращение, за исключением следующих случаев:

1) в случае операндов в виде квадрослов, операндов формата D, G и операндов с полем переменной длины, операндом является конкатенация содержимого регистра N и содержимого регистра N+1;

2) в случае операндов в виде октаслов и операндов формата H операндом является конкатенация содержимого регистра N и регистров N+1, N+2, N+3.

В каждом из этих случаев самые младшие байты операнда содержатся в регистре N, а самые старшие в самом старшем из используемых регистров, в N+1, либо в N+3.

Результат операции является непредсказуемым, если при регистровом режиме используется счетчик инструкций PC,

либо, если при работе с данными расширенного типа операнд распространяется на PC.

Форматы

RN

AP

FP

SP

где N - число в диапазоне от 0 до 12.

Пример.

```
        CMPB    R0, R1      сравнить младшие байты R0
                            и R1
        BEQL    LABEL       если они совпадают, то пе-
                            рейти на LABEL
        CLRB    R0          очистить младший байт R0
        INCB    R1          добавить единицу к младшему
                            байту R1
LABEL:  CLRQ    R2          очистить регистры R2 и R3
```

4.1.2. Косвенно-регистровый режим

При косвенно-регистровом режиме регистр содержит адрес операнда. Косвенно-регистровый режим можно использовать при индексации.

Форматы

(RN)

(AP)

(FP)

(SP)

где N - число в диапазоне от 0 до 12.

Пример.

```
        MOVAL   DATA1,R1        получить адрес данного DATA1 в R1
        MOVAL   DATA2,R2        получить адрес данного DATA2 в R1
        CMPL    (R1),(R2) ;     сравнить значение
        BEQL    LAB             если равны, то перейти на LAB
        INCL    (R1)            если нет, то увеличить на 1 зна-
                                чение данного DATA1
LAB:    MOVL    (SP),R0         переслать верхний элемент стека
                                в R0
```

4.1.3. Режим адресации с автоувеличением

При адресации с автоувеличением указанный регистр содержит адрес операнда. После обработки адреса операнда, содержащегося в операнде, процессор производит приращение адреса операнда на размер типа данных операнда. Процессор производит приращение содержимого регистра на 1, 2, 4, 8 или 16 для соответствующих операндов в формате байта, слова, длинного слова, квадрослова, октаслова.

Режим адресации с автоувеличением может быть использован при индексации, однако индексный регистр не может быть тем самым регистром, который указан в адресации с автоувеличением.

Форматы

(RN)+

(AP)+

(FP)+

(SP)+

где N - число в диапазоне от 0 до 12.

Пример.

```
MOVAL FILE1,R0          Переслать адрес FILE1 в R0
INCB  (R0)+             Добавить 1 к первому байту
                        файла FILE1
CLRB  (R0)+             Очистить второй байт файла FILE 1

MOVAL FILE2,R1          Переслать адрес FILE2 в R1
CLRQ  (R1)+             Очистить первое и второе длинное
                        слово файла FILE2

INCL  (R1)+             ; Добавить 1 к третьему длинному
                        слову файла FILE2
```

4.1.4. Режим косвенной адресации с автоувеличением

При косвенной адресации с автоувеличением указанный регистр содержит адрес, который является адресом, по которому хранится адрес операнда (указатель операнда). После обработки адреса операнда процессор увеличивает содержимое указанного регистра на 4 (количество байтов в адресе).

Режим косвенной адресации с автоувеличением может быть использован совместно с индексацией, однако индексный регистр не может быть тем же самым регистром, что указан в

косвенной адресации с автоувеличением.

Форматы

@(RN)+

@(AP)+

@(FP)+

@(SP)+

где N - число в диапазоне от 0 до 12.

Пример.

```
MOVAL TABLE, R1     Получить адрес таблицы адресов в R1
INCL  @(R1)+        Добавить 1 к длинному слову, адрес ко-
                    торого является первым в таблице адре-
                    сов, содержимое регистра R1 увеличива-
                  ; ется на 4
CLRB  @(R1)+        Очистить байт, адрес которого является
                    вторым в таблице адресов, затем к соде-
                    ржимому R1 добавляется число 4
MOVL  R0,@(R1)+     Переслать содержимое R0 по адресу, ко-
                    торый является третьим в таблице адре-
                  ; сов, затем к содержимому R1 добавляется
                  ; число 4
```

4.1.5. Режим адресации с автоуменьшением

При адресации с автоуменьшением процессор уменьшает содержимое указанного регистра на размер типа данных операнда, после чего регистр содержит адрес операнда. Процессор уменьшает содержимое регистра на 1, 2, 4, 8 и 16 для,

соответствующих операндов в формате байта, слова, длинного слова, квадрослова и октослова.

Режим адресации с автоуменьшением может быть использован совместно с индексацией, однако индексный регистр не может быть тем же самым, что указан в режиме адресации с автоуменьшением.

Форматы

-(RN)

-(AP)

-(FP)

-(SP)

где N - число в диапазоне от 0 до 12.

Пример.

```
INCB    -(R1)     вычесть 1 из содержимого R1 и добавить
                  ; 1 к байту, адрес которого содержится в
                  ; R1
CLRL    -(R2)     вычесть 4 из содержимого R2 и очистить
                  ; длинное слово, адрес которого содержится
                  ; в R2
```

4.1.6. Режим адресации по смещению

При использовании режима адресации по смещению адресом операнда является сумма содержимого регистра и смещения (знак распространен на длинное слово).

Режим адресации по смещению может быть использован

совместно с индексацией.

Форматы

Смщ(RN)

Смщ(AP)

Смщ(FP)

Смщ(SP)

где N - число от 0 до 12;

смщ - выражение, определяющее смещение; данное выражение может сопровождаться одним из указателей длины смещения, которые указывают на количество байтов, необходимых для хранения данного смещения. Указатели приведены в табл. 9.

Таблица 9

Указатель длины смещения	Значение
B^	Для хранения смещения требуется один байт
W^	Для хранения смещения требуется одно слово (два байта)
L^	Для хранения смещения требуется одно длинное слово (4 байта)

Если выражению не предшествует ни один указатель длины смещения, а значение выражения известно, тогда программа макроассемблер распределяет наименьшее количество байтов (1, 2 или 4), необходимых для хранения смещения. Если выражению не предшествует ни один указатель длины смещения, и

значения выражения не известно, то резервируется для этого смещения одно слово (2 байта). Отметим, что как перемещаемые смещения, так и смещения, определяемые в исходной программе позже, программа макроассемблер рассматривает как неизвестные. Если действительное значение смещения не укладывается в отведенную память, компоновщик выдает сообщение об ошибке.

Пример.

```
MOVAL TAB_ARG,R0      Получить в R0 адрес таблицы значений
                      аргументов
CLRW  L^DCP(R0)       Очистить слово, адрес которого равен
                      значению выражения DSP плюс содержи-
                    ; мое регистра R0
                      смещение хранится в виде длинного
                      слова
MOVL  RED(R0), R1     Переслать в R1 длинное слово,адрес
                    ; которого равен адресу таблицы
                      TAB_ARG плюс значение выражения RED
                      смещение хранится в виде слова, по-
                      скольку символ RED не определен
INCB  B^10(R0)      ; Добавить 1 к байту, адрес которого
                    ; равен адресу таблицы TAB_ARG плюс 10
                    ; смещение хранится в виде байта
```

Примечание. Если значение смещения равно 0 и не определена длина смещения, то выбирается косвенно-регистровый режим, вместо режима адресации по смещению.

4.1.7. Режим косвенной адресации по смещению

В режиме косвенной адресации по смещению сумма содержимого регистра и значения смещения (знак распространяется на длинное слово) является адресом адреса операнда (указатель операнда).

Режим косвенной адресации по смещению может быть использован совместно с индексацией.

Форматы

@смщ(RN)

@смщ(AP)

@смщ(FP)

@смщ(SP)

где N - число в диапазоне от 0 до 12;

смщ - выражение, определяющее смещение; данному выражению может предшествовать один из указателей длины смещения, который определяет количество байтов, требующихся для размещения данного смещения. Указатели приведены в табл. 10.

Таблица 10

Указатель длины смещения	Значение
B^	Для хранения смещения требуется один байт
W^	Для хранения смещения требуется одно слово (два байта)

Продолжение табл. 10

Указатель длины смещения	Значение
L^	Для хранения смещения требуется одно длинное слово (4 байта)

Если выражению не предшествует ни один указатель длины смещения, а значение выражения известно, то программа макроассемблер размещает минимальное количество байтов (1, 2 или 4), необходимых для хранения значения смещения. Если выражению не предшествует ни один указатель длины смещения, и значение выражения неизвестно, то для смещения резервируется одно слово (2 байта). Отметим, что если смещение является либо перемещаемым, либо определяется в исходной программе позже, программа макроассемблер считает его неизвестным. Если действительное значения смещения не помещается в отведенную память, компоновщик выдает сообщение об ошибке.

Пример.

```
MOVAL   FILE_PTR,R2     ; Получить в R2 адрес файла указа-
                        ; телей
CLRL    @W^MAN(R2)      ; Очистить длинное слово, адрес ко-
                        ; торого хранится по адресу FILE_PTR
                        ; плюс значение MAN
                        ; для хранения смещения используется
```

```
                        слово
MOVL    @B^MAP(R2),R3   Получить в R3 длинное слово, адрес
                        которого хранится по адресу FILE_
                        PTR плюс значение MAP
                      ; для хранения смещения используется
                        слово
TSTL    @32(R2)         Проверить длинное слово, адрес ко-
                      ; торого хранится по адресу FILE_PTR
                        плюс 32
                        для хранения смещения используется
                        слово
```

4.1.8. Режим литеральной адресации

При литеральной адресации значение литерала хранится в самом байте, указывающем на режим адресации.

Форматы

#литерал

S^литерал

где литерал - выражение, целая константа или константа с плавающей запятой.

Литерал должен быть представлен в короткой литеральной форме. Другими словами, целые числа должны быть в диапазоне от 0 до 63, а константы с плавающей запятой должны принимать одно из 64-х значений, указанных в табл. 11. Короткие литералы с плавающей запятой хранятся в виде трехразрядной дробной части. В табл. 11 показаны также значения (для каж-

дого литерала) показателя степени и дробной части. Информация по формату коротких литералов приведена в документе [2].

Таблица 11

Короткие литералы с плавающей запятой

1) пс	Дробная часть							
	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

1) пс - показатель степени.

Пример.

```
MOVB    #2,R0          Переслать 2 в младший байт R0
MOVL    S^LITER,R1 ; Значение LITER пересылается в R1
                   ; если значение LITER выходит из
```

```
                    ; диапазона 0-63, то генерируется
                      ошибка
MOVF    #2.25,R4    ; В R0 занесено число 2.25
```

Примечания:

1. Когда используется формат #литерал, программа макроассемблер осуществляет выбор между литеральной адресацией и непосредственной адресацией. Программа макроассемблер выбирает непосредственную адресацию, если удовлетворяется любое из следующих условий:

- значение литерала не укладывается в формат короткого литерала;

- литерал является перемещаемым или внешним выражением;

- литерал представляет собой выражение, содержащее неопределенные символы.

Различие между литеральной и непосредственной адресацией заключается в об'еме памяти, необходимой для хранения литерала в инструкции.

2. Формат S^#литерал принуждает программу макроассемблер использовать литеральную адресацию.

4.2. Режимы адресации через счетчик инструкций

В режимах адресации через счетчик инструкций PC в качестве регистра общего назначения используется счетчик инструкций PC. Существует пять различных режимов адресации через счетчик инструкций:

1) относительная адресация;

2) косвенная адресация;

3) абсолютная адресация;

4) непосредственная адресация;

5) адресация общего вида.

4.2.1. Режим относительной адресации

При относительной адресации адресом операнда является указанный адрес. Программа макроассемблер запоминает этот адрес как смещение относительно счетчика PC.

Режим относительной адресации может использоваться совместно с индексацией.

Формат

Адрес

где адрес - выражение, определяющее адрес; данному выражению может предшествовать один из указателей длины смещения, который указывает на количество байтов, необходимых для хранения смещения. Обозначение и использование указателей длины смещения приведено в табл. 12.

Таблица 12

Указатель длины смещения	Значение
B^	Для хранения смещения требуется один байт
W^	Для хранения смещения требуется одно слово (2 байта)
L^	Для хранения смещения требуется одно длинное слово (4 байта)

Если выражению, определяющему адрес, не предшествует никакой указатель длины смещения, и значение данного выражения известно, то программа макроассемблер выбирает наименьшее количество байт (1, 2 или 4), необходимых для хранения смещения. Если адресному выражению не предшествует никакой указатель длины смещения, и значение выражения неизвестно, то программа макроассемблер использует указатель длины смещения, принятый по умолчанию. Если адресное выражение определено в программе после оператора, в котором оно используется, или определено в другой программной секции, то программа макроассемблер считает, что его значение неизвестно.

Пример.

```
MOVL W^<TAB+8>, R0      Получить в R0 длинное слово, расположенное по адресу TAB+8.
                        Для хранения смещения используется
```

```
                    ; слово
MOVL L^POINT, R1    ; Получить в R1 длинное слово, распо-
                      ложенное по адресу POINT.
                    ; Для хранения смещения используется
                      длинное слово.
MOVL LABEL, R2        Получить в R2 длинное слово, распо-
                    ; ложенное по адресу LABEL,
                    ; если метка LABEL не определена в
                      данной программной секции, то ис-
                      пользуется длина смещения по умол-
                      чанию.
```

4.2.2. Режим косвенной относительной адресации

При косвенной относительной адресации указанный адрес является адресом адреса операнда (указатель на операнд). Программа макроассемблер хранит адрес в виде смещения относительно счетчика инструкций PC.

Косвенная относительная адресация может быть использована совместно с индексацией.

Формат

@адрес

где адрес - выражение, определяющее адрес; данному выражению может предшествовать один из указателей длины смещения, который указывает количество байтов, необходимых для хранения смещения. Указатели приведены в табл. 13.

Таблица 13

Указатель длины смещения	Значение
B^	Для хранения смещения требуется один байт
W^	Для хранения смещения требуется одно слово (два байта).
L^	Для хранения смещения требуется одно длинное слово (4 байта).

Если адресному выражению не предшествует ни один из указателей длины смещения, и значение этого выражения известно, то программа макроассемблер выбирает наименьшее количество байт (1, 2 или 4), необходимых для хранения этого смещения. Если адресному выражению не предшествует ни один из указателей длины смещения, и значение выражения неизвестно, программа макроассемблер использует длину смещения, принятую по умолчанию. Если адресное выражение определено в программе после оператора, в котором оно используется, или определено в другой программной секции, программа макроассемблер считает, что его значение неизвестно.

Пример.

```
CLRB    @L^APTR+8       очистить байт, адрес которого хра-
                        нится по адресу APTR+8
                        для хранения смещения используется
                        длинное слово
TSTL    @W^APTR         проверить длинное слово, адрес ко-
```

торого хранится по адресу APTR
для хранения смещения используется
слово

4.2.3. Режим абсолютной адресации

В режиме абсолютной адресации указанный адрес является адресом операнда. Этот адрес хранится как абсолютный виртуальный адрес (сравните с относительной адресацией, при которой адрес хранится в виде смещения относительно счетчика инструкций PC).

Режим абсолютной адресации может использоваться совместно с индексацией.

Формат

@#адрес

где адрес - выражение, определяющее адрес.

Пример.

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

4.2.4. Режим непосредственной адресации

При непосредственной адресации операндом является указанный литерал.

Форматы

#литерал

I^#литерал

где литерал - выражение, целая константа, константа с плавающей запятой.

Пример.

```
MOVL    #1100,R0 ; Получить в R0 значение 1100.
                   Значение 1100 хранится в длинном
                 ; слове.
MOVB    #DISK,R1   Значение символа  DISK пересылается
                   в младший байт регистра R1.
MOVF    #0.3,R2    Получить в R2 значение 0.3
                   оно хранится в формате с плавающей
                   запятой в четырех байтах (оно не мо-
                   жет быть представлено в виде коротко-
                   го литерала)
CMPL    I^#^#4,R3; Сравнить 4 с содержимым R3
                   число 4 хранится в длинном слове,
                   поскольку операция I^ указывает на не-
                 ; посредственную адресацию
```

Примечания:

1. Когда используется формат #литерал программа макроассемблер осуществляет выбор, использовать литеральную адресацию (п. 4.1.8) или непосредственную адресацию. Если литерал представляет собой целое число в диапазоне от 0 до 63 или константу с плавающей запятой, удовлетворяющую формату короткого литерала, программа макроассемблер исполь-

зует литеральную адресацию. Если литералом является выражение, то программа макроассемблер использует литеральную адресацию при существовании всех следующих условий:

- выражение является абсолютным;

- выражение не содержит непосредственных символов;

- значение выражения удовлетворяет формату короткого литерала.

Во всех других случаях программа макроассемблер использует режим непосредственной адресации.

Различие между непосредственной и литеральной адресацией заключается в об'еме памяти, требуемом для хранения литерала в инструкции. При непосредственной адресации в зависимости от типа данных операнда литерал хранится в виде байта, слова или длинного слова.

2. I^#литерал - это формат вынуждает программу макроассемблер использовать непосредственную адресацию.

3. Существует два способа спецификации чисел с плавающей запятой - в виде числового значения или в виде символического имени. Программа макроассемблер по разному обрабатывает эти величины:

- числовые значения преобразуются в соответствующее внутреннее представление чисел с плавающей запятой;

- символы не преобразуются, программа макроассемблер предполагает, что эти величины уже приведены к внутреннему представлению чисел с плавающей запятой.

После получения значения программа макроассемблер пытается преобразовать внутреннее представление этого зна-

чения к формату короткого литерала с плавающей запятой. Если такая попытка не удается, используется непосредственная адресация; если же такая попытка удается, используется литеральная адресация для коротких литералов с плавающей запятой.

4.2.5. Режим адресации общего вида

При адресации общего вида адрес является адресом операнда. Компоновщик преобразует данный режим адресации либо к относительной адресации, либо к абсолютной адресации. Если адрес является перемещаемым, компоновщик преобразует адресацию общего вида в относительную адресацию. Если адрес является абсолютным, компоновщик преобразует адресацию общего вида в абсолютную адресацию. Режим адресации общего вида предназначается для написания позиционно-независимых кодов, когда программист не знает, является ли адрес перемещаемым или абсолютным. В случае использования режима адресации общего вида для хранения операнда требуется 5 байтов.

Адресация общего вида может использоваться совместно с индексацией.

Формат

G^адрес

где адрес - выражение, определяющее адрес.

Пример.

```
TSTL  G^ADR          проверяется длинное слово, поме-
                     ченное меткой ADR
```

```
                    если метка определена как абсо-
                ; лютная, то используется абсолютная
                    адресация, если метка определена
                    как перемещаемая, то - относитель-
                    ная адресация
CALLS #4,G^COUNT    вызывается процедура COUNT с
                    четырьмя аргументами в стеке
```

4.3. Режим индексации

Режим индексации является режимом адресации регистров общего назначения, который может использоваться только в комбинации с другим режимом адресации, называемым базовым режимом. Базовым режимом может быть любой режим адресации, за исключением прямого обращения к регистру, непосредственной адресации, литеральной адресации, индексации и адресации в инструкциях относительного перехода. Для получения базового адреса первым программой макроассемблер обрабатывается базовый режим адресации. Затем добавляется к базовому адресу произведение содержимого индексного регистра и числа байтов операнда данного типа данных. Полученная сумма является адресом операнда.

Сочетая индексацию с другими режимами адресации можно образовать следующие режимы адресации:

- косвенно-регистровый с индексацией;
- индексация с автоувеличением;
- косвенная адресация с автоувеличением и индексацией;

- индексация с автоуменьшением;
- адресация по смещению с индексацией;
- косвенная адресация по смещению с индексацией;
- относительная адресация с индексацией;
- абсолютная адресация с индексацией;
- адресация общего вида с индексацией.

Для каждого из этих режимов процесс обработки базового режима адресации и добавление индексного регистра одинаков.

Форматы

База[RX]

База[AP]

База[FP]

База[SP]

где база - любой режим адресации, по которому определяется базовый адрес, за исключением прямого обращения к регистру, непосредственной адресации, литеральной адресации, индексации и адресации в инструкциях относительного перехода;

x - число в диапазоне от 0 до 12, определяющее индексный регистр.

В табл. 14 приведен список режимов адресации с индексацией.

Пример.

косвенно-регистровый 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 ра-
                                вен адресу SYS плюс 10*1
        CLRL    (R4)[R1]        очистить длинное слово, чей
                                адрес равен адресу SYS плюс
                                10*4
        CLRQ    (R4)[R1]        очистить квадрослово,
                                адрес которого равен адресу
                                SYS плюс 10*8
        CLRO    (R4)[R1]        очистить октаслово,
                                чей адрес равен адресу SYS
                                плюс 10*16
```

индексация с автоувеличением

```
        CLRQ    (R4)+[R1]       очистить квадрослово,
                                чей адрес равен адресу SYS
                                плюс 10*8
```

косвенная индексация 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, хра-
```

няцийся в LIST
R3 теперь содержит адрес
SYS+8

косвенная индексация 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 плавающей запятой, чей ад-
                        рес равен 80*4 плюс адрес,
                        хранящийся по адресу (ADA+16)
```

Таблица 14

Режимы адресации с индексацией

Режим	Формат
	1) 2)
Косвенно-регистровый с индексацией	(RN) [RX]
Индексация с автоувеличением	(RN) + [RX]
Косвенная адресация с автоувеличением и индексацией	@(RN) + [RX]
Индексация с автоуменьшением	-(RN)[RX]
	3)
Адресация по смещению с индексацией	смщ (RN)[RX]
Косвенная адресация по смещению с индексацией	@смщ(RN)[RX]

Продолжение табл. 14

Режим	Формат
	4)
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]

1). (RN) - любой регистр общего назначения: R0...R12, AP, FP, SP.

2). [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.

3). смщ - выражение, определяющее смещение.

4) адрес - выражение, определяющее адрес.

Примечания:

1. Если при базовом режиме адресации меняются содержимое указанного регистра (адресация с автоувеличением, косвенная адресация с автоувеличением, адресация с автоумень-

шением), то этот же самый регистр не может быть указан при индексации.

2. Индексный регистр добавляется к адресу после того, как полностью отработан базовый режим адресации. Например, при косвенной адресации с автоувеличением и индексации базовый регистр содержит адрес адреса операнда. Содержимое индексного регистра (умноженное на количество единиц длины операнда данного типа данных) складывается с адресом операнда, а не с адресом, хранящемся в базовом регистре.

4.4. Адресация в инструкциях относительного перехода

При этом режиме адресации адрес хранится как подразумеваемое смещение относительно счетчика инструкций PC. Этот режим адресации используется только в инструкциях относительного перехода. Смещение для инструкций условного перехода и для инструкции BRB хранится в байте. Смещение для инструкции BRW хранится в слове (2 байта). Смещение в байте допускает адресацию на 127 байт вперед и на 128 байт назад. Смещение в слове допускает адресацию на 32 767 байт вперед и на 32 768 байт назад. При данном режиме адресации смещение считается относительно скорректированного значения программного счетчика инструкций PC, то есть относительно байта, следующего за байтом или словом, содержащим значение смещения. Более подробная информация об инструкциях относительного перехода приведена в документе [2].

Формат

адрес

где адрес - выражение, представляющее адрес.

Пример.

```
CMPL R0,R1          сравнить содержимое R0 и R1
BEQL TEST1          по равенству - переход на метку
                    TEST1
BRW  TEST2          безусловный переход на TEST2
```

5. ОБЩИЕ ДИРЕКТИВЫ ЯЗЫКА МАКРОАССЕМБЛЕР

Общие директивы языка макроассемблер предоставляют средства, позволяющие выполнять функции различных типов. В табл. 15 приведено распределение общих директив по группам, в соответствии с выполняемыми ими функциями. В данном разделе подробно описывается каждая из директив, приводятся их форматы и примеры использования. В приложении 2 приведена краткая информация по всем директивам языка макроассемблер.

Таблица 15

Краткие сведения

об общих директивах языка макроассемблер

Группа	Директива
Директивы управления листингом	.SHOW (.LIST)
	.NOSHOW (.NLIST)
	.TITLE
	.SUBTITLE (.SBTTL)
	.IDENT
	.PAGE
Директивы вывода сообщений	.PRINT
	.WARN
	.ERROR
Директивы дополнительных возможностей	.ENABLE (.ENABL)
	.DISABLE (.DSABL)
	.DEFAULT

Продолжение табл. 15

Группа	Директива
Директивы хранения данных	.BYTE
	.WORD
	.LONG
	.ADDRESS
	.QUAD
	.OCTA
	.PACKED
	.ASCII
	.ASCIC
	.ASCID
	.ASCIZ
	.F_FLOATING (.FLOAT)
	.D_FLOATING (.DOUBLE)
	.G_FLOATING
	.H_FLOATING
	.SIGNED_BYTE
	.SIGNED_WORD
Директивы управления счетчиком инструкций	.ALIGN
	.EVEN
	.ODD
	.BLKA
	.BLKB
	.BLKD

Продолжение табл. 15

Группа	Директива	
	.BLKF	
	.BLKG	
	.BLKH	
	.BLKL	
	.BLKO	
	.BLKQ	
	.BLKW	
	.END	
Директивы секционирования	.PSECT	
программы	.SAVE_PSECT	(.SAVE)
	.RESTORE_PSECT	(.RESTORE)
Директивы управления символами	.GLOBAL	(.GLOBL)
	.EXTERNAL	(.EXTRN)
	.DEBUG	
	.WEAK	
Директивы определения входной	.ENTRY	
точки программы	.TRANSFER	
	.MASK	
Директивы и поддирективы	.IF	
условной трансляции	.ENDC	
	.IF_FALSE	(.IFF)
	.IF_TRUE	(.IFT)
	.IF_TRUE_FALSE	(.IFTF)

Продолжение табл. 15

Группа	Директива
	.IIF
Директивы перекрестных ссылок	.CROSS
	.NOCROSS
Директивы генерирования	.ORDEF
инструкций	.REF1
	.REF2
	.REF4
	.REF8
	.REF16
Директива дополнительной	.LINK
компоновки	

5.1. Директивы управления листингом

.IDENT - директива идентификации

Директива .IDENT является средством языка макроассемблер для идентификации об'ектного модуля. Такая идентификация дается в дополнение к имени, присвоенному об'ектному модулю с помощью директивы .TITLE. В директиве .IDENT может быть указана строка символов, которой помечается об'ектный модуль. Данная строка символов распечатывается в заголовке файла листинга, а также появляется в об'ектном модуле.

Формат

.IDENT строка

Параметр

строка

- последовательность символов, содержащая от 1 до 31 символа, с помощью которой выполняется идентификация данного модуля; данная строка символов подобна строке, указывающей номер версии. Данная строка должна быть ограничена. Ограничителями могут служить любые парные печатные знаки, кроме левой угловой скобки (<) и точки с запятой (;). Ограничительный знак не должен использоваться в самой строке.

Пример.

.IDENT #4-6# номер версии и редакции

строка символов 4-6 включена в об'ектный модуль.

Примечания:

1. Если в исходном модуле содержится более одной директивы .IDENT, то идентификация об'ектного модуля осуществляется на основе строки символов последней из всех директив .IDENT.

2. Если символы ограничения строки не совпадают друг с другом, или указаны недопустимые символы, то выводится сообщение об ошибке.

.PAGE - директива перевода страницы

Директива .PAGE начинает в листинге новую страницу. Сама директива в листинге не распечатывается.

Директива .PAGE, входящая в состав макроопределения, игнорируется. Операция разбиения листинга на страницы производится только во время расширения макрокоманды. Более подробные сведения (см. раздел 6).

Формат

.PAGE

.SHOW и .NOSHOW - директивы управления листингом

Директивы .SHOW и .NOSHOW определяют дополнительные возможности управления листингом исходного текста программы. В директивах .SHOW и .NOSHOW может присутствовать или отсутствовать спецификация списка аргументов.

Если эти директивы заданы со списком аргументов, то директива .SHOW вызывает включение в файл листинга строк определенных типов, а директива .NOSHOW вызывает исключение из файла листинга строк определенных типов. Директивы .SHOW и .NOSHOW осуществляют управление листингом исходного текста блоков условной трансляции, макрокоманд и блоков повторения.

Если эти директивы заданы без аргументов, то они изменяют значение счетчика уровня листинга. В исходном состоянии значение счетчика листинга уровня равно нулю. При каждом задании директивы .SHOW значение счетчика уровня лис-

тинга увеличивается, а при каждом задании директивы .NOSHOW значение счетчика уровня листинга уменьшается на 1.

Если значение счетчика уровня является отрицательным, то распечатка листинга подавляется (если строки не содержат ошибку). Наоборот, если значение счетчика уровня листинга является положительным, листинг формируется. Если значение счетчика равно нулю, то строка либо распечатывается, либо подавляется, в зависимости от значений символических аргументов управления листингом.

Форматы

.SHOW

.SHOW список аргументов

.NOSHOW

.NOSHOW список аргументов

Параметр

список аргументов

- один или несколько символических аргументов, определение которых приведено в табл. 16. Допустимо использование как полного, так и краткого форматов указанных аргументов. Каждый из символических аргументов может быть указан отдельно или в сочетании с другими аргументами. Если указывается несколько символических аргументов, то они должны быть разделены пробелами или символами табуляции, запятыми. Если какой-либо аргумент не включен явным образом в директиву управления, то на протяжении всей исходной программы подразумевается его значение по умолчанию.

Таблица 16

Символические аргументы директив .SHOW и .NOSHOW

Полный формат	Краткий формат	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	MC	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

Пример.

```
    .MACRO  XX

    .SHOW                       распечатать следующую
                                строку
X=.
    .NOSHOW                     не распечатывать остаток
      .                         расширения макрокоманды

    .ENDM
    .NOSHOW EXPANSIONS          не распечатывать
                                расширения макрокоманд
    XX                          вызов макрокоманды XX
X=.
```

Примечания:

1. Счетчик уровня листинга позволяет выполнить избирательную распечатку макрокоманд. В начале определения должна быть указана директива .NOSHOW с целью уменьшения значения счетчика листинга, а в конце макроопределения можно указать директиву .SHOW с целью восстановления значения счетчика листинга до его прежнего значения.

2. Альтернативными формами директив .SHOW и .NOSHOW являются директивы .LIST и .NLIST.

.SUBTITLE - директива подзаголовка

По директиве .SUBTITLE макроассемблер распечатывает строку текста в таблице содержания, которая формируется непосредственно перед листингом трансляции. Кроме того, макроассемблер распечатывает данную строку текста в качестве подзаголовка на второй строке каждого листинга трансляции. Данный текст подзаголовка распечатывается на каждой странице до тех пор пока в программе не будет задана следующая директива .SUBTITLE с другим текстом.

Формат

.SUBTITLE комментарий

Параметр

комментарий

- строка символов кода КОИ-8 размером от 1 до 40 символов. Лишние символы отбрасываются. Данная строка символов представляет собой текстовую строку, которая должна распечатываться в таблице содержания и в качестве подзаголовка в листинге трансляции.

Пример.

.SUBTITLE условная трансляция

по данной директиве программа макроассемблер распечатывает следующий текст в качестве подзаголовка в листинге трансляции:

Условная трансляция

кроме того, этот же текст распечатывается в таблице содер-

жания листинга, в сопровождении номера страницы исходного текста и номера строки предложения, в котором была указана директива .SUBTITLE. Таблица содержания изображена на рис. 1.

Таблица содержания листинга

(1) 5000 директивы макроассемблера

(2) 300 определения макрокоманд

(2) 2300 таблицы данных и исходные установки

(3) 4800 главная программа

(4). 2800 вычисления

(4) 5000 программы ввода-вывода

(5) 1300 условная трансляция

рис. 1

Примечание. Альтернативной формой директивы .SUBTITLE является директива .SBTTL.

.TITLE - директива заголовка

Посредством директивы .TITLE об'ектному модулю присваивается имя. Это имя может состоть из 31 символа, отличных от пробела, следующих за директивой.

Формат

.TITLE имя модуля комментарий

Параметры

имя модуля

- идентификатор размером от 1 до 31 символа.

комментарий

- строка символов кода КОИ-8 размером от 1 до 40 символов, лишние символы отбрасываются.

Пример.

.TITLE печать сообщений

Примечания:

1. Имя модуля, присваиваемое с помощью директивы .TITLE, не имеет никакого отношения к спецификации файла об'ектного модуля, которая указывается в командной строке для программы макроассемблер. Это то имя, которое появляется в карте загрузки компоновщика, а также то имя, которое распознают системные программы библиотекарь или отладчик.

2. Если директива .TITLE не указаное то об'ектному модулю присваивается имя по умолчанию .MAIN. Если в исходной программе указано более одной директивы .TITLE, то имя всего модуля определяется последней из обработанных директив .TITLE.

3. При обработке аргумента "имя модуля" программа макроассемблер игнорирует все пробелы и символы табуляции вплоть до первого символа после директивы .TITLE, отличного от пробела и символа табуляции.

5.2. Директивы вывода сообщений

.ERROR - директива вывода сообщения об ошибках

Задание директивы .ERROR заставляет программу макроассемблер выводить сообщения об ошибках на терминал, в файл групповой регистрации и в файл листинга (если он существует).

Формат

.ERROR [выражение]; комментарий

Параметры

выражение

- выражение, значение которого выводится, когда во время трансляции встречается директива .ERROR.

Комментарий

- комментарий, который выводится во время трансляции по директиве .ERROR. Текст комментария должен предваряться точкой с запятой (;).

Пример.

```
.IF GREATER     ARG-100
.ERROR 5                    недопустимый аргумент
.ENDC
```

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

%MACRO-E-GENERR, GENERATED ERROR:5 недопустимый аргумент

Примечания:

1. Директивы .ERROR, .WARN, .PRINT называются директивами вывода сообщений. Они могут использоваться для вывода информации, указывающей на то, что макровызов содержит ошибку или недопустимый набор условий.

2. По завершении трансляции на экране терминала выводятся сообщения об ошибках, предупреждающие и информационные сообщения, а также порядковые номера строк, которые вызывали появление сообщений об ошибке и предупреждающих сообщений. Более подробную информацию относительно сообщений об ошибках и предупреждающих сообщений содержит [1].

3. Если директива .ERROR включена в библиотеку мокрокоманд [1], то текст комментария должен завершаться дополнительным символом точка с запятой. В противном случае программа библиотекарь лишит данную директиву комментария, и этот комментарий не будет выведен при вызове макрокоманды.

4. Строки, содержащие директивы .ERROR не включаются в файл листинга.

5. Если указанное в директиве .ERROR выражение имеет значение 0, оно не выводится в сообщении об ошибке.

.PRINT - директива вывода сообщений

Директива .RPINT позволяет транслировать информационные сообщения. Данные сообщения состоят из значения некоторого выражения и текста комментария, которые указаны в директиве .PRINT. Для интерактивных заданий информационные

сообщения выводятся на терминал, а для пакетных заданий в файл регистрации. Сообщение, выводимое по директиве .PRINT, не считается сообщением об ошибке или предупреждающим сообщением.

Формат

.PRINT [выражение] ;комментарий

Параметры

выражение

- выражение, значение которого подлежит выводу в составе информационного сообщения, когда программа макроассемблер обрабатывает директиву .PRINт.

Комментарий

- текст комментария, который выводится в составе информационного сообщения, когда программа макроассемблер обрабатывает директиву .PRINT. Тексту комментария должна предшествовать точка с запятой (;).

Пример.

.PRINT 4 ; аргумент может принимать любые значения

Примечания:

1. Директивы .PRINT, .ERROR и .WARN называются директивами сообщений. Они могут быть использованы для вывода информации, указывающей на то, что макровызов содержит ошибку или недопустимый набор условий.

2. Если директива .PRINT включена в библиотеку макрокоманд [1], то комментарий должен завершаться дополнительной точкой с запятой. В противном случае текст будет исключен из директивы и не будет выводится при вызове макроко-

манды.

3. Если выражение имеет хначение нуль, то это значение не выводится в составе информационного сообщения.

.WARN - директива вывода предупреждающего сообщения

По директиве .WARN выводится предупреждающее сообщение на терминал, в файл групповой регистрации либо в файл листинга (если он существует).

Формат

.WARN [выражение] ; комментарий

Параметры

выражение

- выражение, значение которого выводится при обработке директивы .WARN во время трансляции.

Комментарий

- текст комментария, который выводится при обработке директивы .WARN. Текст комментария должен предваряться точкой с запятой.

Пример.

```
.IF DEFINED     AREA
.IF GREATER     200-EXP
.WARN             COPY DATA
.ENDC
.ENDC
```

Если символ AREA определен и значение символа EXP меньше или равно 200, то выводится следующее предупреждаю-

щее сообщение: •

#MACRO-W-GENWRN, GENERATED WARNING:

COPY DATA

Примечания:

1. Директивы .WARN, .ERROR, .PRINT называются директивами вывода сообщений. Они могут быть использованы для вывода информации, указывающей на то, что вызов макрокоманды содержит ошибку или недопустимый набор услозий (более полная информация по вызовам макрокоманд приведена в разделе 6).

2. По завершении процесса трансляуии программ, макроассемблер выводит на терминал общее количество ошибок, предупреждающих и информационных сообщений, а также номера страниц и последовательные номера строк для строк, которые вызвали вывод сообщений об ошибках и предупреждающих сообщений (вывод может производится в групповой файл регистрации). Более полная информация по сообщениям об ошибках и предупреждающим сообщениям приведена в документе [1].

3. Если директива .WARN включена в библиотеку макрокоманд [1], то текст комментария должен завершаться дополнительной точкой с запятой. В противном случае директива .WARN будет лишена этого параметра и при вызове макрокоманды текст комментария не будет выводиться.

4. Строка, содержащая директиву .WARN не включается в файл листинга.

5. Если выражение, указываемое в директиве .WARN, имеет значение, равное нулю, то оно не выводится при выводе предупреждающего сообщения.

5.3. Директивы дополнительных возможностей

.DEFAULT - директива управления значениями по умолчанию

Директива .DEFAULT определяет длину смещения по умолчанию для относительной адресации и для косвенной относительной адресации.

Формат

.DEFAULT DISPLACEMENT, ключевое слово

Параметр

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 смеще-
                                 ние в один байт
.DEFAULT DISPLACEMENT, WORD      значением по умолчанию
                                 является слово
DECB    @DEF+6                   если не определена метка
                                 DEF, используется смеще-
```

ние в одно слово

Примечания:

1. Директива .DEFAULT не оказывает воздействия на значение смещения по умолчанию для адресации по смещению и для косвенной адресации по смещению.

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

.DISABLE - директива запрещения выполнения функций

Задание директивы .DISABLE подавляет или запрещает указанные функции макроассемблера. Более подробная информация содержится в описании директивы .ENABLE.

Формат

.DISABLE список аргументов

Параметр

список аргументов

- один или несколько символических аргументов, перечисленных в табл. 17. В директиве допустимо использование как полного формата, так и краткого формата символических аргументов. Если указывается несколько символических аргументов, они должны быть разделены запятыми, пробелами или символами табуляции.

Примечание. Альтернативной формой директивы .DISABLE является директива .DSABL.

.ENABLE - директива управления выбором функций

Задание директивы .ENABLE разрешает указанные функции макроассемблера. Директива .ENABLE и ее противоположность директива .DISABLE управляют следующими функциями программы макроассемблер:

- создание блоков локальных меток;
- об'явление всех локальных символов доступными для отладчика и включение средств обратной трассировки;
- указание о том, что ссылки на неопределенные символы являются внешними ссылками;
- усечение или округление чисел с плавающей запятой одинарной точности;
- подавление распечатки символов, которые были определены, но на которые отсутствуют ссылки;
- указание о том, что все ссылки на программный счетчик pc являются абсолютными, а не относительными.

Формат

.ENABLE список аргументов

Параметр

список аргументов

- один или несколько символических аргументов, (см. Табл. 17). Допустим как краткий, так и полный формат символических аргументов.

Если указывается несколько символических аргументов, то они должны быть разделены запятыми, пробелами или символами табуляции.

Пример.

```
.ENABLE GLOBAL                  все неопределенные символы
                                рассматриваются как внешние
.ENABLE LOCAL_BLOCK             завершается текущий
                                блок локальных меток
                              ; и начинается новый
```

Примечание. Альтернативной формой директивы .ENABLE является директива .ENABL.

Таблица 17

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 символы включаются в таблицу символов данного об'ектного модуля, которая используется отладчиком
GLOBAL	GBL	Разрешение	При разрешении этой функ-

Продолжение табл. 17

Формат		Значение по умолчанию	Выполняемая функция
Полный	Краткий		
			ции все неопределенные символы рассматриваются как внешние. При запрещении этой функции любой неопределенный символ, который не указан в 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 просто завершается
SUPPRESSION	SUP	Запрещение	При разрешении этой функции все символы, которые определены, но к которым нет ссылок, не включаются в таблицу символов.

Продолжение табл. 17

Формат		Значение по умолчанию	Выполняемая функция
Полный	Краткий		
			При запрещении этой функции все определенные символы включаются в таблицу символов
TRACEBACK	TBK	Разрешение	При разрешении этой функции имена программных секций и их длины, имена модулей и имена программ, включаются в об'ектный модуль для использования отладчиком; при запрещении макроассемблер исключает указанную информацию и, кроме того, делает информацию по любому локальному символу недоступной для отладчика
TRUNCATION	FPT	Запрещение	При разрешении этой функции числа одинарной точности с плавающей запятой усекаются. При запрещении

Продолжение табл. 17

Формат		Значение по умолчанию	Выполняемая функция
Полный	Краткий		
			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

5.4. Директивы хранения данных

.ADDRESS - директива хранения адреса

По директиве .ADDRESS выполняется хранение последовательности длинных слов, содержащей адреса в об'ектном модуле. Рекомендуется для обеспечения компоновщика дополнительной информацией при хранении адресов данных пользоваться директивой .ADDRESS вместо директивы .LONG. В разделяемых образах адреса, специфицированные с помощью директивы .ADDRESS, порождают позиционно-независимый код. Спецификация адресов в позиционно-независимом коде обсуждается в документе [1].

Формат

.ADDRESS список адресов

Параметр

список адресов

- список символов или выражений, разделенных запятыми, которые интерпретируются программой макроассемблер как адреса. Коэффициент повторения недопустим.

Пример.

```
LIST:  .ADDRESS   POINT, PAGE, BYTE4      список адресов
```

.ASCIX - директива хранения строки кодов КОИ-8

Язык макроассемблер имеет четыре директивы хранения символов кода КОИ-8:

1). .ASCII - хранение строки кодов КОИ-8;

2) .ASCIC - хранение пересчитанной строки кодов КОИ-8;

3) .ASCID - хранение строки кодов КОИ-8 с описателем строки;

4) .ASCIZ - хранение строки кодов КОИ-8 с завершающим нулем.

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

алфавитно-цифровые символы.

В строке может появляться любой символ, кроме символов нуля, возврата каретки и перевода формата. Программа макроассемблер не выполняет преобразования строчных букв в прописные.

Директивы хранения символов кода КОИ-8 выполняют преобразование символов в восьмиразрядные значения кодов КОИ-8 и хранение этих кодов, при этом на один символ отводится один байт.

Любой символ, включая символы нуля, возврата каретки и перевода формата, может также представлен выражением, заключенным в угловые скобки вне ограничителей. При этом вы должны определить значения кода КОИ-8 для символов нуля, возврата каретки и перевода формата с помощью оператора прямого присваивания. Директивы хранения символов кода КОИ-8 запоминают восьмиразрядное двоичное значение, специфицированное данным выражением.

Строки символов кода КОИ-8 могут продолжаться на несколько строк текста программы, однако строка, расположенная на одной строке текста программы, должна быть ограничена с обоих концов, причем для каждой строки символов кода КОИ-8 могут быть использованы различные пары ограничителей.

Пример.

```
CR=13
LF=10
 .ASCIC ?HELLO!? <CR><LF>!MY FRIEND!
 .ASCIZ /GO BACK!/
```

```
.ASCIC @GENERAL REGISTER@
.ASCII #HELLO,# <CR><LF>?MY FRIEND!?
```

.ASCII - директива хранения строки символов кода КОИ-8

Директива .ASCII преобразует строку символов в коде КОИ-8 в последовательность байтов. Каждый символ размещается в одном байте.

Формат

.ASCII строка

Параметр

строка

- ограниченная строка символов кода КОИ-8.

Пример.

```
.ASCII   /GOOD BYE!/                  ограничитель /
.ASCII   @UNARY OPERATORS +,-,^@ ; ограничитель @
```

.ASCIC - директива хранения строки символов Кода КОИ-8 со счетчиком

Директива .ASCIC выполняет такую же функцию, что и директива .ASCII, за исключением того, что при задании директивы .ASCIC перед данными (строкой символов) включается счетный байт (счетчик). В этом счетном байте содержится длина строки в байтах. В указанной длине учитывается любой символ КОИ-8 (по одному байту на символ), расположенный внутри ограниченной строки, однако не учитывается счетный байт.

Директива .ASCIC оказывается полезной при копировании текста, поскольку счетчик указывает на длину текста, предназначенного для копирования.

Формат

.ASCIC строка

Параметр

строка

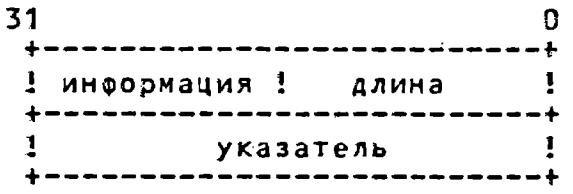
- ограниченная строка символов кода КОИ-8.

Пример.

```
CR=13                    определение символа CR
 .ASCIC /NUMBER/ <CR>    эта строка кодов .ASCIC эквива-
 .BYTE  7                лентна комбинации: счетчик, а за
 .ASCII /NUMBER/ <CR> ;  ним строка кодов .ASCII
```

.ASCID - директива хранения строки символов кода КОИ-8 с описателем

Директива .ASCID выполняет ту же самую функцию, что и директива .ASCII, за исключением того, что перед данными (строкой символов) при задании директивы .ASCID вставляется описатель строки. Этот описатель строки имеет формат изображенный на рис.2



длина

- длина строки (2 байта);

информация

- описательная информация (не используется) (2 байта);

указатель

- позиционно-независимый указатель на строку (4 байта).

Рис. 2

Описатели строк используются в процедурах вызова. Понятие позиционной независимости обсуждается в документе [1].

Формат

.ASCID строка

Параметр

строка

- ограниченная строка символов кода КОИ-8.

Пример.

```
DATA_1: .ASCID /STRING_1/   ; описатель строки 1
DATA_2: .ASCID /STRING_2/     описатель строки 2
DATA_3: .ASCID /STRING_3/     описатель строки 3
 PUSHAL  DATA_1               поместить адреса
 PUSHAL  DATA_2               описателей в стек
 PUSHAL  DATA_3
```

CALLS #3, PROC вызов процедуры

.ASCIZ - директива хранения строки символов кода КОИ-8 с завершающим нулем

Директива .ASCIZ выполняет ту же самую функцию, что и директива .ASCII, за исключением того, что при задании директивы .ASCIZ в качестве завершающего символа строки символов добавляется нулевой байт. Таким образом, если создан с помощью директивы .ASCIZ список текстовых строк, то для фиксации конца строки пользователь должен выполнить поиск только нулевых символов в последнем байте.

Формат

.ASCIZ строка

Параметр

строка

- ограниченная строка символов кода КОИ-8.

Пример.

```
.ASCIZ  ?CONDITION          9 символов в строке
                            данные занимают 10 байтов
.ASCIZ  /FIRST ARGUMENT/    14 символов в строке
                            данные занимают 15 байтов
```

.BYTE - директива хранения данных в виде байтов

При задании директивы .BYTE в об'ектном модуле генерируются двоичные значения в виде последовательности байтов.

Формат

.BYTE список выражений

Параметр

список выражений

- одно или несколько выражений разделенных запятыми. Каждое выражение обрабатывается сначала как выражение в формате длинного слова, затем значение каждого выражения усекается до одного байта. Значение любого выражения должно 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 повторения ограниченный квадратными скобками. Выражение, за которым следует коэффициент повторения имеет следующий формат:

Выражение1[выражение2]

где выражение1 - выражение, указывающее значение, которое должно быть запомнено;

[выражение2] - коэффициент повторений, указывающий на число повторений запоминаемого значения. Данное выражение не должно содержать каких-либо неопределенных символов и должно быть абсолютным. Квадратные ско-

бки являются обязательной частью формата.

Пример.

```
.BYTE  <10+20>*2              запоминается значение 60
.BYTE  10,64-<12*2>,^xa       запоминается 3 байта данных
.BYTE  0                    ; запоминается 1 байт данных
.BYTE  15,a+1[8+2],abc        запоминается 12 байтов данных
```

Примечания:

1. Если три старшыь байта выражения в длинном слове имеют значение отличное от 0 или от ^XFFFFFF, то выдается сообщение об ошибке.

2. Относительное выражение может приняьь во время компоновки значение, превосходящее 1 байт. В этом случае в обрабатываемом об'ектном модуле компоновщик выдает диагностическое сообщение об усечении, например:

```
a:   .BYTE A    ; если виртуальный адрес предложения a равен 256 или больше, то употребление относительного символа "a" приведет к тому, что система эыдаст диагностическое сообщение об усечении
```

3. Диркетива .SIGNED_BYTE является той же директивой .BYTE, за исключением того, что выдается диагностическое сообщение в том случае, если указано значение в диапазоне от 128 до 255.

.D_FLOATING(DOUBLE) - директива хранения данных с плавающей запятой в формате D

При задании директивы .D_FLOATING программа макроассемблер обрабатывает указанные константы с плавающей запятой и запоминает результаты в об'ектном модуле. По директиве .D_FLOATING генерируется шестидесятичетырехразрядное число двойной точности с плавающей запятой (1 разряд на знак, 8 разрядов на показатель степени и 55 разрядов на дробную часть). Сведения о том, как хранятся числа с плавающей запятой одинарной точности, приведены в описании директивы .F_FLOATING.

Форматы

.D_FLOATING список констант

.DOUBLE список констант

Параметр

список констант

- список констант с плавающей запятой. Данные константы не могут содержать каких-либо унарных или бинарных операций, кроме унарного плюса и унарного минуса.

Пример.

```
.D_FLOATING     1000,1.0E3,1.0000000E-9          список
.DOIBLE         3.1415928,1.107153423828         констант
.D_FLOATING     5,10,15,0,0.5
```

Примечания:

1. Числа с плабающей запятой двойной точности всегда

округляются. На них не оказывает влияния директива .ENABLE TRUNCATION.

2. Константам с плавающей запятой, входящим в состав списка литералов не должна предшествовать операция формата с плавающей запятой (^F).

.F_FLOATING (.FLOAT) - директива хранениия данных с плавающей запятой в формате F

Директива .F_FLOATING выполняет обработку констант с плавающей запятой и запоминает результаты в об'ектном модуле. Директива .F_FLOATING генерирует тридцатидвухразрядное данное с плавающей запятой одинарной точности (1 разряд на знак, 8 разрядов на показатель степени и 23 разряда на дробную часть).

Формат

.F_FLOATING список констант

.FLOAT список констант

Параметр

список констант

- список констант с плавающей запятой. Данные константы не должны содержать каких-либо 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 случаях
                                      генерируется число
```

```
                                        134,2
.F_FLOATING -0.75
.FLOAT       0,25,50                    список
.F_FLOATING -0.75,1E38,-1.0E-37         данных
```

Примечания:

1. Сведения относительно выбора функции округления или усечения чисел с плавающей запятой приведены в описании директивы .ENABLE.

2. Константам с плавающей запятой, входящим в состав списка литералов директив .F_FLOATING, не должна предшествовать унарная операция формата с плавающей запятой (^F).

.G_FLOATING - директива хранения данных с плавающей запятой в формате G

Директива .G_FLOATING обрабатывает константы с плавающей запятой и хранит результаты в об'ектном модуле. Директива G_FLOATING генерирует шестидесятичетырехразрядное данное (1 разряд на символ, 11 разрядов на показатель степени, 52 разряда на дробную часть).

Формат .G_FLOATING список констант

Параметр

список константт

- список констант с плавающей запятой. Данные константы не могут содержать каких-либо операций кроме унарного плюса и унарного минуса.

Пример.

```
.G_FLOATING 1.75,1.0000e-6,2.0e4     ;список констант
```

Примечания:

1. Число с плавающей запятой формата G всегда являются округленными. На них не оказывает воздействия задание директивы .ENABLE TRUNCATION.

2. Константам с плавающей запятой, входящим в состав списка литералов директивы .G_FLOATING, не должен предшествовать унарный оператор формата с плавающей запятой (^F).

.H_FLOATING - директива хранения данных с плавающей запятой в формате H

Директива H_FLOATING производит обработку констант с плавающей запятой в формате H и сохраняет результаты в об'ектном модуле. Директива .H_FLOATING генерирует 128-разрядное данное (1 разряд на символ, 15 разрядов на показатель степени, 112 разрядов на дробную часть).

Формат

.H_FLOATING список констант

Параметр

список константТ

- список констант с плавающей запятой. Данные константы не могут содержать каких-либо операций кроме унарного плюса и унарного минуса.

Пример.

```
.H_FLOATING 4814,3.0E16,2.00000E-4     ;список констант
```

Примечания:

1. Числа с плавающей запятой формата H всегда являются округленными. На них не оказывает никакого воздействия задание директивы .ENABLE TRUNCATION.

2. Константа с плавающей запятой, входящим в состав списка литералов директивы .H_FLOATING, не должен предшествовать унарный оператор формата с плавающей запятой (^F).

.LONG - директива хранения данных в виде длинных слов

Директива .LONG генерирует в об'ектном модуле последовательные длинные слова данных (4 байта).

Формат

.LONG список выражений

Параметр

список выражений

- одно из нескольких выражений, разделенных запятыми. За каждым выражением может следовать необязательный коэффициент повторения, ограниченный квадратными скобками.

Выражение, за которым следует коэффициент повторения имеет следующий формат:

Выражение1 [выражение2]

где выражение1 - выражение, указывающее значение, предназначенное для запоминания;

[выражение2] - коэффициент повторений, определяющий количество повторений хранимого значения. Данное выражение не должно содержать каких - либо неопределенных сим-

волов и должно быть обязательной частью формата.

Пример.

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

Примечание. Каждое выражение в списке должно иметь значение, которое может быть представлено 32-я разрядами.

.OCTA - директива хранения данных в виде октаслов

Директива .OCTA генерирует 128 разрядов (16 байт) двоичных данных.

Формат

.OCTA литерал

.OCTA символ

Параметры

литерал

- любая постоянная величина. Этой величине могут предшествовать операторы ^O,^B,^X или ^D для указания, соответственно, восьмеричного, двоичного, шестнадцатеричного или десятичного основания счисления. Кроме того, данной величине может предшествовать оператор ^A, специфицирующий текст в коде КОИ-8. В качестве основания по умолчанию принимается

десятичное основание счисления.

Символ.

- символ, определенный в каком-либо месте программы. Этот символ представляется в виде 32-разрядного значения с распространением знака, которое хранится в октаслове.

Пример.

```
.OCTA   ^A/ARG=3.1415926/       каждый символ КОИ-8
                                хранится в байте
.OCTA   ^B101010101010          ; двоичное число хранится
                                в октаслове
.OCTA   ^X7FFFFF                шестнадцатеричное число
                                хранится в октаслове
```

Примечание. Директива .OCTA похожа на директиву .QUAD и отлична от остальных директив хранения данных (.BYTE, .WORD, .LONG) тем, что воспринимает только одно значение и не обрабатывает выражения.

.PACKED - директива хранения в упакованном формате данных в виде десятичных строк

Директива .PACKED генерирует упакованные десятичные данные, где две цифры занимают один байт. Представление десятичных чисел в упакованном формате в виде десятичных строк полезно при вычислениях, в которых требуется высокая точность. Данные этого типа обрабатываются инструкциями, выполняющими операции над десятичными числами в упакованном формате.

Формат

.PACKED десятичное число [,символ]

Параметры

десятичное число

- десятичное число с количеством цифр от 0 до 31 с необязательным знаком. Каждая цифра может иметь значение в диапазоне от 0 до 9.

[,символ]

- необязательный символ, которому присваивается значение, равное количеству десятичных цифр в строке, при этом знак не считается в качестве цифры.

Пример.

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

.QUAD - директива хранения данных в виде квадрослов

Директива .QUAD генерирует двоичное данное размером в 64 разряда (8 байт).

Формат

.QUAD литерал

.QUAD символ

Параметры

символ

- символ, определенный в произвольном месте программы.

перед запоминанием в квадрослове значение данного символа представляется в виде 32-разрядной величины с распространенным знаком.

Литерал

- любое постоянное значение. Данному значению может предшествовать оператор ^O,^B,^X,^D, который указывает на, соответственно, восьмеричное, двоичное, шестнадцатеричное и десятичное основание счисления. Кроме того, данному значению может предшествовать оператор ^a, указывающая на текст в коде КОИ-8.

Пример.

```
.QUAD   ^A!NOVEMBER!          данное занимает 8 байтов
.QUAD   ^XA9FFFF              шестнадцатеричное число
                              хранится в квадрослове
.QUAD   ^B11111000001         двоичное число хранится
                              в квадрослове
.QUAD   ^O44444               восьмеричное число
                              хранится в квадрослове
.QUAD   APRIL                 APRIL имеет 32 разрядное
                            ; значение с
                              распространением нуля
```

Примечание. Директива .QUAD похожа на директиву .OCTA и отличается от других директив хранения данных (.BYTE, .WORD, .LONG) тем, что она не обрабатывает выражения и воспринимает только одно значение. Эта директива не воспринимает список значений.

.SIGNED_BYTE - директива хранения данных со знаком в виде байтов

Директива .SIGNED_BYTE эквивалентна директиве .BYTE за исключением того, что она особо указывает на наличие знака у данных. Компоновщик использует данную информацию для проверки на условие переполнения.

Формат

.SIGNED_BYTE список выражений

Параметр

список выражений

- выражение или список выражений, разделенных запятыми. За каждым из этих выражений может следовать коэффициент повторения, заключенный в квадратные скобки ([ ]).

Выражение, за которым следует коэффициент повторений имеет следующий формат:

Выражение1[выражение2]

где выражение1 - выражение, указывающее на значение, которое должно быть запомнено. Это значение должно лежать в диапазоне от -128 до +127

[выражение2] - коэффициент повторений, указывающий, сколько раз должно быть повторено специфицированное значение. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным. Квадратные скобки являются обя-

зательным элементом формата.

Пример.

```
.SIGNED_BYTE    ALPHA-DELTA         данные
.SIGNED_BYTE    BETA[2*6]           хранятся
.SIGNED_BYTE    SIN,COS[4]          в виде байтов
```

Примечание. Использование директивы .SIGNED_BYTE позволяет компоновщику выявлять условия переполнения, когда значение выражения находится в диапазоне от 128 до 255. Значения без знака из указанного диапазона могут храниться в байтах, а значения со знаком - нет.

.SIGNED_WORD - директива хранения данных со знаком в виде слов

Директива .SIGNED_WORD эквивалентна директиве .WORD, за исключением того, что она указывает на наличие знака у данных, хранящихся в об'ектном модуле. Компоновщик использует данную информацию для проверки наличия условий переполнения. Директива .SIGNED_WORD оказывается полезной после инструкции выбора (CASE), при этом гарантируется, что смещение помещается в слове.

Формат

.SIGNED_WORD список выражений

Параметр

список выражений

- выражение или список выражений, разделенных запятыми. За каждым из выражений может следовать необязательный коэффициент повторений, заключенный в квад-

ратные скобки.

Выражение, за которым следует коэффициент повторений, имеет следующий формат:

Выражение1[выражение2]

где выражение1 - выражение, указывающее значение, которое должно быть запомнено. Данное значение должно находится в диапазоне от -32768 до +32767;

[выражение2] - коэффициент повторений, указывающий, сколько раз будет повторено специфицированное значение. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным.

Пример.

```
.SIGNED_WORD    LOC2-LOC1          данные хранятся
.SIGNED_WORD    SORT[20]           в виде слов
```

Примечание. Использование директивы .SIGNED_WORD позволяет компоновщику выявлять условия переполнения, когда значение выражения лежит в диапАзоне от 32768 до 65535. Значения без знака из этого диапАзона могут храниться в словах, а значения со знаком - нет.

.WORD - директива хранения данных в виде слов

Директива .WORD генерирует последовательные слова (2 байта) данных в об'ектном модуле.

Формат

.WORD список выражений

Параметр

список выражений

- одно или несколько выражений, разделенных запятыми. За каждым из выражений может следовать необязательный коэффициент повторения, заключенный в квадратные скобки.

Выражение, за которым следует коэффициент повторения, имеет следующий формат:

Выражение1[выражение2]

где выражение1 - выражение, указывающее на значение, которое должно быть запомнено;

[выражение2] - выражение, определяющее число раз, которое указанное значение будет запоминаться. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным.
Квадратные скобки являются обязательным элементом формата.

Пример.

.WORD ^X3FF,SORT[3],ASIA,25

Примечания:

1. Выражение обрабатывается сначала как длинное слово, а затем усекается до слова. Значение выражения должно лежать в диапАзоне значений от -32768 до +32767 для данных со знаком и в диапазоне от 0 до 65535 для данных без знака. Если два старших байта длинного слова имеют значение,

отличное от 0 и ^XFFFF, выводится сообщение об ошибке.

2. Директива .SIGNED_WORD эквивалентна директиве .WORD, за исключением того, что в случае использования этой директивы выводится диагностическое сообщение, если значение выражения лежит в диапазоне от 32768 до 65535.

5.5. Директивы управления счетчиком адресов

.ALIGN - директива настройки счетчика адресов

Директива .ALIGN позволяет настроить счетчик инструкций на определенную границу, указанную либо целым числом, либо ключевым словом.

Форматы

.ALIGN целое[,выражение]

.ALIGN ключ[,выражение]

Параметры

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 настраивается на адрес, следующий за текущим адресом и кратный соответствующему значению из перечисленных в табл. 18.

Таблица 18

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

Выражение

- указывает значение, которым заполняется каждый байт. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным.

Пример.

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

Примечания:

1. Настройка адреса на определенную границу с помощью директивы .ALIGN имеет более низкий приоритет по сравнению с настройкой программной секции, в которой производится попытка данной настройки. Например, если действует настройка программной секции (BYTE) по умолчанию, а директива .ALIGN издана с аргументом WORD или с другим аргументом,

большим по значению, то выдается сообщение об ошибке.

2. Если указано необязательное выражение, то байты, которые пропускает счетчик адресов программы (если такие есть), заполняются значением этого выражения.

3. Хотя большинство инструкций не требует какой-либо настройки, кроме настройки адреса на границу байта, при настройках, описанных в табл. 19, повышается скорость выполнения.

Таблица 19

Длина данных	Настройка
Слово	На слово
Длинное слово	На длинное слово
Квадрослово	На квадрослово

.BLKX - директивы резервирования блоков памяти

В языке макроассемблер имеется десять директив резервирования блоков памяти:

1) .BLKA-резервирует память для адресов (длинные слова);
2) .BLKB-резервирует память для данных в байтах;
3) .BLKD-резервирует память для данных двойной точности с плавающей запятой (квадрослова);
4) .BLKF-резервирует память для данных одинарной точности с плавающей запятой (длинные слова);
5) .BLKG-резервирует память для данных формата G (квадрослова);

6) .BLKH-резервирует память для данных формата H (октаслова);

7). .BLKL-резервирует память для данных в длинных словах;

8). .BLKO-резервирует память для данных в октасловах;

9) .BLKQ-резервирует память для данных в квадрословах;

10).BLKW-резервирует память для данных в словах.

Каждая из директив резервирует память для данных различных типов. Значение выражения определяет количество элементов данных, для которых программа макроассемблер резервирует память. Например, директива .BLKL 4 резервирует память для данных в об'еме четырех длинных слов, а директива .BLKB 2 резервирует память для данных в об'еме двух байтов.

Общее количество зарезервированных байтов равно длине элементов данных данного типа, умноженной на значение выражения. Количество зарезервированных байтов в соответствии с директивой определяется следующим образом:

.BLKB	значение выражения
.BLKW	2 x (значение выражения)
.BLKA	
.BLKF	4 x (значение выражения)
.BLKL	
.BLKD	8 x (значение выражения)
.BLKG	
.BLKQ	
.BLKH	16 x (значение выражения)
.BLKO	

Форматы

.BLKA выражение

.BLKB выражение

.BLKD выражение

.BLKF выражение

.BLKG выражение

.BLKH выражение

.BLKL выражение

.BLKO выражение

.BLKQ выражение

.BLKW выражение

Параметр

выражение

- выражение, определяющее об'ем памяти, подлежащий резервированию. Все символы в этом выражении должны быть определены, а само выражение должно быть абсолютным. Если выражение опущено, принимается значение по умолчанию равное 1.

Пример.

```
.BLKB  25          облать для 25 байтов
.BLKW  10          область для 10 слов (20 байтов)
.BLKL  10          область для 10 длинных слов (40 байтов)
.BLKO  2           область для 2 октаслов (32 байта)
.BLKF  <3*2>       область для 6 значений в формате
                   ; с плавающей запятой (24 байта)
```

.END - директива завершения трансляции

Директивой .END завершается исходная программа. В текущем исходном файле или в любых дополнительных файлах, указанных в командной строке для программы макроассемблер, за точкой программы, в которой задана директива .END, не должно появляться никакого дополнительного текста. Если такой дополнительный текст существует, то программа макроассемблер его игнорирует. Этот дополнительный текст не появится ни в файле листинга, ни в файле об'ектного модуля.

Формат

.END [имя]

Параметр

имя

- адрес (называемый адресом передачи управления), с которого начинается выполнение программы.

Пример.

```
.ENTRY          BEGIN,0          ; маска входа
                                   основная программа


.END            BEGIN
```

Примечания:

1. Адрес передачи управления должен находиться в такой программной секции, которая имеет атрибут EXE.

2. Когда компонуется выполняемый образ, состоящий из нескольких об'ектных модулей, только один об'ектный модуль

должен завершаться директивой .END, в которой указан адрес передачи управления. Все остальные об'ектные модули должны завершаться директивой .END, в которой не указывается адрес передачи управления. Если выполняемый образ либо не содержит ни одного адреса передачи управления, либо содержит более одного адреса передачи управления, то компоновщик выдает сообщение об ошибке.

3. Если в момент указания директивы .END в исходной программе содержится незавершенный блок условной трансляции, то выдается сообщение об ошибке.

.EVEN - директива установки четности текущего счетчика адресов программы

Задание директивы .EVEN обеспечивает установку четности значения текущего счетчика адресов программы. Это выполняется путем добавления к значению счетчика адресов программы единицы, если это значение нечетно. Если значение текущего счетчика адресов является уже четным, то никаких действий не выполняется.

Формат

```
.EVEN
```

.ODD - директива установки нечетности текущего счетчика адресов прграммы

Задание директивы .ODD обеспечивает установку нечетности значения текущего счетчика адресов программы. Это производится путем добавления единицы к текущему значению счетчика адресов программы, если это значение четно. Если значение счетчика адресов программы уже является нечетным, то никаких действий не выполняется.

Формат

.ODD

5.6. Директивы секционирования программ

.PSECT - директива секционирования программы

С помощью директивы .PSECT осуществляется определение программной секции и ее атрибутов, а также происходит ссылка к программной секции, один раз уже определенной. Разбиение программы на программные секции может иметь следующие цели:

- разработка модульных программ;
- отделение инструкций от данных;
- предоставление возможности различным модулям иметь доступ к одним и тем же данным;
- защита данных, предназначенных только для чтения, и инструкций от модификации;
- идентификация секций об'ектного модуля для отладчи-

ка;

- управление порядком, в котором программные секции запоминаются в виртуальной памяти.

Программа макроассемблер автоматически определяет две программные секции:

1) абсолютную программную секцию;

2) неименованную (именем является пробел) программную секцию.

Любые определения символов, появляющиеся в программе до инструкций, данных или директивы .PSECT, помещаются в абсолютную программную секцию. Все инструкции и данные, которые появляются до того, как определена первая поименованная программная секция, размещаются в неименованной программной секции. Любая директива .PSECT, не содержащая спецификации имени программной секции, относится к неименованной программной секции.

Пользователь может определить максимум 254 поименованных программных секций.

Когда программа макроассемблер обрабатывает директиву .PSECT, в которой указана спецификация нового имени программной секции, она создает новую программную секцию и запоминает имя, атрибуты и группировку данной программной секции. Программа макроассемблер включает все данные и инструкции, которые следуют за директивой .PSECT, в эту программную секцию, до тех пор пока не встретится другая директива .PSECT. Все программные секции, являющиеся перемещаемыми, начинаются со значения счетчика адресов програм-

мы равного нулю.

Если программа макроассемблер встречает директиву .PSECT со спецификацией имени ранее определенной программной секции, она 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 перечисленный 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. На самом деле, программа макроассемблер не проверяет, действительно ли содержимое программной секции включает перечисленные атрибуты. Однако и программа макроассемблер и программа компоновщик проверяют, все ли программные секции с одинаковыми именами имеют в точности одни и те же атрибуты. Как программа макроассемблер, так и программа компоновщик выводят сообщения об ошибке, если атрибуты программных секций не согласуются.

Имена программных секций являются независимыми символами по отношению к локальным символам, глобальным символам и именам макрокоманд. Таким образом, одно и тоже символичес-

кое имя может употребляться для обозначения локального символа или имени макрокоманды.

Формат

.PSECT

.PSECT [имя,[список аргументов]]

Параметры

имя

- имя программной секции. В состав указанного имени может входить до 31 символа, любой алфавитно-цифровой символ, символ подчеркивания (_), символ денежной единицы (¤) и точка (.). При этом первый символ не может быть цифрой.

Список аргументов

- список, содержащий атрибуты программной секции и информацию о расположении программной секции. В табл. 20 перечислены атрибуты программной секции и приведены их функции. В табл. 21 приведены сведения об атрибутах программной секции (и их взаимоисключающих противоположностях), принятых по умолчанию. Программные секции настраиваются на определенную границу, если указано целое число в диапазоне от 0 до 9 или одно из пяти ключевых слов. Если указано целое число, то программная секция компонуется таким образом, чтобы начинаться со следующего виртуального адреса, равного числу, которое кратно числу два в степени данного целого числа. Если указано ключевое слово, то программная секция компонуется таким обра-

зом, чтобы начинаться со следующего виртуального адреса, кратного значениям, перечисленным в табл. 22. Ключевым словом по умолчанию является BYTE.

Пример.

```
.PSECT  INSTR,EXE,LONG      ; секция INSTR содержит только
                            ; инструкции
.PSECT  DATA,NOEXE,WORD       секция DATA содержит только
                            ; данные
```

Таблица 20

Атрибуты программной секции

Имя атрибута	Функция
ABS	Признак абсолютной программной секции (неперемещаемой). Компоновщик присваивает данной программной секции абсолютный адрес. Программная секция с таким атрибутом может содержать лишь определения символов (обычно определения символических смещений относительно структур данных, которые используются в транслируемых программах). Абсолютная программная секция не вносит в образ никакого двоичного кода, так что ее требование к размещению в байтах, передаваемое компоновщику, имеет значение, разное нулю. размер определяемой структуры данных равен размеру абсолютной программной секции, напечатанному в

Продолжение табл. 20

Имя атрибута	Функция
	"краткой сводке по программной секции" в конце листинга. 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) сливаются в одну программную секцию. Содержимое этих программных секций сливается в порядке, в котором их обрабатывает компоновщик размещенное пространство виртуальных адресов равно сумме индивидуальных запросов на размещение
EXE	Признак выполняемости. Данная программная секция содержит инструкции. Атрибут EXE является средством отделения инстукций 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, даже когда программные секции расположены в различных группах. Данный атрибут указывается для программных секций с фортрановскими блоками COMMON. Сравнит

Продолжение табл. 20

Имя атрибута	Функция
	данный атрибут с его противоположностью, с атрибуто LCL
LCL	Признак локальной программной секции. Программна секция ограничена ее группой. Сравните данный ат рибут с его противоположностью, с атрибутом GL
LIB	Признак библиотечного сегмента. Резервируется дл будущего использования
NOEXE	Признак невыполняемости. Программная секция содержи только данные и не содержит инструкций
NOPIC	Признак позиционной зависимости содержимого програм мной секции. Данная программная секция предназначен для фиксированного размещения в виртуальной памят (когда это происходит в разделяемом образе)
NORD	Признак нечитаемости. Резервируется для дальнейшег использования
NOSHR	Признак запрета разделяемости. Программная секци резервируется для частого использования инициирующи процессом во время выполнения
NOWRT	Признак запрета записи. Содержимое программной сек ции не может быть изменено во время выполнения
OVR	Признак перекрытия. Программные секции с одинаковы

Продолжение табл. 20

Имя атрибута	Функция
	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. Сравните данный атрибут с его противоположностью, с атрибутом CON
PIC	Признак позиционной независимости содержимого программной секции. Данная программная секция допускает перемещение, то есть ей может быть назначена другая область памяти (когда это происходит в разделяемом образе)
RD	Признак читаемости. Резервируется для дальнейшего использования
REL	Признак перемещаемой программной секции. Компоновщик назначает данной программной секции относительный базовый адрес. Программная секция может содержать инструкции или данные. Сравните с атрибутом ABS
SHR	Признак разделяемости программной секции. Программная секция допускает разделение между несколькими процессами во время выполнения. Данный атрибут наз

Продолжение табл. 20

Имя атрибута	Функция
	начается программным секциям, которые могут быт скомпонованы в разделяемый образ
USR	Признак пользовательского сегмента. Резервируетс для дальнейшего использования
VEC	Признак содержания вектора. Программная секция со держит вектор изменения режима, который указывает н привилегированный разделяемый образ. Атрибут SH должен использоваться с атрибутом VEC
WRT	Признак записи. Содержимое программной секции в время выполнения может быть изменено

Таблица 21

Атрибуты программной секции по умолчанию

Атрибут по умолчанию	Противоположный атрибут
CON	OVR
EXE	NOEXE
LCL	GLB
NOPIC	PIC
NOSHR	SHR
RD	NORD

Продолжение табл. 21

Атрибут по умолчанию	Противоположный атрибут
REL	ABS
WRT	NOWRT
NOVEC	VEC

Таблица 22

Ключевое слово	Размер (в байтах)
BYTE	2^0 = 1
WORD	2^0 = 2
LONG	2^0 = 4
QUAD	2^0 = 8
PAGE	2^0 = 512

Пример.

```
.PSECT      INSTR,EXE,LONG          секция INSTR содержит
                                    только инструкции
.PSECT      DATA,NOEXE,WORD         секция DATA содержит
                                  ; только данные
```

Примечания:

1. Директива .ALIGN не может указать размер настройки адреса больший, чем размер текущей программной секции, следовательно в директиве .PSECT следует указать наибольший размер настройки адреса, необходимый в данной программной секции. Для повышения эффективности выполнения для всех

программных секций, содержащих данные в виде длинных слов, рекомендуется и настройка на длинное слово или на больший размер.

2. В табл. 23 приведены атрибуты, принятые по умолчанию, в абсолютных и неименованных программных секциях. Следует заметить, что имена данных программных секций содержат точки и пробелы.

Таблица 23

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

.RESTORE_PSECT - директива восстановления программной секции

Директива .RESTORE_PSECT восстанавливает программную секцию из вершины стека контекста программной секции, если она была сохранена с помощью директивы SAVE_PSECT. Этот стек является внутренним стеком программы макроассемблер. 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 при сохранении программной секции.

Формат

.RESTORE_PSECT

Пример.

```
        .MACRO  DEF_SYMB            определение символов
                                    и областей
        .SAVE_PSECT                 ; сохранение текущей
                                    секции
        .PSECT MACR1,NOWRT,WORD     определить секцию
                                    MACR1
A=17
B=13
C=11
        .PSECT MACR2,NOEXE,WORD     определить секцию
                                    MACR2
        .BLKW  200
        .BLKB  100
        .RESTORE_PSECT              восстановить
                                    программную
                                    секцию
        .ENDM
```

Директивы .RESTORE_PSECT и SAVE_PSECT особенно полезны в макрокомандах, в которых вводится определение программных секций. В приведенном 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 программной секции каждый раз, когда данная макрокоманда вызывается, то данная программная секция изменилась бы.

Примечание. Альтернативной формой директивы .RESTORE_PSECT является директива .RESTORE.

.SAVE_PSECT - директива сохранения программной секции

Директива .SAVE_RSECT сохраняет контекст текущей программной секции в вершине стека контекста программной секции, когда осуществляется выход из контекста текущей программной секции. Данный стек является внутренним стеком программы макроассемблер. Стек контекста программной секции 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 заполненым, когда встречается директива .SAVE_PSECT, возникает ошибка.

Директивы .SAVE_PSECt и .RESToRE_PSECT оказываются особенно полезными в макрокомандах, в которых вводятся определения программных секций. Другой пример использования директивы .SAVE_PSECT приведен в описании директивы

.RESTORE_PSECT.

Формат

.SAVE_PSECT [LOCAL_BLOCK]

Параметр

[LOCAL_BLOCK]

- необязательное ключевое слово, которое указывает на то, что вместе с контекстом программной секции должен быть сохранен и текущий блок локальных меток.

Пример.

```
MACRO:
        .MACRO ERROR_MES,NEW_TEXT       Формирование таблицы
                                        указателей и
                                        сообщений об ошибках
        .IIF    NOT_DEFINED  MES_NUMBER  MES_NUMBER=0
        .SAVE_PSECT     LOCAL_BLOCK     Сохранить блок
                                        локальных меток
        .PSECT  TEXT                    Таблица сообщений
                                        об ошибках
MES::    ASCIZ  /NEW_TEXT/
        .PSECT  POINTER                 Таблица указателей
        .ADRESS MES
        .RESTORE_PSECT                  Восстановить блок
                                        локальных меток
        PUSHL   #MES_NUMBER
                #1,PRINT_MES            Печать сообщения об
                                        ошибке
MES_NUMBER=MES_NUMBER+1
```

```
        .ENDM   ERROR_MES
        INCL    R4
        CLRL    R2
        BLBC    R1,4¤
        ERROR_MES <SYMBOL NOT FOUND>   Ввести в таблицу
                                       сообщение "символ не найден"
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 локальная метка не определена в блоке локальных меток, в котором к ней производится ссылка, то выводится сообщение об ошибке следующего вида:

%MACRO-E-UNDEFSYM, UNDEFINED SUMBOL

Примечание. Альтернативной формой директивы .SAVE_PSECT является директива .SAVE.

5.7. Директивы управления символами

.DEBUG - директива об'явления символа отладки

Директива .DEBUG определяет список символов, используемых символическим отладчиком. Во время процесса интерактивной отладки эти символы могут использоваться для ссылки к ячейкам памяти или для проверки значений, присвоенных этим символам.

Формат

.DEBUG список символов

Параметр

список символов

- список разрешенных символических имен, разделенных запятыми.

Пример.

```
.DEBUG  PTR1,PTR2,DELTA      эти символы известны
                             отладчику
```

Примечание. Программа макроассемблер добавляет символы, указанные в списке символов, в таблицу символов данного об'ектного модуля. В директиве .DEBUG пользователю не нужно указывать глобальные символы, поскольку глобальные символы автоматически помещаются в таблицу символов об'ектного модуля.

.EXTERNAL - директива об'явления внешнего символа

Задание директивы .EXTERNAL указывает на то, что определенные символы являются внешними или, другими словами, данные символы определены в другом об'ектном модуле и не могут быть определены до этапа компоновки.

Формат

.EXTERNAL список символов

Параметр

список символов

- список допустимых символов, разделенных запятыми.

Пример.

.EXTERNAL AB,CD,EF внешние символы

Примечания:

1. Если директивой .ENABLE разрешен аргумент GLOBAL, все неразрешенные ссылки будут помечены как глобальные и внешние. Таким образом, если разрешен аргумент GLOBAL, у программиста нет необходимости задавать директиву .EXTERNAL. Однако если аргумент GLOBAL запрещен, то программист обязан явно указать директиву .EXTERNAL, чтобы об'явить некоторые символы, которые определяются как внешние, но к которым ссылки выполняются в текущем модуле.

2. Если директивой .DISABLE аргумент GLOBAL запрещен, и программа макроассемблер обнаруживает символы, которые не определены в текущем модуле и не перечислены в директиве .EXTERNAL, то выдается сообщение об ошибке.

3. Альтернативной формой директивы .EXTERNAL является директива .EXTRN.

.GLOBAL - директива об'явления глобального символа

Директива .GLOBAL показывает, что определенные в директиве символы либо определены в текущем модуле, либо определены как внешние в других модулях.

Формат

.GLOBAL список символов

Параметр

список символов

- список разрешенных символических имен, разделенных запятыми.

Пример.

```
.GLOBAL SIN,COS          Эти символы известны
                         всем модулям
.GLOBAL TAN              ; Участвующим в компоновке
```

Примечания:

1. Директива .GLOBAL введена в язык макроассемблер для обеспечения совместимости с языком макроассемблер для ЭВМ СМ 4. Рекомендуется вводить глобальные определения с помощью двойных символов двоеточия и равенства, а внешние ссылки указывать в директиве .EXTERNAL.

2. АЛЬТЕРНАТИВНОЙ ФОРМОЙ ДИРЕКТИВЫ .GLOBAL ЯВЛЯЕТСЯ директива .GLOBL.

.WEAK - директива подавления поиска символа

Директива .WEAK указывает символы, которые являются либо внешне определенными в другом модуле, либо определенными глобально в текущем модуле. Директива .WEAK подавляет поиск символа в любой библиотеке об'ектных модулей.

Если в директиве .WEAK указан символ, не определенный в текущем модуле, этот символ определяется как внешний. Если компоновщик находит определение этого символа в другом модуле, он разрешает символ с использованием этого определения. Если компоновщик не находит внешнего определения, символ приобретает значение нуль и компоновщик при этом не сообщает об ошибке. Компоновщик не производит поиск указанного символа в библиотеке, однако если модуль, извлеченный из библиотеки по другим причинам, содержит определение это-

го символа, компоновщик разрешает символ с использованием этого определения.

Если в директиве .WEAK указан символ, определенный в текущем модуле, то этот символ рассматривается как глобальный. Однако если данный модуль включен в библиотеку об'ектных модулей, то этот символ не включается в таблицу символов библиотеки. Следовательно, просмотр библиотеки во время компоновки с целью разрешения данного символа не влечет за собой включение данного модуля в образ.

Формат

.WEAK список символов

Параметр

список символов

- список разрешенных символических имен, разделенных запятыми.

Пример.

```
.WEAK       JUNE,JULE,MAY_30
```

5.8. Директивы определения входной точки программы

.ENTRY - директива определения входной точки

Директива .ENTRY определяет символическое имя входной точки и маску сохранения регистров (2 байта). Указанный символ определяется как глобальный символ, значение которого равно значению счетчика инструкций PC в точке задания директивы .ENTRY. Входная точка может быть использована в

качестве адреса передачи управления программы. Маска сохранения регистров используется для указания, какие регистры были сохранены до вызова данной процедуры. Эти сохраненные регистры автоматически восстанавливаются, когда данная процедура возвращает управление в вызывающую программу. Описание инструкций вызова процедур приведено в документе [2].

Формат

.ENTRY символ,выражение

Параметр

символ

- символическое имя входной точки

выражение

- маска сохранения регистра для входной точки. Данное выражение должно быть абсолютным выражением и не должно содержать никаких неопределенных символов.

Пример.

```
.ENTRY SYM,^M<R2,R4,R7>     процедура начинается
                            в данной точке
                            R2,R4,R7 сохранены
                            с помощью инструкций
                            CALL, RET
```

Примечания:

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

2. Если указанное выражение имеет установленные разряды 0,1,12 и 13, возникает ошибка трансляции. Данные разряды соответствуют регистрам R0, R1, AP и FP и резервируются

операционной системой МОС ВП.

3. Рекомендуется использовать директиву .ENTRY для определения всех вызываемых точек входа, включая адрес передачи управления программы. Хотя следующая конструкция также определяет входную точку, но применять ее не рекомендуется:

SYMBOL:: .WORD зыражение

хотя процедура, начинающаяся с данной конструкции, может быть вызвана, входная маска не проверяется для любых недопустимых регистров и указанный символ (SYMBOL) не может быть использован с директивой .MASK.

4. Директиву .ENTRY следует использовать для процедур, которые будут вызываться инструкцией CALLS или CALLG. Программа, в которую входят инструкции BSB или JSB, не должна использовать наличие маски сохранения регистров. Такие программы должны начинаться со следующей конструкции:

SYMBOL:: первая инструкция

первая инструкция непосредственно следует за символом (SYMBOL)

.MASK - директива маскирования

По директиве .MASK происходит резервирование слова для маски сохранения регистров предназначенной для вектора передачи управления. Более полная информация по данной директиве и примеры ее использования приведены в описании директивы .TRANSFER.

Формат

.MASK символ [,выражение]

Параметры

символ

- символ, определенный в директиве .ENTRY.

[,выражение]

- маска сохранения регистров.

Примечания:

1. Если в директиве .MASK не указан параметр "выражение", то программа макроассемблер указывает компоновщику скопировать в слово, зарезервированное по директиве .MASK, маску сохранения регистров, указанную в директиве .ENTRY.

2. Если в директиве .MASK указан параметр "выражение", то программа макроассемблер указывает программе компоновщику об'единить в слове, зарезервированном по директиве .MASK, данное выражение и маску сохранения регистров, указанную в директиве .ENTRY. Об'единение маски во входной точке программы и значения указанного выражения компоновщик выполняет по функции логического "ИЛИ". Следовательно, в об'единенную маску сохранения регистров будут включены как регистры указываемые в директиве .ENTRY, так и регистры, указываемые в директиве .MASK.

.TRANSFER - директива передачи управления

Директива .TRANSFER переопределяет глобальный символ для использования в разделяемом образе. После компоновки разделяемого образа компоновщик переопределяет данный сим-

вэл как значение счетчика адресов программы в точке директивы .TRANSFER.

Программу, по возможности, не следует перекомпоновывать, когда изменяется разделяемый образ, к которому эта программа прикомпонована. Однако это достижимо только при выполнении следующих двух условий:

1) не изменился общий об'ем разделяемого образа;

2) точки входа разделяемого образа не изменили своих адресов при изменении кода разделяемого образа и при перекомпоновке образа.

Чтобы избежать изменение об'ема разделяемого образа, при первичном создании этого образа резервируется избыточное пространство. Для уверенности в том, что входные точки не изменились, стройте об'ектный модуль, содержащий вектор передачи управления для каждой входной точки, и не изменяйте порядок этих векторов передачи управления. Компонуйте этот об'ектный модуль в начало разделяемого образа и тогда адреса останутся фиксированными, даже если исходный код программы изменится. После каждой директивы .TRANSFER должна появляться маска сохранения регистров (только для процедур) и инструкция перехода к первой инструкции программы.

Директива .TRANSFER не требует никакой памяти и не генерирует двоичный код. Она только формирует инструкции для компоновщика по определению символа при создании разделяемого образа.

Директива .TRANSFER может быть использована в процедурах, вход в которые осуществляется по инструкциям CALLS и

CALLG. В этом случае директива .TRANSFER используется совместно с директивами .ENTRY и .MASK. Переход к действующей программе должен осуществляться по адресу точки входа плюс два. Добавление числа два к этому адресу нужно для того, чтобы пропустить два байта, относящиеся к маске сохранения регистров.

Формат

.TRANSFER символ

Параметр

символ

- глобальный символ, который является входной точкой процедуры или программы.

Пример.

```
.TRANSFER ROUT_TEST
.MASK     ROUT_TEST,^M<R4,R5>   копирование входной
                                маски по новому адресу
BRW       ROUT_TEST+2           переход к ROUT_TEST
                                минус 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 входной адрес, указанный директивой .TRANSFER. Если программа входит в состав разделяемого образа и вызывается, регистры 2,3,4 и 5 сохраняются.

5.9. Директивы и поддирективы трансляции

.ENDC - директива конца блока условной трансляции

Директива .ENDC завершает блок условной трансляции, начатый диретивой .IF.

Формат

```
.ENDC
```

.IF - директива блока условной трансляции

Блоком условной трансляции называется последовательность исходных предложений, трансляция которых осуществляется только при соблюдении некоторого условия. Блок условной трансляции начинается директивой .IF, а заканчивается директивой .ENDC. Для каждой директивы .IF должна существовать соответствующая директива .ENDC. Директива .IF содержит правило проверки условия и один или два аргумента. Указанное правило проверки условия распространяется и на данный аргумент (аргументы). Если проверка условия проходит успешно, то все предложения языка макроассемблер, заключенные между директивами .IF и .ENDC, транслируются. Если же

проверка показывает, что условие не выполняется, то указанные предложения не транслируются. Исключения из данного положения возникают, если используются поддирективы условной трансляции.

Блоки условной трансляции могут быть вложенными друг в друга, то есть в состав одного блока условной трансляции может входить другой блок условной трансляции. В этом случае вложенный блок условной трансляции будет транслироваться только в том случае, если выполняются условия для внешнего блока и для внутреннего одновременно.

Формат

.IF условие аргумент(ы)

блок

.ENDC

Параметры

условие

- специальное условие, которое должно быть выполнено, чтобы данный блок условной трансляции был включен в трансляцию. В табл. 24 приведены условия, которые могут быть проверены директивами условной трансляции. Условие должно быть отделено от аргумента

(аргументов) запятой, пробелом или символом табуляции.

аргумент(ы)

- аргумент (аргументы) или выражение (выражения) для указанного правила проверки условия. Если аргументом является выражение, то оно не должно содержать неопределенных символов и должно быть абсолютным.

блок

- блок предложений исходного текста программы, который включается в трансляцию при выполнении условия, указанного в директиве.

Таблица 24

Условия, допустимые в директивах условной трансляции

Условие проверки		Дополнительное условие проверки		Тип аргумента	Количество аргументов	Условие, при котором блок транслируется
Формат						
Полный	Краткий	Полный	Краткий			
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

Продолжение табл. 24

Условие проверки		Дополнительное условие проверки		Тип аргумента	Количество аргументов	Условие, при котором блок транслируется
Формат						
Полный	Краткий	Полный	Краткий			
DEFINED	DF	NOT_DEFINED	NDF	Жение символ		(или >=0) символ определен (или не определен)
1)		1)				
BLANK	B	NOT_BLANK	NB	Макрокоманда	1	Аргумент пропущен (или не пропущен)
		1)				
IDENTICAL	IDN	DIFFERENT	DIF	Макрокоманда	2	Аргументы идентичны (или различны)

1) Условия BLANK, NOT_BLANK, IDENTICAL и DIFFERENT полезны только в определениях макрокоманд. В разделе 6

директивы макрокоманд рассмотрены подробно.

Пример.

```
пример директивы условной трансляции
.IF EQUAL POINT+1    транслировать блок, если POINT+1=0
                     не транслировать блок, если POINT+1=0


.ENDC
 вложенные директивы условной трансляции
.IF      условие, аргумент(ы)
.IF      условие, аргумент(ы)

 .
.ENDC
.ENDC
 директивы условной трансляции определяют, будет ли
 производиться трансляция
.IF DEFINED  SYM1
.IF DEFINED  SYM2

 .
.ENDC
.ENDC
```

В последнем примере не происходит обработки вложенных условных предложений программы, если не выполняется самое

внешнее условие. Поэтому для того, чтобы исходный текст программы был оттранслирован, необходимо, чтобы был определен как символ SYM1, так и символ SYM2.

Примечания:

1. Если директива .ENDC появляется вне блока условной трансляции, то выводится сообщение об ошибке.

2. Программа МАКРОАССЕМБЛЕР допускает глубину вложения до 31 уровня условной трансляции. Если предложение пытается превысить допустимый уровень условной трансляции, то выводится сообщение об ошибке.

3. перед сравнением, аргументы в виде строки из строчных букв преобразуются в строки прописных букв, если эти строки не окружены ограничителями.

4. Программа МАКРОАССЕМБЛЕР выводит сообщение об ошибке, если в директиве .IF указано что-либо из следующего:

- условие проверки, отличное от приведенных в табл. 24;
- недопустимый аргумент;
- пустой аргумент.

5. Директивы .SHOW и .NOSHOW определяют, будет ли включен в файл листинга блок условной трансляции, который не транслировался.

.IF_X - поддирективы блока условной трансляции

Поддирективы - это директивы условной трансляции, которые могут быть использованы только в блоках условной трансляции директивы .IF.

В языке макроассемблер существует три поддирективы блока условной трансляции, которые описаны в табл. 25.

Таблица 25

Поддиректива	Функция
.IF_FALSE	Программа должна включать исходные предложения, следующие за директивой .IF_FALSE и продолжающиеся до следующей поддирективы условной трансляции или до конца блока условной трансляции, если условие проверенное при входе в блок условной трансляции было ложным
.IF_TRUE	Программа должна включать исходные предложения, следующие за директивой .IF_FALSE и продолжающиеся до следующей поддирективы условной трансляции или до конца блока условной трансляции, если условие, проверенное при входе в блок условной трансляции было истинным
.IF_TRUE_FALSE	Программа всегда включает в себя исходные предложения, следующие за директивой и продолжающийся 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 ус-

Продолжение табл. 25

Поддиректива	Функция
	! ловие, проверенное при входе в блок ус-
	! ловной трансляции, истинным или ложным

Подразумеваемым аргументом поддирективы условной трансляции является результат проверки указанного условия при входе в блок условной трансляции. Поддирективы директивы условной транляции не обрабатываются в блоках условной трансляции, если условие внешнего блока не выполняется.

Блок условной трансляции, содержащий поддирективы условной трансляции отличается от блока условной трансляции. Если условие в директиве .IF не выполняется, то внутренний блок (блоки) условной трансляции не транслируются, тогда как поддирективы условной трансляции могут вызывать трансляцию блока.

Форматы

.IF_FALSE

.IF_TRUE

.IF_TRUE_FALSE

Пример 1.

Предположим, что символ SYM определен:

```
.IF DEFINED   SYM        условие выполняется, поскольку
                         символ SYM определен
                         трансляция следующего кода
```

```
.IF_FALSE                условие является ложным,
                         поскольку условие предыдущей
                         директивы .IF было истинным.
                         следующий код не
                         транслируется
  .
.IF_TRUE                 условие проверки истинно.
                         символ SYM определен
                         трансляция следующего кода


.IF_TRUE_FALSE           следующий код транслируется
                         безусловно
                         трансляция остатка блока
                         условной трансляции
.IF_TRUE
                         трансляция остатка блока
                         условной трансляции

.ENDC
```

Пример 2.

Предположим, что символ X определен, а символ Y нет:

```
.IF DEFINED X ; условие является истинным. Символ X
                определен
.IF DEFINED Y   условие является ложным. Символ Y не
                определен
```

```
.IF_FALSE       условие является истинным. Символ Y не
                ; определен
                ; трансляция следующего кода


.IF_TRUE        ; условие является ложным. Символ Y не
                ; определен

                  следующий код не транслируется


.ENDC
.ENDC
```

Пример 3.

Предположим, что символ a определен, а символ в нет:

```
.IF DEFINED  A        ; условие является истинным. Символ
                        a определен
                        следующий код транслируется


.IF_FALSE               условие является ложным. Символ a
                        определен. Следующий код не
                        транслируется

.IF NOT_DEFINED B       вложенная директива условной
                        трансляции не обрабатывается
```

```
.ENDC
.ENDC
```

Пример 4.

Предположим, что символ x не определен, а символ Y определен:

```
.IF DEFINED     X       условие является ложным. Символ X
                        не определен
                        следующий код не транслируется

.IF DEFINED     Y       вложенная директива условной
                      ; трансляции не обрабатывается

.IF_FALSE               вложенная поддиректива условной
                        трансляции не обрабатывается


.IF_TRUE                вложенная поддиректива условной
                        трансляции не обрабатывается


.ENDC
.ENDC
```

Примечания:

1. Если поддиректива условной трансляции поевдяется вне блока условной трансляции, то выводится сообщение об ошибке.

2. Альтернативными формами директив .IF_FALSE,

.IF_TRUE и IF_TRUE_FALSE являются соответственно .IFF, .IFT и .IFTF.

.IIF - директива непосредственной условной трансляции

Директива .IIF представляет собой средство языка макроассемблер 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

.IIF условие [,] аргумент(ы), предложение

Параметры

условие

- одно из допустимых условий, определенных в табл. 24 для блоков условной трансляции. Данное условие должно отделяться от аргумента (аргументов) запятой, пробелом или символом табуляции, однако, если первый аргумент может быть пробелом, то условие должно отделяться от аргумента (аргументов) запятой.

аргумент(ы)

- аргумент, связанный с непосредственной директивой условной трансляции, (см. Табл. 25). Если аргументом является выражение, то оно не должно содержать никаких неопределенных символов и должно быть абсолютным. Аргумент (аргументы) должен отделяться от предложения запятой.

предложение

- предложение, которое должно транслироваться, если условие выполняется.

Пример.

```
.IIF DEFINED     HELENA,        BEQL NAME
```

если символ HELENA определен в исходной программе, данная директива генерирует следующий код:

```
BEQL NAME
```

Примечание. Программа макроассемблер выводит сообщение об ошибке, если в директиве .IIF указано что-либо из следующего:

- условие проверки, отличное от приведенных в табл. 24;
- недопустимый аргумент;
- пустой аргумент.

5.10. Директивы перекрестных ссылок

.CROSS и NOCROSS - директивы управления таблицей перекрестных ссылок

Если в командной строке для программы МАКРОАССЕМБЛЕР указан квалификатор /CROSSE_REFERENCE, то формируется таблица перекрестных ссылок. Директивы .CROSSE и .NOCROSSE действительны только в том случае, если в командной строке для MACRO был указан квалификатор /CROSS_REFERENGE.

По умолчанию таблица перекрестных ссылок включает в себя определения и все ссылки на каждый символ в модуле. Таблица перекрестных ссылок может подавляться для всех символов или для определенного списка символов.

Задание директивы .NOCROSS без списка символов запрещает формирование таблицы ссылок для всех символов. Задание директивы .CROSS снова разрешает формирование таблицы перекрестных ссылок (если директива задается без указания списка символов). Любое определение символа или ссылка на символ, появляющиеся в программе после задания директивы .NOCROSS без списка символов и до задания директивы .CROSS без списка аргументов, исключается из таблицы перекрестных ссылок.

Задание директивы .NOSROSS со списком символов подавляет формирование таблицы перекрестных ссылок для перечисленных символов. Задание директивы .CROSS со списком символов снова разрешает формирование таблицы перекрестных ссылок для перечисленных символов.

Форматы

.CROSS

.CROSS список символов

.NOCROSS

.NOCROSS список символов

Параметр

список символов

- список разрешенных символических имен, разделенных запятыми.

Пример.

```
        .NOCROSS                прекращение формирования
                                таблицы перекрестных ссылок
COPY1:  MOVL    SYM1,SYM2       копирование данных
        .CROSS                  продолжение формирования
                              ; таблицы перекрестных ссылок
```

Определение метки COPY1 и ссылки к символам SYM1 и SYM2 не входят в таблицы перекрестных ссылок.

```
        .NOCROSS SYM1         ; не включать символ SYM1
                                в таблицу перекрестных ссылок
COPY2:  MOVL     SYM1,SYM2      копирование данных
        .CROSS   SYM1         ; продолжение формирования
                                таблицы перекрестных ссылок
                                для символа SYM1
```

Определение символа COPY2 и ссылка к символу SYM2 входят в таблицу перекрестных ссылок, а символ SYM1 не входит в таблицу перекрестных ссылок.

Примечания:

1. Директива .CROSS, заданная без списка символов, не вызывает продолжения формирования таблицы перекрестных ссылок для символов, указанных в списке символов директивы .NOCROSS.

2. Если формирование таблицы перекресных ссылок запрещено для всех символов, задание директивы .CROSS со списком символов не имеет действия до тех пор, пока формирование таблицы перекрестных ссылок не будет продолжено заданием директивы .CROSS без списка символов.

.NOCROSS - директива управления таблицей перекрестных ссылок

Если в командной строке программы макроассемблер указан квалификатор /CROSS_REFERENCE, то выполняется построение таблицы перекрестных ссылок. Директивы .CROSS и .NOCROSS определяют символы, которые включаются в таблицу перекрестных ссылок. Описание директивы .NOCROSS приведено в описании директивы .CROSS.

Форматы

.NOCROSS

.NOCROSS список символов

Параметр

список символов

- список разрешенных символических имен, разделенных запятыми.

5.11. Директивы генерирования инструкций

.OPDEF - директива определения кода операции

По директиве .OPDEF происходит определение кода операции, который включается в таблицу кодов операций, определяемых пользователем. Программа макроассемблер выполняет поиск символа по этой таблице поиска его в таблице постоянных символов. С помощью этой директивы можно переопределить существующий код операции или создать новый.

Формат

.OPDEF код значение,список описателей операнда

Параметры

код

- строка кодов КОИ-8,определяющая имя кода операции. Данная строка может быть длиной до 31 символа и может содержать латинские буквы, цифры от 0 до 9, а также специальные символы: символ подчеркивания (_), символ денежной единицы (¤) и точку (.). Строка символов не может начинаться с цифры и не должна быть заключена в ограничители.

значение

- выражение, определяющее значение кода операции. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным. Данное выражение может иметь значение в диапазоне от 0 до 65 535 (шестнадцатеричное FFFF), однако значения от 252 до 255 не могут быть использованы. Указанное

выражение представляется следующим образом:

Если 0< выражение < 251, то выражение является однобайтным кодом операции;

Если выражение >255, то разряды выражения 7...0 являются первым байтом кода операции, а разряды 15...8 эторым байтом кода операции.

Значения 252...255 не могут быть использованы, поскольку архитектура определяет их как начало двухбайтного кода операции.

Список описателей операнда

- список описателей операнда, которые указывают количество операндов и тип каждого из них. В списке допускается употребление до 16 описателей операнда. Описатели операндов перечислены в табл. 26.

Таблица 26

Описатели операндов

Тип доступа	Типы данных								
	1) B	2) W	3) L	4) FF	5) FD	6) FG	7) FH	8) Q	9) O
Адрес	AB	AW	AL	AF	AD	AG	AH	AQ	AO
только чтение	RB	RW	RL	RF	RD	RG	RH	RQ	RO
модификация	MB	MW	ML	MF	MD	MG	MH	MQ	MO
только запись	WB	WW	WL	WF	WD	WG	WH	WQ	WO
поле	VB	VW	VL	VF	VD	VG	VH	VG	VO
переход	BB	BW	BL	BF	BD	BG	BH	BQ	BO

1). B - байт

2). W - слово

3). L - длинное слово

4) FF - с плавающей запятой формата F

5) FD - с плавающей запятой формата D

6). FG - с плавающей запятой формата G

7)
FH - с плавающей запятой формата H

8)
Q - квадрослово

9)
O - октаслово

Пример.

```
.OPDEF   MOVL3  ^XA9FF,RL,ML,WL ; определяется инструкция
                                ; MOLV3, которая исполь-
                                  зует зарезервированный
                                  код операции FF
.OPDEF   DIVF2  ^X46,RF,MF        переопределяются
.OPDEF   MOVC5  ^X2C,RW,AB,AB,RW,AB;инструкции DIVF2 и
                                ; MOVC5
.OPDEF   CALL   ^X10, BB          эквивалентно BSBB
```

Примечания:

1. Для переопределения кода операции возможно использование макрокоманды. Отметим, что поиск символа выполняется сначала в таблице имен макрокоманд, а затем в таблице кодов операций, определенных пользователем.

2. Директива .OPDEF оказывается полезной при создании "пользовательских" инструкций, которые выполняются по микропрограмме, написанной пользователем. Данная директива позволяет программистам выполнять их микропрограммы в составе программ на языке макроассемблер.

3. Описатели операнда указываются в таком же формате,

что и запись указателя операнда, описанная в документе [2], а именно: первый символ указывает на тип доступа к операнду, а второй символ указывает на тип данных операнда.

.REF - директивы генерирования операнда

Язык макроассемблер имеет пять директив генерирования операндов, представленных в табл. 27, которые используются в макрокомандах для определения новых кодов операций.

Таблица 27

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 октаслова

Директивы .REF обеспечивают совместимость с МАКРОАССЕМБЛЕРОМ ранних версий. Директива .OPDEF предоставляет более широкие функциональные возможности и по сравнению с директивами .REF ею проще пользоваться. Поэтому вместо директив .REF следует использовать директиву .OPDEF.

Форматы

.REF1 операнд

.REF2 операнд

.REF4 операнд

.REF8 операнд

.REF16 операнд

Параметр

операнд

- операнд в виде, соответственно, байта, слова, длинного слова, квадрослова или октаслова.

Пример.

```
.MACRO  MOVL3, M, N, L
.BYTE   ^XFF, ^XA9
.REF4   M                       ; операнды
                                  представлены
.REF4   N                       ; в виде
                                ; длинных слов
.REF4   L
.ENDM   MOVL3
MOVL3   R0,@MN-1,(R7)+[R10]
```

В данном примере используется директива .REF4, с помощью которой создается новая инструкция MOVL3. Для этой инструкции используется зарезервированный код операции FF. В примере, приведенном для директивы .OPDEF, используется более предпочтительный метод создания новых инструкций.

5.12. Директива дополнительной компоновки

.LINK - директива дополнительной компоновки

Директива .LINK позволяет включать во время трансляции программы дополнительные записи для компоновщика. Директива .LINK используется для ссылок, которые необходимо включить в отдельный образ. Использование директивы .LINK позволяет не указывать имена этих ссылок в команде LINK языка DCL. Квалификаторы директивы .LINK выполняют функции, сходные с функциями, выполняемыми такими же квалификаторами команды LINK языка DCL.

Формат

.LINK "спец.файла" [/квалиф.[=(имя модуля],...])],...]

Параметры

спец.файла

- спецификация файла, представляющая ограниченную строку знаков, которая определяет один или более входных файлов. Ограничителями строки может быть любая пара знаков, кроме пробела, знака табуляции, знака равенства (=), точки с запятой (;), левой угловой скобки (<). Знак, используемый в качестве ограничителя, не должен появляться в самой строке. Алфавитно-цифровые знаки могут быть использованы в качестве ограничителей, однако, для избежания недоразумений не рекомендуется их использовать.

Входными файлами могут быть об'ектные модули, которые необходимо скомпоновать, библиотеки, которые определены для внешних ссылок или из которых специфицированные модули должны быть включены, или разделяемые образы, которые должны быть включены в выходной образ. Если определяется несколько входных файлов, то они должны быть разделены запятыми (,).

Если тип файла не определяется в спецификации входного файла, то компоновщик использует типы файлов по умолчанию, основываясь на характере файла. Предполагается, что все об'ектные файлы имеют тип файла OBJ.

Следует заметить, что спецификации входных файлов должны быть известны к моменту компоновки. Необходимо, чтобы все ссылки были определенными, так как, если об'ектный модуль, созданный программой макроассемблер, скомпонован в другой директории, чем та, в которой он был создан, то компоновщик будет искать файлы, на которые производятся ссылки, в директиве .LINK.

Знак звездочка в спецификации файла не допускается.

Квалификаторы файла

/INCLUDE=(имя модуля[,...])

- квалификатор показывает, что входной файл является об'ектной библиотекой или библиотекой разделяемого образа, а так же, что специфицированные имена модулей являются обязательными входными модулями для компоновщика.

По крайней мере одно имя модуля должно быть специфицировано. Если специфицируется более, чем одно имя модуля, то их имена разделяются запятыми и весь список имен заключается в круглые скобки.

При использовании квалификатора /INCLUDE можно так же использовать квалификатор /LIBRARY, при этом для определения оставшихся неразрешенных ссылок используется библиотека.

Знак звездочка в спецификации имени файла не допускается. Имена модулей не могут содержать более 31 символа.

/LIBRARY

- квалификатор показывает, что соответствующий входной файл является библиотекой, в которой производится поиск модулей для разрешения любых неопределенных символов во входных файлах.

Если спецификация соответствующего входного файла не содержит тип файла, то компоновщик по умолчанию предполагает тип файла OLB. Можно использовать оба квалификатора /INCLUDE и /LIBRARY для определения спецификации файла. В этом случае явное включение модулей производится в первую очередь, а затем используется библиотека для поиска неразрешенных ссылок.

/SELECTIVE_SEARCH

- квалификатор указывает компоновщику о необходимости добавить к таблице символов из специфицированного файла только те глобальные символы, которые определены в файле и являются текущими неразрешенными.

если этот квалификатор не указан, то компоновщик включает все символы из специфицированного файла в таблицу глобальных символов.

/SHAREABLE

- квалификатор требует, чтобы компоновщик включил файл разделяемого образа. Символ звездочка в спецификации файла не допускается.

Табл. 28 показывает сокращенные наименования квалификаторов для директивы .LINK. Следует отметить, что для удобства чтения, а так же для совместимости с последующими версиями, рекомендуется пользоваться полными наименованиями квалификаторов.

Таблица 28

Квалификатор	Сокращенное наименование
/INCLUDE	/I
/LIBRARY	/L
/SELECTIVE_SEARCH	/SE
/SHAREABLE	/SH

Пример 1.

.LINK "SYS$LIBRARY:MYLIB"_/INCLUDE=(MOD1, MOD2, MOD6)

При обработке данного предложения в файле MYPROG.MAR макроассемблер посылает компоновщику запрос о необходимости скомпоновать MYPROG.OBJ с модулями MOD1, MOD2, MOD6 из библиотеки SYS$LIBRARY:MYLIB.OLB (где SYS$LIBRARY - логическое имя для диска и директории, в котором записан файл

MYLIB.OLB). Для других неразрешенных ссылок поиск библиотеки не производится. Это предложение эквивалентно компоновке файла с помощью команд языка DCL.

Пример 2.

```
$ LINK MYPROG, SYS$LIBRARY:MYLIB /INCLUDE=(MOD1,MOD2,MOD6)
 скомпоновать с об'ектным модулем SYS$LIBRARY:MYOBJ.OBJ
.LINK \SYS$LIBRARY:MYOBJ\
 найти об'ектную библиотеку SYS$LIBRARY:YOURLIB.OLB для
 неразрешенных ссылок
.LINK 'SYS$LIBRARY:YOURLIB' /LIBRARY
 найти таблицу символов SYS$LIBRARY:MYSTB.STB  для
 неразрешенных ссылок
.LINK *SYS$LIBRARY:MYSTB.STB* /SELECTIVE_SEARCH
 скомпоновать с разделяемым образом
.LINK "SYS$LIBRARY:MYSHR.EXE" /SHAREABLE
```

В целях оптимизации выполнения несколько входных файлов могут быть включены в одну директиву .LINK. В примере 3 показано как пять предыдущих предложений могут быть выполнены в одном.

Пример 3.

```
.LINK  'SYS$LIBRARY:MYOBJ',-
       'SYS$LIBRARY:YOURLIB' /LIBRARY,-
       'SYS$LIBRARY:MYLIB' /INCLUDE=(MOD1, MOD2, MOD6),-
       'SYS$LIBRARY:MYSTB.STB' /SELECTIVE_SEARCH,-
       'SYS$LIBRARY:MYSHR.EXE' /SHAREABLE
```

6. МАКРОКОМАНДЫ

С помощью макрокоманд программист может заменить целую последовательность строк исходного текста программы всего лишь одной строкой.

Исходный текст макрокоманды содержится в определении макрокоманды. Определение макрокоманды может содержать формальные аргументы (а может и не содержать формальных аргументов). Данные формальные аргументы могут использоваться в пределах последовательности исходных строк. Позже формальные аргументы заменяются фактическими аргументами; это происходит при вызове макрокоманды.

Команда вызова макрокоманды включает в себя имя данной макрокоманды, за которым могут следовать фактические параметры. Программа макроассемблер производит замену строки, содержащей макровызов на исходный текст, который содержится в макроопределении данной макрокоманды. Программа макроассемблер также производит замену формальных аргументов всюду, где они присутствуют в макроопределении, на соответствующие фактические аргументы, указанные в макровызове. Этот процесс называется расширением макрокоманды.

По умолчанию макрорасширение не распечатывается в листинге трансляции. Распечатка макроопределения производится только в том случае, если в директиве .SHOW (см. Раздел 5) или в квалификаторе /SHOW, который описан в документе [1], указан аргумент EXPANSIONS. Во всех примерах, приведенных в данном разделе, макрорасширение распечатывается, как будто

была указана директива .SHOW EXPANSION в исходном файле или в командной строке для макроассемблера был указан квалификатор /SHOW = EXPANSIONS.

Примечание. В дальнейшем вместо терминов: определение макрокоманды, вызов макрокоманды и расширение макрокоманды будут употребляться более удобные термины: макроопределение, макровызов и макрорасширение.

Директивы макрокоманд обеспечивают средства для выполнения функций различных видов. В табл. 29 перечислены эти виды функций и приведены директивы, которые их обеспечивают. Для облегчения ссылок все директивы макрокоманд представлены в алфавитном порядке.

Таблица 29

Краткие сведения о директивах макрокоманд

Вид функции	Директива
Директивы макроопределений	.MACRO
	.ENDM
Директивы библиотеки макрокоманд	.LIBRARY
	.MCALL
Директива удаления макрокоманд	.MDELETE
Директива выхода из макрокоманды	.MEXIT
Директивы атрибутов аргументов	.NSHR

Продолжение табл. 29

Вид функции	Директива
	.NTYPE
	1)
Директива блока повторений	.REPEAT(.REPT)
Директива конца блока повторений	.ENDR
Директивы блоков неопределенных	.IRP
повторений	.IRPC

1) Альтернативные формы директив (если они существуют) приведены в круглых скобках.

6.1. Аргументы макрокоманд

Макрокоманды имеют аргументы двух типов: фактические аргументы и формальные аргументы. Фактические аргументы - это строки, заданные в макровызове после имени макрокоманды. Формальные аргументы указываются по имени в макроопределении, то есть после имени макрокоманды в директиве .MACRO. Фактические аргументы в макровызовах и формальные аргументы в макроопределении могут разделяться запятыми, символами табуляции или пробелами.

Количество фактических аргументов в макровызове может быть равно или меньше количества формальных аргументов в макроопределении. Но если только количество фактических аргументов больше, чем количество формальных аргументов,

макроассемблер выводит сообщение об ошибке.

Обычно между формальными и фактическими аргументами установлено строгое позиционное соответствие. То есть первый фактический аргумент в макровызове замещает первый формальный аргумент в макроопределении во всех точках, где он только появляется. Однако в случае употребления ключевых аргументов это строгое позиционное соответствие может быть отвергнуто.

В примере 1 дается макроопределение, в котором используются формальные аргументы.

Пример 1.

```
.MACRO    SAVE      ARG1,ARG2,ARG3
.LONG     ARG1                    ; ARG1 первый аргумент
.WORD     ARG3                      ARG3 третий аргумент
.BYTE     ARG2                      ARG2 второй аргумент
.ENDM     SAVE
```

Примеры 2 ,3 иллюстрируют возможные макровызовы и макрорасширения приведенного макроопределения.

Пример 2.

```
SAVE      30,20,10      первый макровызов
.LONG     30            30 - первый аргумент
.WORD     10            10 - третий аргумент
.BYTE     20            20 - второй аргумент
```

Пример 3.

```
 SAVE  X,X-Y,Z          второй макровызов
 .LONG X                X первый аргумент
 .WORD Z                Z третий аргумент
```

```
.BYTE X-Y              X-Y второй аргумент
```

6.1.1. Значения по умолчанию

Значениями по умолчанию являются те значения, которые определены в макроопределении. Эти значения используются в том случае, если в макровызове для данного формального аргумента не указано никакого значения.

Значения по умолчанию указываются в директиве .MACRO следующим образом:

Имя формального аргумента = значение по умолчанию

В примере 1 макроопределения указываются значения по умолчанию.

Пример 1.

```
    .MACRO  DEFINE  B=5,W=10,Q=1000
    .BYTE   B
    .WORD   W
    .QUAD   Q
    .ENDM   DEFINE
```

Примеры 2,3,4 иллюстрируют возможные макровызовы и макрорасширения приведенного макроопределения.

Пример 2.

```
DEFINE              аргументы не указаны
.BYTE  5
.WORD  10
.QUAD  1000
```

Пример 3.

```
DEFINE /30/40     указаны второй и третий аргументы
.BYTE  5
.WORD  30
.QUAD  40
```

Пример 4.

```
DEFINE XYZ        указан первый аргумент
.BYTE  XYZ
.WORD  10
.QUAD  1000
```

6.1.2. Ключевые аргументы

Использование ключевых аргументов позволяет указывать аргументы в макровызове в произвольном порядке, однако при этом в макровызове должны указываться те же самые имена формальных аргументов, которые присутствуют в макроопределении. Ключевые аргументы оказываются полезными в тех случаях, когда макроопределение имеет много формальных аргументов, и только некоторые из них нужно указать в макровызове.

В одном макровызове аргументы должны быть либо все позиционно-зависимые, либо все ключевые. Если в одной макрокоманде смешаны позиционно-зависимые и ключевые аргументы, то только позиционно-зависимые аргументы соответствуют по позиции формальным аргументам, а ключевые аргументы не используются вовсе. Если формальный аргумент соответствует как позиционно-зависимому аргументу, так и ключевому аргу-

менту, то аргумент, появившийся в макровызове последним, отвергает любое другое определение аргумента для данного аргумента.

В примере 1 макроопределения указываются три аргумента.

Пример 1.

```
.MACRO  SAVE    A1,A2,A3
.LONG   A1
.WORD   A2
.WORD   A3
.ENDM   SAVE
```

В примере 2 макровызова указаны ключевые аргументы.

Пример 2.

```
SAVE    A3=5,A2=17,A1=COS
.LONG   COS
.WORD   5
.WORD   17
```

6.1.3. Аргументы в виде строки символов

Если фактический аргумент представляет собой строку символов, в состав которой входят такие символы, как например, запятая, пробел, символ табуляции, которые макроассемблер интерпретирует как разделители, то данная строка символов должна быть заключена между ограничителями. Такими ограничителями обычно являются парные угловые скобки (< >). Кроме того, макроассемблер интерпретирует в качестве ограничителей любые символы, следующие за уголком вверх (^).

таким образом, в случае, если угловые скобки должны использоваться в качестве элементов самой строки символов, программист может применить формат ограничителя с уголком вверх.

Рассмотрим примеры аргументов макрокоманд, заключенных между ограничителями.

Примеры:

```
1.      <THIS   IS A TEST! ISN'T IT?>
2.      <DATE:  1-MAY-1985>
3.      <CLEAR: CLRW    @(R2)+[R4]>
4.      ^%STRING IS FIRST <LAST> FOR AREA%
5.      ^?STACK POINTER <SP OR R14>?
```

В примерах 4,5 начальный уголок вверх указывает на то, что символ процента (%) и символ вопроса (?) являются ограничителями. Заметьте, что уголок вверх указывается только с левым ограничителем.

Макроассемблер воспринимает фактический аргумент в виде строки символов, заключенной между ограничителями, как один фактический аргумент и ставит ему в соответствие только один формальный аргумент. Если аргумент в виде строки символов, содержащий разделительные символы, не заключен между ограничителями, то макроассемблер интерпретирует его как группу последовательных аргументов и ставит ему в соответствие последовательные формальные аргументы.

Пример 6 иллюстрирует макровызов, который имеет один формальный аргумент.

Пример 6.

```
.MACRO   REPEAT STRING
.ASCII  /STRING/
.ASCII  /STRING/
.ENDM   REPEAT
```

Примеры 7 и 8 макровызова демонстрируют фактические аргументы, один из которых указан без ограничителей, а другой с ограничителями.

Пример 7.

```
REPEAT   <D R I V E R>
.ASCII   /D R I V E R/
.ASCII   /D R I V E R/
REPEAT   D R I V E R
```

%MACRO-E-TOOMNYARGS, TOO MANY ARGUMENTS IN MACRO CALL

макроассемблер выдает сообщение "в макровызове слишком много аргументов".

Отметим, что макроассемблер интерпретирует второй макровызов как макровызов, содержащий шесть фактических аргументов, а не один аргумент с пробелами.

Когда происходит вызов макрокоманды, до того, как поставить формальные аргументы в соответствие указанной строке символов, программа макроассемблер удаляет ограничители, между которыми заключена указанная строка символов (если они существуют).

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 обозна-

чать начало поля комментария.

Строка символов, заключенная между ограничителями, не может быть продолжена на следующую строку исходной программы.

Чтобы передать в строке символов число, содержащее операцию указания основания счисления или унарную операцию (например, ^XF19), необходимо, чтобы аргумент целиком был заключен между ограничителями, иначе макроассемблер будет интерпретировать операцию основания счисления как ограничитель. Рассмотрим пример аргументов макрокоманд, которые заключены между ограничителями, поскольку они содержат операции указания основания счисления.

Пример 8.

```
<^B11001100>
<^XFFFF>
<^F2.4>
```

Макрокоманды могут быть вложенными, то есть макроопределение может содержать вызов другой макрокоманды. Если в рамках одного макроопределения вызывается другая макрокоманда и при этом передается аргумент в виде строки символов, программист должен ввести ограничение аргумента так, чтобы во вторую макрокоманду полная строка передавалась как один аргумент.

Пример 9 макроопределения содержит вызов макрокоманды REPEAT.

Пример 9.

```
.MACRO  PART    ARG1,ARG2,STRING
.WORD   ARG1
.WORD   ARG2
REPEAT  <STRING>
.ENDM   PART
```

Следует заметить, что аргумент в макровызове RePeat заключен в угловые скобки, несмотря на то, что фактический аргумент не содержит каких-либо разделительных символов. Это делается по той причине, что фактический аргумент в макровызове REPEAT в то же время является формальным аргументом в этом макроопределении и будет замещаться фактическим аргументом, который может содержать разделительные символы.

В примере 10 осуществляется вызов макрокоманды PART, из которой в свою очередь производится вызов макрокоманды REPEAT.

Пример 10.

```
PART    1945,1985,<40 YEARS>
.WORD   1945
.WORD   1985
REPEAT  <40 YEARS>
.ASCII  /40 YEARS/
.ASCII  /40 YEARS/
```

Другой метод передачи аргументов в виде строки символов во вложенных макрокомандах заключается в ограничении аргументов макрокоманд вложенными ограничителями. В этом

случае макровызовы, входящие в состав макроопределения, не должны содержать ограничителей. Каждый раз, когда аргумент, заключенный между ограничителями, используется в макровызове, макроассемблер до того, как поставить его в соответствие с формальным аргументом, удаляет внешнюю пару ограничителей. Такой метод не рекомендуется использовать, поскольку в этом случае требуется, чтобы программист знал глубину вложения макрокоманд.

В примере 11 макроопределение также содержит вызов макрокоманды REPEAT.

Пример 11.

```
.MACRO  PART2   ARG1,ARG2,STRING
.WORD   ARG1
.WORD   ARG2
REPEAT  STRING
.ENDM   PART2
```

В макроопределении аргумент в макровызове REPEAT не заключен в угловые скобки.

В примере 12 содержится вызов макрокоманды PART2.

Пример 12.

```
PART2   1917,1985,<<68 YEARS>>
.WORD   1917
.WORD   1985
REPEAT  <68 YEARS>
.ASCII  /68 YEARS/
.ASCII  /68 YEARS/
```

Следует заметить, что хотя макровызов REPEAT в мак-

роопределении не заключен в ограничители, однако вызов в макрорасширении оказывается заключенным в ограничители, поскольку аргумент в виде строки символов в макровызове PART2 заключен во вложенные ограничители.

6.1.4. Конкатенация аргументов

Операция конкатенации аргументов, символ апостроф ('), выполняет об'единение аргументов макрокоманд в единый неделимый текст. Апостроф может либо следовать за именем формального аргумента, либо предшествовать ему.

Если апостроф предшествует имени аргумента, то текст до апострофа об'единяется с фактическим аргументом (при расширении макрокоманды). Например, если символ ARG1 является формальным аргументом, которому поставлен в соответствие фактический аргумент TEST, то форма TEXT'ARG1 расширяется в TEXTTEST.

Если апостроф следует за именем формального аргумента, то при расширении макрокоманды фактический аргумент об'единяется с текстом, следующим за апострофом. Например, если символ ARG2 является формальным аргументом, которому поставлен в соответствие фактический аргумент MOV, то форма ARG2'L расширяется в MOVL. Отметим, что сам символ апостроф не появляется в макрорасширении.

Для конкатенации двух аргументов разделите два формальных аргумента двумя последовательными апострофами. Два апострофа необходимы по той причине, что каждая операция конкатенации отбрасывает из расширения один апостроф.

В примере макроопределения используется конкатенация аргументов.

Пример.

```
        .MACRO  CMND    INST,SIZE,NUM
CMN'NUM':
        INST''SIZE      R'NUM
CMN'NUM'Z:
        .ENDM   CMND
```

Отметим, что для конкатенации двух формальных документов INST и SIZE использованы два последовательных апострофа.

Рассмотрим для данного макроопределения макровызов и макрорасширение.

```
        CMND    CLR,W,2
CMN2:   CLRW    R2
CMN2Z:
```

6.1.5. Символьное представление числовых аргументов

Если в качестве фактического аргумента указан символ, то в макрокоманду передается имя этого символа, а не его числовое значение. Однако посредством включения перед символом в вызове обратной косой черты (\) может быть передано значение данного символа. Затем программа макроассемблер передает в макрокоманду символы, представляющие десятичное значение символа. Например, если символ COUNT имеет значение 2, и указан фактический аргумент /COUNT, то в макроко-

манду передается строка "2", а не имя символа "COUNT".

Особенно полезной передача числового значения символа оказывается при создании новых символов совместно с операцией конкатенации аргументов (').

В примере 1 рассматривается макроопределение для передачи числовых значений символов.

Пример 1.

```
 .MACRO SPRING,ARG1,ARG2=^?^M<>?
 .ENTRY DAY'ARG1,ARG2
 .ENDM  SPRING
```

Пример 2 демонстрирует возможный макровызов и макрорасширение для рассмотренного макроопределения SPRING.

Пример 2.

```
MARCH=8
        SPRING  \MARCH
        .ENTRY  DAY8,^M<>

MARCH=MARCH+23
        SPRING  \MARCH,^?^M<R0,R1>?
        .ENTRY  DAY31,^M<R0,R1>
```

6.1.6. Автоматически создаваемые локальные метки

В макрокомандах локальные метки часто оказываются очень полезным средством. Хотя программист и имеет право указывать локальные метки в макроопределениях, но эти локальные метки могут оказаться сдублированными в одном и том же блоке локальных меток, что приводит к ошибке. Однако программист может использовать средства макроассемблера для создания локальных меток в макрорасширении, которые не вступят в конфликт с другими локальными метками. Такие локальные метки называются автоматически создаваемыми локальными метками.

Автоматически создаваемые локальные метки имеют диапазон значений от 30000¤ до 65535¤. 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.

Программист указывает на создаваемую локальную метку символом вопроса (?), который ставится перед именем формального аргумента. При расширении макрокоманды программа макроассемблер создает новую локальную метку в том случае, если пропущен фактический аргумент. Если соответствующий фактический аргумент указан, программа макроассемблер производит его подстановку вместо формального аргумента. Автоматически создаваемые локальные метки могут использоваться только для первых 31 формальных аргументов, указан-

ных в директиве .MACRO.

Автоматически создаваемые локальные метки не могут сочетаться с позиционно-зависимыми фактическими аргументами. Автоматически создаваемые локальные метки не могут сочетаться с фактическими аргументами, представленными ключевыми словами.

В примере 1 макроопределения употребляются автоматически создаваемые локальные метки.

Пример 1.

```
        .MACRO  WINTER          FEB,?LAB
        TSTW    FEB
        BGTR    LAB
        MOVW    FEB,R0
LAB:
        .ENDM   WINTER
```

В примере 2 иллюстрируется как употребление локальных меток, определяемых пользователем, так и употребление автоматически создаваемых локальных меток.

Пример 2.

```
        WINTER  R1
        TSTW    R1
        BGTR    30000¤
        MOVW    R1,R0
30000¤:
        WINTER  DEC
        TSTW    DEC
        BGTR    30001¤
```

```
        MOVW    DEC,R0
30001$:
        WINTER  JAN,20$
        TSTW    JAN
        BGTR    20$
        MOVW    JAN,R0
20$:
```

6.1.7. Строковые операции макрокоманд

Строковыми операциями макрокоманд для обработки строки символов являются следующие:

%LENGTH

%LOCATE

%EXTRACT

Эти операции производят определенные действия над аргументами макрокоманд, состоящими из последовательности символов, и над строками кодов КОИ-8. Они могут использоваться только в макрокомандах и в блоках повторений. В подпунктах 6.1.7.1 - 6.1.7.3 даны описания этих операций, приведены их форматы и представлены примеры их использования.

6.1.7.1. Операция %LENGTH

Операция %LENGTH определяет значение длины строки символов. Например, значение операции %LENGTH(<ABCDE>) равно 5.

Формат

%LENGTH(строка)

Параметр

строка

- аргумент макрокоманды или ограниченная строка символов, которая может быть заключена между угловыми скобками или между одинаковыми символами, первому из которых предшествует уголок вверх.

Пример.

Макроопределение:

```
.MACRO SIZE,STR
.IF GREATER    8-%LENGTH(STR)     если длина строки меньше
                                  8, то вывести сообщение
                                  об ошибке
.ERROR            STRING STR TOO SHORT
.ENDC
.ENDM  SIZE
```

Макровызовы и макрорасширения:

```
    SIZE    DISK
    .IF GREATER     8-4
%MACRO-E-GENERR, GENERATED ERROR: STRING DISK TOO SHORT
    .ENDC
```

```
SIZE    INTERRUPT
.IF GREATER     8-9
.ERROR          .STRING INTERRUPT TOO SHORT
.ENDC
```

При первом макровызове будет напечатано сообщение об ошибке, а при втором - нет.

6.1.7.2. Операция %LOCATE

Операция %LOCATE определяет местонахождение фрагмента строки символов. Если операция %LOCATE осуществляет поиск совпадающего фрагмента строки символов, то она возвращает информацию о позиции первого символа совпадения в строке символов. Например, значение операции %LOCATE(<D>,<ABCDEF>) равно трем. Заметьте, что первая символьная позиция в строке обозначается нулем. Если операция %LOCATE не осуществляет поиск совпадения, она возвращает значение, равное длине указанной строки символов. Например, значение операции %LOCATE (<Z>,<ABCDEF>) равно шести.

Операция %LOCATE определяет числовое значение, которое может быть использовано в любом выражении.

Формат

%LOCATE(строка1,строка2[,символ])

Параметры

строка1

- строка символов, являющаяся фрагментом другой строки символов. Данный фрагмент может быть либо аргументом макрокоманды, либо строкой символов, заключенной

между ограничителями. Ограничителями строки символов могут быть угловые скобки либо одинаковые символы, первому из которых предшествует уголок вверх.

строка2

- строка символов, в которой ищется совпадение фрагмента. Данная строка символов может быть либо аргументом макрокоманды, либо строкой символов, заключенной между ограничителями. Ограничителями строки символов могут быть угловые скобки либо одинаковые символы, первому из которых предшествует уголок вверх.

символ

- необязательный символ или десятичное число, которое указывает позицию в строке символов "строка2" с которой макроассемблер должен начать поиск. Если этот параметр пропущен, макроассемблер начинает поиск совпадения с позиции 0 (начало строки символов); данный символ должен быть абсолютным символом, определенным заранее, а число должно быть десятичным числом без знака. Недопустимы выражения и операции указания основания системы счисления.

Пример.

Макроопределение:

```
.MACRO TEST    STR
.IF EQUAL    %LOCATE(STR,<AABBCCDDEE>)-10
.ERROR                    ; STR NOT IN LIST
.ENDC
```

```
.ENDM  TEST
```

Макровызовы и макрорасширения:

```
    TEST    BB
    .IF EQUAL       2-10
    .ERROR                  BB NOT IN LIST
    .ENDC
    TEST    MM
    .IF EQUAL       10-10
%MACRO-E-GENERR, GENERATED ERROR: MM NOT IN LIST
```

Примечание. Если указан необязательный параметр "символ", поиск в строке "строка2" начинается с символьной позиции, на которую указывает данный параметр. Например, значение операции %LOCATE(<ACE>,<SPACE_PARAMETR>,5) равно 15, поскольку после пятой символовой позиции совпадения не существует.

6.1.7.3. Операция %EXTRACT

Операция %EXTRACT осуществляет извлечение фрагмента из строки символов. Эта операция определяет фрагмент строки символов, который начинается с указанной позиции и имеет указанный размер. Например, значением операции %EXTRACT(2,3,<ABCDEF>) является строка CDE. Отметим, что первый символ строки имеет позицию 0.

Формат

%EXTRACT(символ1,символ2,строка)

Параметры

символ1

- символ или десятичное число, которое указывает начальную позицию фрагмента строки символов. Данный символ должен быть абсолютным и заранее определенным, а число должно быть десятичным числом без знака. Недопустимы выражения и операции указания основания счисления.

символ2

- символ или десятичное число, которое указывает длину фрагмента строки символов. Дачный символ должен 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 между ограничителями. Данная строка символов может быть заключена между угловыми скобками или между двумя одинаковыми символами, первому из которых предшествует уголок вверх.

Пример.

Макроопределение:

```
.MACRO  EXTRA   ARG
.IF_EQUAL       14-%LENGTH(ARG)
.WARN             MAX LENGTH
.ENDC
.BLKB   %EXTRACT (2,2, ARG)
```

```
   .ENDM   EXTRA

     Макровызовы и макрорасширения:

 EXTRA           1985_AUGUST
 .IF EQUAL       14-11
 .WARN             MAX LENGTH
 .ENDC
 .BLKB           35
 EXTRA           1945_SEPTEMBER
 .IF EQUAL       14-14
%MACRO-W-GENWRN, GENERATED WARNING: MAX LENGTH
 .ENDC
 .BLKB           45
```

Примечание. 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.

6.2. Директивы макрокоманд

Данный подраздел посвящен подробному описанию директив макрокоманд. Приводятся форматы директив макрокоманд и даются примеры их использования. Директивы представлены в алфавитном порядке.

.ENDM - директива конца макроопределения

Директивой .ENDM завершается макроопределение. Пример использования директивы .ENDM приведен в описании директивы .MACRO.

Формат

.ENDM МАкроимя

Параметр

мАкроимя

- имя макрокоманды, чье определение должно быть завершено. Имя макрокоманды является необязательным параметром, однако, если оно указано, то оно должно совпадать с именем макрокоманды соответствующей директивы .MACRO. Имя макрокоманды полезно указывать в директиве .ENDM для того, чтобы макроассемблер смог обнаружить те макроопределения, в которых нарушились правила вложения.

Примечание. Если директива .ENDM встречается вне макроопределения, макроассемблер выводит сообщение об ошибке.

.ENDR - директива конца блока повторений

Директива .ENDR указывает на конец блока повторений. Эта директива должна быть завершающим предложением для любой директивы блока неопределенных повторений (.IRP и .IRPC), а также для любой директивы блока повторений (.REPEAT).

Формат

.ENDR

.IRP - директива блока неопределенных повторений с аргументами

По директиве .IRP осуществляется замена формального аргумента последовательными фактическими аргументами, которые указаны в списке аргументов. Этот процесс замены аргументов протекает во время расширения тела блока неопределенных повторений. На конец тела блока неопределенных повторений указывает директива .ENDR.

Директива .IRP аналогична макроопределению, у которого есть только один формальный аргумент. При каждом расширении тела блока повторений этот формальный аргумент заменяется последовательными элементами из списка аргументов. Данная директива и ее тело кодируются построчно в пределах исходной программы. Данный тип макроопределения и его расширение не требуют вызова макрокоманды по имени, тогда как этого требуют другие макрокоманды, описанные в данном разделе.

Директива .IRP может появляться как внутри, так и вне какого-то другого макроопределения, блока неопределенных повторений, блока повторений. Правила указания аргументов директивы .IRP те же самые, что и для указания аргументов макрокоманд.

Формат

.IRP символ,<список аргументов>

Блок

.ENDR

Параметры

символ

- 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

- 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. Директива .MEXIT разрешена в теле этого блока.

Пример.

Макроопределение:

```
.MACRO ALGEBRA          FNC,A1,A2,A3,A4,A5
.NARG  CNT
.IRP   ARG,<A5,A4,A3,A2,A1>
.IIF   NOT_BLANK ,      ARG,    FUNCTION ARG
.ENDR
CALLS  #<CNT-1>,FNC
.ENDM  ALGEBRA
```

Макровызовы и макрорасширения:

```
ALGEBRA         KEY,EXP,LIM,ABS,COS,SIN
.NARG           CNT
.IRP            ARG,<,,ABS,LIM,EXP>
.IIF NOT_BLANK ,ARG,    FUNCTION ARG
.ENDR
.IIF NOT_BLANK ,        ,FUNCTION
.IIF NOT_BLANK ,        ,FUNCTION
.IIF NOT_BLANK ,ABS     ,FUNCTION ABS
.IIF NOT_BLANK ,LIM     ,FUNCTION LIM
.IIF NOT_BLANK ,EXP     ,FUNCTION EXP
CALLS  #<CNT-1>,KEY
```

В приведенном примере для подсчета числа аргументов используется директива NARG, а для определения, не пропущен ли фактический аргумент, используется директива .IIF NOT_BLANK. Если фактический аргумент пропущен, то никакого

двоичного кода не формируется.

.IRPC - директива начала блока неопределенных повторений с символами

Директива .IRPC во всем подобна директиве .IRP, за исключением того, что директива .IRPC допускает подстановку в виде одного символа, а не в виде аргумента. В каждой итерации тела блока неопределенных повторений формальный аргумент заменяется следующим символом из указанной строки. На конец блока неопределенных повторений указывает директива .ENDR.

Формат

```
.IRPC символ,<строка>

блок

.ENDR
```

Параметры

символ.

- формальный аргумент, который последозательно заменяется 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, их использование рекомендуется с целью повышения четкости программы.

блок

- блок исходного текста, который должен быть повторен один раз для каждого символа из списка. Этот блок может содержать макроопределения и блоки повторений. Директива .MEXIT разрешена в теле этого блока.

Пример.

Макроопределение:

```
.MACRO  MODE            STRING
.NCHR   NUM,<BWLQO>
.IRPC   SIZE,<STRING>
.WARN     ADDRESS INSTRUCTION MOV'SIZE
.ENDR
.ENDM   MODE
```

Макровызов и макрорасширение:

```
MODE    <BWLQO>
.NCHR   NUM,<BWLQO>
.IRPC   SIZE,<BWLQO>
.WARN     ADDRESS INSTRUCTION MOV'SIZE
.ENDR
.WARN     ADDRESS INSTRUCTION MOVB
```

```
.WARN     ADDRESS INSTRUCTION MOVW
.WARN     ADDRESS INSTRUCTION MOVL
.WARN     ADDRESS INSTRUCTION MOVQ
.WARN     ADDRESS INSTRUCTION MOVO
```

Для подсчета числа символов в фактическом аргументе в данном примере используется директива .NCHR.

.LIBRARY - директива библиотеки макрокоманд

Директива .LIBRARY добавляет имя к списку библиотек макрокоманд, который просматривается каждый раз, когда встречается директива .MCALL или неопределенный код операции. Библиотеки просматриваются в порядке, обратному тому, в котором они были указаны.

Если программист опускает какие-то данные относительно аргумента имени библиотеки макрокоманд, то принимаются значения по умолчанию. Устройством по умолчанию является пользовательский диск, каталогом по умолчанию является пользовательский каталог, а типом файла по умолчанию является MLB.

Рекомендуется, чтобы библиотеки указывались в командной строке для макроассемблера с помощью квалификатора /LIBRARY, а не с помощью директивы .LIBRARY. Использование директивы .LIBRARY делает затруднительным перемещение файлов.

Формат

```
.LIBRARY имя макробиблиотеки
```

Параметр

имя макробиблиотеки

- ограниченная строка символов, которая является спецификацией файла, содержащего библиотеку макрокоманд.

Пример.

```
.LIBRARY   /DB1:[JOB]ARTIC/
.LIBRARY   ?DB1:MYFILE.MLB?
.LIBRARY   !WORK.MLB!
```

.MACRO - директива начала макроопределения

Директива .MACRO указывает на начало макроопределения. В этой директиве указываются имя макрокоманды и список формальных аргументов. Если указанное имя совпадает с именем ранее определенной макрокоманды, то прежнее макроопределение уничтожается и замещается новым. За директивой .MACRO следует исходный текст, который должен быть включен в макрорасширение. Конец тела макроопределения указывается с помощью директивы .ENDM.

Имена макрокоманд не вступают в конфликт с символами, определяемые пользователем. И макрокоманды, и символы, определяемые пользователем, могут иметь совпадающие имена.

Когда макроассемблер встречает директиву .MACRO, он добавляет имя макрокоманды, указанное в директиве .MACRO, в таблицу имен макрокоманд и запоминает исходный текст этой макрокоманды (до соответствующей директивы .ENDM). Никаких других действий до расширения макрокоманды не производится.

Символы из списка формальных аргументов связываются с данным именем макрокоманды и их действие ограничено пределами данного макроопределения. По этой причине символы, появляющиеся в списке формальных аргументов, могут также появляться в любом месте программы.

Формат

.MACRO имя макрокоманды [список формальных аргументов]

блок

.ENDM имя макрокоманды

Параметры

имя макрокоманды

- имя макрокоманды, которая должна быть определена. Этим именем может быть любое разрешенное символическое имя, длиной до 31 символа.

[список формальных аргументов]

- символы, разделенные запятыми. При макровызове эти символы замещаются фактическими аргументами.

блок

- исходный текст, который должен быть включен в макрорасширение.

Пример.

Макроопределение:

```
        .MACRO  DEFIN
        .PSECT  MINMAX,ABS,LONG
MIN=0
MAX=^XFFFF
        .PSECT  DATA,NOEXE,LONG
LIST:   .BLKB   1000
        .BLKW    100
        .BLKL   10

        .MACRO  DEFIN           переопределить макрокоманду
        .ENDM   DEFIN
        .ENDM   DEFIN
```

Макровызовы и макрорасширения:

```
        DEFIN
        .PSECT  MINMAX,ABS,LONG
MIN=0
MAX=^XFFFF
        .PSECT  DATA,LONG,NOEXE
LIST:   .BLKB   1000
        .BLKW   100
        .BLKL   10
        .MACRO  DEFIN
        .ENDM   DEFIN
        DEFIN               макрорасширение не содержит
```

исходного текста

В данном примере макрокоманда, вызванная в первый раз, определяет некоторые символы и области хранения данных, а также в конце переопределяет себя. Поэтому при вызове данной макрокоманды во второй раз, макрорасширение не содержит никакого исходного текста.

Примечания:

1. Если совпадает имена макрокоманды и кода инструкции ЭВМ СМ 1700, то вместо инструкции используется макрокоманда. Данное свойство позволяет программисту временно переопределить код инструкции.

2. Если совпадают имена макрокоманды и кода инструкции ЭВМ СМ 1700, и при этом данная макрокоманда содержится в библиотеке макрокоманд, то для определения этой макрокоманды должна быть использована директива .MCALL, поскольку в противном случае программа макроассемблер не будет просматривать библиотеки макрокоманд, так как данный символ уже определен (как код инструкции).

3. Во время трансляции программист может переопределить исходный текст макрокоманды путем указания второй директивы .MACRO с тем же самым именем макрокоманды. Путем включения второй директивы .MACRO в исходное макроопределение, можно обусловить переопределение макрокоманды при первом ее вызове. Это особенно удобно при установке исходных значений или при определении символов с помощью макрокоманды, то есть когда нужное действие выполняется только один раз. Новое макроопределение может удалить ненужный исходный

текст в макрокоманде или же полностью удалить всю макрокоманду. Для удаления макрокоманд используется так же директива .MDELETE.

.MCALL - директива вызова макрокоманд

В директиве .MCALL указываются имена системных и/или определенных пользователем макрокоманд, которые требуются для трансляции исходной программы, но не определены в исходном файле.

Если какое-то имя макрокоманды не найдено по завершению поиска (то есть, если данная макрокоманда не определена в какой-либо из библиотек макрокоманд), то выводится сообщение об ошибке.

Формат

.MCALL список макрокоманд

Параметр

список макрокоманд

- список макрокоманд, которые должны быть определены в данном процессе трансляции. Имена макрокоманд должны быть разделены запятыми.

Пример.

```
.MCALL SPANC    ; инструкцию SPANC заменить
                  макрокомандой из библиотеки
```

Примечание. Директива .MCALL обеспечивает совместимость с языком макроассемблер для см 4, однако не рекомендуется этим пользоваться. Когда программа макроассемблер находит неизвестный символ в поле инструкции, он автомати-

чески производит поиск этого символа во всех библиотеках макрокоманд. Если этот символ обнаруживается в библиотеке, то программа макроассемблер использует соответствующее макроопределение и производит расширение ссылки на макрокоманду. Если же программа макроассемблер не обнаруживает неизвестный символ в библиотеке, выводится сообщение об ошибке. Существует одно исключение, когда директива .MCALL обязательно должна быть использована. Это нужно в том случае, если макрокоманда и код инструкции имеют одинаковое имя.

.MDELETE - директива удаления макрокоманды

Директива .MDELETE производит удаление макроопределений для указанных макрокоманд. Количество удаленных Макрокоманд печатается в листинге трансляции в той же самой строке, в которой указана директива .MDELETE.

Директива .MDELETE полностью удаляет макрокоманду, освобождая по возможности память, тогда как способ переопределения макрокоманды, описанный для директивы .MACRO, позволяет только переопределить эту макрокоманду.

Формат

.MDELETE список макрокоманд

Параметр

список макрокоманд

- список имен макрокоманд, макроопределения которых подлежат удалению. Имена макрокоманд должны быть разделены запятыми.

Пример.

```
.MDELETE        LINE,TIME,#BASE
```

.MEXIT - директива выхода из макрокоманды

Директива .MEXIT завершает макрорасширение до того, как наступит конец макрокоманды. Завершение макрорасширения при этом ничем не отличается от завершения, когда встречается директива .ENDM. Данная директива может быть использована также и в теле блока повторений. Директива .MEXIT особенно полезна при условном расширении макрокоманд, поскольку она позволяет обходить сложности вложенных условных директив и изменять направления трансляции.

Формат

.MEXIT

Пример.

```
.MACRO TAPE     N,A,B


.IF EQ          N       начало блока условной трансляции


.MEXIT                  завершение макрорасширения
.ENDC                   и конец блока условной трансляции


.ENDM  TAPE             нормальный конец макрокоманды
```

В данном примере, в случае, если фактическое значение

формального аргумента N было бы равно нулю, то блок условной трансляции был бы транслирован и данная макрокоманда завершилась бы по директиве .MEXIT.

Примечания:

1. Если директива .MEXIT появляется в блоке повторений, то программа макроассемблер завершает текущее повторение тела блока и подавляет дальнейшее расширение тела блока повторений.

2. Если макрокоманды и блоки повторений являются вложенными, то директива .MEXIT вызывает переход к макрорасширению более высокого уровня.

3. Если дирекива .MEXIT возникает независимо от: макроопределения или блока повторений, то выводится сообщение об ошибке.

.NARG - директива определения количества аргументов

Директива .NARG определяет количество аргументов в текущем макровызове.

Директива .NARG производит подсчет всех позиционно-зависимых аргументов, указанных в макровызове, включая пустые аргументы (они обозначаются двумя соседними запятыми). В значение, присваиваемое указанному символу, не входит ни количество аргументов в виде ключевых слов, ни количество аргументов, имеющих значение по умолчанию.

Формат

.NARG символ

Параметр символ:

- символ, которому присваивается значение, равное количеству аргументов в макровызове.

Пример.

Макроопределение:

```
.MACRO ACCOUNT          A,B,C,D=ABC,D=DEF
.NARG  CNT              подсчет числа аргументов
.WORD  CNT              сохранить в CNT значение
                        числа аргументов
.ENDM  ACCOUNT
```

Макровызовы и макрорасширения:

```
ACCOUNT MAR,APR,MAY     значение счетчика будет 3
.NARG   CNT
.WORD   CNT

ACCOUNT                 значение счетчика будет 0
.NARG   CNT
.WORD   CNT

ACCOUNT MAR,APR=30,MAY=31    ; значение счетчика
.NARG   CNT                    будет 1
.WORD   CNT

ACCOUNT JUNE,JULE,,,    значение счетчика будет 5
.NARG   CNT
.WORD   CNT
```

Примечание. Если директива .NARG появляется вне макрокоманды, программа макроассемблер выводит сообщение об ошибке.

.NCHR - директива определения количества символов

Директива .NCHR определяет количество символов в указанной строке символов. Эта директива может появляться в любом месте программы, написанной на языке макроассемблер, и оказывается полезной при вычислении длины аргументов макрокоманд.

Формат

.NCHR символ,<строка>

Параметры

символ.

- символ, которому присваивается значение, равное количеству символов в указачной строке символов.

<строка>

- последовательность печатных символов. Данная строка символов должна быть ограничена угловыми скобками или двумя одинаковыми символами, первому из которых предшествует уголок вверх, причем лишь в том случае, если указанная строка символов содержит разрешенные разделительные символы (запятая, пробел, символ табуляции) или точку с запятой.

Пример.

Макроопределение:

```
.MACRO COUNT   STRING
.NCHR  CNT,<STRING>
.WORD  CNT
.ASCIZ /STRING/
.ENDM  COUNT
```

Макровызовы и макрорасширения:

```
COUNT   <SOVIET UNION>
.NCHR   CNT,<SOVIET UNION>        значение счетчика
.WORD   CNT                       будет равно 12
.ASCIZ  /SOVIET UNION/
```

.NTYPE - директива определения режима адресации

Директива .NTYPE определяет режим адресации для указанного операнда.

Значение символа устанавливается соответствующим образом для определенного адресного режима. В большинстве случаев возвращается 8-разрядное значение (1 байт). Разряды 0...3 указывают на регистр, который связан с данным режимом адресации, а разряды 4...7 указывают на режим адресации. Для обеспечения сжатой формы информации об адресации разряды 4...7 не точно воспроизводят то числовое значение, присвоенное определенному режиму адресации (приложение 2, табл. 6). В частности, литеральный режим обозначается нулем в разрядах 4...7, вместо значений 0...3 (см. Приложение 2, табл. 6). Режим 1 указывает на операнд непосредственной

адресации, режим 2 указывает на операнд абсолютной адресации, а режим 3 указывает на операнд адресации общего вида.

Для адресации с индексацией определяется 16-разрядное значение (2 байта). Старший байт содержит номер режима адресации для базового операнда, а младший байт содержит номер режима адресации первичного операнда (индексный регистр).

Формат

.NTYPE символ,Операнд

Параметр

символ

- любой разрешенный символ. Данному символу присваивается значение, равное 8- или 16-разрядному указателю режима адресации для операнда, который следует далее в качестве аргумента директивы.

операнд

- любое разрешенное адресное выражение, что используется вместе с кодом инструкции. Если "операнд" не указан, то подразумевается нуль.

Пример.

Макроопределение:

макрокоманда PUSHADR используется для помещения адреса в стек. Она осуществляет проверку типа операнда (путем использования директивы .NTYPE) с целью определения, является ли операнд адресом. Если операнд не является адресом, данная макрокоманда просто помещает аргумент в стек и формирует предупреждающее сообщение.

```
    .MACRO  PUSHADR ADDR
    .NTYPE  A,ADDR
A = A@-4@^XF
    .IF_IDENTICAL   0,<ADDR>
    PUSHL   #0
    .MEXIT
    .ENDC
ERR = 0
    IIF LESS_EQUAL A-1, ERR=1
    .IIF EQUAL A-5, ERR=1
    .IF EQUAL  ERR
    PUSHAL  ADDR
    .IFF
    PUSHL   ADDR
    .WARN     ADDR IS NOT AN ADDRESS
    .ENDC
    .ENDM   PUSHADR
```

Макровызовы и макрорасширения:

```
   PUSHADR (R0)
   PUSHL   (R0)
   PUSHADR (R1) [R4]
   PUSHAL  (R1) [R4]
   PUSHADR 0
   PUSHL   #0
   PUSHADR #1
   PUSHL   #1
%MACRO-W-GENWRN, GENERATED WARNING: #1 IS NOT AN ADDRESS
   PUSHADR R0
   PUSHL   R0
%MACRO-W-GENWRN, GENERATED WARNING: R0 IS NOT AN ADDRESS
```

.REPEAT - директива начала блока повторений

По директиве .REPEAT выполняется повторение блока исходного текста программы указанное количество раз. Такое повторение выполняется последовательно с остальным исходным текстом. Конец тела блока повторений указывается с помощью директивы .ENDR.

Формат

```
.REPEAT выражение
   .
   .
   .
Блок
```

.ENDR

Параметры

выражение

- значение данного выражения определяет число раз, которое должен транслироваться в программе данный блок. Если значение данного выражения меньше или равно нулю, блок повторений не транслируется. Данное выражение не должно содержать никаких неопределенных символов и должно быть абсолютным.

блок

- исходный текст, который должен быть повторен. Число повторений определяется значением выражения, указанного в директиве .REPEAT. Блок повторений может содержать макроопределения, блоки неопределенных повторений и другие блоки повторений. Допустимо использование в блоке повторений директивы .MEXIT.

Пример.

Макроопределение:

```
.MACRO COPY    STRING,NUM
.REPEAT NUM
.ASCII /STRING/
.ENDR
.WORD  0
.ENDM  COPY
```

Макровызовы и макрорасширения:

```
COPY    <WHAT'S YOUR NAME?>,4
.REPEAT 4
.ASCII  /WHAT'S YOUR NAME?/
.ENDR
.ASCII  /WHAT'S YOUR NAME?/
.ASCII  /WHAT'S YOUR NAME?/
.ASCII  /WHAT'S YOUR NAME?/
.ASCII  /WHAT'S YOUR NAME?/
```

Примечание. Альтернативной формой директивы .REPEAT является директива .REPT.

Приложение 1

Таблица кодов инструкций

Таблица кодов инструкций содержит постоянные символы, которые программа макроассемблер распознает автоматически. В таблице приводятся коды инструкций в алфавитном порядке.

Подробное описание системы инструкций представлено в документе [2].

Коды инструкций (в алфавитном порядке)

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
9D	ACBB	Сложение байтов, сравнение и переход
6F	ACBD	Сложение, сравнение и переход для данных формата D
4F	ACBF	Сложение, сравнение и переход для данных формата F
4FFD	ACBG	Сложение, сравнение и переход для данных формата G
6FFD	ACBH	Сложение, сравнение и переход для данных формата H

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
F1	ACBL	Сложение длинных слов, сравнение и переход
3D	ACBW	Сложение слов, сравнение и переход
58	ADAWI	Блокированное сложение выровненных слов
80	ADDB2	Сложение байтов, 2 операнда
81	ADDB3	Сложение байтов, 3 операнда
60	ADDD2	Сложение данных формата D, участвуют 2 операнда
61	ADDD3	Сложение данных формата D, участвуют 3 операнда
40	ADDF2	Сложение данных формата F, участвуют 2 операнда
41	ADDF3	Сложение данных формата F, участвуют 3 операнда
40FD	ADDG2	Сложение данных формата G, участвуют 2 операнда
41FD	ADDG3	Сложение данных формата G, участвуют 3 операнда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
6DFD	ADDH2	Сложение данных формата H, участвуют 2 операнда
61FD	ADDH3	Сложение данных формата H, участвуют 3 операнда
C0	aDDL2	Сложение длинных слов, участвуют 2 операнда
C1	ADDL3	Сложение длинных слов, участвуют 3 операнда
20	ADDP4	Сложение данных в упакованном формате, участвуют 4 операнда
21	ADDP6	Сложение данных в упакованном формате, участвуют 6 операндов
A0	ADDW2	Сложение слов, участвуют 2 операнда
a1	ADDW3	Сложение слов, участвуют 3 операнда
D8	ADWC	Сложение с переносом
F3	aoBLEQ	прибавление единицы и переход по меньше или равно
F2	AOBLSS	Прибавление единицы и пе-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		переход по меньше
78	ASHL	Арифметический сдвиг длинного слова
F8	aSHP	Арифметический сдвиг и округление данных в упакованном формате
79	ASHQ	Арифметический сдвиг квадрослова
E1	BBC	Переход по сбросу разряда
E5	BBCC	переход по сбросу разряда и сброс
E7	BBCCI	Блокированный переход по сбросу разряда и сброс
E3	BBCS	Переход по сбросу разряда и установка
E0	BBS	Переход по установке разряда
E4	BBSC	Переход по установке разряда и сброс
E2	BBSS	Переход по установке разряда и установка
E6	BBSSI	Блокированный переход по

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		установке разряда и уста-
		новка
1E	BCC	Переход по сброшенному би-
		ту переноса
1F	BCS	Переход по установленному
		биту переноса
13	BEQL	Переход по равенству
13	BEQLU	Переход по равенству (без
		знака)
18	BGEQ	Переход по больше или рав-
		но
1E	BGEQU	Переход по больше или рав-
		но (без знака)
14	BGTR	Переход по больше
1A	BGTRU	Переход по больше (без
		знака)
8A	BICB2	Очистка разрядов в байте,
		участвуют 2 операнда
8B	BICB3	Очистка разрядов в байте,
		участвуют 3 операнда
CA	BICL2	Очистка разрядов в длинном
		слове, участвуют 2 операн-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		да
C3	BICL3	Очистка разрядов в длинном слове, участвуют 3 операн-
		да
B9	BICPSW	Очистка разрядов в слове состояния программы
AA	BICW2	Очистка разрядов в слове, участвуют 2 операнда
A3	BICW3	Очистка разрядов в слове, участвуют 3 операнда
88	BISB2	Установка разрядов в байте участвуют 2 операнда
89	BISB3	Установка разрядов в байте участвуют 3 операнда
C8	BISL2	Установка разрядов в длин- ном слове, участвуют 2 операнда
C9	BISL3	Установка разрядов в длин- ном слове, участвуют 3 операнда
B8	BISPSW	Установка разрядов в слове состояния программы

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
A8	BISW2	Установка разрядов в слове участвуют 2 операнда
A9	BISW3	Установка разрядов в слове участвуют 3 операнда.
93	BITB	Проверка разрядов в байте
D3	BITL	Проверка разрядов в длинном слове
33	BITW	Проверка разрядов в слове
E9	BLBC	Переход по сбросу младшего разряда
E8	BLBS	Переход по установке младшего разряда
15	BLEQ	Переход по меньше или равно
13	BLEQU	Переход по меньше или равно (без знака)
19	BLSS	Переход по меньше
1F	BLSSU	Переход по меньше (без знака)
12	BNEQ	Переход по неравенству
12	BNEQU	Переход по неравенству (без знака)

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
03	BPT	Внутреннее прерывание по точке останова
11	BRB	Безусловный переход с смещением в байте
31	BRW	Безусловный переход со смещением в слове
10	BSBB	Переход на подпрограмму с смещением в байте
30	BSBW	Переход на подпрограмму со смещением в слове
1C	BVC	Переход по сбросу бита переполнения
1D	BVS	Переход по установке бита переполнения
FA	CALLG	Вызов с обычным списком аргумента
FB	CALLS	Вызов с занесением в стек
8F	CASEB	Выбор перехода со смещением в байте
CF	CASEL	Выбор перехода со смещением в длинном слове
AF	CASEW	Выбор перехода со смеще-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		нием в слове
BD	CHME	Изменить режим на управ-
		ляющую программу
BC	CHMK	Изменить режим на ядро
BE	CHMS	Изменить режим на суперви-
		зор
BF	CHMU	Изменить режим на пользо-
		вательский
94	CLRB	Очистить байт
7C	CLRD	Очистить данное формата D
DF	CLRF	Очистить данное формата F
7C	CLRG	Очистить данное формата G
7CFD	CLRH	Очистить данное формата H
D4	CLRL	Очистить длинное слово
7CFD	CLRO	Очистить октаслово
7C	CLRQ	Очистить квадрослово
B4	CLRW	Очистить слово
91	CMPB	Сравнение байтов
29	CMPC3	Сравнение символов, участ-
		вуют 3 операнда
2D	CMPC5	Сравнение символов, участ-
		вуют 5 операндов

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
71.	CMPD	Сравнение данных формата D
51.	CMPF	Сравнение данных формата F
51FD	CMPG	Сравнение данных формата G
71FD	CMPH	Сравнение данных формата H
D1.	CMPL	Сравнение длинных слов
35	CMPP3	Сравнение данных упакованного формата, участвуют 3 операнда
37.	CMPP4	Сравнение данных упакованного формата, участвуют 4 операнда
EC.	CMPV	Сравнение полей
BI	CMPW	Сравнение слов
ED	CMPZV	Сравнение полей с распространением нуля
OB	CRC	Расчитать циклическую проверку на избыточность
6C.	CVTBD	Преобразовать байт в формат D
4C.	CVTBF	Преобразовать байт в формат F
4CFD	CVTBG	Преобразовать байт в фор-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		мат G
6CFD	CVTBH	Преобразовать байт в формат H
98	CVTBL	Преобразовать байт в длинное слово
99	CVTBW	Преобразовать байт в слово
68	CVTDB	Преобразовать формат D в байт
76	CVTDF	Преобразовать формат D в формат F
32FD	CVTDH	Преобразовать формат D в формат H
6A	CVTDL	Преобразовать формат D в длинное слово
69	CVTDW	Преобразовать формат D в слово
48	CVTFB	Преобразовать формат F в байт
56	CVTFD	Преобразовать формат F в формат D
99FD	CVTFG	Преобразовать формат F в формат H

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
4A	CVTFL	Преобразовать формат F в длинное слово
49	CVTFW	Преобразовать формат F в слово
48FD	CVTGB	Преобразовать формат G в байт
33FD	CVTGF	Преобразовать формат G в формат F
56FD	CVTGH	Преобразовать формат G в формат H
4AFD	CVTGL	Преобразовать формат G в длинное слово
49FD	CVTGW	Преобразовать формат G в слово
68FD	CVTHB	Преобразовать формат H в байт
F7FD	CVTHD	Преобразовать формат H в формат D
F6FD	CVTHF	Преобразовать формат H в формат F
76FD	CVTHG	Преобразовать формат H в формат G

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
6AFD	CVTHL	Преобразовать формат H в длинное слово
69FD	CVTHW	Преобразовать формат H в слово
F5	CVTLB	Преобразовать длинное слово в байт
6E	CVTLD	Преобразовать длинное слово в байт
4E	CVTLF	Преобразовать длинное слово в формат F
4EFD	CVTLG	Преобразовать длинное слово в формат G
6EFD	CVTLH	Преобразовать длинное слово в формат H
F9	CVTLP	Преобразовать длинное слово в упакованный формат
F7	CVTLW	Преобразовать длинное слово в слово
36	CVTPL	Преобразовать упакованный формат в длинное слово
08	CVTPS	Преобразовать упакованный формат в формат с ведущим

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		разделителем
24	CVTPT	Преобразовать упакованный формат в формат с остатком
6B	CVTRDL	Преобразовать формат D в длинное слово (с округлением)
4B	CVTRFL	Преобразовать формат F в длинное слово (с округлением)
4BFG	CVTRGL	Преобразовать формат G в длинное слово (с округлением)
6BFD	CVTRHL	Преобразовать формат H в длинное слово (с округлением)
09	CVTSP	Преобразовать формат с ведущими разделителями в упакованный формат
26	CVTTP	Преобразовать формат с остатком в упакованный формат
33	CVTWB	Преобразовать слово в байт

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
6D	CVTWD	Преобразовать слово в формат D
4D	CVTWF	Преобразовать слово в формат F
4DFD	CVTWG	Преобразовать слово в формат G
6DFD	CVTWH	Преобразовать слово в формат H
32	CVTWL	Преобразовать слово в длинное слово
97	DECB	Уменьшение байта на 1
37	DECW	Уменьшение слова на 1
86	DIVB2	Деление байтов, участвуют 2 операнда
87	DIVB3	Деление байтов, участвуют 3 операнда
66	DIVD2	Деление данных формата D, участвуют 2 операнда
67	DIVD3	Деление данных формата D, участвуют 3 операнда
46	DIVF2	Деление данных формата F, участвуют 2 операнда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
47	DIVF3	Деление данных формата F, участвуют 3 операнда
46FD	DIVG2	Деление данных формата G, участвуют 2 операнда
47FD	DIVG3	Деление данных формата G, участвуют 3 операнда
66FD	DIVH2	Деление данных формата H, участвуют 2 операнда
67FD	DIVH3	Деление данных формата H, участвуют 3 операнда
C6	DIVL2	Деление длинных слов, участвуют 2 операнда
C7	DIVL3	Деление длинных слов, участвуют 3 операнда
27	DIVP	Деление данных в упакованном формате
A6	DIVW2	Деление слов, участвуют 2 операнда
A7	DIVW3	Деление слов, участвуют 3 операнда
3B	EDITPC	Редактирование упакованного формата в символь-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		ную строку
73	EDIV	Расширенное деление
74	EMODD	Расширенное умножение с выделением целой части формата D
54	EMODF	Расширенное умножение с выделением целой части формата F
54FD	EMODG	Расширенное умножение с выделением целой части формата G
74FD	EMODH	Расширенное умножение с выделением целой части формата H
7A	EMUL	Расширенное умножение
EE	EXTV	Выделение битового поля
EF	EXTZV	Выделение битового поля с распространением нуля
ea	FFC	Нахождение первого сброшенного разряда
EA	FFS	Нахождение первого установленного разряда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
00	HALT	Останов
96	INCB	Увеличение байта на 1
D6	INCL	Увеличение длинного слова на 1
B6	INCW	Увеличение слова на 1
0A	INDEX	Вычисление указателя
5C	INSQHI	Блокированное включение в начало очереди
5D	INSQTI	Блокированное включение в конец очереди
0E	IHSQUE	Включение в очередь
F0	IHSV	Включение битового поля
17	JMP	Переход
16	JSB	Переход на подпрограмму
06	LDPCTX	Загрузка контекста программы
3A	LOCC	Определить местонахождение символа
39	MATCHC	Сравнить символы на совпадение
92	MCOMB	Переслать байт в дополнительном коде

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
D2	MCOML	Переслать длинное слово в дополнительном коде
B2	MCOMW	Переслать слово в дополнительном коде
D3	MFPR	Переслать из регистра процессора
8E	MNEGB	Переслать байт с отрицанием
72	MNEGD	Переслать данное формата D с отрицанием
52	MNECF	Переслать данное формата F с отрицанием
52FD	MNEGG	Переслать данное формата G с отрицанием
72FD	MNEGH	Переслать данное формата H с отрицанием
CE	MNEGL	Переслать длинное слово с отрицанием
AE	MNEGW	Переслать слово с отрицанием
9E	MOVAB	Переслать адрес байта
7E	MOVAD	Переслать адрес данного

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		формата D
DE	MOVAF	Переслать адрес данного формата F
7E	MOVAG	Переслать адрес данного формата G
7EFD	MOVAH	Переслать адрес данного формата H
DE	MOVAL	Переслать адрес длинного слова
7EFD	MOVAO	Переслать адрес октаслова
7E	MOVAQ	Переслать адрес квадро- слова
3E	MOVAW	Переслать адрес слова
90	MOVB	Переслать байт
28	MOVC3	Переслать символы, участ- вуют 3 операнда
2C	MOVC5	Переслать символы, участ- вуют 5 операндов
70	MOVD	Переслать данное формата D
50	MOVF	Переслать данное формата F
50FD	MOVG	Переслать данное формата G
70FD	MOVH	Переслать данное формата H

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
DD	MOVL	Переслать длинное слово
7DFD	MOVO	Переслать октаслово
34	MOVP	Переслать данные упакованного вида
DC	MOVPSL	Переслать длинное слово состояния программы
7D	MOVQ	Переслать квадрослово
2E	MOVTC	Переслать транслированные символы
2F	MOVTUC	Переслать транслированные символы до символа
B0	MOVW	Переслать слово
DA	MOVZBL	Переслать байт в длинное слово с распространением нуля
9B	MOVZBW	Переслать байт в слово с распространением нуля
3C	MOVZWL	Переслать слово в длинное слово с распространением нуля
DA	MTPR	Переслать в регистр процессора

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
84	MULB2	Умножение байтов, участвуют 2 операнда
85	MULB3	Умножение байтов, участвуют 3 операнда
64	MULD2	Умножение данных формата D участвуют 2 Операнда
65	MULD3	Умножение данных формата D участвуют 3 Операнда
44	MULF2	Умножение данных формата F участвуют 2 Операнда
45	MULF3	Умножение данных формата F участвуют 3 Операнда
44FD	MULG2	Умножение данных формата G участвуют 2 Операнда
45FD	MULG3	Умножение данных формата G участвуют 3 Операнда
64FD	MULH2	Умножение данных формата H участвуют 2 Операнда
65FD	MULH3	Умножение данных формата H участвуют 3 Операнда
C4	MULL2	Умножение длинных слов, участвуют 2 операнда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
C5	MULL3	Умножение длинных слов, участвуют 3 операнда
25	MULP	Умножение данных в упакованном формате
A4	MULW2	Умножение слов, участвуют 2 операнда
A5	MULW3	Умножение слов, участвуют 3 операнда
01	NOP	Нет операции
75	POLYD	Обработка многочлена формата D
55	POLYF	Обработка многочлена формата F
55FD	POLYG	Обработка многочлена формата G
75FD	POLYH	Обработка многочлена формата H
BA	POPR	Извлечение регистров из стека
0C	PROBER	Проверить доступ для чтения
0D	PROBEW	Проверить доступ для запи-

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		си
9F	PUSHAB	Записать в стек адрес байта
7F	PUSHAD	Записать в стек адрес данного формата D
DF	PUSHAF	Записать в стек адрес данного формата F
7F	PUSHAG	Записать в стек адрес данного формата G
7FFD	PUSHAH	Записать в стек адрес данного формата H
DF	PUSHAL	Записать в стек адрес длинного слова
7FFD	PUSHAO	Записать в стек адрес октаслова
7F	PUSHAQ	Записать в стек адрес квадрослова
3F	PUSHAW	Записать в стек адрес слова
DD	PUSHL	Записать в стек длинное слово
BB	PUSHR	Запись регистров в стек

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
02	REI	Возврат из исключения или прерывания
5E	REMQHI	Удаление из начала очереди, блокированное
5F	REMQTI	Удаление из конца очереди, блокированное
0F	REMQUE	Удаление из очереди
04	RET	Возврат из вызванной процедуры
9C	ROTL	Циклический сдвиг длинного слова
05	RSB	Возврат из подпрограммы
D9	SBWC	Вычитание с переносом
2A	SCANC	Поиск символов
3B	SKPC	Пропуск символов
F4	SOBGEQ	Вычитание единицы и переход по больше или равно
F5	SOBGTR	Вычитание единицы и переход по больше
2B	SPANC	Пропуск символов
B2	SUBB2	Вычитание байтов, участвуют 2 операнда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
83	SUBB3	Вычитание байтов, участвуют 3 операнда
62	SUBD2	Вычитание данных формата D участвуют 2 операнда
63	SUBD3	Вычитание данных формата D участвуют 3 операнда
42	SUBF2	Вычитание данных формата F участвуют 2 операнда
43	SUBF3	Вычитание данных формата F участвуют 3 операнда
42FD	SUBG2	Вычитание данных формата G участвуют 2 операнда
43FD	SUBG3	Вычитание данных формата G участвуют 3 операнда
62FD	SUBH2	Вычитание данных формата H участвуют 2 операнда
63FD	SUBH3	Вычитание данных формата H участвуют 3 операнда
C2	SUBL2	Вычитание длинных слов, участвуют 2 операнда
C3	SUBL3	Вычитание длинных слов, участвуют 3 операнда

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
22	SUBP4	Вычитание данных в упакованном формате, участвуют 4 операнда
23	SUBP6	Вычитание данных в упакованном формате, участвуют 6 операндов
A2	SUBW2	Вычитание слов, участвуют 2 операнда
A3	SUBW3	Вычитание слов, участвуют 3 операнда
07	SVPCTX	Сохранение контекста процесса
95	TSTB	Проверка байта
73	TSTD	Проверка данного формата D
53	TSTF	Проверка данного формата F
53FD	TSTG	Проверка данного формата G
73FD	TSTH	Проверка данного формата H
D5	TSTL	Проверка длинного слова
B5	TSTW	Проверка слова
FC	XFC	Распространенный функциональный вызов
8C	XORB2	Исключающее "или" байтов,

Продолжение

Шестнадцатеричное значение	Мнемоническое обозначение	Выполняемое действие
		участвуют 2 операнда
8D	XORB3	Исключающее "или" бэйтов, участвуют 3 операнда
CC	XORL2	Исключающее "или" длинных слов, участвуют 2 операнда
CD	XORL3	Исключающее "или" длинных слов, участвуют 3 операнда
AC	XORW2	Исключающее "или" слов, участвуют 2 операнда
AD	XORW3	Исключающее "или" слов, участвуют 3 операнда

Приложение 2

Краткие сведения по языку макроассемблер

1. ДИРЕКТИВЫ МАКРОАССЕМБЛЕРА

В табл. 1 сведены все директивы языка макроассемблер, как общие, так и директивы макрокоманд. Все директивы представлены в порядке алфавита.

Таблица 1

Директивы макроссемблера

Формат	Выполняемое действие
.ADDRESS список адресов	Хранит последовательные длинные слова, содержащие адреса данных
.ALIGN Ключ [,выражение]	Настраивает счетчик адресов на границу, указываемую параметром "ключ" (ключевое слово)
.ALIGN целое [,выражение]	Настраивает счетчик адресов на границу, указываемую числом 2 в степени значения параметра "целое" (целое число)

Продолжение табл. 1

Формат	Выполняемое действие
.ASCIC строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), перед которой следует счетный байт
.ASCID строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), перед которой следует описатель строки
.ASCII строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями)
.ASCIZ строка	Хранит строку кодов КОИ-8 (заключенную между ограничителями), за которой следует нулевой байт
.BLKA выражение	Резервирует длинные слова, для адресов данных
.BLKB выражение	Резервирует байты под данные
.BLKD выражение	Резервирует квадрослова для хранения данных двойной точности с плавающей запятой запятой в формате D

Продолжение табл. 1

Формат	Выполняемое действие
.BLKF выражение	Резервирует длинные слова для хранения данных с плавающей запятой одинарной точности в формате F
.BLKG выражение	Резервирует квадрослова для хранения данных с плавающей запятой в формате G
.BLKH выражение	Резервирует октослова для хранения данных с плавающей запятой расширенной точности в формате H
.BLKL выражение	Резервирует длинные слова для хранения данных
.BLKO выражение	Резервирует октослова для хранения данных
.BLKQ выражение	Резервирует квадрослова для хранения данных
.BLKW выражение	Резервирует слова для хранения данных
.BYTE список выражений	Генерирует последовательные байты данных, при этом каждый байт содержит значение, которое определяется значе-

Продолжение табл. 1

Формат	Выполняемое действие
	нием выражения
.CROSS	Разрешает формирование таблицы перекрестных ссылок для всех символов
.CROSS список символов	Таблица перекрестных ссылок формируется для указанных символов
.DEBUG список символов	Делает известными отладчику имена указанных символов
.DEFAULT смещение,ключ	Указывает длину смещения, принимаемую по умолчанию, для режимов относительной адресации
.D_FLOATING список литералов	Генерирует 8-байтное данное с плавающей запятой двойной точности формата D
.DISABLE список аргументов	Запрещает функцию (функции), указанную в списке аргументов
.DOUBLE список литералов	Эквивалентна директиве .D_FLOATING
.DSABL список аргументов	Эквивалентна директиве .DISABLE

Продолжение табл. 1

Формат	Выполняемое действие
.ENABL список аргументов	Эквивалентна директиве .ENABLE
.ENABLE список аргументов	Разрешает функцию (функции), указанную в списке аргументов
.END [имя]	Указывает на логический конец исходной программы; необязательный аргумент имя указывает на адрес точка входа
.ENDC	Указывает на конец блока условной трансляции
.ENDM [макроимя]	Указывает на конец макроопределения
.ENDR	Указывает на конец блока повторений
.ENTRY символ [,выражение]	Определяет входную точку программы
.ERROR [выражение] ;коммент	Выводит определенное сообщение об ошибке
.EVEN	Гарантирует, что значение текущего счетчика адресов является четным (если

Продолжение табл. 1

Формат	Выполняемое действие
	значение нечетно, к нему добавляется единица)
.EXTERNAL список символов	Указывает на то, что символы, определенные в спецификации, являются внешними
.EXTRN список символов	Эквивалентна директиве .EXTERNAL
.F_FLOATING список литералов	Генерирует 4-байтные данные с плавающей запятой одинарной точности формата F
.FLOAT список литералов	Эквивалентна директиве .F_FLOATING
.G_FLOATING список литералов	Генерирует 8-байтные данные с плавающей запятой в формате G
.GLOBAL список символов	Указывает на то, что символы, определенные в спецификации, являются глобальными
.GLOBL	Эквивалентна директиве .GLOBAL

Продолжение табл. 1

Формат	Выполняемое действие
.H_FLOATING список литералов	Генерирует 16-разрядные данные с плавающей запятой расширенной точности в формате H
.IDENT строка	Обеспечивает средство снабжения об'ектного модуля дополнительной информацией
.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
.IFF	Эквивалентна директиве .IF_FALSE
.IF_FALSE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит

Продолжение табл. 1

Формат	Выполняемое действие
	трансляции в том случае, если исходное условие ложно
.IFT	Эквивалентна директиве .IF_TRUE
.IFTF	Эквивалентна директиве .IF_TRUE_FALSE
.IF_TRUE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит трансляции в том случае, если исходное условие является истинным
.IF_TRUE_FALSE	Появляется только в пределах блока условной трансляции. Начинает блок кодов, который подлежит трансляции безусловно
.IIF условие аргумент(ы) предложение	То же самое, что однострочный блок условной трансляции, когда условие проверяется для указанного

Продолжение табл. 1

Формат	Выполняемое действие
	аргумента. Операция подлежит трансляции только в том случае, если проверочное условие истинно
.IRP символ, <список аргументов>	Заменяет формальный аргумент последовательными фактическими аргументами, указанными в списке аргументом
.IRPC символ,<строка>	Заменяет формальный аргумент последовательными одиночными символами, указанными в строке
.LIBRARY имя макробиблиотеки	Определяет библиотеку макрокоманд
.LIST [список аргументов]	Эквивалентна директиве .SHOW
.LINK	Включает дополнительные записи для компоновщика
.LONG список выражений	Генерирует последовательные слова данных. Каждое длинное слово содержит значение, определяемое

Продолжение табл. 1

Формат	Выполняемое действие
	указанным выражением
.MACRO макроимя, Список аргументов	Начинает макроопределение
.MASK символ [,выражение]	Резервирует слово и копирует в него маску сохранения регистров
.MCALL список макроимен	Указывает системные и/или определяемые пользователем макрокоманды из библиотек, которые требуются для трансляции исходной программы
.MDELETE список макроимен	Удаляет из памяти макроопределения тех макрокоманд, которые указаны в списке
.MEXIT	Выход из расширения макрокоманды до завершения этой макрокоманды
.NARG символ	Определяет количество аргументов в текущем макровызове
.NCHR символ,<строка>	Определяет количество зна-

Продолжение табл. 1

Формат	Выполняемое действие
	ков в указанной строке
.NLIST [список аргументов]	Эквивалентна директиве .NOSHOW
.NOCROSS	Запрещает формирование таблицы перекрестных ссылок для указанных символов
.NOCROSS список символов	Запрещает формирование таблицы перекрестных ссылок для всех символов
.NOSHOW	Уменьшает значение счетчика уровня трансляции
.NOSHOW список аргументов	Управляет распечаткой листинга макрокоманд и блоков условной трансляции
.NTYPE символ,Операнд	Может появляться только в рамках макроопределения. присваивает символу значение режима адресации указанного операнда
.OCTA литерал	Запоминает 16 байт данных
.OCTA символ	Запоминает 16 байт данных
.ODD	Дает гарантию того, что значение текущего счетчика

Продолжение табл. 1

Формат	Выполняемое действие
	адресов является нечетным (если оно четное, то к нему добавляется единица)
.OPDEF код значения, список описателей операнда	Определяет код операции и список его операндов
.PACKED десятичная строка [,символ]	Генерирует десятичные числа в упакованном формате, две цифры на байт
.PAGE	Вызывает перевод распечатки листинга трансляции на начало следующей страницы
.PRINT [выражение] ;коммент	Выводит указанное сообщение
.PSECT	Начинает или продолжает неименованную программную секцию
.PSECT имя секции	Начинает или продолжает программную секцию, определяемую пользователем
.QUAD литерал	Запоминает 8 байт данных
.QUAD символ	Запоминает 8 байт данных
.REF1 операнд	Генерирует операнд байта
.REF2 Операнд	Генерирует операнд слова
.REF4 операнд	Генерирует операнд длин-

Продолжение табл. 1

Формат	Выполняемое действие
	ного слова
.REF8 Операнд	Генерирует операнд квадро-
	слова
.REF16 операнд	Генерирует операнд окто-
	слова
.RAPEAT выражение	Начинает блок повторений.
	часть кода до следующей ди-
	рективы .ENDR повторяется
	такое количество раз, кото-
	рое определяется значением
	выражения
.REPT.	Эквивалентна директиве
	.REPEAT
.RESTORE	Эквивалентна директиве
	.RESTORE_PSECT
.RESTORE_PSECT	Восстанавливает контекст
	программной секции из стека
	контекста программной сек-
	ции
.SAVE [локальный блок]	Эквивалентна директиве
	.SAVE_PSECT
.SAVE_PSECT [локальный	Сохраняет контекст текущей
блок]	программной секции в стеке

Продолжение табл. 1

Формат	Выполняемое действие
	контекста программной секции
.SBTTL комментарий	Эквивалентна директиве .SUBTITLE
.SHOW	дает приращение счетчику уровня листинга
.SHOW список аргументов	Управляет процессом распечатки макрокоманд и блоков условной трансляции
.SIGNED_BYTE список выражений	Запоминает последовательные байты (8 разрядов), в которых хранятся данные со знаком
.SIGNED_WORD список выражений	Запоминает последовательные слова (16 разрядов), в которых хранятся данные со знаком
.SUBTITLE комментарий	Распечатывает указанную строку как часть заголовка страницы листинга трансляции. Кроме того, строки, указанные в каждой из директив

Продолжение табл. 1

Формат	Выполняемое действие
	.SUBTITLE, сводятся в таблицу содержания, помещаемую в начале листинга трансляции
.TITLE имя модуля комментарий	Принимает в качестве имени об'ектного модуля первые 15 символов указанного имени модуля, а также обуславливает появление его на каждой странице листинга трансляции
.TRANSFER символ	Указывает компоновщику переопределить значение глобального символа для использования в разделяемом образе
.WARN [выражение] ;коммент	Выводит указанное предупреждающее сообщение
.WEAK список символов	Указывает на то, что каждый из перечисленных символов имеет атрибут подавления
.WORD список выражений	Генерирует последовательные слова данных. Каждое слово содержит значение, соот-

Продолжение табл. 1

Формат	Выполняемое действие
	ветствующее указанному выражению

2. СПЕЦИАЛЬНЫЕ ЗНАКИ

В табл. 2 собраны сведения о специальных знаках языка макроассемблер.

Таблица 2

Специальные знаки, употребляемые в предложениях языка макроассемблер

Знак	Наименование	Функция
_	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
	Знак горизон-	Ограничитель поля предложения

Продолжение табл. 2

Знак	Наименование	Функция
	тальной табуляции	
	Пробел	Ограничитель поля предложения
#	Знак номера	Индикатор режима непосредственной адресации
@	Коммерческое "эт"	Индикатор режима косвенной адресации и операция арифметического сдвига
,	Запятая	Разделитель полей, операндов и пунктов
	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"
м	Восклицательный знак	Операция логического "или"

Продолжение табл. 2

Знак	Наименование	Функция
\	Обратная дробная черта	Операция исключающее "или" и индикатор числового преобразования в аргументах макрокоманд
^	Стрелка аверх	Ограничитель унарной операции 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

3. ОПЕРАЦИИ

3.1. Унарные операции

В табл. 3 собраны сведения об унарных операциях языка макроассемблер.

Таблица 3

Унарные операции

Унарная операция	Имя операции	Пример	Выполняемая операция
+	Знак плюс	+A	Дает положительное значение a
-	Знак 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	^XFGF9	Интерпретирует число FGF9 как шестнадцатеричное
^A	Знак оператора	^A/ABC/	Порождает строку кодов КОИ-8; символы

Продолжение табл. 3

Унарная операция	Имя операции	Пример	Выполняемая операция
	КОИ-8		Заключенные между парными ограничителями, преобразуются в коды КОИ-8
^M	Знак оператора 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 дополнения	^C24	Выполняет преобразование десятичного значения 24 в дополнительный код (дополнение до единицы)

3.2. Бинарные операции

В табл. 4 собраны сведения о бинарных операциях языка макроассемблер.

Таблица 4

Бинарные операции

Знак	Наименование	Пример	Операция
+	Знак плюс	A+B	Сложение
-	Знак 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"

3.3. Строковые операции макрокоманд

В табл. 5 сведены данные по операциям для обработки аргументов макрокоманд в виде символьных строк. Эти операции могут использоваться только в макрокомандах.

Таблица 5

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"

4. РЕЖИМЫ АДРЕСАЦИИ

В табл. 6 собраны краткие сведения о режимах адресации языка макроассемблер.

Таблица 6

Режимы адресации

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
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	@(RN)+	9	Регистр содержит адрес адреса	Да

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
	ресация с автоувеличением	1) -(RN)	7	операнда; процессор увеличивает содержимое регистра на 4	
	Адресация с автоуменьшением			Процессор уменьшает содержимое регистра на размер типа данных операнда, после чего регистр содержит адрес операнда	Да
	Адресация по смещению	2) смщ(RN) B^смщ(RN) W^смщ(RN) L^смщ(RN)	A C E	Сумма содержимого регистра и смещения является адресом операнда; операции	Да

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
				B^, W^, L^ указывают, соответственно, на смещение в байтах, словах и длинных словах	
	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; операции B^,W^, L^ указывают, соответственно, на смещение в байтах, словах и в длинных словах	Да
	Литеральная	#литерал S^литерал	0-3	Указанный литерал является	Нет

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение		Описание	Возможна индексация
	адресация				операндом; литерал хранится в формате короткого литерала	
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; операторы B^, W^, L^ указывают на смещение, соответственно, в байтах, словах и длинных словах	Да
	Косвенная относительная	@адрес @B^адрес		B	Указанный адрес является адресом	Да

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
Относительная адресация	Относительная адресация	@W^адрес	D	адреса операнда; указанный адрес хранится в виде смещения относительно PC; операторы @^,W^,L^ указывают на смещение, соответственно, в байтах, словах и длинных словах	
		@L^адрес	F		
Абсолютная адресация	Абсолютная адресация	@#адрес	9	Указанный адрес является адресом операнда; указанный адрес хранится как абсолютный действительный адрес, а не как смещение	Да

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
Непосредственная адресация	#литерал	I^#литерал	8	Указанный литерал является операндом; этот литерал хранится как байт, слово, длинное слово или как квадрослово	Нет
Адресация объекта вида	G^адрес		-	Указанный адрес является адресом операнда; если адрес определен как перемещаемый, компоновщик запоминает этот адрес в виде смещения относительно ноль; если этот адрес определен	Да

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
				как абсолютный действительный адрес, компонов-щик запоминает его как абсолют-ное значение	
		3)			
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 определять любой	Нет

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
				режим адресации, кроме прямого обращения к регистру, индексации непосредственной адресации лиитеральной 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 рс; данный режим адресации может использо-	Нет

Продолжение табл. 6

Тип	Режим адресации	Формат	Шестнадцатеричное значение	Описание	Возможна индексация
				ваться только в инструкциях относительного перехода	

1) RN - любой регистр общего назначения с R0 по R12. Отметим, что вместо RN может использоваться любой из регистров AP, FP и SP.

2) смд - выражение, определяющее смещение.

3) RX - любой регистр общего назначения с R0 по R12. Отметим, что вместо RX может использоваться любой из регистров AP, FP и SP. RX не может быть тем же самым регистром, что RN, который определен в параметре "база" для определенных базовых режимов.

Приложение 3

Шестнадцатерично-десятичное преобразование чисел

При преобразовании шестнадцатеричных чисел в десятичные часто оказываются полезными десятичные значения степеней 2 и 16, которые представлены на рис. 1

На рис. 2 приведены десятичные значения для каждого возможного шестнадцатеричного значения в каждом байте длинного слова.

Для того, чтобы преобразовать шестнадцатеричное число в десятичное, для каждой цифровой позиции шестнадцатеричного числа найдите на рис. 1 в соответствующей колонке число и выпишите его десятичный эквивалент. Для получения десятичного числа сложите все выписанные эквиваленты.

Пример 1.

D0500AD0 (16) = ?(10)

D0000000	=	3,489,660,928
500000	=	5,242,880
a00	=	2,560
D0	=	208
--------		----------
D0500AD0		3,494,904,576

Для преобразования десятичных чисел в шестнадцатерич-

ные нужно выполнить четыре действия:

1) найдите на рис. 1 наибольшее десятичное число, не превосходящее число, подлежащее преобразованию;

2) запишите шестнадцатеричный эквивалент, за которым следуют нули в количестве, соответствующем числу колонок минус один;

3) вычтите найденное на рис. 2 десятичное значение из десятичного значения, которое подлежит преобразованию;

4) повторять шаги с первого по третий до тех пор, пока результат вычитания не станет равным нулю. Сложите полученные шестнадцатеричные эквиваленты и в результате получится шестнадцатеричное число.

Пример 2.

```
22,466 (10)      = ?(16)

20,480           = 5000          22,466
 1,792           =  700         -20,480
   192           =   C0          ------
     2           =    2           1,986
------           ----            -1,792
22,466           = 57c2           -----
                                    194
                                  -  192
                                   -----
                                      2
                                  -   2
                                   -----
                                      0
```

00152-01 35 01

Степени 2		Степени 16	
2 ** N	N	16 ** N	N
256	8	1	0
512	9	16	1
1024	10	255	2
2048	11	4096	3
4096	12	65536	4
8192	13	1048576	5
16384	14	16777216	6
32768	15	268435456	7
65536	16	4294967296	8
131072	17	68719476736	9
262144	18	1099511627776	10
524288	19	17592186044416	11
1048576	20	281474976710656	12
2097152	21	4503599627370496	13
4194304	22	72057594037927936	14
8388608	23	1152921504606846976	15
1677772	24		

рис.1

Шестнадцатерично-десятичное преобразование

16сс	10сс	16сс	10сс	16сс	10сс	16сс	10сс
0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608
4	1,073,741,824	4	67,108,804	4	4,194,304	4	262,144
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752
8	2,147,483,643	8	134,217,728	8	8,388,608	8	524,288
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,82
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040

байт	байт

слово

16сс	10сс	16сс	10сс	16сс	10сс	16сс	10сс
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

байт	байт

слово

16сс - шестнадцатеричное значение;
10сс - десятичное значение

рис.2

Перечень ссылочных документов

1. Операционнаб система МОС ВП. Система программирования на языке макроассемблер. Руководство программиста. 00152-01 33 01.

перечень ссылочных документов

1. Операционная система МОС ВП. Система программирования на языке макроассемблер. Руководство программиста. 26.00152-01 33 01.

Лист регистрации изменений

Номера листов (страниц)					Всего листов (страниц) в докум.	N документа	Входящий N сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					