**ZSM4**

Макроассемблер

**для операционных систем CP/M-80, RSX180 и UZI180**

Справочное руководство

Версия 4.2

Гектор Пераза {Hector Peraza}, январь 2021 года

**Содержание**

1 Введение 4

2 Запуск ZSM4 4

2.1 Вызов ассемблера 4

2.2 Формат команды 4

2.2.1 Дополнительные переключатели 5

2.3 Формат исходного файла 6

2.3.1 Операторы 6

2.3.2 Символы 6

2.3.3 Числовые константы 7

2.3.4 Строки 7

2.4 Вычисление выражений 8

2.4.1 Арифметические и логические операторы 8

2.4.2 Режимы 9

2.4.3 Внешние 9

2.5 Псевдо-операторы 10

2.5.1 ASEG 10

2.5.2 COMMON 10

2.5.3 CSEG 10

2.5.4 DEFB, DEFM, DB (Определить байт) 10

2.5.5 DEFC, DC (Определить символ) 11

2.5.6 DEFS, DS (Определение пространства) 11

2.5.7 DEFW, DW (Определить слово) 11

2.5.8 DEFZ (Определить строку с нулевым окончанием) 11

2.5.9 DSEG 11

2.5.10 END 12

2.5.11 ENTRY, GLOBAL, PUBLIC 12

2.5.12 EQU 12

2.5.13 EXT, EXTRN 12

2.5.14 INCLUDE, MACLIB 12

2.5.15 NAME 13

2.5.16 IDENT 13

2.5.17 ORG (Определение источника) 13

2.5.18 EJECT, FORM, PAGE 13

2.5.19 DEFL, ASET 14

2.5.20 SUBTTL 14

2.5.21 TITLE 14

2.5.22.COMMENT 14

2.5.23 .PRINTX 15

2.5.24 .RADIX 15

2.5.25 .Z80, .Z180, .Z280 15

2.5.26 RQST, .REQUEST 16

2.5.27 .EVEN и .ODD 16

2.5.28 Условные псевдооперации 16

2.5.29 Псевдо-операции управления листингом 17

2.5.30 Псевдо-операции по перемещению 18

2.5.31 Перемещение Перед Погрузкой 19

2.6 МАКРОСЫ и Блок Псевдоопераций 19

2.6.1 Условия 19

2.6.2 REPT-ENDM 20

2.6.3 IRP-ENDM 20

2.6.4 IRPC-ENDM 21

2.6.5 MACRO 21

2.6.6 ENDM 22

2.6.7 EXITM 22

2.6.8 LOCAL 22

2.6.9 Специальные макро-операторы и формы 23

2.7 Ошибки ZSM4 24

2.8 Совместимость с другими ассемблерами 25

2.9 Дополнительные примечания 25

A Работа в CP/M 27

B Работа в RSX180 28

C Работа в UZI180 29

D Работа в Linux или Windows 30

# 1 Введение

ZSM4 - это макроассемблер для микрокомпьютерных систем с Z80 или совместимым процессором с операционной системой CP/M или RSX180. Это серьезная переработка ассемблера Z80ASM, который впервые появился в CP/M US User Group, Vol 16. Первоначальный источник был защищен авторским правом (C) 1977, Lehman Consulting Services и был передан в общественное достояние. В дальнейшем он был модифицирован Рэем Холлсом {Ray Halls} в 1982 году и Нилом Харрисоном {Neil Harrison} в 1983 году. Последней известной была версия 2.8.

Основными особенностями ZSM4 являются:

* Ассемблирует код Z80, Z180 или Z280 с использованием стандартной мнемотехники Zilog/Mostek.
* Поддерживает условное ассемблирование, расширенное за счет дополнительного набора условных псевдо-операций, которые включают тестирование прохода ассемблирования, определение символов и параметров макросов. Условные выражения могут иметь до 8 уровней вложенности.
* Поддерживает полный набор стандартных макросов, включая IRP, IRPC, REPEAT, локальные переменные и EXITM. Вложенность макросов ограничена только объемом памяти.
* Создает файл REL, подходящий для компоновки с помощью L80 Microsoft или LINK Digital Research.

# 2 Запуск ZSM4

## 2.1 Вызов ассемблера

Команда для запуска ZSM4

**ZSM4 [команда]**

Формат команды описан в следующем разделе. Если команда не указана, ассемблер переходит в интерактивный режим, обозначенный строкой "\*", указывающей на готовность принимать команды.

## 2.2 Формат команды

Команда ZSM4 имеет следующий формат:

**[objfile],[lstfile]=srcfile[/option][/option...]**

где objfile, lstfile и srcfile - это спецификации файлов для выходного объектного файла, выходного файла листинга и входного исходного файла соответственно. Все имена файлов должны соответствовать соглашениям операционной системы об именах файлов и расширениях, а также могут указывать имя устройства.

Расширения по умолчанию следующие:

|  |  |  |
| --- | --- | --- |
| **Файл** | **CP/M** | **RSX180** |
| Перемещаемый объектный файл | REL | OBJ |
| Файла листинга | PRN | LST |
| Исходный файл | MAC | MAC |

Имя устройства по умолчанию в операционной системе CP/M - это текущий зарегистрированный диск. Имя устройства по умолчанию с операционной системе RSX180 - SY0:, пользовательское устройство по умолчанию.

Имена устройств следующие:

|  |  |  |
| --- | --- | --- |
| **Устройство** | **CP/M** | **RSX180** |
| Дисководы | A:, B:, C: | SY:, LB:, DYn:, DUn и т.д. |
| Линейный принтер | LST: или PRN: | LP: |
| Консоль | TTY: или CON: | TI: |

Примеры:

**,TTY:=TES**T Ассемблирует исходный файл TEST.MAC и выводит программу на консоль. Объектный код не генерируется. Полезно для проверки ошибок.

**SMALL,LST:=B:TEST** Ассемблирует TEST.MAC (на диске B), помещает в объектный файл SMALL.REL и распечатывает программу на линейном принтере.

### 2.2.1 Дополнительные переключатели

Параметры задаются с помощью переключателя, состоящего из косой черты, за которой следует одна буква. Некоторые параметры требуют дополнительного аргумента, как описано ниже. Можно использовать несколько переключателей, но каждому из них должна предшествовать косая черта. Все переключатели являются необязательными.

Доступные переключатели следующие:

|  |  |
| --- | --- |
| **/L** | Принудительная генерация файла листинга. |
| **/Dsymbol[=value]** | Определяет символ и при необходимости назначает значение. Значение является числовой константой, следующей формату, описанному в *разделе 2.3.3*. Если значение не указано, принимается 0. |
| **/Sn** | Устанавливает максимальную длину имени символа в файле REL. Значение n может быть в пределах от 5 до 8, а по умолчанию 6 |
| **/Zn** | Выберите начальный целевой тип процессора, n может быть 0 для Z80 (по умолчанию), 1 для Z180 и 2 для Z280. Этот параметр может быть переопределен псевдо-оператором .Z80, .Z180 или .Z280 (см. *Раздел 2.5.25*) |
| **/M** | Инициализирует области данных блока, определенные псевдооперацией DEFS или DS, в нули. Если переключатель не указан, пространство не будет содержать нулей. |
| **/U** | Рассматривает все неопределенные символы как внешние. |
| **/Iddn:[каталог]** | Указывает путь поиска для включаемых файлов (только версия RSX180.) |

Примеры:

**=TEST/L** Ассемблирует TEST.MAC, помещает объектный в файл TEST.REL и в файл листинга TEST.PRN

**LAST=TEST/U** Ассемблирует TEST.MAC и помещает в объектный файл LAST.REL. Файл листинга не будет сгенерирован. Все неопределенные символы будут объявлены внешними.

## 2.3 Формат исходного файла

Допустимы входные исходные строки длиной до 160 символов. ZSM4 сохраняет строчные буквы в строках в кавычках и комментариях. Все символы, коды операций и псевдо-коды, набранные в нижнем регистре, будут преобразованы внутри в верхний регистр.

### 2.3.1 Операторы

Исходные файлы, входящие в ZSM4, состоят из операторов формы:

**[метка[:[:]]] [оператор] [аргументы] [;комментарий]**

Операторы не должны начинаться в столбце 1. Для улучшения читаемости можно использовать несколько пробелов или символов табуляции.

Если метка присутствует, она является первым элементом в инструкции и может быть сразу же отмечена двоеточием. Если метка является единственным элементом в строке, то двоеточие является обязательным. Если за меткой следуют две двоеточия, она объявляется PUBLIC (см. *Раздел 2.5.11*). Например:

**FOO:: RET**

эквивалентно

**PUBLIC FOO**

**FOO: RET**

Следующий элемент после метки или первый элемент в строке, если метка отсутствует, является оператором. Оператор может быть мнемоникой Z80, псевдооперацией, вызовом макроса или выражением. Оценка следующая:

1. Вызов макроса
2. Мнемоника / псевдооперация
3. Выражение

Аргументы, следующие за оператором, будут, конечно, различаться по форме в зависимости от оператора.

*Комментарий* всегда начинается с точки с запятой и заканчивается символом возврата каретки. Комментарий может быть отдельной строкой или может быть добавлен к строке, содержащей оператор. Расширенные комментарии можно вводить с помощью псевдооперации .COMMENT (см. *Раздел 2.5.22*).

### 2.3.2 Символы

Символы ZSM4 могут иметь любую длину, однако значимы только первые 12 символов, и, кроме того, в объектном файле хранятся только первые шесть (если они не изменены переключателем опции S, см. *Раздел 2.2.1*). В символическом имени допустимы следующие символы:

**A-Z 0-9 $ . ? \_ @**

Символ не может начинаться с цифры. Когда символ считывается, нижний регистр переводится в верхний. Если за ссылкой на символ следует символ ##, он объявляется внешним (см. также псевдооперацию EXTRN, *Раздел 2.5.13*).

### 2.3.3 Числовые константы

Основание по умолчанию для числовых констант - десятичное. Оно может быть изменено псевдооперацией .RADIX (см. *Раздел 2.5.24*). Может быть выбрано любое основание от 2 (двоичное) до 16 (шестнадцатеричное). Когда основание больше 10, A-F - это цифры, следующие за 9. Если первая цифра числа не числовая, то перед ней должен стоять ноль.

Числа - это 16-битные величины без знака. Число всегда вычисляется в текущем основании, если не используется одно из следующих специальных обозначений:

**nnnnB** Двоичное

**nnnnD** Десятичное

**nnnnO** Восьмеричное

**nnnnQ** Восьмеричное

**nnnnH** Шестнадцатеричное

Переполнение числа, превышающего два байта, игнорируется, и результатом являются младшие 16 разрядов.

*Символьная константа* - это строка, состоящая из нуля, одного или двух символов ASCII, разделенная кавычками и используемая в непростом выражении. Например, в заявлении

**DB 'A'+1**

'A' - это символьная константа. Но оператор

**DB 'A'**

использует 'A' как строку, потому что это простое выражение. Правила для разделителей символьных констант такие же, как и для строк.

Символьная константа, состоящая из одного символа, имеет в качестве своего значения значение ASCII этого символа. То есть старший байт значения равен нулю, а младший байт - это ASCII значение символа. Например, значение константы 'A' равно 41H.

Символьная константа, состоящая из двух символов, имеет в качестве своего значения значение ASCII первого символа в младшем байте и значение ASCII второго символа в старшем байте. Например, значение символьной константы 'AB' равно 41H+42H\*256.

### 2.3.4 Строки

*Строка* состоит из нуля или более символов, разделенных кавычками. В качестве разделителей строк могут использоваться как одинарные, так и двойные кавычки. Кавычки-разделители могут использоваться в качестве символов, если они появляются дважды для каждого желаемого вхождения символов. Например, оператор

**DB "I am ""great"" today"**

сохраняет строку

**I am "great" today**

Если между разделителями есть нулевые символы, строка является *пустой строкой*.

## 2.4 Вычисление выражений

### 2.4.1 Арифметические и логические операторы

В следующей таблице перечислены разрешенные операторы и их приоритет:

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Функция** | **Приоритет** |
| NUL | тест для пустого аргумента | 1 |
| LOW | взять младший байт | 2 |
| HIGH | взять старший байт | 2 |
| \* | умножение без знака | 3 |
| / | деление без знака | 3 |
| MOD | без знака модуль | 3 |
| SHR | сдвиг вправо | 3 |
| SHL | сдвиг влево | 3 |
| - | унарный минус | 4 |
| + | сложение | 5 |
| - | вычитание | 5 |
| EQ | равенство | 6 |
| NE | не равно | 6 |
| LT | меньше, чем | 6 |
| LE | меньше или равно | 6 |
| GT | больше | 6 |
| GE | больше или равно | 6 |
| LESS | меньше чем со знаком | 6 |
| NOT | побитовое не | 7 |
| AND | побитовое и | 8 |
| OR | побитовое или | 9 |
| XOR | побитовое исключающее или | 9 |

Скобки используются для изменения порядка приоритета. Во время вычисления выражения, как только встречается новый оператор, который имеет приоритет меньше или равный последнему встреченному оператору, выполняются все операции вплоть до нового оператора. То есть сначала вычисляются подвыражения, включающие операторы с более высоким приоритетом.

Все операторы, кроме +, -, \*, / должны быть отделены от своих операндов хотя бы одним пробелом.

Операторы изоляции байтов (HIGH, LOW) изолируют 8 бит высокого или низкого порядка абсолютного 16-битного значения. Если перемещаемое значение предоставляется как операнд, HIGH и LOW будут обрабатывать его, как если бы оно было относительно нулевого местоположения.

### 2.4.2 Режимы

Все символы, используемые в качестве операндов в выражениях, находятся в одном из следующих режимов: абсолютном, перемещаемых данных, перемещаемых программ (кода) или COMMON. (См. *Раздел 2.5* для ASEG, CSEG, DSEG и COMMON псевдо-операций) Символы, ассемблированные под воздействием псевдо-операций ASEG, CSEG (по умолчанию) или DSEG, находятся в абсолютном режиме, режиме перемещаемого кода или перемещаемых данных соответственно.

Количество COMMON режимов в программе определяется количеством COMMON блоков, которые были названы псевдо-операцией COMMON. Два COMMON символа не находятся в одном и том же режиме, если они не находятся в одном и том же общем блоке. В любой операции, отличной от сложения или вычитания, режим обоих операндов должен быть абсолютным.

Если операция является сложением, то применяются следующие правила:

1. По крайней мере один из операндов должен быть абсолютным.
2. Абсолютный + режим = режим

Если операция представляет собой вычитание, то применяются следующие правила:

1. режим - Абсолютный = режим
2. режим - режим = Абсолютный

где два режима одинаковы.

Каждый промежуточный шаг вычисления выражения должен соответствовать вышеприведенным правилам для мод, иначе возникнет ошибка. Например, если FOO, BAZ и ZAZ являются тремя программно-перемещаемыми символами, то выражение

**FOO + (BAZ - ZAZ)**

законно, потому что первый шаг (BAZ - ZAZ) генерирует абсолютное значение, которое затем добавляется к программно-перемещаемому значению FOO.

### 2.4.3 Внешние

Помимо классификации по режиму, символ является либо внешним, либо не внешним. (См. EXTRN, *Раздел 2.5.13*). Внешнее значение должно быть ассемблировано в двухбайтовое поле (внешние значения не поддерживаются). К использованию внешних элементов в выражениях применяются следующие правила:

1. Внешнее допустимо только при сложении и вычитании.
2. Если в выражении используется внешний символ, результат выражения всегда является внешним.
3. Когда операция сложения выполняется, любой операнд (но не оба) может быть внешним.
4. Когда операция является вычитанием, только первый операнд может быть внешним.

## 2.5 Псевдо-операторы

### 2.5.1 ASEG

Синтаксис:

**ASEG**

ASEG устанавливает счетчик местоположения в абсолютный сегменте памяти. Расположение абсолютного счетчика будет таким же, как у последнего ASEG (по умолчанию 0), если только после ASEG не будет выполнена операция ORG для изменения местоположения. Эффект ASEG также достигается с помощью псевдо-операции сегмента кода (CSEG) и переключателя [P] в LINK DR. См. Также *Раздел 2.5.30*.

### 2.5.2 COMMON

Синтаксис:

**COMMON /имя\_блока/**

COMMON устанавливает счетчик местоположения в памяти в выбранном общем блоке. Местоположение всегда является началом области, так что совместимость с оператором COMMON FORTRAN сохраняется. Если имя\_блока опущено или состоит из пробелов, оно считается неименованным общим. См. Также *Раздел 2.5.30*.

### 2.5.3 CSEG

Синтаксис:

**CSEG**

CSEG устанавливает счетчик местоположения в сегменте перемещаемого кода памяти. Это будет местоположение последнего CSEG (по умолчанию 0), если только после CSEG не будет выполнена операция ORG для изменения местоположения. CSEG - это условие ассемблера по умолчанию (см. также *Раздел 2.5.30*.)

### 2.5.4 DEFB, DEFM, DB (Определить байт)

Синтаксис:

**DB exp[,exp...][,string...]**

**DB string[,string...][,exp...]**

**DEFB exp[,exp...][,string...]**

**DEFB string[,string...][,exp...]**

**DEFM string[,string...]**

Аргументы в DB являются либо выражениями, либо строками. DB хранит значения выражений или символы строк в последовательных ячейках памяти, начиная со счетчика текущего местоположения.

Выражения должны вычисляться с точностью до одного байта. (если высокий байт результата равен 0 или 255, ошибка не появляется; в противном случае приводит к ошибке A.)

Строки из трех и более символов не могут использоваться в выражениях (то есть за ними должна сразу следовать запятая или конец строки). Символы в строке хранятся в порядке появления, каждый в виде однобайтового значения со старшим битом, равным нулю.

### 2.5.5 DEFC, DC (Определить символ)

Синтаксис:

**DC string**

**DEFC string**

DC хранит символы в строке в последовательных ячейках памяти, начиная с текущего счетчика местоположений. Как и в случае с DB, символы хранятся в порядке появления, каждый в виде однобайтового значения с битом старшего порядка, равным нулю. Однако DC сохраняет последний символ строки с битом старшего порядка, установленным равным единице. Если аргументом DC является нулевой строкой, то возникнет ошибка.

### 2.5.6 DEFS, DS (Определение пространства)

Синтаксис:

**DS exp**

**DEFS exp**

DS резервирует область памяти. Значение exp указывает количество выделяемых байтов. Все имена, используемые в exp, должны быть предварительно определены (то есть все имена, известные в этот момент на проходе 1). В противном случае во время прохода 1 генерируется ошибка V, а во время прохода 2 может генерироваться ошибка U. Если ошибка U не генерируется во время прохода 2, , вероятно, будет генерироваться фазовая ошибка, потому что DS не генерировал код на проходе 1.

### 2.5.7 DEFW, DW (Определить слово)

Синтаксис:

**DW exp[,exp...]**

**DEFW exp[,exp...]**

DW хранит значения выражений в последовательных ячейках памяти, начиная с текущего счетчика местоположения. Выражения вычисляются как 2-байтовые (слово) значения.

### 2.5.8 DEFZ (Определить строку с нулевым окончанием)

Синтаксис:

**DEFZ строка**

DEFZ хранит символы в строке в последовательных ячейках памяти, начиная с текущего счетчика местоположения, и добавляет дополнительный нулевой байт в конце строки.

### 2.5.9 DSEG

Синтаксис:

**DSEG**

DSEG устанавливает счетчик местоположения в памяти перемещаемого сегмента данных. Расположение относительного счетчика данных будет таким же, как у последнего DSEG (по умолчанию 0), если не будет выполнена операция ORG после DSEG для изменения местоположения. См. Также *Раздел 2.5.30*.

### 2.5.10 END

Синтаксис:

**END [exp]**

Оператор END указывает конец программы. Если exp присутствует, это начальный адрес программы. Если exp отсутствует, то стартовый адрес не передается компоновщику для этой программы.

### 2.5.11 ENTRY, GLOBAL, PUBLIC

Синтаксис:

**ENTRY имя[,имя...]**

**GLOBAL имя[,имя...]**

**PUBLIC имя[,имя...]**

ENTRY, GLOBAL или PUBLIC объявляет каждое имя в списке внутренним и, следовательно, доступным для использования этой программой и другими программами, которые будут загружаться одновременно. Все имена в списке должны быть определены в текущей программе, иначе возникнет ошибка U. Ошибка M генерируется, если имя является внешним именем или именем COMMON блока.

### 2.5.12 EQU

Синтаксис:

**имя EQU exp**

EQU присваивает имя значению exp. Если exp является внешним, генерируется ошибка. Если имя уже имеет значение , отличное от exp, возникает ошибка M.

### 2.5.13 EXT, EXTRN

Синтаксис:

**EXT имя[,имя...]**

**EXTRN имя[,имя...]**

EXT или EXTRN объявляют, что имена в списке являются внешними (т.е. определенными в другой программе). Если какой-либо элемент в списке ссылается на имя, определенное в текущей программе, возникает ошибка M. Ссылка на имя, за которым сразу следуют два знака фунта (например, NAME##), также объявляет имя внешним.

### 2.5.14 INCLUDE, MACLIB

Синтаксис:

**INCLUDE filename**

**MACLIB filename**

Псевдо-операция INCLUDE собирает исходные операторы из альтернативного исходного файла в текущий исходный файл. Использование INCLUDE устраняет необходимость повторять часто используемую последовательность операторов в текущем исходном файле. INCLUDE и MACLIB - это синонимы.

Параметр filename – это любая допустимая спецификация файла, определенная операционной системой. Значения по умолчанию для расширений имен файлов и имен устройств такие же, как и в командной строке ZSM4.

INCLUDE файл открывается и ассемблируется в текущий исходный файл сразу после оператора INCLUDE. Когда достигнут конец файла, ассемблирование возобновляется с инструкцией, следующей за INCLUDE.

В листинге символ C печатается между собранным кодом и исходной строкой в каждой строке, ассемблированной из INCLUDE файла.

Вложенные включения разрешены до уровня 5.

Файл, указанный в поле операнд, должен существовать. Если файл не найден, генерируется ошибка V (value error), а включение игнорируется.

### 2.5.15 NAME

Синтаксис:

**NAME ('modname')**

**NAME 'modname'**

NAME определяет имя модуля. В имени модуля значимы только первые шесть символов. Имя модуля также может быть определено с помощью псевдо-операции TITLE. При отсутствии псевдо-операций NAME и TITLE имя модуля создается из имени исходного файла.

### 2.5.16 IDENT

Синтаксис:

**IDENT 'ident'**

IDENT определяет идентификационную строку для модуля, которая может использоваться для отслеживания версий. Значимы только первые шесть символов.

### 2.5.17 ORG (Определение источника)

Синтаксис:

**ORG exp**

Счетчик местоположения устанавливается равным значению exp, и ассемблер присваивает код ячейкам памяти, начиная с этого значения. Все имена, используемые в exp, должны быть известны на проходе 1, и значение должно быть либо абсолютным, либо находиться в той же области, что и счетчик местоположения.

### 2.5.18 EJECT, FORM, PAGE

Синтаксис:

**PAGE [exp]**

**EJECT**

**FORM**

EJECT, FORM или PAGE заставляет ассемблер запускать новую выходную страницу. Значение exp, если оно включено в инструкцию PAGE, становится новым размером страницы (измеряется в строках на страницу) и должно находиться в диапазоне от 10 до 255. Размер страницы по умолчанию составляет 60 строк на страницу. Ассемблер помещает символ подачи страницы в файл листинга в конце страницы.

### 2.5.19 DEFL, ASET

Синтаксис:

**имя ASET exp**

**имя DEFL exp**

DEFL - это то же самое, что и EQU, за исключением того, что ошибка не генерируется, если имя уже определено. DEFL и ASET - синонимы.

### 2.5.20 SUBTTL

Синтаксис:

**SUBTTL text**

SUBTTL указывает подзаголовок, который должен быть указан в строке после заголовка (см. TITLE, *Раздел 2.5.21*) на каждом заголовке страницы. Текст усекается после 60 символов. В программе может быть задано любое количество подстрок.

### 2.5.21 TITLE

Синтаксис:

**TITLE text**

TITLE указывает заголовок, который должен быть указан в первой строке каждой страницы. Если задано более одного заголовка, возникает ошибка Q. Первые шесть символов заголовка используются в качестве имени модуля, если не используется псевдооперация NAME. Если ни NAME, ни TITLE псевдооперации не используются, имя модуля создается из имени исходного файла.

### 2.5.22.COMMENT

Синтаксис:

**.COMMENT разделитель...текст... разделитель**

Первый непустой символ, встречающийся после .COMMENT - это разделитель. Следующий текст содержит блок комментариев, который продолжается до тех пор, пока не будет найдено следующее вхождение разделителя. Например, если в качестве разделителя использовать звездочку, то формат блока комментариев будет следующим:

**.COMMENT \***

**любое количество введенного здесь**

**текста в качестве блока комментариев**

**.**

**.**

**. \***

;возврат в нормальный режим

### 2.5.23 .PRINTX

Синтаксис:

**.PRINTX разделитель...текст... разделитель**

Первый непустой символ, встречающийся после .PRINTX, является разделителем. Следующий текст отображается на клемме во время сборки до тех пор, пока не встретится другое вхождение разделителя. .PRINTX полезен для отображения прогресса через длинное ассемблирование или для отображения значения условных переключателей ассемблера. Например:

**IF CPM**

**.PRINTX /CPM version/**

**ENDIF**

**Примечание**

.PRINTX будет выводиться на обоих проходах. Если требуется только одна распечатка, используйте псевдо-операции IF1 или IF2. Например:

**IF2**

**IF CPM**

**.PRINTX /CPM/**

**ENDIF**

**ENDIF**

будет только печатать, если CPM имеет значение true, а ZSM4 находится в проходе 2.

### 2.5.24 .RADIX

Синтаксис:

**.RADIX exp**

Основание (или система счисления) по умолчанию для всех констант - десятичное. Оператор .RADIX позволяет изменить значение по умолчанию основания на любое основание в диапазоне от 2 до 16. Например:

**LD A,0FFH**

**.RADIX**

**LD B,0FF**

Две команды LD в приведенном выше примере идентичны. Параметр exp в операторе .RADIX всегда находится в десятичном исчислении, независимо от текущего основания.

### 2.5.25 .Z80, .Z180, .Z280

Синтаксис:

**.Z80**

**.Z180**

**.Z280**

.Z80 позволяет ассемблеру принимать коды операций Z80. Это условие по умолчанию. Режим Z80 также может быть установлен путем добавления переключателя /Z0 к командной строке ZSM4 (см. *Раздел 2.2.1*).

.Z180 позволяет ассемблеру принимать коды операций Z80/HD64180. Режим Z180 также можно установить, добавив переключатель /Z1 в командную строку ZSM4.

.Z280 позволяет ассемблеру принимать коды операций Z280. Режим Z280 также может быть установлен путем добавления параметра /Z2 в командную строку ZSM4.

### 2.5.26 RQST, .REQUEST

Синтаксис:

**RQST filename[,filename...]**

**.REQUEST filename[,filename...]**

.REQUEST отправляет запрос к загрузчику компоновщика для поиска неопределенных глобальных переменных в именах файлов в списке. Имена файлов в списке должны быть в виде допустимых символов. Они не должны включать расширения имен файлов или спецификации диска. Обычно компоновщик предоставляет расширение по умолчанию и использует дисковый накопитель по умолчанию.

### 2.5.27 .EVEN и .ODD

Синтаксис:

**.EVEN**

**.ODD**

.EVEN заставляет ассемблер выдавать нулевой байт (инструкция NOP), если текущий адрес нечетный, поэтому следующая инструкция или байт данных будут выровнены по границе слова (четный адрес).

Аналогично, .ODD заставляет ассемблер выдавать нулевой байт (инструкция NOP), если текущий адрес четный, поэтому следующая инструкция или байт данных будут выровнены по нечетному адресу.

Обратите внимание, что для правильной работы .EVEN и .ODD во время выполнения соответствующий модуль должен быть загружен компоновщиком по четному адресу.

### 2.5.28 Условные псевдооперации

Условными псевдооперациями являются:

|  |  |
| --- | --- |
| **IF/IFT exp** | истина, если exp не равен 0. |
| **IFF exp** | истина, если exp равно 0. |
| **IF1** | истина, если проход 1. |
| **IF2** | истина, если проход 2. |
| **IFZ80** | истина, если ассемблер находится в режиме Z80 (см. *Раздел 2.5.25*). |
| **IFZ180** | истина, если ассемблер находится в режиме Z180 (см. *Раздел 2.5.25*). |
| **IFZ280** | истина, если ассемблер находится в режиме Z280 (см. *Раздел 2.5.25*). |
| **IFDEF symbol** | истина, если символ определен или объявлен внешним. |
| **IFNDEF symbol** | истина, если символ не определен или не объявлен внешним. |
| **IFB <arg>** | истина, если arg пуст. Угловые скобки вокруг arg обязательны. |
| **IFNB <arg>** | истина, если arg не пуст. Используется для тестирования при подаче фиктивных параметров. Угловые скобки вокруг arg обязательны. |
| **IFIDN <arg1>,><arg2>** | истина, если строка arg1 идентична строке arg2. Угловые скобки вокруг arg1 и arg2 необходимы. |
| **IFDIF <arg1>,><arg2>** | истина, если строка arg1 отличается от строки arg2. Угловые скобки вокруг arg1 и arg2 необходимы. |

Все условные выражения используют следующий формат:

**IFxx [аргумент]**

**.**

**.**

**.**

**[ELSE**

**.**

**.**

**. ]**

**ENDIF**

Условные выражения могут быть вложены максимум на 10 уровней. Любой аргумент условного выражения должен быть известен на проходе 1, чтобы избежать ошибок V и неправильной оценки. Для IF, IFT и IFF выражение должно включать значения, которые были определены ранее, и выражение должно быть абсолютным. Если имя определено после IFDEF или IFNDEF, то проход 1 считает это имя неопределенным, но оно будет определено на проходе 2.

#### 2.5.28.1 ELSE

Каждая условная псевдо-операция может опционально использоваться с псевдо-операцией ELSE, которая позволяет генерировать альтернативный код при наличии противоположного условия. Только один ELSE разрешен для данного IF, и ELSE всегда привязан к самому последнему, открытому IF. Условное выражение с более чем одним ELSE или ELSE без условного выражения вызовет ошибку C.

#### 2.5.28.2 ENDIF

Каждый IF должен иметь соответствующий ENDIF, чтобы завершить условное выражение. В противном случае в конце каждого прохода генерируется ошибка T. ENDIF без совпадения IF вызывает ошибку C.

### 2.5.29 Псевдо-операции управления листингом

Вывод в файл листинга может управляться псевдо-операцией LIST:

**LIST option,option,...**

Если листинг не составляется, то псевдо-операция LIST не оказывает никакого эффекта. В одном операторе LIST можно указать несколько параметров, разделенных запятыми. Возможны следующие варианты:

**OFF** Подавляет листинг до тех пор, пока не появится команда LIST ON.

**ON** Включает файл листинга после инструкции LIST OFF. Это условие по умолчанию.

**COND** Включает листинг ложных условных блоков. Это условие по умолчанию.

**NOCOND** Подавляет листинг ложных условных обозначений.

**SYMBOL** Генерирует таблицу символов в конце листинга. Это условие по умолчанию.

**NOSYMBOL** Подавляет печать таблицы символов в конце файла листинга.

**SORT** Сортирует таблицу символов, сгенерированной с помощью опции SYMBOL. Это условие по умолчанию.

**NOSORT** Не сортирует таблицу символов. Это обычно приводит к сокращению времени ассемблирования.

**MACROS** Выводит полный текст макросов для всех расширений MACRO/REPT/IRP/IRPC.

**NOMACROS** Подавляет листинг всего текста и кода, создаваемого макросами.

**XMACROS** Выводит только исходные строки макросов, которые генерируют объектный код. Это условие по умолчанию.

Для совместимости с RMAC и M80 также распознаются следующие псевдо операторы управления списком:

**.XLIST** то же самое, что и LIST OFF

**.LIST** такой же, как LIST ON

**.LALL** то же самое, что и LIST MACROS

**.SALL** то же самое, что LIST NOMACROS

**.XALL** то же самое, что и LIST XMACROS

### 2.5.30 Псевдо-операции по перемещению

ZSM4 поддерживает все четыре перемещаемые области, определенные форматом REL: ASEG (абсолютный), CSEG (кода), DSEG (данных) и именованных COMMON.

Режим по умолчанию для ассемблера - перемещаемый код. То есть ассемблирование начинается с автоматически выполняемого CSEG и счетчика местоположения в режиме перемещаемого кода, указывающего на местоположение 0 в памяти перемещаемого сегмента. Все последующие инструкции будут ассемблированы в соответствующий памяти сегмента кода до тех пор, пока не будет выполнена псевдо-операция ASEG или DSEG или COMMON. Например, первый обнаруженный DSEG устанавливает счетчик местоположения в нулевое местоположение в памяти перемещаемого сегмента данных. Следующий код ассемблируется в сегменте памяти, относящемся к данным. Если обнаружится следующий CSEG, счетчик местоположения вернется к следующему свободному местоположению в перемещаемом сегменте кода и так далее.

Псевдо-операции ASEG, DSEG, CSEG никогда не имеют операндов. Если вы хотите изменить текущее значение счетчика местоположений, используйте псевдо-операцию ORG.

#### 2.5.30.1 Псевдо-операция ORG

В любой момент значение счетчика местоположения может быть изменено с помощью псевдо-операции ORG. Форма оператора ORG следующая:

**ORG exp**

где значение exp будет новым значением счетчика местоположения в текущем режиме. Все имена, используемые в exp, должны быть известны на проходе 1, а значение exp должно быть либо абсолютным, либо в текущем режиме счетчика местоположения. Например, операторы

**DSEG**

**ORG 50**

установят перемещаемый счетчик данных на 50 относительно начала памяти перемещаемого сегмента данных.

### 2.5.31 Перемещение перед загрузкой

Две псевдо-операции .PHASE и .DEPHASE, позволяют коду располагаться в одной области, но выполняться только в другой, указанной области.

Например:

**0000' .PHASE 100H**

**0100 CD 0106 FOO: CALL BAZ**

**0103 C3 0007' JP ZOO**

**0106 C9 BAZ: RET**

**.DEPHASE**

**0007' C3 0005 ZOO: JP 5**

Все метки внутри блока .PHASE определяется как абсолютная величина от начала координат фазовой области. Код, однако, загружается в текущей области (то есть от 0000' в данном примере). Код внутри блока может быть позже перемещен в 100H и выполнен.

## 2.6 МАКРОСЫ и Блок Псевдоопераций

Макрокоманды, предоставляемые ZSM4, включают в себя три псевдооперации повтора: repeat (REPT), indefinite repeat (IRP) и indefinite repeat character (IRPC). Также предусмотрена операция определения макроса (MACRO). Каждая из этих четырех макро-операций завершается псевдо-операцией ENDM.

### 2.6.1 Условия

Для целей обсуждения макросов и блочных операций будут использоваться следующие термины:

1. фиктивный используется для представления фиктивного параметра. Все фиктивные параметры являются юридическими символами, которые появляются в теле макроса расширения.
2. dummylist - это список фиктивных параметров, разделенных запятыми.
3. arglist - это список аргументов, разделенных запятыми. arglist должен быть разделен угловыми скобками. Две угловые скобки без промежуточных символов (<>) или две запятые без промежуточных символов вводят в список аргумент null. В противном случае аргумент представляет собой символ или ряд символов, заканчивающихся запятой или >. С угловыми скобками, вложенными в arglist, один уровень скобок удаляется при каждом использовании заключенного в скобки аргумента в arglist (см. Пример, Раздел 2.6.5.) Строка в кавычках является приемлемым аргументом и передается как таковая. Если не заключено в скобки или строку в кавычках, начальные и конечные пробелы удаляются из аргументов.
4. paramlist используется для представления списка фактических параметров, разделенных запятыми. Никаких разделителей не требуется (список заканчивается концом строки или комментарием), но правила ввода нулевых параметров и вложенных скобок такие же, как описано для arglist (см. Пример, Раздел 2.6.5.)

### 2.6.2 REPT-ENDM

Синтаксис:

**REPT exp**

**.**

**.**

**.**

**ENDM**

Блок операторов между REPT и ENDM повторяется exp раз. exp оценивается как 16-битное беззнаковое число. Если exp содержит какие-либо внешние или неопределенные термины, создается ошибка. Пример:

**X DEFL 0**

**REPT 10 ;генерирует DB 1 - DB 10**

**X DEFL X+1**

**DB X**

**ENDM**

### 2.6.3 IRP-ENDM

Синтаксис:

**IRP dummy,<arglist>**

**.**

**.**

**.**

**ENDM**

Параметр arglist должен быть заключен в угловые скобки. Количество аргументов в arglist определяет количество повторений блока операторов. Каждое повторение заменяет следующий элемент в arglist для каждого вхождения манекена в блоке. Если arglist равен null (т. е. <>), то блок обрабатывается один раз с каждым dummy удаленным вхождением dummy. Например:

**IRP X,<1,2,3,4,5,6,7,8,9,10>**

**DB X**

**ENDM**

генерирует те же байты, что и в примере REPT.

### 2.6.4 IRPC-ENDM

Синтаксис:

**IRPC dummy,[<]string[>]**

**.**

**.**

**.**

**ENDM**

IRPC похож на IRP, но arglist заменяется строкой текста, а угловые скобки вокруг строки являются необязательными. Операторы в блоке повторяются один раз для каждого символа в строке. Каждое повторение заменяет следующий символ в строке для каждого вхождения манекена в блоке. Например:

**IRPC X,0123456789**

**DB Х+1**

**ENDM**

генерирует тот же код, что и в двух предыдущих примерах.

### 2.6.5 MACRO

Часто удобно иметь возможность генерировать заданную последовательность операторов из различных мест в программе, даже если при каждом использовании последовательности могут потребоваться различные параметры. Эта возможность обеспечивается оператором MACRO.

Форма

**name MACRO dummylist**

**.**

**.**

**.**

**ENDM**

где name соответствует правилам формирования символов и является именем, которое будет использоваться для вызова макроса. Параметры dummy в dummylisэто t - параметры, которые будут изменяться (заменяться) при каждом вызове MACRO. Операторы перед ENDM составляют тело макроса. Во время сборки макрос раскрывается каждый раз, когда он вызывается, но, в отличие от REPT/IRP/IRPC, MACRO не раскрывается при его обнаружении.

Форма вызова макроса

**name paramlist**

где name - это имя, указанное в определении MACRO, а параметры в paramlist будут заменять фиктивные в MACRO - dummylist один к одному. Количество элементов в dummylist и paramlist ограничено только длиной строки. Количество параметров, используемых при вызове макроса, не обязательно должно совпадать с количеством фиктивных в dummylist. Если параметров больше, чем фиктивных, то дополнительные параметры игнорируются. Если их будет меньше, то лишние фиктивные будут обнулены. Собранный код будет содержать код расширения макроса после каждого вызова макроса.

Примечание

Фиктивный параметр в MACRO/REPT/IRP/IRPC всегда распознается исключительно как фиктивный параметр. Имена регистров, такие как A и B, будут изменены в расширении, если они были использованы в качестве фиктивных параметров.

Вот пример определения макроса, который определяет макрос под названием FOO:

**FOO MACRO X**

**Y DEFL 0**

**REPT X**

**Y DEFL Y+1**

**DB Y**

**ENDM**

**ENDM**

Этот макрос генерирует тот же код, что и предыдущие три примера, когда вызов

**FOO 10**

исполняется.

Другой пример, который генерирует тот же код, иллюстрирует удаление одного уровня скобок при использовании аргумента в качестве arglist:

**FOO MACRO X**

**IRP Y,<X>**

**DB Y**

**ENDM**

**ENDM**

Когда вызов

**FOO <1,2,3,4,5,6,7,8,9,10>**

сделано, расширение макроса выглядит так:

**IRP Y,<1,2,3,4,5,6,7,8,9,10>**

**DB Y**

**ENDM**

### 2.6.6 ENDM

Каждый псевдо-оператор REPT, IRP, IRPC и MACRO должен быть завершен псевдо- оператором ENDM. В противном случае в конце каждого прохода генерируется ошибка T. Несопоставимый ENDM вызывает ошибку O.

### 2.6.7 EXITM

Псевдо-операция EXITM используется для завершения вызова REPT/IRP/IRPC или MACRO. При выполнении EXITM расширение выводится немедленно, а любое оставшееся расширение или повторение не генерируется. Если блок, содержащий EXITM, вложен в другой блок, внешний уровень продолжает расширяться.

### 2.6.8 LOCAL

Синтаксис:

**LOCAL dummylist**

Псевдо-операция LOCAL разрешена только внутри определения MACRO. При выполнении LOCAL ассемблер создает уникальный символ для каждого манекена в dummylist и заменяет этот символ для каждого вхождения манекена в расширение. Эти уникальные символы обычно используются для определения метки внутри макроса, тем самым устраняя многократно определенные метки на последовательных расширениях макроса. Символы, созданные ассемблером, варьируются от ??0001 до ??FFFF. Поэтому пользователи должны избегать имена типа ??nnnn для своих собственных символов. Если используются операторы LOCAL, они должны быть первыми операторами в определении MACRO.

### 2.6.9 Специальные макро-операторы и формы

**&** Амперсанд используется в макро-расширении для объединения текста или символов. Фиктивный параметр, который находится в строке в кавычках, не будет заменен в расширении, если ему непосредственно не предшествует &. Чтобы сформировать символ из текста и манекена, поместите & между ними. Например:

**ERRGEN MACRO X**

**ERROR&X: PUSH BC**

**LD C,'&X'**

**JP ERROR**

**ENDM**

В этом примере вызов ERRGEN A будет генерировать:

**ERRORA: PUSH BC**

**LD C,'A'**

**ERROR JP**

**;;** В блочной операции комментарий, которому предшествуют две точки с запятой, не сохраняется как часть расширения (т. е. он не будет отображаться в списке даже под .LALL). Однако комментарий, которому предшествует одна точка с запятой, будет сохранен и появится в расширении.

**!** Когда восклицательный знак используется в аргументе, следующий символ вводится буквально (т. е. !; и <;> эквивалентны).

**NUL** NUL-оператор, который возвращает true, если его аргумент (параметр) равен null. Остаток строки после NUL считается аргументом к NUL. Условное

**IF NUL аргумент**

имеет значение false, если во время расширения первый символ аргумента не является точкой с запятой или возвращением каретки. Рекомендуется проводить тестирование нулевых параметров с помощью условных выражений IFB и IFNB.

**%** Знак процента используется только в аргументе макроса. % преобразует выражение, которое следует за ним (обычно символ) , в число в текущем корне. Во время расширения макроса число, полученное в результате преобразования выражения, заменяется фиктивным. Использование специального оператора % позволяет вызывать макрос по значению. (Обычно вызов макроса-это вызов по ссылке с текстом аргумента макроса, заменяющего именно фиктивный.)

Выражение, следующее за %, должно соответствовать тем же правилам, что и псевдо-операция DS (Define Space). Требуется допустимое выражение, возвращающее неперемещаемую константу.

В приведенном ниже примере LB (аргумент в MAKLAB) обычно заменяется Y (аргумент в MACRO) в виде строки. Однако % приводит к тому, что LB преобразуется в неперемещаемую константу, которая затем заменяется на Y. Без специального оператора % результатом сборки будет "Error LB", а не "Error 1" и т. д.

**MAKLAB MACRO Y**

**ERR&Y: DB 'Error &Y',0**

**ENDM**

**MACRO MAKERR X**

**LB DEFL 0**

**REPT X**

**LB DEFL LB+1**

**MAKLAB %LB**

**ENDM**

**ENDM**

При вызове как MAKERR 3, ассемблер будет генерировать:

**ERR1: DB 'Error 1',0**

**ERR2: DB 'Error 2',0**

**ERR3: DB 'Error 3',0**

## 2.7 Ошибки ZSM4

Ошибки ZSM4 обозначаются односимвольным флагом в первом столбце файла листинга. Если файл листинга не печатается на терминале, то каждая ошибочная строка также печатается или отображается на терминале. Ниже приведен список кодов ошибок ZSM4:

|  |  |
| --- | --- |
| **A** | Слишком много операторов IF  Достигнут максимальный уровень условной вложенности. |
| **B** | ENDIF без оператора IF |
| **C** | ELSE без оператора IF |
| **D** | Ошибка диапазона относительного jump |
| **E** | Ошибка выражения  Недопустимый оператор, два последовательных оператора и т. д. |
| **L** | Недопустимый идентификатор  Идентификатор содержит недопустимые символы. |
| **M** | Символ определен несколько раз |
| **N** | Недопустимый код операции |
| **O** | Плохой код операции или нежелательный синтаксис  ENDM, LOCAL вне блока; DELF, EQU или MACRO без имени; плохой синтаксис в коде операции; или плохой синтаксис в выражении (несоответствующие скобки, кавычки, последовательные операторы и т.д.). |
| **P** | Ошибка фазы  Значение , метка или имя EQU отличается на проходе 2. |
| **Q** | Отсутствует закрывающая кавычка  неправильно закрытая строка в операторе DB и т.д. |
| **R** | Ошибка перемещения  Недопустимое использование перемещения в выражении, например abs-rel. Области данные, кода и COMMON можно перемещать. |
| **T** | Отсутствует ENDM или ENDIF  Обычно появляется в конце листинга, указывая на неоконченное условие или макрос. |
| **U** | Неопределенный символ  Символ, на который ссылается выражение, не определен. |
| **V** | Ошибка значения |
| **W** | Переполнение таблицы символов |
| **Z** | Деление на ноль  Вычисляемое выражение содержит деление на ноль. |

## 2.8 Совместимость с другими ассемблерами

Мы позаботились о том, чтобы сделать ZSM4 максимально совместимым с M80 Microsoft и RMAC Digital Research. Есть, однако, некоторые различия, которые перечислены ниже:

* Псевдо-оператор .TFCOND (переключение списка ложных условных обозначений) не реализован.
* ZSM4 позволяет нескольким псевдо-операторам TITLE появляться в исходном файле, в то время как M80 позволяет только один.
* Если псевдо-оператор NAME не указан, M80 использует первое слово оператора TITLE в качестве имени модуля по умолчанию, в то время как ZSM4 использует имя исходного файла.
* ZSM4 хранит до 8 символов имен символов в REL-файлах (см. переключатель параметров /Sn в разделе 2.2.1), в то время как M80 и RMAC хранят до 6 символов.
* ZSM4 не поддерживает расширения формата REL, такие как M80 v3.44 или SLR. Во всяком случае, лишь немногие компоновщики поддерживают их.
* Порядок байтов символьных констант, состоящих из двух символов, такой же, как и в RMAC, но в M80 является противоположным (см. Раздел 2.3.3). Таким образом, оператор типа

LD HL,'AB'

загрузит в регистр L 'A' и в регистр H 'B'.

## 2.9 Дополнительные примечания

Режим Z280 имеет несколько предостережений:

* Операнд инструкций, принимающий 8 или 16 разрядное значение или смещение, должен быть известен на проходе 1. Примерами таких инструкций являются:

ADD A,(IX+d8) и ADD A,(IX+d16)

LD (HL),d8 и LD (HL),d16

и т.д.

Это потому, что разные формы одной и той же инструкции генерируют разные опкоды и/или имеют разную длину. Если операнд не известен на проходе 1, то ассемблер должен будет угадать или выбрать произвольно одну из форм; когда размер операнда, наконец, известен на проходе 2, произойдет фазовая ошибка, если его размер не совпадает с тем, который угадан на проходе 1. Чтобы устранить такое ограничение, ассемблер должен будет выполнять дополнительные проходы над исходным кодом, становясь более сложным и медленным.

* Как описано в Руководстве Z280, синтаксис четырех расширенных инструкций EPUF, EPUI, EPUM и MEPU неясен. Инструкции требуют четырехбайтового аргумента шаблона, который загружается в сопроцессор или декодируется им. Руководство Z280 даже определяет формат для него, но нигде нет описания или примера того, как на самом деле выглядит мнемоническая инструкция. Даже если шаблон предназначен для сопроцессора, процессор Z280 может декодировать его часть. В частности, количество байтов, передаваемых инструкциями EPUM и MEPU, извлекается из четвертого байта шаблона, и, таким образом, похоже, что шаблон должен быть частью мнемонической инструкции.

В этой версии ZSM4 байты шаблона задаются отдельно от инструкции и поэтому пользователь должен следовать соответствующему формату. Например:

**EPUM (HL)**

**DEFB t1,t2,t3,t4**

где t1, t2, t3 и t4 - байты шаблона.

* Псевдо-операторы нечетного выравнивания .EVEN и .ODD (см. раздел 2.5.27) требуют соответствующей поддержки компоновщика. В частности, любой код или сегмент данных, использующий эти псевдо-операторы, должен начинаться с четной границы адреса. Если компоновщик не предусматривает такой функции, то тот же результат может быть достигнут путем связывания всех сегментов четного размера и связывания конечной программы, начинающейся по четному адресу. Чтобы убедиться, что все сегменты имеют четный размер, завершите их псевдо-оператором .EVEN.

# A Работа в CP/M

Исполняемый файл ZSM4 CP/M имеет размер около 22 Кбайт, поэтому для запуска ассемблера рекомендуется использовать систему с большим TPA, особенно если требуется собрать большие исходные файлы или если исходные файлы используют большое количество макросов. С другой стороны, формат таблицы символов намного компактнее, чем, например, M80.

Спецификации файлов соответствуют стандартной конвенции CP/M:

**drive:filename.ext**

Обязательным является только filename, все остальные поля являются необязательными и по умолчанию имеют следующие значения:

**drive** текущий дисковод

**ext** MAC для источника, REL для вывода объектного файла и PRN для файла листинга

Если командной строки нет, ассемблер перейдет в интерактивный режим и запросит командную строку:

**A>ZSM4**

**Z80/Z180/Z280 Macro-Assembler V4.1**

**\***

В интерактивном режиме можно ввести несколько команд для обработки нескольких файлов без необходимости перезагрузки ассемблера каждый раз.

В CP/M 3.0 и MP/M дата и время ассемблирования отображаются в листинге.

# B Работа в RSX180

ZSM4 построен с именем задачи по умолчанию ...MAC; например, в команде INStall можно указать другое имя:

**INS ZSM4/TASK=...ZSM**

Задача также строится с приращением памяти по умолчанию в размере 8000 байт, чего должно быть достаточно для сборки файлов малого и среднего размера. Однако для больших файлов указание большего приращения памяти с помощью команды INStall не является необходимым, так как ассемблер может автоматически запрашивать больше памяти у системы, если/когда это становится необходимым. Если запрос не будет удовлетворен, то и ошибка будет выведена на экран.

Спецификации файлов соответствуют стандартному соглашению RSX180:

**dev:[каталог]filename.ext;vers**

Обратите внимание, что квадратные скобки выше являются частью синтаксиса и поэтому необходимы при указании каталога. Только filename является обязательным, все остальные поля являются необязательными и по умолчанию имеют следующие значения:

|  |  |
| --- | --- |
| dev | SY0: (устройство входа пользователя) |
| каталог | каталог текущего пользователя |
| ext | MAC для источника, OBJ для объектного файла и LST для файла листинга |
| vers | самая высокая (последняя) версия файла. |

Если задача была установлена, ее можно вызвать, введя имя установленной задачи, например

**MAC [команда]**

если задача не установлена, ее можно вызвать с помощью команды RUN, например:

**RUN $ZSM4**

Если командной строки нет, ассемблер перейдет в интерактивный режим и запросит командную строку:

>MAC

MAC>

ZSM4 также поддерживает косвенные командные файлы под RSX180 стандартным способом:

**MAC @filespec**

где filespec - допустимая спецификация командного файла. Если расширение не указано, предполагается CMD. Командный файл просто содержит список команд, выполняемых ZSM4, по одной в строке, и может, в свою очередь, включать вызовы дополнительных командных файлов до уровня вложенности 3.

# C Работа в UZI180

Поскольку UZI180 имеет встроенный эмулятор CP/M, версия ZSM4 для CP/M может быть запущена непосредственно из командной строки UZI180. Используется обычное ограничение формата имени файла 8.3 CP/M.

В настоящее время разрабатывается собственная версия ZSM4 для UZI180.

# D Работа в Linux или Windows

CP/M версия ZSM4 может использоваться для кросс-сборки в системе Linux или Windows с помощью программы zxcc Джона Эллиота {John Elliott}

(<http://www.seasip.demon.co.uk/Unix/Zxcc/index.html> ). Программа zxcc эмулирует как среду Z80, так и среду CP/M таким образом, что приложение CP/M может быть запущено из командной строки Linux или Windows, как если бы это было собственное приложение.

Поскольку ZSM принимает исходные файлы как с CP/M, так и с конечными соглашениями Unix, преобразование в формат CP/M не требуется.

Будучи программой CP/M, ZSM4 накладывает некоторые ограничения:

* Имена файлов ограничены форматом 8.3 и указываются строчными буквами.
* Если исходные файлы ВКЛЮЧАЮТ дополнительные файлы (см. Раздел 2.5.14), то команда zxcc, вызывающая ZSM4, должна быть запущена в каталоге, где находятся включаемые файлы.
* Коммутаторы опций должны быть экранированы должным образом (см. Документацию zxcc для получения подробной информации), или же оболочка Unix может интерпретировать их как спецификации каталога.

Примеры:

zxcc zsm4 test,test=test

zxcc zsm4 -"=test /L"

Обратите внимание, как вся команда во втором примере экранируется, так как она содержит параметр /L.

Программа zxcc позволяет вызывать ZSM4 из файла Makefile, например:

.PREFIX:

.PREFIX: .mac .rel

SRCS = main.mac functions.mac misc.mac

OBJS = $(SRCS:.mac=.rel)

all: example.com

$(OBJS): %.rel: %.mac \*.inc

zxcc zsm4 -" =$</l"

example.com: $(OBJS) mylib.lib

zxcc link -"$@=main,functions,misc,mylib.lib[s]"