**Перемещающий макроассемблер ZMAC  
 и редактор связей ZML**

Авторское право 1988, 1989 Аль Хоули {A.E. Hawley}

Торговые марки: **Z80**, **Z180**, Zilog, Inc.; **HD64180**, Hitachi; **CP**/**M**, **ZSID**, Digital Research, Inc.

Предупреждение

*Эта программа и инструкция по эксплуатации защищены американским Законом об авторском праве (Заголовок 17 свода законов США). Несанкционированные продажи и/или воспроизведение могут привести к тюремному заключению до одного года и штрафам до 10,000$ (17 USC 506). Нарушители авторского права могут также подвергнуться гражданско-правовой ответственности.*

Преднамеренно пустая страница

**Содержание**

1 Введение 5

Установка 6

Вызов ZMAC 6

Службы ZCPR3/33/34 8

Параметры командной строки 8

2 Введение в ассемблер и редактор связей 11

Процесс ассемблирования/компоновки 12

Объектные файлы HEX и REL 13

3 Форматы исходных и выходных файлов 15

Исходные файлы 15

Поле метки 16

Поле оператора 16

Поле операнда 17

Поле комментария 19

Выходные файлы 20

Листинг ассемблера 20

Листинг таблицы символов 22

Файл регистрации ошибок 22

4 Выражения 24

Числа 24

Символьные константы 25

Символы и метки 26

Счетчик адреса 26

Операторы и приоритет операторов 27

Определение перемещаемости 30

5 Псевдо-операции (директивы ассемблера) 31

6 Перемещение и основы перемещения 42

Перемещение 42

Основы перемещения 45

7 Обработка макросов 47

Форма макросов 48

Макросы псевдо-операции 49

Макро-операторы 50

Встроенные (RPT) макросы 50

Сохраняемые макросы 52

Определение и вложенность 52

Макро-вызов 52

Правила макрорасширения 54

8 Компоновка с помощью ZML 55

Вызов 55

Основы перемещения, порядок загрузки 57

Неинициализированные данные 59

Файлы PRL и SPR 59

Реализация $MEMRY 61

Синтаксис команды ZML 62

Параметры спецификации файла (таблица) 63

Параметры (таблица) 63

Формальный синтаксис (таблица) 64

Параметры, полное описание 65

Изменение конфигурации ZML 68

9 Настройка параметров с помощью ZCNFG 69

Вызов ZCNFG 69

Параметры ZMAC 70

Параметры ZML 74

Приложение 1 - Глоссарий 76

Приложение 2 – Список литературы 77

Приложение 3 - Формат объекта REL 80

Приложение 4 - Коды операций 83

Приложение 5 - Сообщения об ошибках 86

Предметный указатель 90

# Введение

ZMAC - перемещающий макроассемблер для компьютеров основанных на процессорах Zilog Z180, Hitachi HD64180 и Z80. Он читает текстовые файлы, содержащие операторы ассемблера, и производит файлы, содержащие объектный код, листинг распечатки кода и источника, листинг таблицы символов и файл регистрации ошибок. Объектный код может быть в формате Intel HEX или Microsoft REL. Объектные файлы HEX подходят для ввода с помощью загрузчика типа утилиты CP/M LOAD или общедоступной утилиты MLOAD написанной Роном Фаулером {Ron Fowler}. Объектные файлы REL совместимы с ZML и другими редакторами связей, которые принимают объектные файлы в формате Mrel.

ZMAC требует минимум около 30 Кбайт памяти TPA для выполнения полезного ассемблирования. Большее количество доступной памяти позволяет выполнять ассемблирование гораздо более крупных и сложных программ.

ZMAC работает в среде CP/M. Он автоматически обнаружит и использует расширенные возможности более мощной среды Z-system, для которой был разработан.

Если вы еще этого не сделали, остановитесь прямо сейчас и сделайте по крайней мере две копии дистрибутивного диска ZMAC! Защитите от записи оригинал и одну из копий и используйте остающуюся копию в качестве рабочего диска. Установка должна быть выполнена только на рабочей копии(ях).

**Файлы дистрибутива**

Вы найдете следующие файлы на своем основном диске ZMAC:

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| ZMAC.COM | Ассемблер; |
| ZMLNK.COM | Редактор связей; |
| ZCNFG.COM | Утилита конфигурации для ZMAC и ZMLNK; |
| ZMAC.CFG | Оверлей конфигурации используется ZCNFG; |
| ZMLNK.CFG | Оверлей конфигурации используется ZCNFG; |
| FILES.DOC | Текстовый файл, содержащий описание любых других файлов, находящихся на диске; |
| Z80.HLP | Разнообразные данные о кодах операций в файле справки ZCPR3. |

Z80.HLP защищен авторским правом © 1988 Кэмерона Коттрилла {Cameron W. Cotrill}.

На дистрибутивном диске присутствуют другие файлы с расширением файла Z80. Эти файлы иллюстрируют многие структуры и псевдооперации, предусмотренные ZMAC. Файл общественного достояния "MUSICBOX" написанный Эдом Крампом {Ed Cramp} является отличным примером применения многих ассемблерных инструкций, доступных в ZMAC.

Вы найдете полезными следующие файлы общественного достояния. Они доступны для скачивания с Z-узлов и многих других CP/M-ориентированных электронных досок объявлений (BBS).

|  |  |
| --- | --- |
| MLOAD24.COM | Загрузчик HEX; |
| XIZI-3.LBR | Переводчик мнемоник Zilog <-> Intel; |
| XLATE5.LBR | Ранняя версия XIZI-3; |
| SEQIO22.LIB | Библиотека макросов, полезная в качестве примеров; |
| Z8E.LBR | Экранно-ориентированный отладчик; |
| PROLNK15.LBR | Microsoft совместимый редактор связей. |

Следующие полезные утилиты НЕ являются общественным достоянием. Они могут быть приобретены у вашего поставщика ZMAC/ZML.

|  |  |
| --- | --- |
| ZTOI.COM | Преобразование мнемоник Zilog в Intel; |
| ITOZ.COM | Преобразование мнемоник Intel в Zilog; |
| ZLIB.COM | Менеджер библиотеки для редактора связей REL; |
| ZREF.COM | Генератор перекрестных ссылок для источников на Z80/HD64180; |
| DSD.COM | Экранно-ориентированный отладчик. Лучший. |

## Установка

ZMAC готов работать в системах CP/M, ZCPR33 или ZCPR34. Если вы используете ZCPR3, то вы должны использовать Z3INS.COM для установки ZMAC.COM. Z3INS не требуется для версий ZCPR выше 3.0.

Утилита установки ZCNFG.COM на вашем рабочем диске отображает выбираемые пользователем настройки по умолчанию и предусматривает изменение в интерактивном режиме.

Если хотите, вызовите ZCNFG сейчас, чтобы увидеть, какие параметры активны в настоящее время. Программа сама выдает инструкции и подсказки. Обычно, по умолчанию установлены подходящие параметры. Вам не нужно менять какие-либо из них прямо сейчас. Выйдите из ZCNFG с помощью команды 'Q', ничего не меняя. После ознакомления с ZMAC и содержанием этого руководства, вы можете пожелать выполнить реконфигурацию. См. [*Главу 9*](#_Настройка_параметров_с) в этом руководстве, когда будете готовы установить параметры с помощью ZCNFG.

ZMAC готов к использованию! Если вы опытный программист на языке ассемблера, вы должны ознакомится с главой [Конфигурация](#_Настройка_параметров_с_1) и просмотреть главу [Псевдооперации](#_Псевдо-операции_(директивы_ассемблера)_4).

Если вы новичок в программировании на языке ассемблера, вы должны прочитать это руководство и использовать как можно больше ссылок в *Приложении 2*. Не бойтесь перечитывать до тех пор, пока не понимаете!

ZMAC и ZML, вероятно, будет видоизменяться. Если вы не удосужились отправить регистрационную форму, можете быть уверены, что ваше имя отсутствует в списке тех, кого уведомляют о новых выпусках!

## Вызов ZMAC

Простейший вызов, это ввести (в командной строке):

ZMAC

или ZMAC / <== '?' может быть использован вместо '/'

*Примечание*: '<==' означает комментарий, которй не является частью набранной строки.

Результатом будет экран справки, который показывает формат команды и список параметров командной строки, вместе с их текущими значениями по умолчанию. Значения параметров по умолчанию являются важной информацией. Они показывают, что сделает ассемблер, без определения параметров. Параметры будут обсуждаться дальше на следующих страницах.

Полный синтаксис командной строки ZMAC:

ZMAC \_ [DIR:] FN [.TYP] [,o...]

\_ (подчеркивание) означает, что в этой позиции требуется пробел. Пробелы в образце командной строки выше присутствуют для удобочитаемости. Они не являются частью командной строки.

[] означает, что заключенный элемент является необязательным.

FN это имя исходного файла для ассемблирования. Это единственный элемент, не имеющий значения по умолчанию.

DIR может быть буквой диска (A, B, и т.д.), или областью пользователя (0, 1 ...), или стандартной комбинацией ZCPR как A1, C4 и т.д. Пользователи ZCPR3 распознают эту форму как используемую ZCPR3. Она идентична. Если ваша система поддерживает именованные каталоги, DIR может быть именем каталога.

'o' расшифровывается как параметр. Параметры командной строки обсуждаются ниже.

... (многоточие) Предыдущий элемент может повторяться несколько раз.

Запятая, предшествующая каждому параметру, отмечает место, где требуются один или несколько разделителей. Допустимыми разделителями являются: запятая, '/', пробел и символ табуляции. Это позволяет вам вводить параметры в различных форматах. Все последующие команды будут интерпретированы правильно:

ZMAC NAME o o o <== Одиночные пробелы или символы табуляции

ZMAC NAME o o o <== Несколько пробелов или символов табуляции

ZMAC NAME/o/o /o <== Наклонные черты и дополнительные пробелы

ZMAC NAME /o o o <== Общая форма ZCPR3

ZMAC NAME /o,o,o <== Общая форма ZCPR3

Если DIR не определен, ZMAC выполняет поиск файла в каталоге зарегистрированном в настоящее время. Если указан альтернативный DU: с помощью программы ZCNFG, то этот каталог также будет участвовать в поиске файла, не найденного в текущем (или определенном) DIR. Поиск в этом альтернативном каталоге особенно полезен для нахождения файлов, на которые имеются ссылки в основном исходном файле в псевдо-операторах .INCLUDE, и более подробно рассматривается в [*Главе 5*](#_Псевдо-операции_(директивы_ассемблера)_3).

Если поле TYP не определено, предполагается расширение файла Z80. ZMAC определит местоположение файла и ассемблирует содержимое для производства объектного файла FN.REL.

Если во время ассемблирования возникнут ошибки - будет произведен файл с названием FN.ERR, вместе с сообщениями об ошибках на консоли.

ZMAC повторно исполняемый файл. Это означает, что вы можете использовать ZCPR3 команду 'GO' (с соответствующим хвостом команды) для ассемблирования следующего исходного файла. Вы можете использовать в своих интересах это свойство в: ALIAS, файле Submit или сценарии ZEX, при наличии нескольких модулей для ассемблирования. Преимущество, безусловно, состоит в том, что вы экономите время, требуемое для перезагрузки ZMAC с диска для каждого вызова. Диск, содержащий ZMAC, даже не должен присутствовать для повторного вызова. Он, например, может быть заменен новым, имеющим свободное место для выходных файлов для последующего ассемблирования.

**Спецификации файла**

Когда делается ссылка на имя файла в командной строке или в источнике на ассемблере, допустима полная спецификация в стиле ZCPR3, как в образце командной строки выше. Поля имени (FN) и расширения не могут содержать символы из следующего списка:

пробел <управляющий символ> = \_ : ; < > , /

**Генерация выходного файла по умолчанию**

ZMAC может быть настроен так, что любой из четырех возможных выходных файлов производится без объявления в командной строке. В этом случае объекты по умолчанию обеспечиваются (также с учетом конфигурации) для всех частей спецификации файла. Любая часть спецификации файла, указанная в командной строке (в параметрах O, P, E и S), переопределяет значения по умолчанию и обязательно используется при производстве этого файла.

Имена файлов по умолчанию появляются в столбце 'default filespec' экрана справки. Их объявления по умолчанию обозначены 'YES' (Да) или 'NO' (Нет) в столбце, озаглавленном 'make it?' (делать это?).

## Службы ZCPR3/33/34

Если ZMAC вызывается в операционной системе, которая включает в себя ZCPR3 - присутствуют следующие возможности. Если ZCPR3 был сконфигурирован, чтобы исключить некоторый модуль системы (например, именованный каталог), то функции требующие эту услугу не будут работать.

ZMAC устанавливает ZCPR3 флаг ошибки программы, если ошибки возникают во время ассемблирования.

При возникновении первой ошибки, ZMAC записывает следующие данные в буфер сообщения Z3: полное имя файла, диск и номер пользователя файла, номер строки в файле и однобуквенный код ошибки, который указывает тип ошибки. Буква соответствует одной из списка сообщений об ошибках находящегося в конце этого руководства. В псевдониме (Alias) эта информация может быть извлечена из буфера сообщения и передана редактору, например PMATE, как часть хвоста команды.

Предупреждающие сообщения не вызывают ни одно из вышеуказанных действий, но отображаются в листинге .PRN.

Экран справки получает доступ к внешнему FCB Z3, чтобы включить имя, по которому была вызвана программа. (Вы можете переименовать ZMAC как подсказывает ваша фантазия, и экран справки будет следовать этому примеру.)

## Параметры командной строки

Параметры командной строки отображаются на экране помощи. Есть два типа параметров: те, которые не принимают никаких аргументов и те, которые могут включать спецификацию файла в качестве аргумента. Каждому параметру предшествует по крайней мере один разделитель. Приемлемыми разделителями являются: пробел, символ табуляции, запятая и наклонная черта (/).

Первая группа параметров - C, L и H. Параметры C и L управляют созданием листинга ассемблирования на консоли и устройстве печати, соответственно. Параметр H выбирает тип кода, который будет произведен в объектный файл. Все являются переключателями. Экран справки показывает условие по умолчанию для каждого параметра. Перед конфигурацией C и L имеют значения 'NO' (отсутствие листинга на любом устройстве), и H установлен для объектного кода REL. Если опция H будет включена в список параметров, то будет произведен объектный файл Intel HEX. Если параметры C и L включены в список, то листинг ассемблирования отправляется на консоль и устройство печати.

Вторая группа параметров управляет созданием, именем и местом назначения выходных файлов. Местом назначения и именем управляют (может отсутствовать) спецификация файла. По умолчанию объектные и файлы регистрации ошибок будут созданы, как обозначенные 'YES' на экране справки. Файлы PRN и SYM не создаются. Если параметры O, P, S и E присутствуют без спецификации файла, то они переключают создание связанных с ними выходных файлов. По умолчанию местом назначения создания этих файлов, является зарегистрированный в настоящее время каталог. Ниже приведены некоторые примеры командной строки:

Предполагая, что MYCODE.Z80 находится в текущем каталоге, которым является B0:

1) ZMAC MYCODE <== MYCODE.REL и MYCODE.ERR

2) ZMAC MYCODE O P <== MYCODE.PRN, MYCODE.ERR

3) ZMAC MYCODE O P E <== MYCODE.PRN

4) ZMAC MYCODE /O,P <== Так же, как 2, другие разделители

5) ZMAC MYCODE Oxxxx <== XXXX.REL, MYCODE.ERR

6) ZMAC MYCODE Pwork:test <== MYCODE.REL, WORK:TEST.PRN

7) ZMAC MYCODE O2: P3: <== B2:MYCODE.REL, B3:MYCODE.PRN

8) ZMAC MYCODE O.zrl <== MYCODE.ZRL, MYCODE.ERR

Пример 1 иллюстрирует вызов ассемблирования, который полагается на значения по умолчанию, показанные на экране справки.

Примеры 2 и 3 иллюстрируют, как изолированная буква параметра ПЕРЕКЛЮЧАЕТ создание файла.

Примеры 5 и 8 иллюстрируют определение выходного имени файла.

Примеры 6 и 7 иллюстрируют безусловное создание выходных файлов, если присутствует любая часть спецификации файла. Соответствующие части спецификации по умолчанию заменяются на указанные в командной строке.

Однобуквенные параметры C, L и H действительно не требуют разделителя перед следующим параметром, потому что они не определяют аргумент. Разделитель ДОЛЖЕН следовать за параметрами O, P, E и S, даже если аргумент отсутствует. Такой случай возникает, когда файл PRN не создается по умолчанию, и вы хотите создать один для текущего ассемблирования, не определяя спецификацию файла (будут использоваться значения по умолчанию DU, имени, и расширения). Все следующие команды интерпретируются правильно:

ZMAC NAME CP <== листинг на консоль и NAME.PRN

ZMAC NAME CPb3: <== то же самое, но PRN выводится в каталог B3:

ZMAC NAME CP S.tbl <== также создать файл символов NAME.tbl

ZMAC NAME CP,h <== создать объект HEX если Objfile равен ON.

ZMAC NAME CP4:xxx.lst,h <== файл XXX.LST в области пользователя 4,

по умолчанию dr.

Следующая команда неправильно истолкована, если вы хотите параметры C и H:

ZMAC NAME PCH <== создает CH.PRN

Любой из параметров может быть включен в виде одной буквы. Если используется однобуквенный параметр, он действует как переключатель. Второе появление просто отменяет первое. Для тех параметров, которым требуется аргумент, присутствие параметра вызывает безусловное переключение в состояние ON для этого параметра. т.е., если файл PRN создается по умолчанию то, P<filespec> НЕ выключает создание файла PRN, а определяет имя и место назначения этого файла из спецификации <filespec>. В той же ситуации 'P', с последующим разделителем, отменяет генерацию файла PRN.

Параметры командной строки имеют силу только во время текущего ассемблирования. Они не становятся значениями по умолчанию для будущих компиляций. Действуют только во время текущего ассемблирования. Постоянные значения по умолчанию устанавливаются ZCNFG.

**Статистика ассемблирования**

После завершения ассемблирования ZMAC делает паузу в своем консольном выводе, если интерактивная информация или сообщения об ошибках использовали экран. Это обеспечивает возможность изучения этих сообщений, прежде чем они прокручиваются на экране. После этой паузы (если она возникает), приводятся статистические данные о текущем ассемблировании:

Количество ошибок;

Спецификация входного файла;

Спецификации для всех сгенерированных файлов;

Число ассемблированных строк;

Число определенных макросов;

Число макрорасширений;

Размер каждого сегмента (ASEG, CSEG, DSEG, BLANK COMMON);

Свободная память неиспользуемая при ассемблировании.

Следующие статистические данные сообщаются только если они существуют:

Имя и размер каждого именованного общего сегмента;

Общий размер кода внутри блоков .PHASE - .DEPHASE.

# Введение в ассемблер и редактор связей

**Введение в программирование на ассемблере**

В этой главе приведен обзор взаимосвязи между машинным кодом, который использует ваш компьютер и файлами и программами, используемыми для генерации этого кода. Эта взаимосвязь рассматривается с точки зрения роли, которую играют ассемблер, редактор связей и связанные с ними входные и выходные файлы данных.

С точки зрения ассемблера/редактора связей конечная цель программирования состоит в создании файла данных, пригодного для загрузки по определенному адресу в память компьютера. Процессор (в нашем случае Z80, Z180 или HD64180) может, учитывая адрес расположения этих данных, изменить его, скопировать их по другому адресу или использовать их в качестве инструкций по управлению его действиями и выполнить задачу, требуемую пользователю компьютера (или операционной системе).

Некоторые из первых микрокомпьютеров (Altair и Imsai, например) имели переключатели на передней панели, с помощью которых программа могла быть введена вручную по ОДНОМУ БИТУ ЗА РАЗ(!). Современные компьютеры получают программы и данные с ленты, диска или входного порта. Наиболее важной (и по отношению к нам) является организация данных в виде файла на носителе. Файл логически последователен и содержит определенное число байтов. В потоке байтов нет никаких 'разрывов', несмотря на то, что многие байты могут иметь неопределенные (читайте 'случайные'), значения.

Файлы генерируются работающими программами (которые самими запускаются как файлы!). РЕДАКТОР является такой программой. Такими же являются АССЕМБЛЕРЫ, РЕДАКТОРЫ СВЯЗЕЙ и ЗАГРУЗЧИКИ. Редактор связей и загрузчик (MLOAD является загрузчиком общественного достояния) выполняют подобные функции. Они преобразуют промежуточный код в файле, созданном ассемблером в файл двоичного кода, подходящий для выполнения процессором (после загрузки в память по правильному адресу). Ассемблер получает свой ввод из простого текстового файла, созданного редактором.

Процесс создания и запуска программы приводится в таблице ниже. В этой таблице 'x' в именах файлов обозначает имя, которое может быть одинаковым для всех показанных файлов. Нет ничего уникального в расширениях имен файлов, показанных в таблице. Показанные расширения обозначают стандартное использование. Мы будет ссылаться на различные типы файлов, используя расширения имен файлов, перечисленных в таблице. Файлы, обозначенные как 'простой текст', обсуждаются в [*Главе 3*](#_Форматы_исходных_и). Наш интерес сосредотачивается на объектных файлах, созданных ассемблером и как они используются редактором связей или загрузчиком для создания абсолютных файлов.

## Процесс ассемблирования/компоновки

**Программа Ввод Вывод**

Редактор Клавиатура ┌────── x.Z80 файл (простой текст)

консоли │ ┌─── x.LIB файл (простой текст)

│ │

Ассемблер x.Z80 файл ─┘ │ x.PRN файл (простой текст)

(ZMAC) x.LIB файл ────┘ x.SYM файл (простой текст)

x.ERR файл (простой текст)

┌────── x.REL файл (перемещаемый объект)

│ ┌─── x.HEX файл (абсолютный объект)

│ │

Редактор x.REL файл ─┘ │ x.COM файл (двоичный образ)

связей (ZML) │ x.BIN файл (двоичный образ)

│ x.SYM файл (простой текст)

│

Загрузчик x.HEX файл ────┘ x.COM файл (двоичный образ)

(MLOAD)

Операционная x.COM файл Компьютер выполняет задачу

система

Файл HEX или REL, созданный ассемблером, называют объектным файлом. Код в таком файле вызывают объектным кодом HEX или объектным кодом REL. Эти термины часто сокращаются до 'объектного кода' или просто 'объект'. Файлы HEX содержат поток байтов, которые интерпретируются загрузчиком HEX, например, MLOAD.COM для создания файла, подходящий для загрузки в определенные адреса памяти. Файлы REL так же интерпретируются редактором связей, например, ZML, чтобы создать подобный файл. Созданный файл в любом случае, называют абсолютным, и код, который он содержит называют абсолютным кодом.

Файлы COM создаются редактором связей и загрузчиком. ZML может также создать исполняемый файл с расширением BIN. Файлы COM и BIN содержат код, который является точным образом памяти, готовым к использованию процессором. Когда файл копируется в память, местоположение, в которое копируется первый байт называется адресом загрузки. Для стандартного COM файла в CP/M, адрес загрузки - 100H. Файлы BIN загружаются в память в расположения ОТЛИЧНЫЕ от 100H.

Предположим, что должна быть написана довольно простая программа. Весь исходный код находится в одном файле Z80. (PRINT.Z80 является примером) есть два способа сделать окончательный файл COM:

1) Z80---->(ZMAC)--->HEX--->(MLOAD)----->COM

2) Z80---->(ZMAC)--->REL--->(ZML)--->COM

Результаты будут идентичны, а также они должны быть!

Редактор связей и загрузчик оба производят файлы COM. В чем разница? Почему два способа для получения того же результата? Ответ заключается в природе этих двух объектных файлов, создаваемых ассемблером: файлах REL и HEX. Эти файлы напрямую не пригодны для использования процессором. Они содержат байты, которые будут переведены в образ памяти плюс информацию, необходимую для процесса ассемблирования.

## Объектные файлы HEX и REL

**HEX файлы.** Файл HEX содержит последовательность записей, каждая из которых определяет до 16 байтов кода и адрес, назначенный для первого байта. Задача загрузчика (MLOAD, например), извлекать байты кода и загрузить их в образ памяти по указанному начальному адресу. Начальный адрес в первой записи предоставляется оператором ORG в файле Z80. Код генерируется для последовательного увеличивающихся адресов, пока не встретится следующий оператор ORG. В этот момент адрес находящийся вне последовательности начинает новую HEX запись, и загрузчик начинает загружать по новому адресу. Этот процесс, в зависимости от операторов ORG, может оставить диапазоны адресов в абсолютном образе, содержание которых не определено. Когда MLOAD (или CP/M LOAD.COM) создает образ COM-файла, эти диапазоны заполняются нулями.

MLOAD позволяет второй режим работы. В этом режиме существующий COM файл указывается в командной строке в дополнение к HEX файлу для загрузки. Файл COM обеспечивает, данные для диапазонов адресов оставленных неуказанными записями HEX файла. В этом случае эффект процесса загрузки состоит в том, чтобы заменить байты в файле COM. Такой процесс называется наложением. Он обеспечивает возможность для быстрого и удобного исправления файла COM. Отладчики DSD, ZDM и DDT содержат HEX загрузчик, который выполняет ту же задачу.

Адрес выполнение кода, полученный из HEX файла, совпадает с адресом загрузки, определенным в первой записи. Все адресные ссылки в коде приведены к абсолютным числам во время ассемблирования для создания HEX файла. Этот факт отличает HEX файлы от REL.

**REL файлы.** REL обозначает перемещаемый. Вместо того, чтобы определить точное расположение и значение каждого байта кода, файл REL содержит данные, которые идентифицируют те байты, значение которых зависит от того, где в памяти окончательный код образа должен быть расположен. В этой схеме необходимо различить абсолютные и перемещаемые байты кода. Это делается путем сортировки кода в логические группы, названные сегментами. Содержимое абсолютного сегмента - очевидно. Адреса меток в этом сегменте не изменяются редактором связей. Для перемещаемых сегментов определяются те значения, которые являются адресами в сегменте. Их значение в файле REL является их смещением от начала сегмента. Позже, когда редактор связей генерирует образ COM, начальный адрес сегмента добавляется к каждому смещению. Это является перемещением. Логически, не существует ограничений на количество возможных сегментов. ZMAC обеспечивает 16 таких сегментов или основ перемещения(relocation bases).

Некоторые функции, такие как консольный ввод-вывод, чтение и запись файлов, вызовы функций BDOS, или сортировка списка неоднократно используются в программах на языке ассемблера. Чтобы не изобретать их каждый раз, когда они необходимы, каждая функция может быть написана как вызываемая подпрограмма в отдельном файле, затем ассемблирована для создания REL файла. Программа, которая использует эти функции, просто использует инструкцию вызова соответствующей подпрограммы с указанием, что подпрограмма не присутствует. Во время компоновки имена этих REL файлов передаются в качестве входящих редактору связей, вместе с REL файлом основной программы. Редактор связей объединяет код из набора REL файлов в один COM файл. Здесь становится понятно, почему перемещение важно. Адрес выполнения этих небольших подпрограмм никогда не известен, пока редактор связей наконец не помещает весь код вместе!

Перейдем к другой ситуации. Вы определяете с помощью оператора ORG в файле Z80, что код должен начинаться в 400H, затем ассемблируете, чтобы создать файлы HEX и REL и наконец файлы COM. Если вы исследуете файл с помощью дисковой утилиты или отладчика, или даже команды 'peek' Z3, вы увидите, что первый байт в файле код, который идет на 400H. Это логически не COM файл, потому что он не будет выполняться правильно, при загрузке по стандартному адресу 100H! Кроме того, если был указан дополнительный код (с помощью оператора ORG) в расположении 1400H, созданный файл будет намного больше. Неиспользованная область между концом кода начинающегося в 400H и началом кода в 1400H будет заполнена нулями. Редактор связей приводит к тому же результату при работе с файлом REL, созданным из кода Z80, в котором присутствует начальное объявление ASEG (абсолютный код). Но посмотрите, что происходит, если объявление ASEG отсутствует.

Редактор связей предполагает, что он собирается создавать COM файл, который будет помещен в память в расположении 100H для выполнения. Поскольку ASEG был упущен из файла Z80 (на этот раз), файл REL был создан в сегменте по умолчанию, являющимся (перемещаемым) CSEG. Во-первых, оператор ORG, в котором указано 400H, теперь будет интерпретироваться как СМЕЩЕНИЕ 400H. Аналогично 1400H тоже является смещением. Файл COM будет теперь содержать 400H нулей перед первым байтом кода! Фактический код будет теперь также перемещен. Он подойдет для выполнения в расположении 500H (100H + 400H смещение), где он будет находиться, если файл будет загружен в 100H. Код с адреса 1400H будет так же перемещен, и 'пустое' пространство между двумя блоками кода будет заполнено нулями.

Существует еще один способ. Удалите 'ORG 400H' из файла Z80 и повторно ассемблируйте в REL файл. При выполнении операции связывания, включите параметр в командной строке /С400. Затем повторите сборку с параметром /R400. Первая произведет файл COM. Вторая - файл BIN. Файл COM будет содержать 300H нулей перед первым кодом. Этот код будет перемещен для выполнения в 400H, фактический адрес, когда файл загружается, начиная со стандартного 100H для COM-файлов. У файла BIN будет точно тот же код, за исключением того, что ведущие нули отсутствуют! 'Дыра' между первыми и вторыми блоками кода в обоих файлах все еще заполнена нулями.

Когда ZMAC создает объектный файл HEX, он работает абсолютном режиме. Когда создает объект REL, он находится в перемещаемом режиме. Все адресные ссылки в абсолютном режиме ассемблирования могут быть разрешены (т.е., назначены фиксированные числовые адреса) на основе операторов в исходном файле. В абсолютном режиме код ассемблируется только в одну основу перемещения, абсолютную. Для абсолютной основы перемещения значение константы перемещения всегда равно 0. Эффект состоит в том, что перемещение абсолютного перемещаемого кода не происходит! Присвоенные адреса, появляющиеся в листинге ассемблирования, являются абсолютными адресами, не изменяются компоновщиком (объект REL) или MLOAD (объект HEX).

Когда создается и соединяется объект REL, ко всем перемещаемым значениям компоновщиком добавляется константа перемещения. Значение этой константы обычно не известно во время ассемблирования (за исключением, как отмечалось выше, для кода в абсолютном режиме). Оно присваивается во время соединения алгоритмом управления памятью, который принимает во внимание любые адреса загрузки, определенные в командной строке.

# Форматы исходных и выходных файлов

## Исходные файлы

ZMAC обрабатывает текстовые файлы, обычно подготовленные редактором или текстовым процессором. Каждая строка файла, как ожидается, заканчивается ASCII символом возврата каретки, сопровождаемого символом перевода строки. Сам файл обычно содержит стандартную метку конца файла CP/M (01AH) в конце текста. Большинство популярных редакторов и текстовых процессоров производят файлы этого типа.

Исходный файл может требовать включения других файлов с помощью INCLUDE, MACLIB или подобных операторов. Формат таких файлов следуют тем же правилам что и файл, содержащий на них ссылки.

Источник обрабатывается построчно. Некоторые строки, и части из нескольких строк, игнорируются ассемблером, за исключением включения в выходной листинг. Такими строками являются:

1. Пустые строки, которые содержат только символы CRLF.
2. Строки, которые содержат точку с запятой как первый печатный символ. Это особый случай поля комментария, обсуждаемого ниже.
3. Строки, которые начинаются со звездочки (\*) в первой позиции символа.
4. Строки, которые следуют определенным ассемблерным инструкциям, которые вы включили в источник. Эти инструкции обсуждаются в Главе [Псевдооперации](#_Псевдо-операции_(директивы_ассемблера)_2) и в действительности являются особым случаем комментариев.

Все остальные строки обрабатываются ассемблером, который сканирует строку и интерпретирует действия, которые должны быть приняты после установки встроенных правил. Текст, который не может быть однозначно интерпретирован этими правилами, будет отмечен как ошибка ассемблирования, потому, что почти наверняка будет производиться ошибочный код.

Вот форма приемлемой входной строки:

┌─ Поле метки

│

│ ┌─ Поле оператора (OPR)

│ │

│ │ ┌─ Поле операнда (OPA)

│ │ │

│ │ │ ┌─ Поле комментария

│ │ │ │

v v v v

**TXT TXT TXT ;STR**

^ ^ ^ ^ ^ ^ ^

│ └─────┘ └─────┘ └────┘ <───── Пробелы или символы табуляции

│

└──────────────────────────── Первый столбец

TXT представляет собой последовательность печатаемых символов из ограниченного набора символов. TXT не содержит пробелов или символов табуляции.

STR является строкой печатаемых символов ASCII, включая пробелы и символы табуляции, но исключая другие непечатаемые символы.

Обратите внимание, что поле комментария ВСЕГДА начинается с точки с запятой (;).

### Поле метки

Поле метки может содержать одно из:

а) 'свободное место' (любая комбинация пробелов и символов табуляции);

б) текст как "СИМВОЛ[[1]](#footnote-1)";

в) текст как "СИМВОЛ:" (Обратите внимание на завершающее двоеточие);

г) текст как "СИМВОЛ::" (Объявляет общедоступный СИМВОЛ).

Поле метки может содержать начальные пробелы. Хотя, это не хорошая форма.

*Совет*: разделители двоеточия однозначно идентифицируют СИМВОЛ (для ассемблера) в качестве метки. Если вы забыли двоеточие, как в пункте б) выше, ассемблер должен работать намного активнее, чтобы избежать неправильного толкования СИМВОЛА. Это занимает время и замедляет процесс ассемблирования. Используйте двоеточия и сэкономьте время!

Символы, которые могут использоваться в СИМВОЛЕ:

а) Буквенные символы верхнего и нижнего регистра

б) ? $ % . \_ @

в) Числа (0..9), за исключением того, что

г) Первый символНЕможет быть числом

СИМВОЛ может иметь длину, какую вы пожелаете. Однако, ZMAC игнорирует любые символы после 15. Хорошая практика, создать СИМВОЛЫ уникальными в первых 6 или 7 символах, потому что многие редакторы связей имеют такие ограничения.

СИМВОЛ в поле метки обычно является меткой, которая является особым случаем СИМВОЛА. Метка сохраняется в таблице символов ассемблера для ссылки на нее другими частями программы.

СИМВОЛ может быть именем, присваиваемым макроопределению. Такие СИМВОЛЫ сохраняются в таблице имен макросов.

СИМВОЛ может быть объектом оператора определения, например SET или EQU. СИМВОЛ вместе с его определенным значением сохраняется в таблице символов вместе с типом, отмечающим его в качестве метки.

### Поле оператора

OPR является сокращением слова оператор. Это поле также упоминается как поле операции и поле оператора. Поскольку слово 'оператор' также используется в совершенно ином контексте для описания выражений, сокращение OPR будет использоваться в данном руководстве для описания поля, когда существует возможность неоднозначности.

Поле OPR может содержать:

а) Код операции (код машинной команды);

б) Псевдо-операцию (код псевдо-операции);

в) Имя макроса.

**Коды операций**: коды машинных команд для Z80, Z180 и HD64180 перечислены в *Приложении 4*. Код операции является начальным словом машинной команды. Инструкция баланса {balance} (если имеется) является полем OPA.

Несколько типичных кодов операций:

LD halt Exx call JR и Otimr TST

Обратите внимание, что верхний/нижний регистр может использоваться как вы пожелаете.

Несмотря на то, что ассемблер не будет запутан, если вы будете использовать коды операций в качестве меток, такая практика мешает ясности программ и считается плохим стилем программирования по этой причине.

**Псевдо-операции**: также известны как директивы ассемблера. Они обеспечивают инструкции, которые направляют процесс ассемблирования. Большинство не генерирует код в объектном файле. Например, псевдооперация .Z80, направляет ассемблер выдавать предупреждение, если в источнике появляется код операции не Z80, и псевдо-операция PAGE заставляет ассемблер вставлять код перевода формата в файл листинга (PRN). [Псевдо-операции](#_Псевдо-операции_(директивы_ассемблера)_1) описаны в *Главе 5*.

**Макросы**: Макрос является последовательностью операторов ассемблера, определенных в исходном файле и присвоенного им имени, которое находится в поле метки. Макрос вызывается, когда его имя занимает поле OPR исходной строки. Вызов заставляет в этот момент ассемблировать операторы макроса, прежде чем перейти к следующей строке. Макросы часто содержат инструкции кодов операций и генерируют код. [Макросы](#_Обработка_макросов) описаны в *Главе 7*.

### Поле операнда

OPA является сокращением слова операнд. Оба термина используются в этом руководстве. OPA используется, когда термин операнд может бы привести к неоднозначности.

Поле OPA содержит список аргументов (если таковые имеются) связанный с операцией, подразумеваемой в поле операции. Если список аргументов не требуется, то поле остается пустым.

Когда поле OPR содержит код операции, поле OPA содержит параметры, подразумеваемые в *Приложении 4*. Три вида аргумента могут потребоваться: имена регистров, 8-битные данные и 16-битных данные.

**Именами регистров** являются:

A B C D E H L I R (8 битные регистры)

AF BC DE HL IX IY SP AF' (16 битные регистры)

Имена регистров являются зарезервированными словами[[2]](#footnote-2) относительно поля OPA.

Когда имя регистра процессора встречается в поле OPA, и вы использовали это имя в качестве метки (в поле метки), ZMAC сталкивается с неоднозначностью, которая не может быть решена с помощью контекста символа. Это должно быть интерпретировано как имя регистра или как Символ? Например, рассмотрим следующий фрагмент кода:

B EQU 23H

...

...

LD A,B

...

Это должно быть интерпретировано как “загрузка A содержимым регистра B” или как “загрузка A значением символа B (=23h)”?

Из-за таких возможных неоднозначностей вы не должны использовать имена регистров в поле метки. ZMAC обнаружит такое использование и выдаст сообщение об ошибке.

Имена макросов, хранятся и используются иначе, чем другие символы. Благодаря этому, нет никаких ограничений к именам, которые присваиваются макросам. Имена регистров могут использоваться в качестве имен макросов без неоднозначности, влияющих на работу ZMAC.

Ссылки на 8 и 16 битные данные могут быть представлены числами, символами или выражениями. Выражение это похожая на алгебраическую формула, составленная из чисел, символов и арифметических/логических операторов. Один или два символа заключенные в кавычки, могут содержать 8 или 16 битные данные, соответственно. Если будут присутствовать неуместные ссылки регистров или значения данных для определенного кода операции - выдается сообщение об ошибке.

Имена регистров не используются в выражениях. Инструкции, с участием регистров IX и IY принимают аргументы в форме (IX+<exp>), которая выглядит как выражение. Само <exp> не может содержать имя регистра. Причина: ассемблер не сможет назначить значение для содержимого регистра в общем случае. Это количество становится реальной, только когда выполняется фактический код.

Когда поле OPR содержит макрос или псевдооперацию, поле OPA может содержать текст ASCII, заключенные в кавычки строки, числа, символы или выражения. Точные требования зависят от потребностей макроса или псевдооперации.

Поле ОРА может быть пустым. Однако, если оно присутствует, ему всегда предшествует непустое поле OPR. Они следуют вместе.

**Текст ASCII** и **строки заключенные в кавычки** обсуждаются в следующих параграфах. Числа, символы и выражения обсуждаются в [*Главе 4*](#_Выражения_1).

Строка ASCII представляет собой последовательность печатных символов ASCII, в котором пробелы и знаки табуляции (пробелы) могут быть включены. Когда строка является частью исходной строки на ассемблере, в ней присутствуют некоторые символы, которые не предназначены быть частью строки. Должен использоваться некоторый метод разграничения строки, чтобы гарантировать, что только нужные символы присутствуют в производимом коде. Обычно, строки заключены в кавычки (") или одиночные апострофы ('). Вот несколько примеров:

db 'Это строка'

db "Это строка "

db "there's more than one way.."

db ' " кавычки также могут быть в последовательности" '

Если вам необходимо включить разделитель как часть текста в строке, вы можете сделать так, просто включите его в два раза. Вот некоторые примеры строк в кавычках, наряду со строками символов, которые фактически включаются в сгенерированный код:

db 'xxxx''yyyy' ; производит xxxx'yyyy

db "foo""bar" ; производит foo"bar

Обратите внимание, что разделители, которые начинают и заканчивают заключенную в кавычки строку, не сохраняются.

В главе, посвященной [псевдо-операциям](#_Псевдо-операции_(директивы_ассемблера)_8), вы будете наблюдать определения синтаксиса нескольких типов строк. Глоссарий (*Приложение 1*) содержит определения этих терминов. Различие между ними заключается в типе используемых разделителей и допустимых символов в строке. Во многих случаях ZMAC правильно интерпретирует текстовую строку без разделителей. Разнообразие обработки строк позволяет ZMAC использовать исходный код, написанный для многих других ассемблеров, которые используют различные соглашения разграничения. Следующие термины используются в этом руководстве, когда необходимо быть точным. Большую часть времени термин 'строка' будет использован, когда различие не важны и будет обычно подразумевать строку заключенную в кавычки.

TXT последовательность символов. Без пробелов, символов табуляции.

QTXT ТХТ заключеные в одинарные или двойные кавычки.

STR последовательность печатаемых символов, включая пробелы и символы табуляции.

QSTR STR заключенный апострофами или символами кавычек. Символ кавычки может быть включен в STR, если ему непосредственно предшествует тот же символ. Символ, предшествующий строке должен совпасть с завершающим ее. Другие пары рассматриваются как нормальные печатаемые символы в строке.

В ZMAC инструкции ассемблера NAME, .TITLE и .SBTTL (описанные в [*Главе 5*](#_Псевдо-операции_(директивы_ассемблера)_9)) предоставляют мягкий набор правил для ввода текстовой информации. В этих случаях даже конечный разделитель может быть опущен! Ниже приведены правила интерпретации аргументов для этих случаев:

а) Заключительная кавычка для заключенных в кавычки строк является необязательной;

б) В строках без кавычек и заключенных в кавычки, у которых отсутствует заключительная кавычка, конец-строки определяет конец строки;

в) Строки слишком длинные для их предполагаемого использования усекаются до максимально используемой длины (в настоящее время 7 символов для NAME, 60 для TITLE, SBTTL).

Синтаксис аргументов для NAME, .TITLE и .SBTTL очень либеральный. Сообщение об ошибке выдается, только если отсутствует аргумент. Иначе, ZMAC максимально использует то, что вы предоставили в источнике. Строка без кавычек точно соответствует строке, завершенной только управляющим символом (кроме символа табуляции). Заключенная в кавычки строка правильно интерпретирует продублированные символы кавычек (') или ("), и управляющий символ заменит обычно требуемую заключительную кавычку.

### Поле комментария

Поле комментария совершенно необязательно. При наличии, оно предваряется символом ';'. Ассемблер рассматривает точку с запятой как логический конец строки и игнорирует все, что следует до начала следующей строки. В макроопределении поле комментария сохраняется для включения его при последующих макрорасширениях. Если это нежелательно, то в начале поле комментария могут использоваться две точки с запятой. Комментарий в этом случае будет также игнорироваться.

Пробелы, символы табуляции и любые печатаемые символы могут присутствовать в поле комментария.

Поле комментария может быть единственным полем, присутствующем в строке. Ему может предшествовать пробел и поэтому он может начинаться везде, где пожелаете.

## Выходные файлы

ZMAC производит от нуля до четырех файлов, в зависимости от выбора параметров по умолчанию и параметров командной строки. При отсутствии переопределения выбора параметров командной строки имена файлов будут <xxx>.REL, <xxx>.PRN, <xxx>.SYM и <xxx>.ERR; <xxx> - часть имени ассемблируемого файла. Эти расширения файлов по умолчанию могут быть 'постоянно' изменены с помощью утилиты ZCNFG. В этом руководстве файлы будут упоминаться приведенными выше расширениями по умолчанию.

### Листинг ассемблера

**Файл PRN** (Листинг ассемблера)

Файл PRN содержит представление произведенного кода, копию исходных строк и информацию о коде и исходном файле(ах). Листинг PRN может быть отправлен в любой комбинации в файл, на консоль и устройство печати (принтер) с помощью разумного выбора параметров командной строки и параметров конфигурации. Форматом листинга PRN управляют многие параметры в утилите ZCNFG. Вывод PRN можно адаптировать к используемого принтера и предпочтениям программистов. Эти три возможности обеспечивают полное управление разбиением на страницы:

а) Прогон страницы формата или отсутствие прогона страницы в начале листинга;

б) Прогон страницы или отсутствие прогона страницы в конце листинга;

в) Условный прогон страницы в конце листинга, чтобы гарантировать, что начало СЛЕДУЮЩЕГО листинга будет иметь аналогичную ориентацию по отношению к фальцованной бумаге. Это называется 'четностью страницы'.

Параметры ZCNFG разрешают устанавливать длину и ширину страницы по умолчанию чтобы соответствовать возможностям принтера, или могут использоваться в качестве значений по умолчанию значения среды Z3. Другими элементами формата листинга можно так же управлять. Большинство из них описано ниже. Все параметры описаны в *Главе 9* ([Конфигурация](#_Настройка_параметров_с_2)).

Ниже приводится пример листинга PRN, немного отредактированный для наглядности. Элементы в угловых скобках являются примечаниями, которые были отредактированы. Как вы можете догадаться по номерам строк, большая часть листинга была удалена.

Обратите внимание на название программы напечатанное чуть ниже номера страницы. Каждая страница листинга содержит текущее название основного файла с исходным кодом в том же месте.

Последние несколько строк указывают на основные функции строк ассемблерного листинга PRN.

Рисунок 3‑1 Пример листинга PRN

ZMAC Relocating Macro Assembler v 0.8 page 1

<Здесь выводится название из псевдооператора .TITLE> TEST .Z80

<Подзаголовок будет распечатан здесь>

1 ;Program: test.z80

71 0000' C3 000B' JP START

84 0011' ED73 0064" MAIN:: ld (stak), sp ;save for c

97 dseg

99 0000" 0000 wbaddr:: dw 0

^ ^ ^ ^ ^ ^ ^

│ │ │ │ │ │ │

**E** **Строка+** **Адрес** └**Поле кода**┘ └────── **Копия исходной строки** ───────┘

^

└── **Маркер для файлов INCLUDE и MACLIB**

**E** указывает расположение кода ошибки, если строка содержит ошибку.

**Строка** определяет поле номера строки. С помощью ZCNFG можно переместить номер строки направо от поля кода, отключить номера строк или изменить метод нумерации. (См. [*Главу 9*](#_Настройка_параметров_с_3)).

**+** маркер, который занимает эту позицию, когда код собирается из файла INCLUDE.

**Адрес** 4 шестнадцатеричных символа адреса текущего кода. Это 16 битное число. В этом примере сначала выводится младший байт числа. ZCNFG позволяет выбрать печать сначала младшего или старшего байта.

**Поле кода** содержит шестнадцатеричный эквивалент кода, сгенерированный и отправленный в объектный файл REL или HEX. Заключительный пробел разделяет код операции (если есть) от любых аргументов. Если представлен 16 битный аргумент, он может быть выведен сначала с младшим или старшим байтом. Порядок определяется в ZCNFG.

**Копия исходной строки** Из-за всей информации, представленной в листинге, длина каждой строки листинга часто будет превышать ограничение 80 столбцов для маленьких принтеров и экранов CRT. Псевдо-оператор PAGE позволяет в аргументе определить допустимую длину строки PRN. ZCNFG позволяет этому же параметру быть установленным как значение по умолчанию. В любом случае строка будет усеченной справа, если она превысит максимальное ограничение строки. Строка 84 в примере листинга выше является такой усеченной строкой. ZMAC устанавливает длину и ширину страницы по умолчанию к сохраненному в Вашей среде Z3 по указанию ZCNFG.

**Знак перемещаемого сегмента** - Справа от каждой 16 битной величины, предусмотрено место для индикации перемещаемого сегмента (ASEG, CSEG, DSEG или COMMON). Отсутствие в нем знака указывает абсолютное значение, апостроф (') перемещаемый код, кавычка (") означает перемещаемые данные и звездочка (\*) указывает BLANK или именованное общее перемещаемое значение.

### Листинг таблицы символов

Листинг таблицы символов обычно добавляется к листингу PRN. Однако, параметр командной строки 'S', переключает генерацию отдельного файла таблицы символов. Формат этого файла подходит для использования с отладчиками DSD, SID и ZSID. Пример листинга типичной таблицы символов, добавленного к листингу PRN, показан ниже.

ZMAC Relocating Macro Assembler v 0.8 page 3

TEST .Z80

\*\*\*\* Symbol table \*\*\*\*

$MEMRY 0066 EXIT 0020 LOCBUF 0002

MAIN 0011 STAK 0064 START 000B

WBADDR 0000 Z3EADR 0009 Z3INIT 000F

### Файл регистрации ошибок

Когда опция 'E' в состоянии on (по умолчанию или в результате вызова командной строки), данные об ошибках отправляются в файл регистрации, имя которого совпадает с именем исходного файла. Если ошибок не было, то ERR файл не производится. Это был бы пустой файл. Данные в файле регистрации в фиксированном формате и включают: имя файла, номер строки, номер столбца и тип ошибки. Данные являются простым текстом так, что файл может быть использован в качестве входных данных редактора. Первая строка начинается с символа '\*' в столбце 1. Символ '\*' может использоваться редактором, чтобы пропустить строки, которые не содержат данные об ошибках.

Пример ниже является частью ERR файла, который появился в результате ассемблирования программы, содержащей преднамеренные ошибки.

**Пример файла регистрации ошибок**

\* FILE SPEC LINE COL CODE

B6:ZILOG2.Z80 68 K

B6:ZILOG2.Z80 69 K

B6:ZILOG2.Z80 106 O

C3:ZILOGM.LIB 191 E

C3:ZILOGM.LIB 198 S

B6:ZILOG2.Z80 35 D

B6:ZILOG2.Z80 39 D

B6:ZILOG2.Z80 68 K

Обратите внимание, что спецификация файла в четвертой и пятой строках отличается, от других. Это файл INCLUDE (или MACLIB). Номера строк, показанные здесь, являются номерами строк в этом файле. Другими словами, нумерация строк начинается снова для каждого включенного файла. ZMAC отслеживает номера строк в каждом файле. Когда 'INCLUDE' файл завершен, нумерация строк в родительском файле продолжается, с места в котором она была прервана. Результат: вы можете перейти на строку 198 в файле zilogm.lib (в вашем редакторе), чтобы исправить ошибку. Некоторые редакторы (например, PMATE) позволяют встроенный макрос, который автоматизирует процесс: загружают ERR файл, а затем использует спецификацию файла и номер строки для перехода к строке с ошибкой. Один нюанс: ZMAC подсчитывает строки, начиная с '1'. Если ваш редактор считает с '0', вам придется делать поправку на этот факт.

Последние три строки показывают номера строк, которые находятся очевидно не в последовательности. Это вызвано тем, что некоторые ошибки обнаружены в проходе 1, некоторые в проходе 2 и некоторые в обеих проходах. Шестая и седьмая строки являются примером ошибок, обнаруженных в проходе 2. Последняя является примером одной обнаруженной в обеих проходах.

Столбец помеченный COL зарезервирован для будущей реализации.

Последний столбец содержит код ошибки: указывающий характер ошибок. Коды ошибок приведены в *Приложении 5*.

# Выражения

В этой главе рассматриваются выражения. Официальное обозначение для выражения является <exp>. Он будет часто обозначается как 'exp' в данном руководстве (см. *Приложение 1*, Глоссарий). В следующих разделах обсуждаются основные элементы выражений и правила построения допустимых выражений.

Выражение представляет собой алгебраическую комбинацию операндов и операторов. Операнды (<opa>) являются числами, символами и символьными константами. Операторы (<opr>) определяют выполняемые арифметические / логические функции. Полный список операторов приводится в *Таблице 4-1*.

Вот очень сокращенное (и рекурсивное) определение выражения:

<exp> = { {<opa>} <opr> } <opa>

<opa> = <num> | <sym> | <char constant> | <exp>

<opr> = <Один из символов оператора из Таблицы 4-1>

Самая простая форма выражения

<exp> = <opa>

где <opa> - число, символ или символьная константа. В следующих разделах мы обсудим эти элементы выражений прежде, чем перейти к операторам и более сложным выражениям.

**Элементы выражений**

## Числа

Три вида данных могут присутствовать в поле OPA строки ассемблерного источника: ссылки на регистры процессора, 8 битовые и 16 битовые величины. Когда поле OPR содержит код операции (см. Список кодов операций в *Приложение 4*), поле OPA содержит не более одной такой величины определяемой как 'n' или 'nn'. 'n' - величина 8 бит, и 'nn' - величина 16 бит. Величина 8 бит имеет числовое значение в диапазоне от 0 до 0ffh (в шестнадцатеричном формате), которое совпадает с значением от 0 до 255 (в десятичном формате). Величина 16 бит имеет значение в диапазоне от 0 до 0ffffh (от 0 до 65535 в десятичной системе счисления). Вы можете заменить ЧИСЛОМ 'n' или 'nn' везде, где они встречаются в качестве аргумента для кода операции.

Один из следующих индикаторов основания должен быть добавлен к числу, если оно представлено в основании отличном от текущего значения по умолчанию:

|  |  |  |
| --- | --- | --- |
| **Индикатор основания** | **Основание** | **Тип числа** |
| **B** | 2 | Двоичное |
| **O** | 8 | Восьмеричное |
| **Q** | 8 | Восьмеричное |
| **D** | 10 | Десятичное (по умолчанию) |
| **H** | 16 | Шестнадцатеричное |

Чтобы избежать путаницы, в этом руководстве основание системы счисления *всегда* выражается в десятичной нотации. ZMAC по умолчанию предполагает, что числа выражаются в десятичной системе. Это означает, что вы не должны добавлять индикатор основания 'D' к десятичным числам, но вы ДОЛЖНЫ добавить 'H' к числу, которое предназначено быть шестнадцатеричным.

Основание по умолчанию, используемое ассемблером, может быть изменено включением оператора .RADIX следующим образом:

.RADIX n <== 'n' предполагается десятичное

После этого, до следующего оператора .RADIX или конца программы, основание по умолчанию будет 'n'. Вы должны будете добавлять индикатор основания к любому числу, не предназначенному, иметь текущее основание. Параметр 'n' псевдо-оператора .RADIX не ограничен вышеупомянутыми четырьмя системами счисления (основаниями). Назначение основания может быть может быть любым от 2 до 255. Использование других оснований системы счисления, однако, может быть вредно для вашего психического здоровья!! Максимального основание, для которого ваш терминал произведет разумный экранный вывод, равняется 36 (цифры 0-9 и буквы A-Z).

В операторе .RADIX аргумент 'n' всегда предполагается ДЕСЯТИЧНЫМ, таким образом, вы не должны делать вычисления для аргумента в непривычной системе счисления. Если вместо этого вы хотите использовать двоичный, восьмеричный или шестнадцатеричный, вы можете просто добавить замыкающий оператор .RADIX. Не много практических применений для своеобразных оснований, но они приходят 'бесплатно' с обобщенным алгоритмом, реализованным в ZMAC.

Файл примера PRINT.Z80 содержит множество присвоений оснований и демонстрирует их использование в макросе с помощью псевдо-оператора .PRINT. Нечетные основания вроде 7 или 9 не будут работать с большинством других ассемблеров.

Обратите внимание, что если вы установите основание в нечетную основу и забудете переключать его обратно на что-то более 'нормальное', то вы получите несколько странного вида результат ассемблирования!

## Символьные константы

Символьной константой является один или два ASCII печатаемые символа, заключенные в одинарные или двойные кавычки. Пробелы и знаки табуляции тоже могут быть частью символьной константы. Символьная константа преобразуется ассемблером в число. Она может использоваться везде, где число допустимо в операторах и выражениях. Число генерируемое ассемблером, является значением ASCII символа. Например, 'A' генерирует значение 41h, и 'a' преобразуется в 61h. Числа, генерируемые с помощью отдельных символов, являются 8-разрядными величинами (байтами). Когда два символа присутствуют внутри разделителей ' или ", 16-разрядное число (слово) формируется и хранится в коде образа в обратном порядке. Символьная константа 'AB' преобразуется в число 4241h. Порядок инвертирован! Это вызвано тем, что текст, который вы вводите, как предполагается, находится в 'человеко-читаемой' форме: сначала старший байт, в то время как процессор ожидает найти в хранящихся 16-разрядных числах первым младший байт.

Символьные константы выглядят как короткие строки. Ассемблер различает символьные константы и строки видом оператора, в котором они появляются. Строки не допускаются, например, в выражениях или в операторах DW. Коды операций (см. *Приложение 4*) часто требуют 8-разрядные или 16-разрядные аргументы. Это числа. Один или два заключенных в кавычки символа будут интерпретироваться как символьные константы.

Ниже приведено несколько примеров символьных констант и генерируемый код, взятый из реального ассемблерного листинга PRN: (порядок байтов в слове: старший, младший).

3 = 0032 var1 equ '2'

4 = 4142 var2 equ 'AB'

130 028C' 3132 DEFW '12'

143 02B8' 0030 DW "0"

150 02CC' 0031 3132 DW '1','12'

156 02E4' 020E 020E DW ADDRS1,ADDRS1

## Символы и метки

СИМВОЛ - имя, которое обозначает число. Мы часто говорим о ЗНАЧЕНИИ символа. Это значение просто соответствующее число. Метка является символом в поле метки любой исходной ассемблерной строки кроме операторов EQU, SET и MACRO. Это - тонкое различие, и термины 'метка' и 'СИМВОЛ' часто используются взаимозаменяемо. Символы, которые могут использоваться в СИМВОЛЕ, идентичны перечисленным для поля метки в главе посвященной [формату исходного файла](#_Поле_метки). Вот некоторые типичные имена символов из реального ассемблера:

\*\*\*\* Symbol table \*\*\*\*

$LABELY 000D %LABEL 000E .LABEL 0017

@LABEL 000B ABCDE 0064 ABCDEF 0064

ASM$MAC 0063 ASMMAC 0062 CLABEL 0000

FGHIJ 0064 G 00C8 GHIJKL 0064

L$ABELZ 000D L%ABEL 000E L.ABEL 0017

L?ABEL 0018 L@ABEL 000B LAB 0000

LABEL07 0007 LABEL08 0008 LABEL09 0009

LABEL? 0018 LABEL@ 000B LABELX$ 000D

LABEL\_ 0013 L\_ABEL 0013 M 00C8

QQQQQQQQQQQQQQ 000E QQQQQQQQQQQQQQQ 000F S 00C8

## Счетчик адреса

Счетчик адреса (также называемый счетчик команд) содержит адрес кода, созданного в процессе ассемблирования каждой исходной строки. Специальный символ используется для ссылки на счетчик адреса: '$' (знак доллара). Он может использоваться как любой другой символ. Когда на счетчик адреса ссылаются во время ассемблирования в абсолютном режиме, он является абсолютным. Когда ссылаются во время ассемблирования в перемещаемом режиме, он перемещаемый. В блоке кода PHASE-DEPHASE, счетчик адреса абсолютный.

**Примеры**

Источник для ZCPR3 содержит много условно ассемблируемых блоков кода. Если включено слишком много, то произведенный машинный код будет больше, чем пространство, выделенное для CCP в большинстве систем. Чтобы предотвратить такую возможность, метка ставится вначале (обычно ZCPR3:) кода. В конце программы размещаются операторы:

IF ($-ZCPR3) > MAXSIZ

Было выбрано слишком много вариантов

ENDIF

Разница между текущим значением счетчика и адресом запуска программы это конечно, длина. Если она больше, чем разрешено (MAXSIZ), то ZMAC пытается ассемблировать следующую строку. Так как эта строка не корректна в качестве ввода в ассемблер (это НЕ строка комментария!), она рассматривается как ошибка. Строка распечатывается в вашей консоли как часть сообщения об ошибке, предупреждая Вас о проблеме. Другой пример:

jp $ ; это бесконечный цикл, переход на себя

## Операторы и приоритет операторов

**Операторы**

Операторы, распознаваемые в выражениях, приведены в следующей таблице. Символы A и B могут быть выражениями. Обратите внимание, что некоторые операторы унарные. Они принимают только один аргумент. Один оператор, NUL, не принимает аргументов вообще.

Таблица 4‑1 Операторы

|  |  |
| --- | --- |
| ***Оператор*** | ***Выполненная арифметическая операция*** |
| A **+**  B | арифметическая сумма |
| A **–** B | арифметическое разность |
| A **\***  B | умножение целого без знака |
| A **/** B | беззнаковое целое деление |
| A **MOD** B | целочисленный остаток после A/B |
| A **SHL** B | сдвиг A влево на B битов с заполнением нулями справа |
| A **SHR** B | сдвиг A вправо на B битов с заполнением нулями слева |

|  |  |
| --- | --- |
| ***Оператор*** | ***Условие для значения "TRUE" (иначе "FALSE")*** |
| A **EQ** B | A численно равен B |
| A **=** B | A численно равен B |
| A **LT** B | A численно меньше B |
| A **<** B | A численно меньше B |
| A **LE** B | A численно меньше или равно B |
| A **<=** B | A численно меньше или равно B |
| A **GT** B | A больше, чем B |
| A **>** B | A больше, чем B |
| A **GE** B | A больше или равно B |
| A **>=** B | A больше или равно B |
| A **NE** B | A не равно B |
| A **<>** B | A не равно B |

|  |  |
| --- | --- |
| ***Оператор*** | ***Поразрядный логический результат*** |
| **NOT** B | Все биты дополнены |
| A **AND** B | Логическое “И” A и B |
| A **OR** B | Логическое “ИЛИ” A и B |
| A **XOR** B | Логическое исключающее “ИЛИ” A и B |

|  |  |
| --- | --- |
| ***Оператор*** | ***Результат операции*** |
| **HIGH** B | То же, что B SHR 8 |
| **LOW** B | То же, что B AND 0FFH |
| **NUL** | Возвращает значение TRUE, если остальная часть текущей строки не содержит ничего, кроме пустого пространства. То, что следует за оператором NUL, будет проигнорировано, но заставит вернуть значение FALSE. NUL полезен для тестирования пустых аргументов в макросах. |

**Приоритет операторов**

Во время вычисления выражения, порядок, в котором выполняются операции контролируется правилами приоритета. Это необходимо, чтобы избежать неоднозначности. Все выражение (по крайней мере, концептуально) отсканированы, и операции в верхней части приведенного ниже списка выполняются первыми, а затем далее ниже по списку. Операторы расположенные в одной строке имеют одинаковый приоритет. Порядок, в котором они выполняются, слева направо в порядке следования в выражении.

( ) [ ] <== Наивысший приоритет

\* / MOD SHL SHR

+ -

EQ = LT < LE <= GT > GE >= NE <>

NOT

AND

OR XOR

HIGH LOW

NUL <== Самый низкий приоритет

Естественно, как в алгебраических выражениях, будут случаи, когда вы захотите управлять (или быть уверенными), как элементы выражения группируются для вычисления. ZMAC позволяет вам заключить выражения в скобки. Далее, вы можете использовать круглые или квадратные скобки как вам нравится. Это означает, что ZMAC должным образом обрабатывает выражения, которые используют старые квадратные скобки ZAS. Однако, будьте в курсе одного различия: некоторые коды операций принимают одну из двух форм в качестве одного из своих аргументов. Единственным отличием между инструкциями "ld hl, (nn)" и "ld hl, nn" является наличие круглых скобок, окружающих один из аргументов. Здесь, квадратные скобки НЕ то же самое, что круглые скобки! Квадратные скобки ВЫНУЖДАЮТ параметр интерпретироваться как сразу загруженную величину. В таблице ниже приведены примеры обоих стилей.

**Стиль: ld rr,nn Стиль: ld rr,(nn)**

ld rr,<exp> ld rr,(<exp>)

ld rr,1+(<exp>\*<exp>) ld rr,(1+[<exp>\*<exp>])

ld rr,[<exp>]

**Ограничения выражений**

Величины, используемые в выражениях были описаны как имеющие значение. Это не совсем так просто. Символы, определенные в качестве меток в сегментах кода, данных и COMMON, помечаются как перемещаемые, в то время как из абсолютного сегмента и блоков .PHASE помечаются как абсолютные. Кроме того, некоторые символы не могут быть определены в текущем модуле вообще. Они объявляются внешними, и их перемещаемость не определена.

Перемещение (добавление подходящей константы к каждой перемещаемой величине) выполняется редактором связей, получающим информацию из битового потока, который включает в себя выходной файл в формате Microsoft REL.

К моменту, генерации потока битов, все выражения, должны быть вычислены к одной из следующих форм, разрешенных форматом REL:

1. 16-ти разрядное значение плюс тег основы перемещения (заметьте, что одна из основ абсолютная);
2. 16-ти разрядное значение плюс тег внешний;
3. имя внешней величины и 16-разрядное смещение, которое будет добавлено;
4. имя внешней величины и 16-разрядное смещение, которое будет вычтено.

Передача выражения редактору связей не предусмотрена! Таким образом любое выражение должно быть уменьшенным во время ассемблирования до одной из вышеупомянутых форм.

**Правила построения выражения**

Ниже приведены правила построения допустимых выражений. Если они будут нарушены, то вы будете осчастливлены сообщением об ошибке и вынуждены снова исследовать свое выражение в свете этих правил.

1. Оба операнда во всех арифметических и логических операциях, за ИСКЛЮЧЕНИЕМ сложения и вычитания должны быть абсолютными. Такие арифметические и логические операции производят величины, которые сами по себе абсолютные.
2. Пусть A абсолютная величина, Е внешний символ и R перемещаемый символ. Следующие формы являются допустимыми и производят в результате типы, показанные на рисунке справа:

E + A Внешний;

E - A Внешний;

R1 - R2 Абсолютный (R1, R2 в той же основе перемещения);

R - A Перемещаемый;

R + A Перемещаемый.

Выражение обычно более сложное, чем подразумевают эти правила. Однако рассматривая рекурсивное определение выражения в начале данного раздела, поймете, что правила также могут применяться рекурсивно, чтобы видеть, допустимо ли выражение.

Есть более простой путь, обсужденный в книге Кейлингерта {Calingaert} (См. Список литературы в *Приложении 2*), и используемый Нилом Колвином {Neil Colvin) в его документации для Ассемблера TDL. Этот метод объясняется и иллюстрируется в следующих параграфах.

## Определение перемещаемости

Редактор связей добавит постоянную r к каждой перемещаемой величине. Предвидя это перемещение, перепишите это выражение с r, алгебраически добавленным к каждому перемещаемому символу. Затем перестройте получающееся выражение в попытке вынести r. Если V, W, X и Y являются перемещаемыми, то можно построить следующее выражение:

Y + 2\*X -3\*W +V

Для вычисления его перемещаемым, добавьте r к каждому перемещаемому символу:

(Y+r) + 2\*(X+r) - 3\*(W+r) + (V+r)

Перестройте, чтобы получить алгебраический эквивалент:

1. Раскройте скобки:

Y + r + 2X + 2r -3W -3r +V +r

1. Соберите элементы, для получения:

Y + 2\*X - 3\*W + r

В общем, если r может быть вынесен за скобки, он может иметь коэффициент. Если r обращается в нуль, то коэффициент равен нулю, конечно. Правила просты:

* если коэффициент равен 0, то выражение абсолютное.
* Если коэффициент равняется 1 (в примере выше), то выражение перемещаемо.
* Если коэффициент не 0 или 1, то оно недопустимо, так как не может быть перемещено.
* Аналогично, если коэффициент r с показателем степени (кроме 1), это выражение не перемещаемо.

# Псевдо-операции (директивы ассемблера)

**Определения мнемоник псевдо-операций ZMAC**

Примечание: для упрощения чтения определения синтаксиса содержат пробелы, которые не являются частью синтаксиса. Где синтаксически требуются пробелы, они обозначены '\_' (подчеркивание).

Термины псевдо-операции и директивы ассемблера являются синонимами.

$INCLUDE FS Синонимы: .IN, MACLIB, INCLUDE

(См. описание .IN)

.180 Синоним: .HD64

Отключает нефатальные предупреждающие сообщения для кодов операций процессора HD64180. Это - нормальный режим ассемблера по умолчанию. (См. .Z80)

.ACCEPT {QSTR ,} <symbol>

Эта псевдо-операция дает вам возможность ввести значение для символа во время ассемблирования файла. Например, Вы могли бы хотеть ввести дату/время, номер версии или логическое значение (Yes/No), которое будет управлять условным ассемблированием некоторых частей исходного файла. Вам, возможно, придется отлаживать код, вкрапленный в исходный код, который должен быть ассемблирован, во время разработки кода, но который должен быть опущен в рабочей версии.

Когда строка .ACCEPT ..... , встречаются во время 1 прохода ассемблирования, на вашей консоли появится подсказка. Вы отвечаете, вводя значение или выражение, сопровождаемое возвратом каретки. <symbol> - имя, которое обычно не определено в программе. Значение по умолчанию в этом случае равно нулю. Если вы ответите только возвратом каретки, то присвоенное значение будет двоичным нулем.

.ACCEPT обрабатывается только один раз (во время прохода 1), таким образом, вам не придется дважды вводить одно и то же. Благодаря этой стратегии, любые переменные, используемые в выражении (такие как YES и NO), должны быть определены, когда встречается оператор .ACCEPT.

QSTR - необязательная подсказка, которая будет распечатана на консоли. Если вы не определяете подсказку, то будет использоваться подсказка по умолчанию, состоящая из имени символа, сопровождаемого '?'.

Вы можете обеспечить значение по умолчанию для <symbol> с помощью псевдо-операции SET. Такое определение значения по умолчанию должно удовлетворять двум условиям: во-первых, оно должно произойти перед оператором .ACCEPT и во-вторых, оно должно быть в условном блоке ассемблирования, который заставляет его быть обработанным только во время первого прохода. Например:

IFP1

obsize set 200h ; Размер буфера по умолчанию

ENDIF

....

.ACCEPT 'Изменение размера буфера с 200h до ', obsize

....

выведет сообщение на консоли:

Изменение размера буфера с 200h до ?

и будет ожидать ввод с клавиатуры. Если вы ответите только возвратом каретки, то obsize все еще будет иметь значение 200h. Если вы введете число или выражение, то это значение будет присвоено obsize.

**.COMMENT** <разделитель>< всё, что угодно><разделитель>

Блок произвольно форматированного текста может быть включен в листинг ассемблера с помощью этой инструкции. <разделитель> - любой печатаемый символ. Первое появление разделителя должно быть одной строке с псевдо-операцией. Завершающий разделитель может появиться в ЛЮБОМ МЕСТЕ в последней строке блока. Блок будет завершен в последующем конце строки.

**.DEPHASE**

Отмечает конец блока PHASE. (См. также .PHASE). После появления инструкции .DEPHASE ассемблирование продолжится с адресов, являющихся продолжением тех, которые действовали, при вызове блока .PHASE, и в той же основе перемещения (ASEG, CSEG, и т.д.).

.HD64 Синоним: .180

Отключает нефатальные предупреждающие сообщения для кодов операций процессора HD64180. Это - нормальный режим по умолчанию ассемблера. (См. .Z80)

**.IF1**

Использует младший значащий бит для вычисления выражений как TRUE или FALSE. Этот метод совместим с используемым Digital Research в их ассемблерах ASM, MAC и RMAC. Выбор между значением по умолчанию ".IF1" или ".IF16" является настраиваемым параметром.

**.IF16**

Использует полные 16 бит значения выражения при вычислении значения TRUE или FALSE. Этот метод совместим с используемым Microsoft в своем ассемблере Macro-80. Выбор по умолчанию между .IF1 и .IF16 определяет параметр конфигурации.

.IN FS Синонимы: MACLIB, $INCLUDE, INCLUDE

Вызывает файл определенный FS для включения в ассемблирование в этой точке. Файл может содержать что-нибудь, обычно приемлемое в основном исходном файле, включая другие операторы .IN. Включаемые файлы, могут быть вложены до 4 уровней. Оператор END ЗАВЕРШАЕТ ассемблирование так же, как если бы он был в основном файле. Вследствие этого, операторы END обычно не присутствуют во включаемых файлах.

**.LALL**

Перечисляет все строки макросов во время макрорасширений. Если .XLIST действует, конечно, ничего не отображается. (См. также .SALL и .XALL),

**.LCTL**

Активизирует перечисление строк, содержащих следующие (управление листингом) псевдо-операции: (См. также .XCTL)

PAGE .TITLE .SBTTL .LFCOND .SFCOND .LCTL

.XLIST .LIST .LALL .SALL .XALL .XCTL

**.LFCOND**

Перечисление ложных условных операторов в листинге ассемблирования. (См. также .SFCOND), гарантирует, что строки внутри условного оператора, который оценивает значение False, будут указаны даже если они не ассемблированы.

**.LIST**

Продолжает перечисление, которое было выключено .XLIST

**.PHASE** <exp>

Начало блока кода, адрес выполнения которого <exp>, где <exp> - абсолютное значение. См. также .DEPHASE. Абсолютный (ASEG) код, который следует за директивой .PHASE, в действительности располагается в сегменте кода (CSEG, например), в том случае, если встречается оператор .PHASE, хотя его адрес выполнения в другом месте. .DEPHASE отмечает конец блока. Код в блоке PHASE должен быть перемещен во время выполнения в адрес, определенный <exp>, если он должен быть выполнен.

Директивы ORG, ASEG, CSEG, DSEG и COMMON не допустимы в блоке PHASE.

**.PRINT** {<pass>,} QSTR | STR Синоним: PRINTX

**.PRINTX** {<pass>,} QSTR | STR Синоним: PRINT

Отображает в кавычках или без строку на консоль. Если присутствует <pass> (обратите внимание на обязательную запятую), он должен быть числом 1 или 2. Отображение будет происходить только во время прохода 1 или прохода 2, соответственно.

**.RADIX** n

Где n - десятичное число, больше 1. Устанавливает основание системы счисления, используемое для преобразования чисел из ассемблерного источника и команды ACCEPT. Текущее основание используется каждый раз, когда требуется преобразование в или из текста ASCII. Это особенно полезно в связи с ведущим символом %, который вызывает вычисление чисел в списке параметров макросов. Если иное не оговорено с помощью .RADIX, по умолчанию основание системы счисления чисел является десятичным.

**.REQUEST** FN {,FN...}

Передает имя (имена) к объектному (обычно .REL) файл. Редактор связей интерпретирует их как библиотеки, в которых будет осуществляться поиск неразрешенных внешних ссылок. Для совместимости с Microsoft имена должны иметь длину меньше чем 7 символов. Каждое имя является только именем (без спецификации каталога или расширения имени файла), и только первые 6 символов имени используются.

**.RLIST**

Восстанавливает флаги управления списком (LIST) из стека. Должна быть ранее использована инструкция .SLIST, чтобы продвинуть флаги управления списком. Стек 5 уровней в глубину. Флаги управления списком генерируются следующими (управления листингом) псевдо-операции:

.LFCOND .SFCOND .XLIST .LIST

.LALL .SALL .XALL .LCTL .XCTL

Эффект заключается в восстановлении окружения управления списком, который был в силе при появлении инструкции .SLIST. Отменяются любые ранее действовавшие операторы управления списком.

**.SALL**

Подавить всех макросов в листинге. Только строка вызова макроса будет отображаться в листинге. (См. также .XALL и .LALL)

**.SBTTL** { QSTR | TXT или [QSTR | TXT] } Синоним: SUBTTL

Определяет строку подзаголовка для использования на последующих страницах листинга ассемблирования. Следует тем же правилам, что и .TITLE.

**.SFCOND**

Подавляет перечисление операторов, которые следуют за оператором IFxx, который оценивает FALSE до следующего ELSE или ENDIF. (См. также .LFCOND).

**.SLIST**

Сохраняет текущие флаги управления списком в стеке. Стек глубиной 5 уровней. Средства управления списком могут быть восстановлены с помощью инструкции .RLIST, которая выталкивает стек. Флаги управления списком генерируются следующими (управления листингом) псевдооперациями:

.LFCOND .SFCOND .XLIST .LIST

.LALL .SALL .XALL .LCTL .XCTL

Типичное использование для этой инструкции перед оператором INCLUDE, который "включает" другой файл. Такой файл может содержать операторы управления списком в его собственных целях. После строки INCLUDE .RLIST вернет управление листингом в прежнее состояние независимо от операторов во включенном файле.

**.TITLE** { QSTR | TXT или [QSTR | TXT] } Синоним: TITLE

Строка в кавычках или без сохраняется и используется в качестве строки заголовка на последующих страницах листинга. Открывающие и закрывающие кавычки не включаются в заголовок, но кавычки в заключенной в кавычки строке включаются один раз для каждого двойного появления (одно вхождение будет завершать строку). Символы кавычек могут быть или ' или ".

Если за псевдо-операцией .TITLE не следует аргумент не следует, то строка заголовка очищается к пустой.

**.XALL**

Исключает из листинга все строки макроса, которые не производят кода. Впоследствии будут перечислены только те строки макросов, которые производят код.

**.XCTL**

Отключение перечисления строк, содержащих следующие (управления листингом) псевдооперации: (См. также .LCTL)

PAGE .TITLE .SBTTL .LFCOND .SFCOND

.XLIST .LIST .LALL .SALL .XALL

.LCTL .XCTL

**.XLIST**

Подавление перечисления, до появления инструкции .LIST. Пока действует XLIST инструкции .TITLE и .SBTTL игнорируются и строки заголовков остаются, какими они были до инструкции .XLIST.

Следующие псевдо-операции подавляются .XLIST. Эффект заключается в предотвращении переводов формата, изменений в строках заголовка или подзаголовка, изменений в листинге макроса, изменений в листинге ложных условных выражений и изменения в отображении строк, содержащих операторы управления списком в результате действия операторов, находящихся в пределах диапазона пары операторов .XLIST - .LIST.

PAGE .TITLE .SBTTL .LFCOND .SFCOND

.LALL .SALL .XALL .LCTL .XCTL

За исключением .XLIST и .LIST они включают группу псевдоопераций управления листингом (PRN).

**.Z80**

Эта псевдо-операция устанавливает ассемблер в режим кода Z80. Любые инструкции, уникальные для HD64180/Z180, ассемблируются правильно, но отмечаются как нефатальные ошибки. (См. .HD64)

**ASEG**

Устанавливает счетчик адреса для абсолютного сегмента кода. Когда ASEG встречается впервые, значение счетчика адреса по умолчанию равно 0000h. Если ассемблерный код не предназначен для выполнения по адресу 0, то должен следовать оператор ORG, устанавливающий адрес выполнения. Обычно оператор ORG будет определять 100H (не забывайте 'H'!), в качестве начального адреса в Z-system, но более высокие адреса также могут быть уместными.

**ASET** Синонимы: SET, DEFL

(Определение см. в SET)

Эта мнемоника псевдо-операции используется ассемблерами, которые не могут различить (из контекста) между псевдо-операцией SET и инструкцией SET Z80. ZMAC способен различать контекст и знает для чего SET предназначен.

**COMMON** {[/]<NAME>[/]}

Устанавливает счетчик адреса именованному общему сегменту. Если <NAME> отсутствует, общая область определяется с именем по умолчанию BLANK. С помощью этой псевдооперации, кроме BLANK, могут быть определены до двенадцати именованных общих областей.

BLANK common назначается с

COMMON <== без аргумента

или COMMON BLANK

или COMMON /BLANK/

Аргумент после псевдо-операции COMMON принимается с или без разделителя '/'. Для именованного COMMON используйте имя по вашему выбору вместо 'BLANK'.

При попытке активировать более 12 именованных общих областей, появляется ошибка 'T'. (Слишком много именованных common).

**CSEG**

Устанавливает счетчик адреса для сегмента перемещаемого кода. Код, сгенерированный в сегменте CSEG, перемещаемый и начинается с предполагаемого значения 0 для счетчика адреса. Оператор ORG ДОБАВЛЯЕТ свой аргумент к текущему счетчику адреса в сегменте кода. Последующий код производится с новым значением счетчика адреса.

CSEG является режимом по умолчанию ZMAC. Все инструкции ассемблируются в сегмент кода до появления определений ASEG, DSEG или COMMON.

**DB** n {,n....} Синонимы: DEFB, DEFM

Определение данных байта. n может быть значением байта, заключенной в кавычки строкой или выражением. Выражения производят 16 разрядные значения. Когда такое значение представлено в существующем контексте, используется младший байт, и старший байт игнорируется по умолчанию. Может использоваться специальный оператор 'HIGH' для определения, что используется старший байт вместо младшего. Обратитесь к Главе [Выражения](#_Выражения_2) для обсуждения HIGH.

**DC** QSTR {,n...} Синоним: DEFC

Определяет символьные данные. n - данные байта (обычно 0dh, 0ah). n не может быть другой заключенной в кавычки строкой. Последний байт потока данных изменяется, так что бит 7 (старший бит) имеет значение 1. (аналогично псевдо-операции TDL .ASCIS)

**DEFB** Синонимы: DB, DEFM

Определение байта (см. описание DB)

**DEFC** Синоним: DC

Определение символов (см. описание DC)

**DEFL** Синонимы: ASET, SET

Определение метки (см. описание SET)

**DEFM** Синонимы: DB, DEFB

Определение памяти (см. описание DB)

**DEFS** Синоним: DS

Определение места для хранения. (См. описание DS)

**DEFW** Синоним: DW

Определение слова (см. описание DW)

**DEFZ** Синоним: DZ

Определение завершенной нулем строки (См. описание DZ)

**DS** <exp> {,<exp>} Синоним: DEFS

Определение места для хранения. Первый аргумент определяет длину пространства в байтах. Второй (необязательный) аргумент определяет значение байта, которым можно заполнить пространство.

**DSEG**

Устанавливает счетчик адреса для сегмента перемещаемых данных. Сегмент перемещаемых данных следует тем же правилам, что и сегмент перемещаемого кода. Несмотря на то, что имя подразумевает только данные, не существует проблем с включением кода (инструкций) в сегмент данных так же, как нет никаких проблем с включением операторов данных в сегмент кода.

**DW** nn {,nn...} Синоним: DEFW

Определение данных слова. nn - 16 разрядные величины. Это может быть выражение. Можно определить несколько слов, разделив их запятыми. Все значения, определенные DW, сохраняются в стандартном формате слова Z80/Z180 с младшим значащим байтом, сохраненным в более низком адресе. Обратите внимание, что это - истина даже при том, что значения в листинге ассемблирования могут отображать сначала старший байт.

**DZ** n {,n...} Синоним: DEFZ

Определение с нулем. Так же, как DB, за исключением двоичного нуля добавляемого к данным. n может быть выражением. (аналогично псевдо-операции TDL .ASCIZ)

**ELSE**

Часть блока IF ... ENDIF. Когда появляется ELSE, логическое значение предыдущего условного оператора IFxx инвертируется и используется новое значение для управления следующим ассемблированием. ELSE устанавливает новый уровень управления потоком. Может быть до 8 отложенных уровней управления потоком (8 уровней вложения), прежде чем встретится с ENDIF.

**END** {<exp>}

Объявляет конец текущего блока. Оператор END необязательный. Если отсутствует, ассемблер обнаруживает метку конца файла и интерпретирует ее как оператор END. Если присутствует дополнительный аргумент, он включается в объектный файл как начальный адрес программы. Редактор связей будет использовать это значение в качестве аргумент команды перехода, которую он установит на 0100H.

ZMAC завершает каждый проход ассемблирования при первом появлении оператора END, игнорируя все последующее. Несмотря на то, что завершение программы в файле INCLUDE является сомнительной практикой программирования, ZMAC предполагает, что программист ПРЕДПОЛАГАЕТ такую структуру, когда это происходит. Все после оператора END будет ПРОИГНОРИРОВАНО, даже после вызова оператора INCLUDE в исходном коде.

**ENDIF**

Завершает блок условного ассемблирования.

**ENDM**

Определяет конец макроопределения.

**ENT** Symbol {,Symbol...} Синонимы: ENTRY, GLOBAL, PUBLIC

(См. описание PUBLIC)

**ENTRY** Symbol {,Symbol...} Синонимы: ENT, GLOBAL, PUBLIC

(См. описание PUBLIC)

**EQU** Синтаксис: <symbol> \_ EQU \_ <exp>

Директива EQU объявляет, что значение <exp> присваивается символу с именем в первом поле. Символ может дополнительно быть завершен ':' или '::'. <exp> должно быть вычислено к абсолютному или перемещаемому значению к концу первого прохода. Внешние символы не могут использоваться в <exp> потому, что они не вычисляются до компоновки.

Типичное использование для EQU:

FALSE EQU 0

TRUE EQU NOT FALSE

YES EQU TRUE

EQU не может использоваться для переопределения ранее определенного символа, если переопределение не является просто копией более раннего EQU. Весьма распространено иметь такие константы, как TRUE и FALSE, избыточно определенные в нескольких модулях, которые могут быть с помощью оператора INCLUDE включены в программу на ассемблере. Такое использование принимается без ошибок, так как переопределение не меняет значения символа.

Переопределение метки не допускается. Результаты такого переопределения могут быть достигнуты более традиционными способами. (Используйте другую метку!)

**EXITM**

Вызывает выход от макрорасширения. Обычно часть условного блока. EXITM завершает расширение, когда он ассемблируется. (см. [*Главу 7*](#_Обработка_макросов_2))

**EXT** Symbol {,Symbol...} Синонимы: EXTERNAL, EXTRN

(См. описание EXTERNAL)

**EXTERNAL** Symbol {,Symbol...} Синонимы: EXTRN, EXT

Каждый символ в списке передается через объектный файл REL редактору связей для определения во время компоновки из некоторого другого модуля, в котором символ был объявлен PUBLIC (доступным). Существует параметр конфигурации ZMAC для исключения этих символов из REL файла, на которые в противном случае нет возможности сослаться в текущем модуле. Такое исключение является желательным, если список EXTERNAL (внешних) символов является частью 'шаблонного' заголовка, который содержит символы, не используемые в текущем модуле. Не имеющие ссылки символы вызовут сообщение редактора связей 'неопределенный символ', если они не встречается в одном из соединяемых модулей. Если не имеющий ссылки символ появится как точка входа в соединяемой библиотеке REL, то код, содержащий этот символ, будет включен в окончательно соединенный код. Это является способом вызвать включение блока кода из библиотеки хотя он никогда не указывается, за исключением оператора EXTERNAL.

**EXTRN** Symbol {,Symbol...} Синонимы: EXTERNAL, EXT

(См. описание EXTERNAL)

**GLOBAL** Symbol {,Symbol...}

Объявляет список символов, которые совместно используются с другими (внешними) модулями. Символы GLOBAL, определенные в текущем модуле, автоматически помечаются 'PUBLIC'. Инструкция PUBLIC не должна использоваться. Список GLOBAL может быть файлом включенным с помощью INCLUDE, используемым в нескольких модулях, чтобы освободить программиста от обязанности вести соответствующие списки PUBLIC и EXTERNAL в каждом модуле.

**IF** <exp> TRUE, если <exp> вычисляется к TRUE

Эта и следующие псевдо-операции IFxx управляют ассемблированием последующих строк, пока не встречаются с оператором ELSE или ENDIF. Когда инструкция IFxx вычисляет TRUE, то последующие строки включены в ассемблирование. Когда IFxx - FALSE, строки не ассемблируются.

Истинность основывается на значении ее младшего значащего бита (0=False, 1=True), ИЛИ на ее общем значении (16 битов) (0=False, иначе True). Какой из этих двух критериев используется определяется параметром конфигурации (if true based on...), который может быть заменен в ассемблировании псевдо-операцией .IF1 или .IF16.

ZMAC вычисляет <exp>, когда встречаются строка IFxx во время ассемблирования. Все элементы <exp>, должно быть определены ранее.

Условие IF/{ELSE}/ENDIF может полностью содержаться в другом условии   
IF ... ELSE, ELSE ... ENDIF или IF ... ENDIF. Это называют вложением. Условное ассемблирование этого вида может быть вложено до 8 уровней. Это означает, что до 8 незаконченных IF или ELSE с незавершенным ENDIF могут быть в силе. Так как макросы могут содержать условия IF/ELSE/ENDIF, необходимо позаботиться, что расширение макроса не вызовет превышение уровня вложенности 8. Переполнение вложенности приводит к ошибке во время ассемблирования.

**IF1** (Синоним: IFP1) TRUE при обработке прохода 1

*Примечание*: это не то же, что псевдо-операция .IF1 !

**IF2** (Синоним: IFP2) TRUE при обработке прохода 1

**IFB** <arg> TRUE, если отсутствует аргумент

Угловые скобки ТРЕБУЮТСЯ.

**IFDEF** <symbol> TRUE, если <symbol> определен

**IFE** <arg> TRUE, если <arg> = 0

**IFF** <arg> TRUE, если <arg> равен FALSE

**IFG** <arg> TRUE, если <arg> > 0

**IFGE** <arg> TRUE, если <arg> >= 0

**IFL** <arg> TRUE, если <arg> < 0

**IFLE** <arg> TRUE, если <arg> <= 0

**IFN** <arg> TRUE, если <arg> <> 0

**IFNB** <arg> TRUE, если аргумент присутствует

Угловые скобки ТРЕБУЮТСЯ.

**IFNDEF** <symbol> TRUE, если <symbol> еще не определен

**IFP1** (Синоним: IF1) TRUE при обработке прохода 1

**IFP2** (Синоним: IF2) TRUE при обработке прохода 2

**IFT** <exp> TRUE, если <exp> равно TRUE (=IF <exp>)

**INCLUDE** FS Синонимы: .IN, MACLIB, $INCLUDE

Включение исходного файла. (см. .IN для определения.)

**IRP** <identifier>,STR {,STR...}

Макрос повторения переменного числа строк (см. [*Главу 7*](#_Обработка_макросов_3))

**IRPC** <identifier>,STR

Макрос повторения переменного числа символов (см. [*Главу 7*](#_Обработка_макросов_4))

**LOCAL** <identifier> { ,<identifier>...}

Объявляет локальные (в макросе) символы. Пространство, выделенное для локальных меток, составляет 127 байт. Суммарная длина локальных символов в любом макроопределении не может превысить это число.

**MACLIB** FS

Так же, как псевдо-операция .IN, за исключением того, что эта инструкция настраивается с помощью ZCNFG для чтения файла только на первом проходе. В этом случае файл не может содержать операторы, которые генерируют код. Операторы EQU/SET соответствуют этому требованию.

**MACRO** Синтаксис: <name> MACRO {arg {,arg...}}

Заголовок макроопределения. <name> должно присутствовать и при необходимости может быть завершено двоеточием. Имя, присвоенное макросу, уникально для макроса и может совпасть с определенной меткой, не вызывая путаницы (для ZMAC). Если имя совпадет с одной из других псевдо-операций, то макрос будет иметь приоритет. Замененная псевдо-операция становится недоступной. См. также ENDM, которая используется для завершения макроопределения.

**NAME** QSTR | TXT или [ QSTR | TXT ]

Аргумент обеспечивает имя текущего модуля. Это имя используется редактором связей и менеджером библиотек, чтобы идентифицировать код, произведенный текущим ассемблированием. Если инструкция NAME не используется, то имя модуля совпадет с именем файла. Вторая форма совместимая с синтаксисом, требуемым M80: ['<symbol>']

**ORG** <exp>

Использует <exp> для изменения значения текущего счетчика адреса. Если текущая основа перемещения - ASEG, значение расположения устанавливается <exp>. Для других основ перемещения счетчик адреса увеличивается на значение <exp>.

**PAGE** { <exp> {,<exp>} }

Без аргументов эта инструкция отправляет символ прогон страницы в устройство листинга.

С параметрами, эта псевдо-операция сбрасывает значения длины и ширины страницы, используемые для управления листингом ассемблирования. Эти параметры имеют значения по умолчанию 66 строк и 132 символа. Значения по умолчанию настраиваются с помощью ZCNFG.

Если присутствует первый параметр, он определяет число строк на странице, которое будет использоваться для последующего разбиения на страницы. Второй (необязательный) параметр определяет ширину страницы листинга. Если первый параметр 0, сохраняется текущее значение по умолчанию. Если второй параметр - меньше 10, сохраняется текущая ширина страницы по умолчанию.

**PUBLIC** Symbol {,Symbol...} Синонимы: ENT, ENTRY, GLOBAL

Объявляет список символов доступных для ссылок из других (внешних) модулей. Символы в этом списке должны быть определены в текущем модуле. Местонахождение метки составляет такое определение. Символы объявленные PUBLIC доступны для использования другими модулями, в которых они объявлены как EXTERNAL. (См. EXTRN)

**REPT** <exp>

Макрос повторения. Последующая группа операторов, завершенная оператором ENDM, будет ассемблирована несколько раз. Число повторений задается значением <exp>. Все переменные, используемые в выражении, должно быть определены ранее. (Недопустимы внешние ссылки или ссылки вперед) (см. [*Главу 7*](#_Обработка_макросов_1))

**SET** Синонимы: ASET, DEFL

Синтаксис: <symbol> \_ SET \_ <exp>

Определяет значение <symbol>, полученного из <exp>. Символ - первый элемент в строке (без предшествующих пробелов) и может быть завершен с двоеточием. Пробелы должны окружать слово SET. Подобен EQU, за исключением того, что SET может использоваться для переопределения символов, которые были ранее определены с помощью SET или EQU (или их синонимами). Будет выдаваться ошибка при попытке переопределения метки.

**SUBTTL** { QSTR | TXT или [QSTR | TXT] } Синоним: .SBTTL

(См. описание.SBTTL)

**TITLE** { QSTR | TXT или [QSTR | TXT] } Синоним: .TITLE

(См. описание .TITLE)

# Перемещение и основы перемещения

## Перемещение

Перемещение обеспечивает средства объединения REL файлов для создания единого составного абсолютного файла, готового к выполнению. Во время объединения код может быть загружен относительно любой из нескольких основ перемещения, каждая из которых определяет уникальное адресное пространство.

Основы перемещения определяются в файле Z80, вызовом одного из следующих операторов:

|  |  |
| --- | --- |
| **Оператор:** | **Активирует основу перемещения:** |
| ASEG | Абсолютную (не перемещаемый код) |
| CSEG | Относительного кода - ZMAC по умолчанию |
| DSEG | Относительных данных |
| COMMON {NAME} | Общих относително именованной основы |

Одну из 13 позволенных COMMON основ, по умолчанию, называют BLANK. Другие 12 доступны для определения с помощью оператора COMMON. (См. [*Главу 5*](#_Псевдо-операции_(директивы_ассемблера)) для синтаксиса и использования этой псевдо-операции.)

По мере обработки ZMAC исходных операторов, байты кода отправляются в объектный файл. Внутренний счетчик увеличивается на единицу для каждого сгенерированного байта. Этот внутренний счетчик называют счетчиком команд или счетчиком адреса. Некоторые исходные строки, как комментарии, не производят объектный код, поэтому, они не увеличивают счетчик команд.

Счетчик команд (PC) содержит 16-разрядное число. Так как PC установлен в 0 в начале ассемблирования, его значение - относительный адрес следующего байта, который будет отправлен в объектный файл. Это - значение, отображается на экране в поле **Адрес** листинга ассемблирования в начале каждой строки, которая производит байты кода. Фраза 'относительный адрес' обычно сокращается до 'адрес'.

Два фрагмента листинга ассемблирования показаны ниже в *Листингах 1* и *2*. Они получены ассемблированием источников Z80, единственным отличием которых является оператор ORG, которые устанавливают адрес начала программы равным 0 в первом случае и 100H во втором. Обратите внимание, что на три 16-разрядных значения являются ссылками: START, Z3EADR и 0FE00H.

**Листинг 1. Ассемблирование с начальным адресом 0000H**

Адрес Объекный код Исходный код

ORG 0

0000 C3 000B JP START

0003 5A 33 45 4E DB 'Z3ENV'

0008 01 DB 1

0009 FE00 Z3EADR: DW 0FE00H

000B 2A 0009 START: LD HL,(Z3EADR)

**Листинг 2. Ассемблирование с начальным адресом 0100H**

Адрес Объекный код Исходный код

ORG 100H

0100 C3 010B JP START

0103 5A 33 45 4E DB 'Z3ENV'

0108 01 DB 1

0109 FE00 Z3EADR: DW 0FE00H

010B 2A 0109 START: LD HL,(Z3EADR)

Меткам Z3EADR: и START: присвоены текущее значения PC в качестве значений их адресов. Эти значения затем используются в качестве аргументов 'nn' для кодов операций в поле кода. 0FE00H - число, предоставленное программистом. Предполагается, что оно является константой и хранится в объектном коде по адресу 0009H.

Адрес выполнения кода в *Листинге 1* равен 0000H, который для кода в Листинге 2 равен 100H.

Сравнивая объектный код в этих двух листингах, очевидно, что 16 разрядные аргументы в первой и четвертой строках изменились. Величина на которую они изменились равна 100H, разнице между двумя адресами исполнения.

Подразумеваемый процесс, просто иллюстрирует **перемещение** объектного кода для изменения его адреса выполнения. Перемещение осуществляется добавлением 16 разрядного числа, **константы перемещения** для 'адаптации' 16 разрядных чисел в объектном коде. Значение константы перемещения является разницей между новыми и старыми адресами выполнения. 'Подходящими' 16 разрядными числами в объектном коде являются те, исходными значениями которых является адрес или адресное выражение. Такие числа являются **перемещаемыми**. Они перемещаемы из-за контекста, в котором они определены в исходном коде.

Много величин исключены из процесса перемещения, потому что их функция или значение не зависят от их адреса. Такие величины являются абсолютными. Байты (как в операторах DB) абсолютные. Часть оператора, коды операций - абсолютные. Данные байта в части операнда кодов операций - абсолютные. Слово является абсолютным, если оно не перемещаемо.

Объектный код в примерах выше представляет байты, которые в конечном счете могут быть загружены в память. Она не содержит никакой информации, чтобы определить, какие байты, если таковые имеются, должны быть 'в паре' как слова (потенциальные адреса). Адрес загрузки или выполнения не указан. Код, отправляемый в объектный ФАЙЛ, содержит такую дополнительную информацию, характер которой зависит от того, какой тип объекта производится.

Файлы HEX содержат информацию об адресах выполнения в дополнение к объектному коду. Так как объектный код в объектных файлах HEX абсолютный, адреса загрузки и выполнения идентичны. При создании HEX файла ассемблер работает в **абсолютном режиме**.

Файлы REL содержат о перемещении данные в дополнение к объектному коду. Перемещаемые слова помечены так, чтобы компоновщик смог выполнить, любое необходимое перемещение при создании абсолютного выходного файла. Адрес выполнения для перемещаемого кода не установлен до момента компоновки. При создании файла REL ассемблер, как говорят, находится в **перемещаемом режиме**.

Файл Z80 содержит последовательность строк на языке ассемблера, каждая из которых может произвести объектный код при компиляции. Произведенный объектный код является смесью элементов двух видов: элементов кода и элементов, которые содержат информацию об элементах кода. **Элементы кода** происходят из операторов с кодами операций Z80 и операторов хранения данных типа DB, DW и т.д. **Информационные элементы** генерируются ассемблером в ответ на определенную псевдо-операции и конструкции исходного кода такие, как использование внешних аргументов.

Исходную программу Z80 можно символически рассматривать как последовательность, наподобие первой строки, на *Рисунке 6-1*, в которой все детали каждой строки скрыты, и обозначены только порядковые номера строк (L1, L2 ...).

Объектный код REL, произведенный ассемблером, не является строчно-ориентированным, но элементы кода находятся в той же последовательности. Объект REL показан схематично во второй строке *Рисунка 6-1*. В данном случае, Ox представляют элементы кода, и I представляют элементы информации из ассемблера. Так как некоторые исходные строки не производят объектный код, индексы в Z80 и в представлении REL на *Рисунке 6-1* не обязательно совпадают. Однако, их ПОРЯДОК сохраняется.

Абсолютный код, произведенный редактором связей, организован как показано в строке 3, при условии, что ни один из информационных элементов не направил редактор связей для изменения последовательности кодов. Информационные элементы обычно используются редактором связей, чтобы изменить ЗНАЧЕНИЯ элементов объектного кода, для создания окончательного кода, представленного в третьей строке *Рисунка 6-1*. Здесь, есть корреспонденция между O1 и M1....On и Mn. Каждый элемент может содержать одним или несколько байтов.

Рисунок 6‑1 Последовательность кода при ассемблировании/компоновке

Источник Z80: [L1, L2, L3, L4........................Ln]

Объект REL: [..O1,I1,..O2,O3,..I2,O4,....On,In]

Машинный код: [M1, M2,............Mn]

В предыдущем обсуждении ничто не было сказано об адресах или выделении памяти. Очевидно, если машинный код является полезным (и сохраняется в файле), ему должен быть выделен диапазон последовательных адресов соответствующий M1,... Mn. Первый адрес (для M1) является **начальным адресом**. Размер программы определен, это - общее количество байтов, выделенных для машинного кода. Начальный адрес присваивается редактором связей. Этот начальный адрес символически упоминается как **основа перемещения**.

## Основы перемещения

Основы перемещения являются символьными именами, присвоенными независимо расположенным областям памяти. Такими областями могут быть отображенные в карте памяти порты ввода-вывода, ПЗУ, совместно используемые общие области или просто область дискретных данных для упрощения отладки. Каждая из этих областей имеет имя. Определены четыре таких имени: абсолютная (ABS), относительного кода, относительных данных и Blank. Blank первая из 13 возможных совместно используемых COMMON основ перемещения. Остальные 12 имен выбираются программистом с помощью оператора COMMON. Во время ассемблирования отдельный счетчик команд используется для перемещения каждой основы, таким образом, каждая ссылка на метку в программе дается относительно одной из этих основ. Фактическое выделение и отображение основ перемещения к физическим адресам откладывается до процесса компоновки.

Основы перемещения определяются в файле Z80, вызовом одного из следующих операторов:

|  |  |
| --- | --- |
| **Оператор:** | **Активирует основу перемещения:** |
| ASEG | Абсолютную (не перемещаемый код) |
| CSEG | Относительного кода - ZMAC по умолчанию |
| DSEG | Относительных данных |
| COMMON {NAME} | Общих относително именованной основы |

См. [*Главу 5*](#_Псевдо-операции_(директивы_ассемблера)) для синтаксиса и использования этой псевдо-операции.

ASEG, CSEG, DSEG и COMMON указывают ассемблеру, пометить последующий код как принадлежащий обозначенной основе перемещения. ZMAC использует счетчик команд, выделенный этой основе при присвоении значений меткам. Основа перемещения может изменяться так часто, как вы хотите, включением вышеупомянутых операторов. Эффект состоит в том, что теперь есть до 16 логических групп кода, отправляемого в объектный файл. Группы называются сегментами, с именами, отражающим имена псевдо-операций выше. Концепция показана на *Рисунке 6-2*. Для источника Z80 C1... и C2... обозначают группы операторов после директивы CSEG. D1... и D2... обозначают группы операторов после директивы DSEG. Как и раньше, ассемблер производит объектный код в той же последовательности, как он встречается в источнике, и I представляет приобщенные информационные данные, которые определяют основу перемещения для каждого байта кода. Редактор связей сортирует код для каждого сегмента и, как подразумевается в третьей строке *Рисунка 6-2*, загружает его в непрерывный набор адресов. Перемещаемые значения в коде перемещаются относительно начала C1 и D1 этих двух сегментов соответственно.

Рисунок 6‑2 Последовательность сегментов при ассемблировании/компоновке

Источник Z80: [C1... D1... C2... D2...]

Объект REL: [I..C1..I.. D1.. I.. C2..I... D2..I.]

Машинный код: [C1,C2......D1,D2]

ZML переносит эту стратегию один шаг вперед при соединении нескольких модулей REL. Модули соединяются по одному в порядке, в котором они названы в командной строке. Соединение следует стратегии, показанной на *Рисунке 6-2*. Сегменты из каждого модуля объединяются в соответствующих им основах перемещения так, чтобы все сегменты **данных** были вместе в одном диапазоне адресов, все области **кода** находятся в другом непрерывном диапазоне, и т.д. для других основ перемещения. Процесс показан на *Рисунке 6-3*. Обратите внимание, что символы были несколько сокращены, игнорируя упоминание об информационных данных в файлах REL и позволив одним символом логически обозначить отдельные сегменты в каждом модуле REL.

Рисунок 6‑3 Соединение нескольких модулей

Источники REL: X.REL, Y.REL, Z.REL

Сегменты REL: [CX, DX], [CY,DY], [AZ,CZ,DZ]

Машинный код: [AZ... CX,CY,CZ... DX,DY,DZ]

Во время ассемблирования оператор ORG приводит к различным результатам для **абсолютного** сегмента, чем для других (перемещаемых) сегментов. Для абсолютного сегмента, 'ORG <exp>' устанавливает счетчик команд в значение выражения. Для перемещаемых сегментов тот же оператор **увеличивает** счетчик адреса на значение <exp>!

Псевдо-операция COMMON может определить одну из 13 именованных общих основ. Первая имеет имя BLANK и используется, когда имя не определено или когда определено имя BLANK или /BLANK/. Остальные 12 могут быть названы, используя любые 6 символьные алфавитно-цифровыми имена, которые начинаются с буквы. Разделители '/' вокруг имени необязательны. Они обеспечивают совместимость с источниками, подготовленными для других ассемблеров, которые требуют их.

При связывании нескольких модулей REL, области COMMON обрабатываются иначе, чем другие. Общие зоны с одинаковыми именами загружаются все по одному адресу. Они перекрываются. В результате только исходные данные в последней соединенной общей области, как может показаться, допустимы. (Они соединяются в той же последовательности в которой появляются в списке модулей REL в командной строке редактора связей).

Никакое перемещение не происходит во время соединения для абсолютной основы. Таким образом код из одного модуля может перезаписать его из другого.

# Обработка макросов

Макросом называется именованная последовательность операторов языка ассемблера, которая определяется в исходном файле. Макрос вызывается, когда его имя занимает поле OPR исходной строки. Вызов заставляет ассемблировать операторы макроса в этой точке прежде, прежде чем приступить к обработке следующей строки исходного кода.

Макрос может иметь список формальных параметров. Эти параметры соответствуют частям операторов языка ассемблера в теле макросов. Во время вызова параметры из строки вызова заменяют свои дубликаты в операторах в теле макроса, прежде чем операторы будут ассемблированы. Операторы в теле макроса можно рассматривать как макеты операторов, которые редактируются, перед ассемблированием в соответствии с параметрами обеспеченными при вызове макроса. Макро-операторы могут неоднократно ассемблироваться с выполнением редактирования во время каждого повторения. Единственный макрос может, например, генерировать исходный код для ряда драйверов устройств ввода-вывода. Ссылки, приведенные под названием "Программирование процессора Z80" в *Приложении 2*, содержат много примеров реализации макросов.

Существует два вида макросов: встроенные и сохраняемые. Имена встроенных макросов предопределены в ассемблере. Имена сохраняемых макросов определяются программистом в исходных строках ассемблера. Такие макросы называют 'Сохраняемыми' потому, что макроопределение (последовательность операторов) сохраняется для будущего вызова.

Макросы реализуются с помощью псевдо-операций, операторов, уникальных для макросов и обычных операторов ассемблера. Встроенные макросы являются псевдо-операциями. Семь из псевдо-операций, перечисленных в [*Главе 5*](#_Псевдо-операции_(директивы_ассемблера)_5), уникальны для макросов и использования макросов. Здесь они приведены с подсказкой назначения в круглых скобках.

|  |  |  |  |
| --- | --- | --- | --- |
| **Макросы псевдо-операции** | | **Встроенные макросы** | |
| **MACRO** | (определение) | **REPT** | (повторять <exp> раз) |
| **LOCAL** | (..метки) | **IRPC** | (повторять для каждого символа) |
| **EXITM** | (выход из макроса) | **IRP** | (повторять для каждой STR) |
| **ENDM** | (окончание макроса) |  |  |

Форма макросов обсуждается в следующем разделе. Последующие разделы описывают макросы псевдо-операции, операторы, уникальные для макросов, встроенные макросы и сохраняемые макросы.

## Форма макросов

Макросы встречаются в источнике ассемблера в двух контекстах: определение и вызов. *Таблица 7-1* показывает в символьной форме, как макросы кодируются в исходном файле ассемблера. Обратите особое внимание на различие между двумя видами макросов. Можно сказать, что определение встроенного макроса совпадает с его вызовом.

Таблица 7‑1 Форматы макросов

**Определение макросов**

*Сохраняемые макросы Встроенные макросы*

**<NAME> MACRO {<dlist>} (не применяется)**

**<body>**

**ENDM**

**Вызов макроса**

*Сохраняемые макросы Встроенные макросы*

**{label:} <NAME> {<plist>} {label:} <INAME> <plist>**

**<body>**

**{label:} ENDM**

**<body>** - (тело) последовательность операторов ассемблера. Единственный оператор, который является особенным, является MACRO. Он может присутствовать в теле сохраняемого макроса, но НЕ может встретиться во встроенном макросе.

**<dlist>** - список из нуля или большего количества параметров, разделенных запятыми. Эти параметры называют **фиктивными параметрами**. Они определяют место в списке параметров, которые, во время вызова, заменяются фактическими параметрами. Фиктивные параметры из <dlist> используются в макете исходного кода в <body>.

**<plist>** - список из одного или более параметров, разделенных запятыми. Параметры в <plist> называют **фактическими параметрами**.

Для сохраняемых макросов <plist> содержит то же число параметров как в <dlist>

Вместе, <plist> и <body> управляют исходным кодом, произведенным макросом.

Примечание: определение сохраняемого макроса должно предшествовать своему вызову в источнике.

*Совет*: Многие макросы, однажды определенные, бывают полезны во многих программах. Чтобы избежать их переписывания, они могут быть размещены в отдельном файле, имеющем традиционное расширение файла '.LIB'. затем, этот файл включается в конкретную программу на ассемблере с помощью оператора MACLIB или INCLUDE недалеко от начала исходного файла ассемблера.

## Макросы псевдо-операции

**ENDM**

Определяет конец макроопределения. Эта инструкция требуется в качестве последней инструкция в теле встроенных макросов (REPT, IRPC и IRP) и макроопределения (..MACRO..).

**EXITM**

Вызывает выход из макрорасширения. Обычно являясь частью условного блока, EXITM завершает расширение текущего макроса. Игнорируются любые оставшиеся строки тела макроса до ENDM. Если текущий макрос вложен в другом (внешнем) макросе, расширение внешнего макроса продолжается. EXITM выходит только из текущего уровня вложенности. EXITM и ENDM не взаимозаменяемые. Макрос, содержащий оператор EXITM, все равно должен быть завершен с помощью ENDM.

**LOCAL** <identifier> { ,<identifier>...}

Объявляет локальные (внутри макроса) символы. <identifier> - имя символа в теле макроса, включая любую метку для оператора ENDM. Каждый раз, когда макрос расширяется, ZMAC создает уникальный символ для каждого из символов, объявленного LOCAL. Созданные символы имеют форму '??nnnn', где nnnn изменяется от '0001' до '9999' для последовательного присвоения символам.

Если символы будут определены в макросе, не будучи объявленными LOCAL, то второй вызов макроса приведет ошибкам дублирования символов. Та же проблема возникает, если метки определены во встроенном макросе. Вторая итерация вызовет ошибку 'дублированный символ' для символов, которые не объявлены LOCAL.

Если используется оператор LOCAL, он должен быть первым оператором в теле макроса.

Место, выделяемое для локальных меток, составляет 127 байтов для каждого макроопределения. Суммарная длина локальных символов в любом отдельном макроопределении не может превысить это число.

**MACRO** Синтаксис: <name> MACRO {arg {,arg...}}

Заголовок сохраненного макроопределения. <name> должен присутствовать и может быть завершен необязательным двоеточием. Имя, присвоенное макросу, уникально для макроса и может совпадать с определенной меткой, не вызывая путаницы (у ZMAC). Если имя совпадет с одной из других псевдо-операции, то макрос будет иметь приоритет. Замененная псевдо-операция становится недоступной.

Список аргументов идентичен <dlist> в *Таблице 7.1*. Каждым параметр (аргумент) является местом размещения фиктивного параметра, цель которого состоит только в том, чтобы установить соответствие между местом в списке аргументов и вхождениями фиктивных параметров в текстовой строке в теле макроса.

Сохраняемые макросы могут быть переопределены с помощью оператора MACRO, который определяет имя ранее определенного макроса. При вызове макроса, самое последнее определение определяет, расширение какого макроса происходит.

## Макро-операторы

**Операторы, уникальные для макросов**

Макросы могут использовать следующие операторы для управления генерацией исходных строк:

**&** (Оператор конкатенации)

**;;** (Управление комментарием)

**^** (Одиночный символ Escape)

**%** (Оператор оценки числа)

**<>** (Разделители для подавления оценки)

Перечисленные выше операторы имеют смысл только в пределах списка параметров макроса и теле макроса. Эти операторы, 7 специальных псевдо-операций и обычные операторы ассемблера используются для определения сохраняемых макросов и вызова встроенных и сохраняемых макросов.

Следующие разделы описывают использование семи псевдо-операций и операторов для создания макроса.

## Встроенные (RPT) макросы

Встроенные макросы также называют предопределенными макросами или макросами повторения.

Встроенные макросы заставляют ZMAC многократно считывать каждую строку в <body> (теле), ассемблировать ее для создания объектного кода в каждой итерации. В макросах IRP и IRPC, части <body>, которые соответствуют <identifier> изменяются. Таким образом произведенный код отличается для каждой итерации. Показанная, необязательная метка, предшествующая макро-вызову, НЕ является частью самого макроса. Ее значением будет адрес первой строки кода, произведенного макросом. Однако, метка присутствующая в операторе ENDM, ассемблируется в каждом проходе <body>, потому что ассемблер должен читать (и ассемблировать) эту строку. Результат: если количество повторений будет больше, чем 1, то будет выдаваться ошибка фазы для меток, которые не переопределены псевдо-операцией LOCAL или с помощью оператора конкатенации &, как описано ниже.

**REPT** <exp>

Макрос повторения. Группа операторов в <body> будет ассемблирована несколько раз. Если метка присутствует в операторе ENDM, она является частью <body>. Она должна быть LOCAL или сочленённой меткой, которая изменяет значение в каждой итерации. Число повторений определяется значением <exp>. Все переменные, используемые в выражении, должны быть ранее определены. (Недопустимы внешние ссылки или ссылки вперед)

Например, следующий макрос генерирует блок данных, содержащий числа 1...10.

.LALL ; обеспечивает листинг в файле PRN

XXX SET 1 ; Устанавливает начальное значение вне макроса

REPT 10 ;; повторить 10 раз

DB XXX ;; генерирует оператор данных

XXX SET XXX+1 ;; постепенно увеличивает значение XXX

ENDM ;; завершает тело макроса

Этот пример иллюстрирует способ, при использовании более сложного выражения, чем приведено (XXX+1), позволяющий создать более полезный набор значений. Для действительно БОЛЬШОГО блока данных этот вид макроса может сэкономить время!

**IRPC** <identifier>,STR

Макрос повторения переменного числа символов. STR - строка символов, которая не включает пробелов. <identifier> - любой допустимый символ.

Для каждого символа в STR, <body> ассемблируется один раз. Таким образом количество повторений для этого макроса является количество символов в STR. Перед ассемблирования каждой строка <body>, любые вхождения <identifier> в строке заменяются текущим символом из STR.

Следующий макро-вызов:

exirpc: IRPC xx,cbed

ld xx,(hl) ;; игнорировать комментарий

inc hl ; повторять комментарий

ENDM

произведет следующий ассемблерный код:

ld c,(hl)

inc hl ; повторять комментарий

ld b,(hl)

inc hl ; повторять комментарий

ld e,(hl)

inc hl ; повторять комментарий

ld d,(hl)

inc hl ; повторять комментарий

**IRP** <identifier>,STR {,STR...}

Макрос повторения переменного числа строк. Стратегия очень походит на стратегию макроса IRPC, за исключением того, что количеством повторений является число строк в которых вся текущая STR заменяет вхождение <identifier>. Простым примером служит создание таблицы переходов, аналогичной в начале BIOS:

bios: IRP xx,cboot,wboot,const,conin

xx&v: jp xx

ENDM

произведет:

bios:

cbootv: jp cboot

wbootv: jp wboot

constv: jp const

coninv: jp conin

......

Обратите внимание на использование оператора конкатенации (&) для создания меток, соответствующих целям переходов. Если бы метка в макросе была 'v&xx:', тогда были бы сгенерированы как 'vcboot:'.

## Сохраняемые макросы

Как показано в *Таблице 7-1*, сохраняемый макрос определяется, и затем позже вызывается по имени. С точки зрения программирования определение означает написание блока кода как в таблице. Вызов еще более прост. Следующие разделы описывают эти два аспекта с точки зрения обработки текста макропроцессором ZMAC и действий, выполняемых специальными макро-операторами.

## Определение и вложенность

**Макроопределение и его хранение**

Когда инструкция MACRO встречается в поле OPR строки ввода, ZMAC сохраняет имя макроса в своих внутренних таблицах. Все строки от этой точки до соответствующего ENDM обрабатываются следующим образом.

1. Список фиктивный параметров сохраняется в буфере, выделенном текущему макро-имени.
2. Метки, перечисленные в операторе LOCAL, сохранены в списке, который является индивидуальным для этого макроса.
3. Другие строки от <body> сохраняются в той же буферной зоне, кроме…
4. Символы от ';;' до конца строки не сохраняются. Сдвоенные символы точки с запятой начинают поле комментария, которое выводится в файл PRN ТОЛЬКО там, где определен макрос. Такие комментарии не появляются в листинге макро-вызова, потому что они не включены в сохраняемое <body>.

**Переопределение макроса**

Макрос может быть переопределен. Переопределение означает, что имя ранее определенного макроса используется повторно в новом определении макроса. Когда имя используется для вызова макроса, самое последнее определение используется для расширения.

**Вложенные макроопределения**

В макроопределении может быть записано второе макроопределение. Каждое такое определение завершается его собственным ENDM. Вы можете вкладывать макросы до 16 уровня, где первый вложенный макрос находится на 1 уровне вложенности.

Вложенный макрос может переопределить макрос, в который он вложен!

Встроенные макросы могут быть вложены в сохраняемый макрос. Обратное, однако, недопустимо. Сохраняемый макрос НЕ может быть определен во встроенном макросе.

## Макро-вызов

Когда макрос вызывается, ZMAC принимает ввод строки из макропроцессора (МП) вместо непосредственно из исходного файла. МП в свою очередь берет свой ввод текста их образа памяти <body> макроса вместе с его параметрами и локальными метками. Встроенные макросы хранятся во временном буфере, который освобождается, когда счетчик повторений завершен. Сохраняемые макросы постоянно находятся в выделенной памяти, откуда их можно вызвать неоднократно по имени. Образ в памяти изменяется только, когда макрос переопределяется (заменяется другим образом).

МП копирует строки из <body> на вход подпрограммы ZMAC. Во время передачи фактических параметров из строки вызова макроса заменяются фиктивными параметрами из <body>. Этот процесс называют **макрорасширением**. Во время расширения фактический параметр может быть изменен. В этом заключается практическая ценность макросов. Следующие пункты описывают процесс макрорасширения для одной типичной строки тела макроса.

1. В строке из <body> ищутся символьные строки, которые подлежат замене строками фактических параметров. Они определяются путем поиска строк, которые совпадают в списке фиктивных параметров, при соблюдении определенных текстовых условий:

Строка НЕ фиктивный параметр если:

а. найдена в поле комментария

б. предшествует ^ (стрелка вверх)

в. окружена пустым пространством или запятыми в пределах QSTR

Строка ЯВЛЯЕТСЯ фиктивным параметром если:

а. предшествующий или последующий символ '&'

б. окружена пустым пространством или запятыми, и НЕ является частью QSTR

В следующих шагах, 'фиктивный параметр' означает те строки, которые соответствуют вышеупомянутым правилам.

1. Каждый фиктивный параметр 'в паре' с соответствующим ФАКТИЧЕСКИМ параметром, заданным в строке вызова как часть <plist> на основе соответствия позиций фиктивных и фактических параметров в <dlist> и <plist>. Предшествующие пробелы игнорируются в параметрах обоих списках.
2. Фиктивные параметры в строках из <body> заменяются соответствующими им фактическими параметрами. Эту замену называют, 'передачей параметров'. Прежде, чем передать фактический параметр, его первый символ исследуется. На основе идентификации первого символа замена происходит в соответствии с одним из четырех возможных наборов правил. Три случая определяют символы:  
   ' (одинарная кавычка), < (левый угол) и % (знак процента). Любой другой начальный символ означает четвертый случай. *Таблица 7-2* содержит четыре случая замены параметра.
3. Если символ в поле метки сохраненной макро-строки соответствует одному из символов в списке LOCAL, то он заменяется следующим доступным уникальным символом в форме '??nnnn:'.

Таблица 7‑2 Передача фактических параметров

|  |  |
| --- | --- |
| **Случай:** | **Действие замены фиктивное параметра:** |
| **'** | Заключенная в кавычки строка. (QSTR). строка, включая в кавычки, заменяет фиктивный параметр. |
| **<** | Это - строка в скобках. Все символы до завершающе правой скобки (>) заменяются на макет. Открывающие и закрывающие скобки НЕ включаются в замену. Скобки могут быть вложены. Вложенные <> пары включаются в замену. Если фактический параметр будет <<TXT>>, то строка, которая заменяет фиктивный параметр, будет <ТХТ>. Когда правые или левые угловые скобки встречаются в кавычках они рассматриваются как обычные символы без особого значения. |
| **%** | Последовательность символов, которая следует передается в вычислитель выражений для немедленной оценки как 16 разрядное значение. Затем это значение преобразуется в его ASCII представление в текущем основании системы счисления. Если псевдо-операция RADIX не использовалось для задания основания, то число будет в десятичном представлении (значения по умолчанию в ZMAC). Строка ASCII заменяет фиктивный параметр. |
| **<другой>** | Если первый символ не является одним из вышеупомянутых, то весь фактический параметр (последовательность символов до следующего пробела, запятой или точки с запятой) заменяет фиктивный параметр. |

## Правила макрорасширения

Во время макрорасширения фактические параметры из <plist> заменяют фиктивные параметры в тексте из <body>.

Фиктивные параметры распознаются и заменяются если они:

|  |  |  |
| --- | --- | --- |
|  | - | являются строкой, которая соответствует одному из элнментов <dlist>, |
| и | - | ей предшествуют пробел, символ табуляции или запятая и за ней следуют пробел, символ табуляции, запятая или конец строки, |
| или | - | ей предшествует амперсанд (&), даже в заключенной в кавычки строке, |
| или | - | за ней следует амперсанд (&), даже в заключенной в кавычки строке. |

Фиктивные параметры, удовлетворяющие другим разумным требованиям изложенным выше, не заменяются, если они:

|  |  |  |
| --- | --- | --- |
|  | - | встречаются в поле комментария, |
| или | - | им предшествует стрелка вверх (^), |
| или | - | встречаются в заключенной в кавычки строке без амперсанда (&). |

Во время замены фиктивных параметров любые предшествующие или последующие '&' удаляются из строки. Стрелка вверх, которая предшествует фиктивному параметру удаляется, и замена этого макета фактическим параметром запрещена.

Фактические параметры передаются в соответствии со следующими правилами:

* Удаляются предшествующие и последующие пробелы;
* Заключенные в кавычки строки передаются нетронутыми, включая определяющие кавычки;
* Произвольный текст в угловых скобках передается дословно. Пара определяющих левых и правых угловых скобок удаляется;
* Предшествующий % вызывает непосредственную оценку в текущей системе счисления;
* Предшествующая стрелка вверх заставляет следующий символ быть переданным буквально без любой специальной интерпретации. Стрелка вверх удаляется (не передается).

# Компоновка с помощью ZML

ZML объединяет перемещаемые объектные модули из входных файлов в абсолютный машинный код в выходном файле. Выходной файл обычно подходит для загрузки для выполнения в Z-system или CP/M. Отсортированная таблица символов, содержащая глобальные символы, используемые в объектных модулях, может быть отправлена в выходной файл и выведена на экран на консоли. Файл таблицы символов подходит для использования символьными отладчиками аналогичными DSD и ZSID.

ZML работает в операционных системах CP/M или Z-system на компьютерах, которые используют центральный процессор Z80, Z180 или HD64180.

Отчет о состоянии выводится на экран после завершения компоновки. Выведенная информация включает неразрешенные или дублированные символы, карту распределения памяти при загрузке и размер памяти, неиспользованной процессом компоновки.

Если размер неиспользованной памяти становится равным нулю, программа стала слишком большой для обработки редактором связей. Ограничение размера программы обычно больше, чем 32 Кбайт. Это зависит от вашего общего размера TPA и числа глобальных символов, используемых в текущей работе редактора связей.

ZML выдаст сообщение об ошибке в таком случае. Если вы используете Z-system, некоторое дополнительное пространство TPA может быть сделано доступным, реконфигурировав операционную систему.

## Вызов

Вызов с '//' отобразит экран справки на вашей консоли.

ZML // <== (для экрана справки)

Конечно, экран справки показывает корректный синтаксис команды. Он также кратко описывает параметры и их значения по умолчанию. Вы можете изменить их (и другие) значения по умолчанию с помощью программы конфигурации ZCNFG.

Операция связывания может быть запущена, введя команду, которая следует следующему, совместимому с L80 и ZLINK, синтаксису:

ZML \_ [FS=]<file-list> \_ [/]o [ (\_|/)o...]

**FS** может быть полной спецификацией файла Z3: [DU: | DIR:] NAME [.TYP]

**<file-list>** является FS[/] [,FS[/]...]

**o** является одной из следующих букв параметров: C, D, L, M, R, S, Y

Примечания:

1. Это - подмножество более гибкого синтаксиса, описанного позже.
2. См. Глоссарий в *Приложении 1* для описания сокращений, используемых в определениях синтаксиса.

**<file-list>** является списком файлов, которые соединяются для создания выходного файла. Элементы списка разделяются запятыми. Элементы, форма которых - FS или FS/ или FS/S, должны быть в формате REL, произведенном ZMAC, и описанным в *Приложении 3* (Формат Microsoft REL). FS/K, FS/P и FS/I определяют входные файлы, которые НЕ в формате REL. ZML не устанавливает ограничение на количество заголовков входных файлов REL или RSX, используемых для завершения компоновки. Вы можете определить несколько заголовков PRL, но использоваться будет только последний названный.

**Поиск в библиотеке REL** – '**/**' или '**/S**', добавляется к элементам <file-list>, предписывая ZML, искать этот файл как библиотеку формата REL, извлекая из него только модули, содержащие точки входа, необходимые для удовлетворения неразрешенных символьных ссылок. Обратите внимание, что, если библиотека будет названа дважды, то поиск будет осуществлен дважды. Это обеспечивает два прохода через библиотеку. Иногда, упреждающие внешние объявления, неизбежны в библиотеке REL. За один проход такая библиотека приведет к неразрешенному символу(ам). Второй проход разрешит такие символы (если не появятся новые объявления!). Например:

ZML MYPROG,MYSUBS,MYLIB/S,MYLIB/S

Еще один способ направить редактор связей, искать библиотеку состоит во включении в исходный модуль оператора .REQUEST, параметрами которого являются имена библиотек в формате REL для поиска. (См. описание псевдо-операции .REQUEST в Руководстве по ассемблеру.) Нередки случаи, когда при соединении нескольких модулей REL общего назначения, встречаются элементы .REQUEST, определяющие одинаковые имена библиотек в нескольких модулях. Чтобы избежать нескольких непредвиденных проходов через каждую библиотеку, ZML игнорирует избыточные поисковые запросы к библиотеке, которые исходят из элементов .REQUEST.

Когда элементы .REQUEST появляются во время обработки входного файла, имена запрашиваемых библиотек REL добавляются в список. Один из параметров конфигурации позволит вам изменить эту стратегию так, что имена ВСТАВЛЯЮТСЯ в список для связывания сразу после завершения текущего модуля. Список входных файлов изменяется динамически! Вложение разрешено. Файл, вставленный в список входных файлов, может в свою очередь также содержать список .REQUEST, который автоматически изменит список файлов снова! (Вложение также позволено, когда файлы добавляются в список - немного легче обнаруживать!). Когда файлы с помощью .REQUEST вставлены, дублирующие поиски в библиотеке рассматриваются, как будто они были частью исходной командной строки. Каждый вызов приводит к проходу через библиотеку, даже при том, что возможно поиск в ней осуществлялся ранее. Этот параметр конфигурации должен использоваться с большой предусмотрительностью! Несколько проходов через библиотеки замедляют работу редактора связей пропорционально числу проходов. Обычно, перестановка модулей в библиотеке устраняет необходимость в нескольких проходах.

**Имена выходных файлов** - совпадают с именем первого введенного (REL) файла в командной строке, если другое не подразумевается в командной строке(ах).

Если элемент <file-list> находится в форме 'FS =' или 'FS/N', то он определяет имя и место назначения объектного выходного файла. Все части спецификации файла необязательные. Значения по умолчанию спецификации каталога - текущий зарегистрированный каталог. Значения по умолчанию имени файла - имя первого файла REL. Точно так же элемент, форма которого - 'FS/M' или 'FS/Y', определяет файл таблицы символов.

**Спецификации файлов**

Синтаксис спецификации файлов (FS) совпадает с синтаксисом ZMAC и ZCPR3x. Настраиваемые значения по умолчанию присутствуют для всех расширений файлов. Они перечислены здесь с первоначальными настройками в дистрибутивной конфигурации ZML.

**Расширение: Использование файла: (входные файлы)**

**REL** Объектные модули и библиотеки в формате Microsoft REL.

**HDR** Двоичный файл, с которым объединяется выходной объект.

**Расширение: Использование файла: (выходные файлы)**

**SYM** Текстовый ASCII файл символов и значений символов.

**COM** Образ двоичного кода, подходящий для загрузки в 100H

**BIN** Образ двоичного кода, адрес загрузки которого *не* 100H

**PRL** Заголовок PRL + образ кода в 100H

**SPR** Заголовок SPR + образ кода в 000H

**BPR** Заголовок типа PRL + образ кода не в 000H или 100H

**RSX** двоичный заголовок + любой из COM, BIN, PRL, SPR или BPR

Все расширения файлов по умолчанию настраиваются с помощью ZCNFG.

**Повторное выполнение**

После вызова, может использоваться ZPCR3 команда GO (с подходящим хвостом команды) для повторного выполнения программы редактора связей, если не использовались другие нерезидентные команды. Эта возможность особенно полезна, если должны быть выполнены несколько компоновок, используя, например, псевдоним (Alias). Экономится время, так как программа редактора связей повторно не загружается с диска при каждом вызове. Во время повторного выполнения диск, содержащий ZML, даже не должен присутствовать в системе.

## Основы перемещения, порядок загрузки

Каждый байт кода или области данных, описанные во входном файле REL, идентифицируется как принадлежащий к именованной основе перемещения, также называемой сегментом. Редактор связей имеет три основные задачи:

1. Он собирает код/данные, выделенный каждому сегменту в каждом REL файле, и объединяет их в одну соответствующую основу перемещения (группу адресов) в выходной файл.
2. Он сортирует в предписанном порядке группы кода, включенного в основы перемещения в окончательный образ кода.
3. Он корректирует все адресные ссылки в операторах кода и данных, для учета изменений в расположении кода, вызванных шагами 1 и 2. Этот процесс является перемещением.

Выходной файл содержит образ кода, предназначенный для загрузки в память с помощью загрузчика (обычно в CCP), чья работа не зависит от редактора связей. Первый байт кода может иметь смещение (определяется параметром C, и 0 по умолчанию) от адреса запуска файла. Макет файла следующий:

Используемые параметры: любые кроме /I, /P, /K (не PRL-типа)

|<...образ кода>..| ( '...' это смещение)

Образ кода включает код из нескольких основ перемещения, только один из которых может быть пустой (нулевой длины). Макет будет такой:

|...<CSEG>...<DSEG><COMMON1,COMMON2, и т.д.>..|

*Основной адрес загрузки* - адрес памяти, в котором первый байт образа кода в выходном файле сохраняется загрузчиком. Адресом загрузки определенного сегмента является адрес памяти, в котором находится первый байт этого сегмента, когда файл загружен по основному адресу загрузки. Адресом перемещения сегмента является адрес, в котором сегмент будет выполняться правильно, и также называется 'адресом выполнения'.

Два вида выходных файлов производятся (типы файлов по умолчанию)

1. .COM - когда адрес загрузки 100H (условие по умолчанию)
2. .BIN - когда адрес загрузки не 100H (параметры A или C)

Если операционная система следует стандартами CP/M, то стандартный адрес загрузки 100H. В таких системах загрузчик (часть CCP) загружает файлы, с расширением .COM по этому адресу. Обычно, сегмент кода (CSEG) является первым кодом в образе, и его адреса загрузки и перемещения совпадают с основным адресом загрузки. ZCPR33 и ZCPR34 содержат расширенные загрузчики, которые могут загрузить файл образа кода в другие адреса. Такие приложения обсуждаются в связи с параметрами /A, /R и /P.

Когда в командной строке указаны имена нескольких файлов REL для компоновки, код в CSEG из каждого файла REL объединяется (для формирования окончательного сегмент CSEG) в том порядке, в котором они появляются во входных файлах. Аналогичным образом объединяются DSEG. Общие сегменты с одинаковыми именами просто перекрываются[[3]](#footnote-3). Получающийся именованный общий сегмент имеет размер, самого большого из составляющих его наложений. Вы можете включать инициализируемые данные в именованные общие сегменты, но имейте в виду, что только ПОСЛЕДНИЕ соединенные инициализируемые данные будут действительны из-за этого процесса перезаписи во время компоновки.

ASEG, если присутствует, это первый блок данных в выходном файле и адрес первого байта в ASEG определяет основной адрес загрузки. Параметр /A применяется ТОЛЬКО к перемещаемым основам. Его использование присвоит адрес загрузки для CSEG (и других). Вы получите сообщение об ошибке 'overlapping segments' (перекрывающиеся сегменты), если адрес, определенный /A будет выбран не правильно!

Порядок, в котором именованные общие сегменты загружаются, является порядком, в котором они встречаются в файлах REL, на которые воздействует редактор связей. Именованные в первом файле REL первыми появятся в выходном файле. В каждом файле REL соблюдается порядок вхождения в файл.

**Начальный переход к входу в программы**

Когда основной адрес загрузки 100H, редактор связей поместит по адресу 100H команду перехода к точке входа в программу, если такая точка входа определена. Такая точка входа определяется в исходном файле ассемблера выражением после оператора 'END'. Код, который (без перехода) был бы загружен в 100H, перемещается для загрузки и выполнения в 103H, чтобы освободить место для вставленной команды перехода. Если несколько модулей в списке компоновки содержит выражение после оператора END, редактор связей не знает какое выражение использовать в качестве цели начального перехода. Такая ситуация заставит прервать работу редактора связей с сообщением о фатальной ошибке. Коварным источником подобной ошибки является объявление точки входа оператором END модуля в библиотеке, которая по иным причинам объявлена дважды в командной строке. В таком случае компоновщик видит ДВА объявления адреса входа, по одному на каждом из запрошенных поисков в библиотеке!

Типичное использование для начального перехода - включение заголовка в начале файла перед фактическим началом кода программы. Например, Z3HDR.REL мог бы содержать заголовок ZCPR3 плюс данные для параметров конфигурации программы, и MYPRG.REL содержит фактический код программы, завершенный оператором 'END START'. START - метка в MYPRG.Z80 в начале программы, и Z3HDR.Z80 не содержит начальный переход. Команда редактора связей:

ZML MYPRG=Z3HDR,MYPRG

произведет файл MYPRG.COM, который запускается с инструкции перехода, целью которого является код в 'START:'. Это хороший способ включить такой стандартный блок предопределенного кода/данных в начале программы.

## Неинициализированные данные

Неинициализированное пространство данных - диапазон адресов в образе кода, которое следует из операторов 'DS n' и ORG в источнике ассемблера для файлов REL. Также оно может появиться между сегментами, если пользователь определяет адреса загрузки для сегментов. Такое пространство заполняется некоторым значением, чтобы гарантировать, что весь код и инициализированные (DW, DB и т.д.) данные сохраняют свои корректные адреса во время загрузки операционной системы.  
В ZML значение заливки по умолчанию - двоичный 0. Неинициализированные данные хранятся в выходном файле, если за ним следует код или инициализированные данные. Любые неинициализированные данные в конце образа кода не хранятся в выходном файле. В результате происходит уменьшение размера файла, когда, например, большой буфер ввода-вывода определяется операторами DS, которые помещают его адрес в конце соединенного кода. Знание порядка загрузки сегментов позволяет предвидеть, какой сегмент будет ПОСЛЕДНИМ. То есть, КУДА поместить операторы DS для определения буфера! Это понятие было описано как усечение неинициализированных данных. Обратите внимание, что расположение $MEMRY ИНИЦИАЛИЗИРУЕТСЯ.

Удобная практика программирования выделять DSEG только для операторов DS. Именованный COMMON может служить аналогичной цели.

## Файлы PRL и SPR

ZML предоставляет средства для создания SPR, PRL, RSX и некоторые пока еще неназванных типов объектных файлов. Они создаются в результате использования двух параметров командной строки: /I и /P. Третий параметр /К, является синонимом /P и присутствует для совместимости с синтаксисом другого редактора связей.

Для нормальной операции связывания последовательность спецификаций файлов и параметров присутствуют в командной строке (может быть расширена). Мы будем именовать ее 'списком компоновки'. Выводом редактора связей является 'образ кода' в выходном файле. Редактор связей ВСЕГДА производит этот образ кода в соответствии с имеющимися инструкциями в списке компоновки. Первый байт образа кода является первым байтом в выходном файле в отсутствие параметров I, P или K.

Структура выходных файлов произведенных с использованием параметров /I или /P зависит от выбора параметров. Возможны три типа:

1. <Заголовок PRL><образ кода><битовый массив>

используемые параметры: /P или /K или FS/P или FS/K

1. <Заголовок RSX><Заголовок PRL><образ кода><битовый массив>

используемые параметры: FS/I и (FS/P или /P или FS/K или /K)

1. <Заголовок RSX><образ кода>

используемые параметры: FS/I или /I

Расположенные спереди заголовки не влияют на содержание образа кода. С точки зрения редакторов связей основной адрес загрузки такой же, как при отсутствии заголовка в выходном файле. Ожидается, что загрузчик переместит образ кода в основной адрес загрузки и после использования отбросит части заголовка файла.

**Файлы PRL типа**

Три вида выходных файлов могут быть созданы с помощью параметров PRL, согласно значений, присвоенных параметром /A<hex>. Эти три, отличающиеся расширениями файлов, являются:

1. .PRL - (обычно по умолчанию), основной адрес загрузки = 0100H
2. .SPR - Основной адрес загрузки = 0000 ( установлен с: /A0000 )
3. .BPR - Основной адрес загрузки **не является** ни 0000, ни 0100H

Заголовок PRL может появиться двумя способами:

1. /P или /K ( БЕЗ спецификации файла! )

Заголовок генерируется ZML

1. FS/P, FS/K (спецификация файла присутствует)

Заголовок КОПИРУЕТСЯ из файла FS

Во всех трех файлах PRL типа ZML устанавливает длину образа кода как обычное (в стиле Intel младший байт сначала) слово в <Заголовок PRL> относительные расположения 1 и 2. Когда заголовок создается ZML (параметры /P или /K), все другие расположения заполняются нулями. Когда заголовок копируется из файла, то в других расположениях отражает содержимое файла. Такие заголовки обычно являются специализированными загрузчиками, которые используют данные смещения 1 и 2 для загрузки и во время выполнения операций перемещения.

**Структура битового массива**

После включения заголовка любой формы, ZML сразу производит битовый массив непосредственно следующий за образом кода. Битовый массив содержит установленный бит для старшего байта каждого перемещаемого слова в образе кода. Все другие биты сброшены (0). Длина битового массива вычисляется как <длина кода>/8, округленная в большую сторону для включения любого остатка от целочисленного деления. Биты в каждом байте данных битового массива присваиваются так, чтобы старший значащий бит байта соответствовал САМОМУ МЛАДШЕМУ из восьми представленных адресов. Битовый массив предназначен для использования во время выполнения перемещения загрузчиком. Заголовок может содержать маленький загрузчик. DDT (Digital Research Inc.) является файлом PRL, который содержит такой загрузчик в первой странице. Программы ZCPR34 Типа 4 являются файлами PRL, загрузчик которых более сложен и использует функции загрузчика, предоставляемые CCP, а также кодом в заголовке.

Когда файл PRL-типа производится, связанный битовый массив сразу следует за последними инициализируемыми данными. Это то же расположение как усеченные неинициализированные данные. В связи с этим, вы были бы очень неблагоразумны, чтобы зависеть от любого конкретного значения в затронутых областях DS!

Стандартные файлы заголовков PRL и SPR - ровно 2 записи CP/М (256 байтов) длиной. Это, длина по умолчанию, произведенная ZML, настраивается с помощью ZCNFG. Если вы изменяете длину по умолчанию, имейте в виду, что создаваемый файл больше не будет стандартным файлом PRL или SPR из-за разной длины заголовка. Длина образа кода все еще будет в расположениях 1 и 2. Любому коду, который использует эту длину, нужно также предоставить длину заголовка. Когда пустой заголовок PRL произведен ZML, исполняемый код должен быть помещен в пространство заголовка отдельной операцией. Как правило, код ассемблируется отдельно и затем накладывается в заголовок, используя MLOAD или отладчик аналогичный DSD, ZSID, Z8E и т.д.

Когда используется форма FS/P, длиной заголовка управляет длина исходного файла. Программист обязан гарантировать, что она - ровно 2 записи! Если она будет более длинной (или короче), то ZML искренне воспроизведет его в выходном файле, включая установку байта длины. Исходный файл заголовка обычно содержит образ кода загрузчика, в котором байты 1 и 2, как ожидается, будут заменены редактором связей. Преимущество этого подхода в форме /P - устранение пост обработки файла PRL для установки код загрузчика. Если единственная ссылка на длину заголовка является внутренней к коду в заголовке, то длины отличные от стандартных могут использоваться безопасно.

**Файлы типа** RSX

Заголовок RSX КОПИРУЕТСЯ из файла FS.ZML не вносит в него никаких изменений. СОДЕРЖАНИЕ этого файла полностью зависит от программиста. Ограничение для ВСЕХ заголовочных файлов налагается блочной структурой файловой системы: заголовочный файл всегда составляет целое число 128-байтовых записей в длину.

Когда объявлен параметр /I (без FS), используется имя по умолчанию (RSX.HDR). То же имя является значением по умолчанию для отсутствующих частей формы FS/P. Это имя по умолчанию настраивается с помощью ZCNFG.

Файлы заголовков (такие как RSX.HDR) должны иметь длину меньше 32 Кбайт. Это ограничение возникает из соображений использования памяти.

## Реализация $MEMRY

Адрес следующего свободного байта памяти, доступного программе в образе кода, может быть включен в эту программу в двухбайтовом поле с меткой $MEMRY. Для этой цели, используется следующий свободный байт расположенный за ВСЕМ пространством выделенным программе. Выделение здесь включает в себя все пространство неинициализированных данных. $MEMRY должен быть объявлен PUBLIC в одном из модулей REL, чтобы быть распознанным редактором связей.

Оператор с меткой $MEMRY: может быть оператором DS или DW. В следующем примере исходного кода показано, каким образом $MEMRY реализуется, используется, и его значение становится доступным для других модулей.

PUBLIC FREBEG,$MEMRY

FREBEG: LD HL,($MEMRY)

RET

... ...

... ...

$MEMRY: DS 2

FREBEG - глобальная подпрограмма, которая возвращает адрес первого свободного байта в памяти в регистре HL. Оно получается из значения, сохраненного в $MEMRY редактором связей.

Примечание: данные $MEMRY являются ИНИЦИАЛИЗИРУЕМЫМИ данными, даже когда объявлены с помощью оператора DS, потому что они инициализируются редактором связей. Любые неинициализированные данные, которые предшествуют $MEMRY, НЕ будут усечены. Поэтому $MEMRY лучше всего располагать (в источнике на ассемблере) вместе с другими инициализируемыми данными или областями кода.

Когда генерируется файл типа PRL, два бита в битовом массиве выделяются для значения $MEMRY. Поскольку целью битового массива является обеспечение идентификации старшего байта каждого перемещаемого слова в изображении кода, соответствующий бит устанавливается для значения $MEMRY. Некоторые другие редакторы связей НЕ устанавливают этот бит (Значение $MEMRY не будет перемещено загрузчиком, который зависит от битового массива). ZML может быть настроен, чтобы НЕ устанавливать этот бит в битовом массиве. Использование значения $MEMRY во время выполнения кода будет отличаться в этих двух случаях.

## Синтаксис команды ZML

В *Таблице 8-3* приведено точное определение синтаксиса командной строки ZML. Основные аспекты синтаксиса обсуждены ниже и в *Таблицах 8-1* и *8-2*. Более детальное обсуждение параметров и их использование следуют за таблицами.

ZML позволяет использовать несколько командных строк. Все, кроме первой, являются интерактивными: они обозначены подсказкой, запрашивающей дополнительные параметры команды. Первой, конечно, является хвост команды вызова. Его максимальная длина определяется размером буфера команд в вашей операционной системе. Он обычно, по крайней мере, длиной 80 символов. Последующие командные строки (подсказка ',' или '/') могут иметь длину 128 символов, включая заключительный символ возврата каретки.

Существуют два вида параметров командной строки: те, которые определяют спецификацию файла (FS) и те, которые определяют только выбор параметров (<optn>). Вы можете сочетать два вида параметров при помощи соответствующего разделителя.

Параметрам FS предшествует запятая и за ними сразу может следовать модификатор (fo), который применяется к этой спецификации файла. Предшествующая запятая необязательна в начале хвоста команды и в начале интерактивной командной строки, подсказка которой ','. Распознаваемые модификаторы приводятся ниже в *Таблице 8-1*.

Параметрам выбора <optn>, предшествует пробел (и необязательный '/').   
Однобуквенные параметры управляют функциями редактора связей, выключая или включая их. Это параметры I, K, L, M, P, S и Y. Когда любому из них предшествует '-', функция выключена. Например, '/-M' запрещает генерацию листинга таблицы символов. Возможно, вы пожелаете использовать возможность замены условий настроенных по умолчанию. Все такие параметры упомянуты ниже в *Таблице 8-2*.

Если командная строка заканчивается пробелом или запятой, то ZML предполагает, что она не закончена и запрашивает дополнительный ввод. Замыкающая запятая повторяется в виде подсказки. Замыкающий пробел преобразуется в '/' и отображается как подсказка. Первый введенный символ не должен быть пробелом или запятой, если символ приглашения уже подходящий для следующего параметра. При подсказке возврат каретки завершает команду и запускает работу редактора связи. Ввод '//' вызывает справку и затем повторяет приглашение к вводу.

Последовательность из пробелов и запятых игнорируются за исключением последней в последовательности. Несколько последовательных пробелов (или запятых) рассматривают в качестве одного пробела (или запятой).

## Параметры спецификации файла (таблица)

Таблица 8‑1 Список модификаторов FS

|  |  |
| --- | --- |
| Следующие модификаторы могут быть добавлены к параметрам FS. | |
| **/** | FS является библиотекой REL. После '/' требуется разделитель для предотвращения иной интерпретации. |
| **=** | FS - выходной файл для образа кода. |
| **/S** | FS - библиотека REL, в которой будет осуществляться поиск |
| **/N** | FS - выходной файл для образа кода. |
| **/M** | FS - имя файла глобальных символов. |
| **/Y** | То же, что /M |
| **/P** | FS - имя заголовочного файла заголовка PRL/SPR для использования. |
| **/K** | То же, что /P |
| **/I** | FS - имя заголовка RSX-стиля для вставки. |
| Эти модификаторы взаимоисключающие. Если вы к FS добавите несколько, то только первый будет исполнен. Остальные будут проигнорированы.  Следующие два будут распознаны, даже если добавлены к одному из вышеупомянутых модификаторов FS. | |
| **/E** | Конец командной строки и начало компоновки |
| **/Q** | Аварийное прекращение работы. Возврат в операционную систему (ОС)  (ничего не сделано) |

## Параметры (таблица)

Таблица 8‑2 Список функций параметров

| **<optn>:** | **Выполняемая функция:** |
| --- | --- |
| **A[:]<hex> \*\*** | Устанавливает основной адрес загрузки (по умолчанию = 100H) |
| **C[:]<hex>** | Устанавливает адрес загрузки CSEG. (по умолчанию = конец ASEG) |
| **D[:]<hex>** | Устанавливает адрес загрузки DSEG. (по умолчанию = конец CSEG) |
| **R[:][seg]<hex>** | Установка перемещения SEG (по умолчанию = CSEG) |
| **T[:]<seg>** | Рассматривать SEG как CSEG или DSEG вместо COMMON |
| **E** | Командная строка завершена и запускается компоновка.  Остальная часть командной строки игнорируется. |
| **Q** | Выход. Аварийное прекращение работы. Выход в ОС. |
| **L \*\*** | Включает вывод листинга таблицы символов на консоль. |
| **M \*\*** | Включает вывод листинга таблицы символов в файл. |
| **Y,S** | То же, что M |
| **P,K \*\*** | Включает генерацию выходного файла PRL/SPR. |
| **I \*\*** | Включает генерацию выходного файла RSX-типа. |

**Примечания:**

<o> включает функцию, -<o> выключает ее

**\*\*** настраиваемые параметры, значение по умолчанию которых может быть установлено ZCNFG.

## Формальный синтаксис (таблица)

**Таблица 8-3 Спецификация синтаксиса ZML**

Символы используемые в определении синтаксиса:

|  |  |
| --- | --- |
| =: | означает, 'определен как' |
| **fs** =: | Спецификация файла ZCPR3 стиля (со значениями по умолчанию). |
| **fm** =: | Модификатор спецификации файла, добавляемый к FS. |
| **< >** =: | Угловые скобки разграничивают элементы синтаксиса. |
| **( )** =: | Окаймляют требуемые компоненты синтаксиса. |
| **[ ]** =: | Элементы, залеченные в квадратные скобки, необязательные. |
| **<hex>** =: | От одного до четырех шестнадцатеричных ASCII символов. |
| **|** =: | Дизъюнкция. Подразумевается только ОДИН из элементов списка. |
| ... =: | Этот элемент синтаксиса может быть повторен. |
| **CR** =: | Возврат каретки. |
| **SP** =: | Пробел или символ табуляции. |

Символы в ВЕРХНЕМ регистре в определении являются дословным вводом с клавиатуры. При вводе они могут быть в верхнем или нижнем регистре. Ввод командной строки не чувствителен к регистру. Запятые и наклонные черты вправо - также дословный ввод с клавиатуры. Пробелы в определении синтаксиса присутствует только для удобочитаемости и не является частью синтаксиса.

**Активизация** хвоста команды =: <arg0> [<arg>...] [ SP | , ] cr

**Запрошенная** командная строка =: [<arg>...] [ SP | , ] cr

**<arg0>** =: [,] [//] (CR | fs [fm]) | (SP | / | //) (CR | <optn>)

**<arg>** =: , [//] (CR | fs [fm]) | SP [/ | //] (CR | <optn>)

**<optn>** =: A<hex> | C<hex> | D<hex> | <r> | T<seg> | <lopts>

**<r>** =: R<hex> | R<seg> SP <hex> | R <seg> , <hex>

**<lopts>** =: E | I | K | L | M | P | Q | S | Y

**fm** =: / | = | /E | /I | /K | /Q | /M | /N | /P | /Y | /S

## Параметры, полное описание

Параметр начинается с одного из символов, показанных в *Таблице 8-2*. Каждому параметру в списке предшествует по крайней мере один разделитель. Допустимыми разделителями являются пробел и символ табуляции. Наклонная черта вправо (/) может следовать за разделителем, и '/' может использоваться вместо пробела или символов табуляции МЕЖДУ элементами параметров. Синтаксически, есть два использования для '/'. Первое - в качестве модификатора в конце спецификации файла. Такие модификаторы показаны в *Таблице 8-1*. Второе - в качестве признака отдельного параметра. Разделитель, предшествующий '/', служит, чтобы различить эти два случая и избежать неоднозначных интерпретаций. Если предыдущий элемент является параметром, такая неоднозначность не возникает, и '/' воспринимается в качестве разделителя параметра, а также признака параметра.

**L – Параметр вывода листинга на консоль**

Параметр L включает или отключает вывод полной таблицы символов на консоль. L вызывает вывод на консоль, '-L' запрещает такой вывод. Список сообщений об ошибках, ПОВТОРЯЮЩИЕСЯ или ОТСУТСВУЮЩИЕ символы отображаются ВСЕГДА.

**M, Y, S - Параметры файлов таблицы символов**

Параметр M вызывает генерацию выходного файла, который содержит глобальные имена символов из объектных файлов REL и их адреса после перемещения и соединения. Параметр '-M' запрещает генерацию файла таблицы символов. Y является синонимом для M. Параметр S в этом контексте, синоним для M. В форме 'FS/S' - S имеет совершенно другой смысл. По этой причине использование /S, для вызова генерации файла символов не поощряется. Параметр /S включен только для тех, кто еще не забыл такое использование в других редакторах связей.

**A[:]<hex> - Основной адрес загрузки для ВСЕХ сегментов**

Это - основной адрес загрузки (выполнения) для соединенного образа кода, созданного ZML. Его значение по умолчанию 100H, таким образом, параметр не нужно указывать при создании кода, предназначенного для выполнения по этому адресу. Значение по умолчанию может быть изменено с помощью ZCNFG.

Сегменты будут соединены и размещены в выходном файле в следующем порядке: CSEG, DSEG, BLANK COMMON, именованные COMMON. Адрес выполнения для CSEG будет иметь значение по умолчанию (100) или <hex>, как задано в командной строке.

Значение, определенное с A<hex> определяет какой из этих трех расширений файла (PRL, SPR или BPR) использовать для выходных файлов PRL-типа.

Этот параметр полезен для создания программы ZCPR33/34 Типа 3, если источник определил сегменты DSEG или COMMON. (См. также параметр /R ниже.)

**C[:]<hex> - Расположение сегмента кода**

Параметр C определяет адрес загрузки сегмента кода. Адрес загрузки, как предполагается, 100H при отсутствии параметра C. Адрес выполнения для сегмента кода также по умолчанию 100H. Синтаксис параметра C - Cnnnn, где nnnn - шестнадцатеричное число.

Предполагается, что абсолютный выходной файл создается для загрузки по адресу 100H. Поэтому, расширение файла по умолчанию используется COM. Так как, параметр C является причиной размещения сегмента кода по некоторому более высокому адресу, файл заполняется необходимым количеством нулей (двоичный ноль) до начала фактического кода.

**D[:]<hex> - Расположение сегмента данных**

Параметр D определяет адрес загрузки сегмента данных. Сегмент данных является последовательно соединенным кодом из основы перемещения DSEG в модулях REL. Если параметр D отсутствует, адрес загрузки сегмента данных следует непосредственно за последним байтом адресного пространства сегментов кода. Адрес выполнения корректируется, чтобы совпасть с адресом загрузки.

Сегмент COMMON загружается сразу после сегмента данных, и его адрес выполнения корректируется, чтобы совпасть с адресом его загрузки.

**R[:][<seg>]<hex> - Параметр перемещение сегмента кода**

Параметр R определяет перемещение сегмента <seg> так, чтобы его адрес выполнения (<hex>) был определен независимо от его адреса загрузки. Если в команде <seg> отсутствует, используется CSEG. Если адрес выполнения CSEG не 100H, расширение файла по умолчанию для абсолютного выходного файла кода изменяется на BIN. Такие файлы обычно не подходят для загрузки и выполнения по обычному системному адресу командного файла 100H. Если адрес выполнения сегмента не определен параметром R, он совпадает с адресом загрузки.

**Пример**:

Программа Типа 3 для ZCPR33/34 загружается по адресу, определенному на относительном смещении 0BH. Слово в этом расположении обычно генерируется ассемблером из символа, который является меткой для первого байта кода. Байт типа окружения на смещении 8, конечно 3, чтобы идентифицировать обработку Типа 3. Предположим, что имя исходной программы - IF.Z80 и она была ассемблирована для создания IF.REL. Следующая команда ZML тогда произвела бы выходной файл IF.COM, который загружается и выполняется по адресу 8000H:

ZML IF /R8000

или ZML IF /R:8000

Если другие сегменты (DSEG, COMMON), присутствуют, то должен использоваться параметр /A, чтобы гарантировать, что их адреса перемещения совпадают с их адресами загрузки.

**T[:]<seg> - Преобразование COMMON в обычную основу перемещения**

где <seg> - основа перемещения CSEG, DSEG или именованный COMMON.

Параметр T помечает перемещаемый сегмент в качестве объединяемого (как CSEG), а не накладываемого (как именованный сегмент COMMON). Вы можете иметь, столько сегментов кода (или данные), сколько ассемблер может поместить в поток REL. В источнике ассемблера основы объявляются как именованные COMMON. Во время компоновки эти именованные COMMON, которые реально должны быть сегментами кода, должны быть помечены, используя эту опцию. CSEG и DSEG уже помечены так, поэтому, для этих основ этот параметр избыточен и бесполезен. Попытка пометить ASEG приведет к сообщению об ошибке.

Этот параметр не нужен, если соединяемые модули REL уже содержат информацию о типе основы перемещения для именованных сегментов. Большинство ассемблеров, которые позволяют именованные основы, идентифицируют их в качестве именованных COMMON.

Можно пожелать генерировать код для нескольких банков в системах с переключением банков. При написании исходного кода используется сегмент именованный COMMON, например, BANK1, редактору связей можно сказать рассматривать этот сегмент в качестве кода (как CSEG). Параметр R используется для присвоения индивидуального адреса перемещения при необходимости.

**P,K - PRL, SPR и BPR выходной файл**

Параметр P (K является синонимом) включает генерацию выходного файла, в котором сначала следует стандартный заголовок типа PRL и за ним размещается битовый массив. Расширение имени выходного файла зависит от адреса загрузки, присвоенного скомпонованному образу кода. Если адрес загрузки равен 0000h, то расширение имени будет .SPR. Если 0100h оно будет .PRL. Любой другой адрес загрузки приведет к расширению имени файла .BPR. BPR - сокращение Binary Page Relocatable (двоичный странично-перемещаемый).

Когда используется параметр '/P', ZML генерирует заголовок, который имеет длину ровно 256 байт. Байты 1 и 2 из заголовка содержат длину образа кода. Байт 0 и байты 3...255 заполнены двоичными 00. Эти заполненные нулями байты резервируют место для предоставленного пользователем кода, который может быть наложен, используя MLOAD, отладчик или другими средствами по усмотрению программиста. CP/M DDT - это программа, которая была создана таким образом.

Когда форма 'FS/P' вызывается из командной строки, ее недостающие части заполняются по умолчанию из PRL.HDR, и файл с этим именем копируется в первые байты выходного файла. Во время операции копирования байты 1 и 2 заменяются длиной образа копируемого кода. Типичный заголовочный файл может содержать перемещающий загрузчик, который перемещает образ кода в его конечное место расположения, перемещает байты (или слова) помеченные битами в битовом массиве, и затем передает управление этому коду. Это - типичная архитектура отладчика типа DDT, ZSID, DSD и большинства других.

Обе формы (/p и FS/p) приводят к созданию стандартного битового массива в выходном файле. Расположение битового массива определяется длиной кода, сохраненной в байтах 1 и 2 заголовка.

Пример одного использования PRL или файлов SPR следующий.

**Пример**:

Программа ZCPR34 Типа 4 загружается для выполнения максимально высоко в памяти, чуть ниже CCP. Такими файлами являются с расширениями SPR или PRL. Программисты выбирают после перемещения кода какая часть кода будет исполняемой (определяется программистом) в заголовке. Имеется стандартный заголовок T4LDR.Z80, который может быть ассемблирован один раз в двоичный объект. Пусть он будет называться T4LDR.BIN. Код в T4LDR предполагает, что образ кода находится в форме PRL (исполняется в 100H). Для создания копии IF.COM, которая загружается максимально высоко в памяти:

ZML IF,T4LDR.BIN/P

Обратите внимание, что в отличие от примера заголовка Типа 3, приведенного при описании параметра R, нет необходимости устанавливать '4' по смещению 8 в программу. Так как T4LDR делает это во время процесса перемещения. Тот же ассемблированный источник может использоваться для любого типа соответствующим выбором команды редактора связей! Программы Типа 4 обсуждены в *THE COMPUTER JOURNAL, Выпуск 32, на страницах 10-17*.

Параметр I приводит к генерации выходного файла, который является объединением заголовочного файла и файла, созданного выбором других параметров. Расширение файла по умолчанию для такого выходного файла будет .RSX, отражая единую структуру, используемую для этого вида файлов. Две возможных формы выходного файла:

<Заголовок RSX><образ кода>

<Заголовок RSX><Заголовок PRL><образ кода><битовый массив>

Когда используется форма '/I' в командной строке, файл под названием RSX.HDR разыскивается в каталоге по умолчанию и альтернативных каталогах. Этот файл, если найден, обеспечивает точный образ <Заголовка RSX>. Байты из этого файла КОПИРУЮТСЯ без изменения или любых других связывающих действий, чтобы стать <Заголовком RSX>. Точно так же, когда в командной строке используется форма FS/I, файл, определенный FS, обеспечивает образ <Заголовка RSX>. Длина заголовочного файла не определена так как нулевая длина файла не имеет никакого смысла и максимальные размеры файлов определяются операционной системой. Недостающие части FS обеспечиваются именем файла по умолчанию RSX.HDR. Это назначение по умолчанию FN.FT может быть изменено с помощью ZCNFG.

## Изменение конфигурации ZML

Все параметры I, K, L, M, P, S и Y имеют значения по умолчанию (YES/NO). Используйте ZCNFG для настройки их максимально удобно для вас, и используйте параметры в хвосте команды в исключительных случаях. Если параметр настроен в состояние ON (YES), то он может быть выключен помещением его буквы в командной строке в виде отрицательного параметра. Например, если создание таблицы символов (параметр M) будет сконфигурирован ON, то параметр '/-M' в хвосте команды запретит создание файла таблицы символов.

Была выполнены заготовки для доступа к буферу сообщений Z-system и номеру системного файла. Хотя эти значения по умолчанию фактически присутствуют и настраиваются, они в настоящее время не используются ZML.

**Сообщения об ошибках**

ZML выводит сообщения об ошибках на консоль при возникновении проблем. Для удобства выводимые сообщения перечислены в *Приложении 5* в конце Руководства.

# Настройка параметров с помощью ZCNFG

ZCNFG является утилитой конфигурации общего назначения, которая может использоваться для редактирования параметров в любой программе, для которой был создан конфигурационный файл. Она, вместе с .CFG файлами, которые являются частью вашего дистрибутивного диска, обеспечивает меню-ориентированное средство просмотра и изменения многих параметров, которые управляют работой ZMAC, ZML и самим ZCNFG. Параметры конфигурации выводятся на экран в одном или более меню. Текущая настройка для каждого параметра выводится на экран как часть информации меню. Также может быть вызван экран СПРАВКИ для помощи в выборе параметров.

ZCNFG является универсальной, потому что она использует файл наложения, чтобы обеспечить расположение(я) меню, экран(ы) справки и данные, определяющие расположение и природу каждого параметра. Соответствующее наложение должно присутствовать для целевой программы. ZCNFG загружает его автоматически или в результате указания в командной строке. Несмотря на то, что ZCNFG является утилитой Z-system, она становится совместимой с CP/M, если отсутствует ZCPR3x.

## Вызов ZCNFG

Одна из следующих командных строк обеспечит экран справки, который объясняет, как вызвать ZCNFG.

ZCNFG

ZCNFG / <== ? может заменить /

ZCNFG // <== ? может заменить //

У ZCNFG есть два метода нахождения файла наложения: подразумеваемый и явно объявленный в командной строке. Явное объявление имеет приоритет над подразумеваемым и использует подразумеваемую спецификацию файла для обеспечения значения по умолчанию для неуказанных частей явной спецификации файла.

**Подразумеваемая спецификация файла**: имя файла наложения находится в целевом файле или имеет то же имя, что и целевой файл и находится в текущем каталоге или альтернативном каталоге DU.

ZCNFG FS

ZMAC, ZML и ZCNFG содержат строку из 5 символов в относительном расположении 0DH (сразу после заголовка Z3), которая является уникальной для каждой программы. Эта строка воспринимается в качестве части имени подразумеваемой спецификации файла. В качестве части каталога по умолчанию используется текущий DU и расширения файла по умолчанию - .CFG. Обратите внимание, что файлы наложения, предоставленные ZMAC имеют эту форму.

**Явная спецификация файла**: файл наложения - вторая спецификация файла в хвосте команды. например:

ZCNFG FS \_ FS

где: FS = [DU: | DIR:][<filename>][.<filetype>]

Первая спецификация файла (FS) определяет (целевую) программу, которая должна быть настроена. Вторая спецификация файла определяет связанный файл наложения. Как подразумевается, все части обеих спецификаций необязательные. Если все будет опущено, будет выведен экран справки. Сообщение об ошибке будет вознаграждением за исключение только первой FS.

**Меню и управление установкой**

**X** или **Esc** = Сохранить изменения и выйти

**/** или**?** = Объяснить пункты меню

**Q** или **^C** = Выход без сохранения изменений

**>** или **.** = Следующее меню

**<** или **,** = Предыдущее меню

## Параметры ZMAC

**Параметры интерфейса системы**

**Z3 Msg Buffer Offset** (по умолчанию: 42H) <== Смещение буфера сообщений Z3

**Z3 System File number** (по умолчанию: 4) <== Номер системного файла Z3

При возникновении первой ошибки ассемблирования ZMAC устанавливает флаг ошибки программы. Имя текущего входного файла записано в буфере системного файла, определенного номером. Тип ошибки, DU спецификация файла и номер строки в файле по вышеназванному смещению в буфере сообщений Z3.

**Alternate DU** (по умолчанию: ??) <== Альтернативный DU

Этот параметр предоставляет DU для поиска входной файл после неуспешного поиска в текущем каталоге. Символ ? в DU является формой подстановочного знака: Она относится к текущему диску или пользователю. Например,

?8 означает, 'смотреть в области пользователя 8 текущего диска'

b? означает 'смотреть на диске B в текущей области пользователя'

c9 означает, конечно, искать файл в каталоге C9:

Альтернативная спецификация DU служит той же цели что и ZRDOS функция PUBLIC Directory. Часто используемые библиотеки (.LIB для ZMAC, .REL для ZML) могут быть сохранены в одном каталоге, всегда доступном из любого зарегистрированного в системе текущего каталога. Они не должны присутствовать в указанном текущем DU.

**Дополнительные параметры**

Возможно, были добавлены новые параметры, так как это руководство было распечатано. Они, а также те уже описанные, объясняются в экранах СПРАВКИ, доступных при выполнении ZCNFG.

**Параметры расширений файлов по умолчанию**

**Source File type** (по умолчанию: Z80)

Это расширение файла будет принято для ассемблируемого исходного файла. Любое расширение, присутствующее в командной строке, будет конечно заменить это значение по умолчанию. Измените его, если вы обычно используете другое расширение (как .ASM или .MAC) для ваших исходных файлов.

**INCLUDE/MACLIB file type** (по умолчанию: LIB)

Это - расширение по умолчанию для имен, заданных в операторах MACLIB или .IN. Если имя в операторе .IN будет включать расширение файла, то оно будет иметь приоритет.

**Listing File type** (по умолчанию: PRN)

Это - расширение файла по умолчанию для файла листинга. Явное расширение в спецификации файла параметра P в командной строке заменяет его.

**Symbol table File type** (по умолчанию: SYM)

Это - расширение файла по умолчанию для файла таблицы символов. Как и для всех других расширений файлов по умолчанию, явное указание в командной строке имеет приоритет.

**REL OBJect File type** (по умолчанию: REL)

У объектного файла, произведенного в формате Microsoft REL, будет это расширение файла по умолчанию.

**HEX OBJect File type** (по умолчанию: HEX)

Это - расширение по умолчанию для выходного файла формата Intel HEX.

**Error Log File type** (по умолчанию: ERR)

Расширение по умолчанию для файла регистрации ошибок.

**Параметры управления вводом-выводом**

**Listing to Console?** (по умолчанию: NO)

Значение по умолчанию для переключателя параметра 'C' командной строки, определяющего вывод листинга на консоль.

**Listing to Disk File?** (по умолчанию: NO)

Значение по умолчанию для переключателя параметра 'P' командной строки, определяющего вывод листинга в файл на диске.

**Listing to Printer?** (по умолчанию: NO)

Значение по умолчанию для переключателя параметра 'L' командной строки, определяющего вывод листинга на принтер.

**Generate OBJect File?** (по умолчанию: YES)

Значение по умолчанию для переключателя параметра 'O' командной строки, определяющего создание объектного файла.

**Generate Symbol File?** (по умолчанию: NO)

Значение по умолчанию для переключателя параметра 'S' командной строки определяющего создания файла с таблицей символов.

**OBJect type (HEX/REL)** (по умолчанию: REL)

Значение по умолчанию для переключателя параметра 'H' командной строки.

Переключатель между HEX и REL, и определяет вид произведенного объектного файла: формат Microsoft REL или формат Intel HEX.

**Make Error Log File?** (по умолчанию: YES)

YES активирует создание файла регистрации ошибок при возникновении ошибок. Это - значение по умолчанию параметра 'E' командной строки.

**STOP if errors exceed xxx** (по умолчанию: xxx = 100)

Этот параметр определяет число нефатальных ошибок, которые могут произойти прежде, чем будет прервано текущее ассемблирование. Значение 0 приводит к игнорированию ограничения. Аварийное прекращение работы не происходит.

**Размеры буферов по умолчанию**

ZMAC использует настраиваемые пользователем буферы файлового ввода-вывода, которые выделяются во время выполнения. Выделение происходит автоматически. Буфера не требуемые для настроек по умолчанию и параметров командной строки не выделяются. Вместо этого выделяется место для резидентных символов и таблиц макросов. Когда такие буферы выделяются, размер им присваивается из настроек, выполненных во время конфигурации.

Размеры буферов определяются в терминах 128-байтовых записей. Таким образом, по умолчанию размер буфера исходного файла равен 16\*128=2 Кбайт. Вы можете изменить это число в пределах между 1 и 32. (нулевая длина не допускается). Буфера меньшего размера оставляют больше места для ассемблирования больших и более сложных исходных файлов. Более крупные буфера увеличивают скорость ассемблирования посредством уменьшения операций дискового ввода-вывода.

**Source File Buffer** (по умолчанию 16)

**Object File Buffer** (по умолчанию 8)

**Listing File Buffer** (по умолчанию 16)

**Error Log File Buffer** (по умолчанию 4)

**Управление форматом листинга**

**Form Feed at Start of listing?** (по умолчанию NO)

Это настройка предполагает, что бумага в принтере установлена по верхнему обрезу страницы, когда запускается печать листинга ассемблирования. Если вы хотите выполнять прогон страницы, прежде чем будет распечатана первая строка, измените ее на YES.

**Form Feed at End of listing?** (по умолчанию YES)

Это настройка вызывает прогон страницы после того, как последняя строка будет распечатана и позволяет избежать оставления принтера посреди страницы. Переключитесь к NO, чтобы предотвратить этот прогон страницы.

**Keep Page Parity in Listing?** (по умолчанию YES)

После завершения печати ZMAC отмечает было ли распечатано нечетное или четное число страниц. Если число нечетное, то выдается дополнительный символ прогона страницы. С перфорированной бумагой у всех листингов тогда будет та же ориентация относительно сгиба. Хотя это стоит в среднем 1/2 листа за листинг, удобство трудно превзойти! Переключитесь к NO, если вам не требуется эта возможность.

**Get Page size from Z3 Environment?** (по умолчанию NO)

Изменение этого параметра в YES заставит ZMAC получать длину и ширину страницы для консоли и принтера от в настоящее время выбранных в среде ZCPR3. Значения длины и ширины по умолчанию, приведенные ниже, будут в этом случае проигнорированы.

**Default Page Length** (по умолчанию 66)

Общая длина страницы в строках. ZMAC допускает верхние и нижние поля. Эта длина страницы используется для принтера и для файла PRN.

**Default Page width for List Device** (по умолчанию 80)

Общая ширина с точки зрения символов печати при выводе на принтер. ZMAC допускает поля.

**Default Page width for PRN file Listing** (по умолчанию 132)

Это число определяет длину строк, отправленных в файл PRN. Строки длиннее этого значения будут усечены справа. Эта длина строки и длина строки для устройства печати (принтера) работают независимо. Длинная строка будет усеченной до 80 символов при отправке на принтер, и до 132 символов в файл PRN.

**Include Line Numbers in Listing?** (по умолчанию YES)

Переключите параметр в NO, если вы хотите удалить номера строк из листинга.

**Statement Sequence (for line numbering)?** ( по умолчанию NO)

Поддерживается отдельная нумерация для каждого исходного файла и номера строк не увеличиваются для расширения макросов. Таким образом номера строк соответствуют строкам в каждом исходном файле. Это упрощает поиск строк в источнике, которым может потребоваться изменение.

Изменение этого параметра в YES заставит номера строк быть присвоенными последовательно независимо от включенных исходных файлов с помощью операторов INCLUDE или макрорасширений.

**Line Number First?** (по умолчанию YES)

Этот параметр заставляет номера строк отображаться в левой части листинга. Изменение этого параметра в NO заставит номера строки быть распечатанными между полем кода и полем исходной строки.

**List False Conditionals?** (по умолчанию NO)

Исходные строки в ложном условном блоке не будут включены в листинг. Они не производят кода. Измените этот параметр в YES, если вы хотите, чтобы эти строки присутствовали в листинге.

**List .TITLE, PAGE, etc. statements?** (по умолчанию NO)

Включение строк, содержащих псевдо-операции управления листингом в выходной файл или листинг запрещено. Это совпадает с вызовом псевдо-операция .XCTL. (См. .XCTL и .LCTL в главе Псевдо-операции о перечислении), ЭФФЕКТ скрытия псевдо-операции по-прежнему присутствует. Установка этого параметра в YES заставляет все строки, содержащие эти псевдо-операция быть включенными в листинг и эквивалентно действию инструкции .LCTL.

**Старший байт первый в:**

**Addresses in Operand** (по умолчанию YES)

**Words in DW stmnts** (по умолчанию YES)

Эти два выбора управляют порядком, в котором старший и младший байты 16 разрядных величин (слова) отображаются в листинге ассемблирования. Людям обычно нравится видеть старший байт сначала. Фактический порядок хранения (т.е. как показано в дампе памяти) является младшим байт сначала. Установите их по своему желанию.

**Macro Expansions: XALL, SALL, LALL** (по умолчанию LALL)

Это три способа переключения, которые вы можете использовать для выбора, какой из параметров перечисления макроса вы предпочитаете использовать в качестве значения по умолчанию. Обратитесь к описаниям .XALL, .SALL и .LALL в главе [Псевдо-операции](#_Псевдо-операции_(директивы_ассемблера)_7), чтобы понять, хотите ли вы изменить настройки по умолчанию.

**Определение значение 'TRUE'**

**Use 16 BITS to test Trueness?**

**Use LOW BIT to test Trueness?** (по умолчанию)

Выражение, используемое в условии IF (условное ассемблирование) уменьшается во время ассемблирования до 16 разрядного двоичного числа, которое проверяется одним из двух методов: 1) равно нулю или не равно нулю, и 2) является четным или нечетным? Z/NZ это 16 БИТНЫЙ тест. Четный/Нечетный эквивалентно тесту младшего значащего бита. Условии IF ассемблируется, если результаты проверки имеют значение TRUE, где TRUE определено как NZ, когда используется 16 БИТНЫЙ тест, и как ODD (lsb = 1), когда выбран тест МЛАДШЕГО бита. Важно знать, какое соглашение используется в ваших условиях IF, чтобы иметь предсказуемое поведение. Ассемблеры Digital Research для тестирования значений используют метод МЛАДШЕГО бита. Ассемблер M80 Microsoft использует 16 БИТНЫЙ тест (Z/NZ).

Этот параметр определяет, какой метод использует ZMAC по умолчанию. Две псевдо-операции .IF1 и .IF16 (в [*Главе 5*](#_Псевдо-операции_(директивы_ассемблера)_6)) разрешают явное управление, если метод используется в источнике ассемблера.

## Параметры ZML

**Параметры расширения файла по умолчанию**

**Source File type** (по умолчанию REL)

Расширение файла по умолчанию для входных файлов в формате Microsoft REL.

**Executable File type** (по умолчанию COM)

Расширение файла по умолчанию для выходного двоичного файла, если его адрес выполнения 100H. Для файлов не выполняемых по адресу 100H, расширение будет BIN. Любое из них может быть изменено любым расширением файла, указанием выходного файла в командной строке.

**Symbol Table File type** (по умолчанию SYM)

Расширение файла по умолчанию для таблицы глобальных символов.

**Параметры управления вводом-выводом**

**List Symbol Table on Console?** (по умолчанию NO)

Если вы хотите по умолчанию отображать таблицу глобальных символов на консоли, то установите параметр в YES. Вы тогда получите выгоду, включая также параметр 'Page Console Display' (ниже)

**Make Symbol Table File?** (по умолчанию NO)

Не создавать файл таблицы глобальных символов, если отсутствует параметр командной строки 'S'. Установка этого параметра в YES заставит ZML создавать таблицу глобальных символов по умолчанию.

**Page Console Display?** (по умолчанию NO)

Параметр 'P' в командной строке вызовет разбивку на страницы отображения на консоль. Она полезно только, когда на консоль отправляется таблица символов или есть большое количество неразрешенных или отсутствующих символов.

**Параметры компоновки**

**Неинициализированные данные**

Параметр 'U' определяет значение заполнения, которое будет присутствовать в выводе неинициализированной области абсолютного файла (COM или BIN). Обратите внимание, что это значение заливки не принимает во внимание значения вытекающие из инструкций ассемблера выделения памяти (DS), которые определяют байты заливки. Этот параметр полезен для обеспечения специальных значений для кода, который должен быть помещен в PROM или ROM.

**Параметры системного интерфейса**

Хотя следующие два параметра конфигурации, существуют, в настоящее время они не используются. Однако, когда происходят ошибки компоновки, флаг ошибки в программе ZCPR3 установлен.

**Z3 Msg Buffer Offset** (по умолчанию 42H)

**Z3 System File number** (по умолчанию 4)

**Alternate DU** (по умолчанию ??)

Этот параметр предоставляет DU для поиска входной файл, если поиск в каталоге по умолчанию оказался неудачным. См. Описание параметра альтернативного DU для ZMAC.

**Дополнительные параметры**

Возможно, были добавлены новые параметры, так как это руководство было распечатано. Они, а также те уже описанные, объясняются в экранах СПРАВКИ, доступных при выполнении ZCNFG.

Приложение 1 - Глоссарий

**Обозначение: Интерпретация:**

{ } Элементы в фигурных скобках необязательные

... Означает, что предыдущий элемент может быть повторен неоднократно.

( ) Скобки группируют элементы как в алгебраической нотации

| вертикальная черта разделяет альтернативные элементы (исключающее ИЛИ).

TXT Последовательность печатных символов. Без пробелов и символов табуляции.

QTXT TXT заключенный в одинарные или двойные кавычки.

STR последовательность печатных символов, включая пробелы и символы табуляции.

QSTR STR заключенная с одинарные или двойные кавычки. Символ кавычки может быть включен в STR, если этому сразу предшествует тот же символ.

< > Описание параметра. т.е. <любой текст>. Угловые скобки не требуются как часть текста.

\_ подчеркивание обозначает пробел. Обычно используется в описаниях синтаксиса, чтобы различить пробелы, требуемые синтаксисом и которые присутствуют только для повышения удобочитаемости.

FN Имя файла. До 8 символов.

TYP Поле расширения в полном имени файла.

FS Полная спецификация файла, совместимая с ZCPR3 в форме

[DIR: | DU:] FN [.TYP]. В спецификации файла пробелы не допустимы. Примеры: WORK:FOO.BAR и A3:GLITCH.Z80.

**Определения**

**Модуль** - Один блок ассемблерного источника, промежуточного или машинного кода, используемый или созданный ассемблерами, редакторами связи, менеджерами библиотек REL и загрузчиками. Промежуточный код - HEX или REL, созданный ассемблером из исходных операторов для одной программы. Два или более модулей REL могут быть соединены в один файл для создания библиотеки REL. Редактор связей обрабатывает один или несколько модулей REL для создания модуля с абсолютным кодом. Загрузчик аналогичный MLOAD обрабатывает модули HEX для создания модуля с абсолютным (машины) кодом. Обычно, ассемблер создает файл, содержащий всего один модуль.

**Объект** - Общее обозначение для модулей с кодом HEX или REL. Также, иногда используется для описания машинного кода, созданного редактором связей.

**Файл PRN** - Текстовый листинг, созданный ассемблером, содержащий представление произведенного объектного кода, копию исходных строк и необязательную таблицу символов с их значениями.

**Исходный файл** - Текстовый файл читаемый ассемблером для создания выходных файлов. Также, входные для ZML файлы REL.

**Переключатель** - переключает между несколькими взаимоисключающим альтернативами.

**Пустое пространство** - Пробелы или символы табуляции в тексте.

Приложение 2 – Список литературы

**Мнемоники ассемблера и срецифичные данные процессора**

Название: **Z80-CPU Technical Manual**

Издатель: Zilog, Inc (1976)

10460 Bubb Road

Cupertino, California 95014

phone: (408) 446-4666

Название: **Hitachi HD64180 8-bit High Integration CMOS**

Microprocessor Data Book

Издатель: Hitachi America, Ltd.

Semiconductor and IC Sales and Service Division

2210 O'Toole Avenue

San Jose, CA 95131

phone: (408) 942-1500

**Основы ассемблера и компоновщика**

Название: **Relocating Assemblers and Linkage Editors: Part 1**

Автор: Andrew Bender

Издатель: Microsystems, V4 N9, Sep 1983, p.86

Название: **Relocating Assemblers and Linkage Editors: Part 2**

Автор: Andrew Bender

Издатель: Microsystems, V4 N10, Oct 1983, p.114

Название: **Relocating Assemblers and Linkage Editors: Part 3**

Автор: Andrew Bender

Издатель: Microsystems, V5 N1, Jan 1984, p.120

Название: **Structured Programming with M80**

Автор: Dennis N. Quinn

Издатель: Micro/Systems Journal, V1 N3, Jul/Aug 1985, p.26

Название: **Assemblers, Compilers, and Program Translation**

Автор: Peter Calingaert

Издатель: Computer Science Press (1979)

11 Taft Court

Rockville, Maryland 20850

ISBN: 0-914894-23-4

**Философия программирования**

Название: **The Mythical Man-Month.**

Автор: Frederick P. Brookes, Jr.

Издатель: Addison-Wesley Publishing Company. (1975)

ISBN: 0-201-00650-2

Название: **Writing Interactive Compilers and Interpreters.**

Автор: P.J. Brown.

Издатель: John Wiley & Sons. (1979)

ISBN: 0-471-27609-X

**Интерфейс CP/M**

Название: **The Programmer's CP/M Handbook.**

Автор: Andy Johnson-Laird.

Издатель: Osbourne/McGraw-Hill. (1983)

ISBN: 0-88134-119-3

Название: **System Programming under CP/M-80.**

Автор: Lawrence E. Hughes.

Издатель: Reston Publishing Co. (1983)

ISBN: 0-8359-7457-X

Название: **CP/M 2.2 Interface Guide.**

(also see CP/M 2.2 Alteration Guide).

Издатель: Digital Research. (1979)

**Интерфейс ZCPR/ZRDOS/ZSDOS**

Название: **ZCPR3, The Manual.**

Автор: Richard Conn.

Издатель: Zoetrope (1985)

ISBN: 0-918432-59-6

Название: **ZCPR2 Concepts Manual and ZCPR2 Rational Manual.**

Автор: Richard Conn. (1983)

Источник: via modem from ZNODE #2, (213) 670-9465

Название: **CP/M 2.2 Interface Guide.**

(also see CP/M 2.2 Alteration Guide).

Издатель: Digital Research. (1979)

Название: **ZRDOS Programmer's Guide.**

Автор: Dennis L. Wright. (1985)

Название: **ZSDOS User's Guide**

Автор: H.F. Bower, C.W. Cotrill, C. Wilson

Издатель: Plu\*Perfect Systems (1988)

410 23rd Street

Santa Monica, CA 90402, (213) 393-6105 (evenings)

**Программирование процессора Z80**

Название: **Programming the Z80.**

Автор: Rodney Zaks.

Издатель: Sybex. (1979)

ISBN: 0-89588-013-X

Название: **8080/8085 Assembly Language Programming.**

Издатель: Intel. (1979) - Manual Order Number 9800940.

Название: **CP/M MAC Macro Assembler:**

Language Manual/ Applications Guide.

Издатель: Digital Research. (1980)

**Программирование и дизайн операционной системы**

**Public Domain Source code listings**:

ZCPR3, ZCPR33

ZCPRnn Utilities

Общественного достояния Введение в программирование на Ассемблере. Превосходное учебное руководство в 10 главах.

MEYERTUT.LBR

Название: **Operating System Design - The XINU approach.**

Автор: Douglas Comer.

Издатель: Prentice-Hall. (1984)

ISBN: 0-13-637539-1

Название: **Algorithms.**

Автор: Robert Sedgewick.

Издатель: Addison-Wesley. (1984)

ISBN: 0-201-06672-6

Приложение 3 - Формат объекта REL

**Формат перемещаемых объектных файлов** ZMAC

В этом разделе описывается формат перемещаемых объектных файлов, создаваемых ZMAC и используемых в качестве ввода в ZML и другие редакторы связей, которые обрабатывают этот формат (широко известный как формат Microsoft REL). Эта информация включена здесь для справки. Она не требуется в качестве обычного раздела по использованию ZMAC или ZML.

Формат Microsoft состоит из битового потока. Поля в потоке битов не выравниваются на границах байта, за исключением в конце модуля. Каждое поле в потоке битов, называется элементом. Первые биты в элементе определяют тип и длину данных в составе поля.

Существует два типа элементов: абсолютный и перемещаемый, обозначенные первым битом поля. Если первый бит 0, следующие восемь битов загружаются как абсолютный байт. Если первый бит равняется 1, то следующие два бита определяют ТИП перемещаемого поля. Если следующие два бита 0, это - специальный элемент Link. В противном случае указывается перемещаемый элемент. Каждый из возможных элементов Link проиллюстрирован в таблице представленной ниже.

Поток битов REL, как показано здесь, обрабатывается слева направо. Данные байта представлены в порядке с первым старшим битом. 16 разрядные данные (слова) представлены в стандартном порядке Intel с младшим байтом первым. Данные ASCII являются часть потока битов для некоторых элементов Link. Каждый символ ASCII 8 битов длиной. Части потока битов, которые не определены как байт, слово или ASCII в таблице, называются битами управления. В таблице поля в элементе изображены точками, и концы элементов обозначены '\*'. Ни один из них не является частью формата. Они присутствуют только для повышения удобочитаемости.

Два из элементов, отмеченных 'Зарезервирован..', не реализованы в ZML. Элемент 100.0100 'зарезервирован для будущего использования' Microsoft. ZML выдаст сообщение об ошибке, если любой из этих 2 элементов появится в потоке битов REL.

**АБСОЛЮТНЫЕ ЭЛЕМЕНТЫ** (первый бит = 0)

'byte' содержит 8 битов абсолютных данных

0.byte\* <== 8 битов абсолютных данных

**ПЕРЕМЕЩАЕМЫЕ ЭЛЕМЕНТЫ** (первый бит = 1)

'word' содержит 16 битов перемещаемых данных

100...... <== (Специальный элемент Link, См. ниже)

101.word\* <== 16 битов кода относительных данных

110.word\* <== 16 битов данных относительных данных

111.word\* <== 16 битов COMMON относительных данных

**СПЕЦИАЛЬНЫЕ ЭЛЕМЕНТЫ LINK**

'zzz' - 3 разрядное значение, число байтов в поле .ascii.

'ascii' - имя состоящее из zzz символов

100.0000.zzz.ascii\* <== Имя символа PUBLIC

100.0001.zzz.ascii\* <== Имя COMMON

100.0010.zzz.ascii\* <== Имя модуля

100.0011.zzz.ascii\* <== Имя библиотеки REL

100.0100.......... <== (Зарезервирован для будущего использования)

**СПЕЦИАЛЬНЫЕ ЭЛЕМЕНТЫ LINK** (продолжение)

yy определяет применимую основу перемещения:

00 = Абсолютная (константа перемещения = 0000)

01 = Относительный код

10 = Относительные данные

11 = Относительные данные COMMON (текуший именнованный COMMON)

'size' и 'offset' являются абсолютными 16 разрядными числами

100.0101.yy.size.zzz.ascii\* <== Размер именнованного COMMON блока

100.0110.yy.word.zzz.ascii\* <== Цепочка внешних/Именнованный символ

100.0111.yy.word.zzz.ascii\* <== Значение и имя символа PUBLIC

100.1000.yy.offset\* <== Внешний - смещение (Зарезервирован)

100.1001.yy.offset\* <== Внешний + смещение

100.1010.yy.size\* <== Размер сегмента данных

100.1011.yy.word\* <== Устанавливает счетчик адреса

100.1100.yy.word\* <== Значение адреса цепочки

100.1101.yy.size\* <== Размер сегмента кода

100.1110.yy.word\* <== Конец модуля, точки входа

100.1111\* <== Отмечает конец файла

**Примечания для специальных элементов Link**

100.0000 Символ PUBLIC: используется для поиска в библиотеке.

100.0001 Выбирает COMMON блока: <ascii> является именем блока.

100.0010 Определяет имя модуля: <ascii> является именем модуля.

100.0011 Поиск в библиотеке. Редактор связей добавляет <ascii>.REL в список исходных файлов, для выполнения компоновки, используя только модули, на которые имеются ссылки.

100.0101 Определяет длину COMMON блока с именем <ascii> равным абсолютному значению <size>.

100.0110 Внешняя цепочка: <word> - заголовок связанного списка, каждый узел которого существует в поле адреса внешней ссылки. <ascii> - имя внешнего символа. Последний узел (с самым младшим адресом) содержит значение абсолютного нуля, чтобы отметить конец цепочки.

100.0111 Определяет точку входа: <word> является адресом, <ascii> - именем.

100.1000 Внешний минус смещение: не поддерживается ZML

100.1001 Внешний плюс смещение: текущий счетчик адреса является внешней ссылкой, к которой должно быть добавлено смещение <offset>. Имя символа, на который ссылаются, содержится в <ascii>. Так как ссылка является частью связанного списка (сцепленных внешних), текущий счетчик адреса и значение смещения <offset>) сохраняются, и обработка откладывается, пока не заполнены все цепочки в конце компоновки.

100.1010 Определяет размер сегмента данных как абсолютное значение <size>.

100.1011 Устанавливает счетчику адреса значение <word>.

100.1100 Цепочка внутренних: <word> - заголовок связанного списка. Все узлы в цепочке находятся в текущем модуле. Каждый заменяется текущим значением счетчика адреса. Используется для разрешения ссылок вперед в однопроходных ассемблерах и компиляторах. Как и цепочка внешних, последний узел содержит абсолютный ноль.

100.1101 Определяет размер сегмента кода как абсолютное значение <size>.

100.1110 Конец модуля (Обратите внимание, что он инициирует ввод до границы байта). Поле <word> содержит адрес входа модуля, который является или относительным расположением 0000 или адресом, заданным оператором END.

100.1111 Конец файла (инициирует ввод до границы байта и заполнение нулями).

**Перемещение 16-битных значений**

Значения 'word' перемещаются, добавлением константы перемещения, уникальной для основы перемещения, на которую ссылается 'yy', или подразумевают биты управления перемещаемых элементов. Константы перемещения (RC) для четырех основ следующие:

00 = Абсолютный - RC = 0000

01 = Относительный код - RC = Основа перемещения + накопленный размер ранее загруженных сегментов кода.

10 = Относительные данные - RC = Основа перемещения + накопленный размер ранее загруженных сегментов данных.

11 = Относительные COMMON (текущее имя COMMON) - RC = Основа перемещения. Не зависит от ранее загруженных COMMON с тем же именем.

Приложение 4 - Коды операций

**Коды операций Z80, Z180/HD64180**

Коды операций Z80/Z180/HD64180, распознаваемые ZMAC, совпадают с опубликованными Zilog и Hitachi. Они перечислены в этом разделе с целю быстрых ссылок на мнемоники. Для получения дополнительной информации о кодах операций, вы можете обратиться к руководствам Zilog или Hitachi, ссылки на которые приведены в *Приложении 2*. Файл Z80.HLP в сочетании с ZCPR3 HELP.COM обеспечивает, экранно-ориентированный набор данных для каждого кода операции.

Многие коды операций требуют операндов. Для определенных кодов операций операндами могут быть регистры, регистровые пары или числа. Используемые символы перечислены в следующей таблице, которая показывает определенные группы регистров или значения представленные каждым символом. Термины соответствуют используемым Zilog и Hitachi.

dd,ss rr pp qq r,r' p

BC BC BC BC A 00H

DE DE DE DE B 08H

HL IY IX HL C 10H

SP SP SP AF D 18H

E 20H

H 28H

L 30H

38H

Следующие символы используются в поле операнда, чтобы представлять значение данных.

n,d = 8 разрядные данные nn = 16 разрядные данные

**Группа вызовов и возврата**

CALL nn RET RETI

CALL cc,nn RET cc RETN

RST p

**Группа переходов**

JP nn JR e DJNZ e

JP cc,nn JR C,e

JP (HL) JR NC,e

JP (IX) JR Z,e

JP (IY) JR NZ,e

е = 7 битное относительное смещение

**Группа 8-битной загрузки**

LD r,r' LD (IY+d),r LD (BC),A

LD r,n LD (HL),n LD (DE),A

LD r,(HL) LD (IX+d),n LD (nn),A

LD r,(IX+d) LD (IY+d),n LD A,I

LD r,(IY+d) LD A,(BC) LD A,R

LD (HL),r LD A,(DE) LD I,A

LD (IX+d),r LD A,(nn) LD R,A

**Группа 16-битной загрузки**

LD dd,nn LD (nn),HL PUSH qq

LD IX,nn LD (nn),dd PUSH IX

LD IY,nn LD (nn),IX PUSH IY

LD HL,(nn) LD (nn),IY POP qq

LD dd,(nn) LD SP,HL POP IX

LD IX,(nn) LD SP,IX POP IY

LD IY,(nn) LD SP,IY

**Группа 16-битной арифметики**

ADD HL,ss SBC HL,ss INC ss

ADC HL,ss DEC ss INC IX

ADD IX,pp DEC IX INC IY

ADD IY,rr DEC IY \* MLT ss

\* эти инструкции уникальны для Z180/HD64180

**Группа арифметики общего назначения и управления процессором**

DAA CCF HALT IM0 \* SLP

CPL SCF DI IM1

NEG NOP EI IM2

\* эти инструкции уникальны для Z180/HD64180

**Група 8-битных арифметических и логических операций**

(ведущий 'A,' является необязательным)

ADD A,r SBC A,n XOR (HL)

ADD A,n SBC A,(HL) XOR (IX+d)

ADD A,(HL) SBC A,(IX+d) XOR (IY+d)

ADD A,(IX+d) SBC A,(IY+d) CP r

ADD A,(IY+d) AND r CP n

ADC A,r AND n CP (HL)

ADC A,n AND (HL) CP (IX+d)

ADC A,(HL) AND (IX+d) CP (IY+d)

ADC A,(IX+d) AND (IY+d) INC r

ADC A,(IY+d) OR r INC (HL)

SUB r OR n INC (IX+d)

SUB n OR (HL) INC (IY+d)

SUB (HL) OR (IX+d) DEC r

SUB (IX+d) OR (IY+d) DEC (HL)

SUB (IY+d) XOR r DEC (IX+d)

SBC A,r XOR n DEC (IY+d)

**Группа сдвига и вращения**

RLCA RRC r SLA r

RLA RRC (HL) SLA (HL)

RRCA RRC (IX+dd) SLA (IX+dd)

RRA RRC (IY+dd) SLA (IY+dd)

RLC r RR r SRA r

RLC (HL) RR (HL) SRA (HL)

RLC (IX+dd) RR (IX+dd) SRA (IX+dd)

RLC (IY+dd) RR (IY+dd) SRA (IY+dd)

RL r SRL r

RL (HL) SRL (HL)

RL (IX+dd) RLD SRL (IX+dd)

RL (IY+dd) RRD SRL (IY+dd)

**Группа обмена, пересылки блоков и поиска**

EX DE,HL EX (SP),HL LDI CPI

EX AF,AF' EX (SP),IX LDIR CPIR

EXX EX (SP),IY LDD CPD

LDDR CPDR

**Круппа установки, сброса и проверки битов**

BIT b,r \* TST r SET b,r RES b,r

BIT b,(HL) \* TST n SET b,(HL) RES b,(HL)

BIT b,(IX+d) \* TST (HL) SET b,(IX+d) RES b,(IX+d)

BIT b,(IY+d) \* TSTIO n SET b,(IY+d) RES b,(IY+d)

\* эти инструкции уникальны для Z180/HD64180

**Группа ввода и вывода**

IN A,(n) OUT (n),A OUTI

IN r,(C) OUT (C),r \* OTIM

\* IN0 r,(n) \* OUT0 (n),r \* OTIMR

INI \* OTDM OTIR

INIR \* OTDMR OUTD

IND OTDR

INDR

\* эти инструкции уникальны для Z180/HD64180

Приложение 5 - Сообщения об ошибках

**Сообщения об ошибках ZMAC**

НЕФАТАЛЬНЫЕ ошибки идентифицируются в файле PRN и на консоли одним из следующих, приведенных ниже, кодов ошибок в первом символе строки. Если активизирован файл регистрации ошибок, то этот же код ошибки, также, будет первым символом строки в файле. Строки операторов с ошибками, обычно, не будут генерировать правильный объектный код. Попытка загрузки (с помощью MLOAD), или компоновка объектного файла, для которого были выведены сообщения об ошибках, очень вероятно, вызовет новые жалобы загрузчика (редактора связей) и, почти наверняка, произведет непригодный для использования .COM или двоичный файл.

| **Код ошибки** | **Его значение** |
| --- | --- |
| **A** | Ошибка параметра. Недопустимый параметр или выражение встречается в поле операнда. |
| **B** | Ошибка баланса. Операторы IF, ELSE и ENDIF не сбалансированы или пара MACRO / ENDM не согласована. Ищите место, где отсутствует один из этих элементов. |
| **C** | Символьная ошибка. Символ (такой, как управляющий символ) присутствует, где он недопустим. |
| **D** | Ошибка дублирования символа. Только события, которые произойдут после первого - помечаются как ошибка. Измените имя для второго вхождения для разрешения конфликта. |
| **E** | Ошибка выражения. Выражение в поле операнда не может быть вычислено. Смотрите главу с названием '[Выражения](#_Выражения)', если причина не очевидна. |
| **I** | Ошибка включения. Не удается найти указанный файл INCLUDE, или превышен допустимый уровень вложенности. |
| **K** | Ошибка ключевого слова.[[4]](#footnote-4) Имя регистра использовалось в поле метки и вызывает неоднозначность, если используется в поле операнда в качестве параметра. |
| **M** | Ошибка режима. Произошел неподходящий способ адресации. Проверьте выражение, которое использует внешний символ в недопустимом контексте. |
| **O** | Ошибка кода операции. Что-либо нераспознанное присутствует в поле Opcode. Ищите недостающий символ табуляции, или другую проблему формата, если это не просто опечатка. |
| **P** | Ошибка фазы. Метки имеет другое значение в проходе 2, чем она имела в проходе 1. Операторы во включаемом файле MACLIB могут содержать код генерирующий операторы только в режиме обработки прохода 1. Также, может быть виноват код (IF...ENDIF) условного ассемблирования, который собирается во время только одного из двух проходов. |
| **S** | Легендарная синтаксическая ошибка. Оператор плохо сформирован и не распознан ассемблером. Ищите неуместные разделители и недопустимые символ в предполагаемом контексте. |
| **T** | Были определены слишком много COMMON блоков. Только 12 COMMON блоков позволены в дополнение к /BLANK/ или неименованному COMMON. Вы должны будете удалить лишние. |
| **U** | Неопределенный символ. Символ в поле операнда никогда не определялся. Проверьте написание или типографскую ошибку в имени символа. |
| **V** | Ошибка значения. Параметр в поле операнда вне допустимого диапазона. Наиболее распространенная причина - относительный переход к цели, расположенной слишком далеко. Вы вводили некоторый новый код между двумя? Перестановка аргументов для регистры IX и IY могут также вызвать эту ошибку. |
| **W** | Предупреждение. Есть что-то сомнительное в текущем операторе. Это сообщение появляется для инструкций HD64180 или Z180, когда действует псевдо-операция .Z80. |

**Фатальные ошибки ZMAC**

ФАТАЛЬНЫЕ ошибки заставляют прервать ассемблирование без генерации объектного модуля. Файл регистрации ошибок, если активирован, создается и содержит запись ошибок до фатальной. Фатальные ошибки вызывают аварийное прекращение работы, потому что ассемблер испытывает недостаток в важной информации или ресурсах, чтобы продолжить работу или потому, что управляющая структура как IF или MACRO не сбалансированы. Если возникнет фатальная ошибка, на консоль будет отправлено одно из следующих сообщений.

**<FN>.<TYP> NOT FOUND**

Файл, определенный в спецификация в командной строке или в операторе INCLUDE не может быть найден. Проверьте орфографические ошибки или неправильную спецификацию DU в любом случае. Если ваша командная строка сложна и является хорошим кандидатом для опечаток, рассмотрите возможность создания псевдонима или файла submit для запуска ассемблера.

**INVALID OPTION SPECIFICATION**

Один или несколько параметров командной строки воспринимаются как недопустимые. Одна из причина, когда забывают разделить параметры запятыми или пробелами.

**MORE THAN 8 IF LEVELS PENDING....**

Вы не учитывали ENDIF? Номер строки сообщит, где начинается девятое вложенное условие IF. Обеспечиваются только 8 уровней вложенности. Ваш код будет намного более читаемым, если уровни вложенности не превышают 6 или 7 уровней.

**UNTERMINATED IF!**

Достигнут конец файла не найдя заключительного ENDIF. После проверки, что ENDIF действительно там, ищите другие нефатальные сообщения об ошибках, которые могут указать код, который никогда не производился.

**MEMORY FULL....**

Не осталось места для внутренних таблиц ассемблера. Вы можете обеспечить больше места: а) предотвратив использование файла PRN, и б) используя ZCNFG для уменьшения размеров буферов файлов до абсолютного минимума. Альтернативным решением является разбиение исходных файлов на более мелкие файлы, которые в дальнейшем будут объединены редактором связей. Не заблуждайтесь, пытаясь использовать файлы INCLUDE! Источник с включенными файлами является логически тем же, как будто файлы физически присутствуют.

**UNTERMINATED MACRO.....**

Оператор ENDM отсутствует в макроопределении, или блуждающее условие IF заставило проигнорировать ENDM.

**LOCAL LABEL LIMIT EXCEEDED**

Достигнут предел для общего количества из 9999 локальных меток для макросов. Ищите способы упростить макросы, чтобы избежать существующий предел. Также ищите способы разбить программу на соединяемые модули, чтобы избежать исчерпания памяти!

**MACRO NESTED PAST 16 LEVELS....**

ZMAC предусматривает вложение макросов максимум до 16 уровней. Вероятно, существует более простой способ закодировать функцию, если вы используете так много уровней!

**MACRO DEFINITION INSIDE AN INLINE MACRO....**

Макросы не могут быть определены во встроенных макросах: REPT, IRPC и IRP.

**Сообщения об ошибках ZML**

**Can't find file <filespec>**

Проверьте каталог указанный в спецификации файла. Файл действительно там? 'Опечатка' в имени файла - частая причина этой ошибки.

**Invalid option specification!**

У редактора связей есть проблема при разборе командной строки. Если в параметре ошибок нет (опечатки, например), то, возможно, отсутствует разделитель.

**Source filespec error - wildcard or bad DIR:!**

1) Подстановочные знаки не распознаны в именах файлов.

2) Часть спецификации файла DU: или DIR: корректна?

**<filespec> is an invalid REL file!**

Один из исходных файлов в командной строке не является файлом в формате REL, который распознает редактор связей. Вы не учли разделитель в командной строке? Определен неправильный файл?

**Memory full!**

Процесс компоновки не может быть завершен, потому что двоичный образ не помещается в доступную память (область TPA). Попытайтесь сократить количество глобальных символов, объединив несколько файлов REL или перестройте источник, чтобы создать несколько модулей кода с меньшим размером во время выполнения, которые могут быть загружены независимо. Если вы используете "полную" операционную систему ZCPR3, вы можете получить некоторую дополнительную память путем установки операционной системы с отключением большей части модулей системы.

**Read error!**

Что-то не так с файлом, который не может быть правильно прочитан. Попробуйте повторное ассемблирование, желательно из файлов резервных копий.

**Multiple main modules!**

Проверьте оператор END в каждом из ваших модулей. Параметр после END определяет обязательный начальный адрес, и по крайней мере два из модулей содержат такую спецификацию. Удалите лишние, которые являются несоответствующими и повторно ассемблируйте. У редактора связей нет критерия для выбора какой адрес использовать.

**Library search limit exceeded!**

Не более десяти библиотек могут быть определены в ассемблере оператором .REQUEST. Если ваш основной исходный файл невинен, посмотрите во включенных файлах запросы библиотек, которые вы, возможно, пропустили.

**Undefined symbols**

Перечисленные имена символов на которые ссылаются с помощью EXTERNAL в одном или более модулей, нигде не объявлены PUBLIC или GLOBAL. Распространенной причиной этой ошибки является не включение библиотеки формата REL в командную строку вызова. Вторая причина - включение не имеющего объявлений внешних ссылок в одном из исходных файлов ассемблера. ZMAC может быть сконфигурирован, чтобы проигнорировать отсутствующие внешние ссылки. Иначе, требует повторное ассемблирование модуля(ей), содержащих имена после добавления надлежащих объявлений PUBLIC.

**Duplicate symbols**

Перечисленные имена символов были объявлены PUBLIC больше чем в одном из соединяемых модулей. Вы должны повторно ассемблировать, по крайней, мере один из модулей с дублированными именами, изменив или удалив их из списка объявлений PUBLIC или GLOBAL.

**\*\*\*Overlapping segments\*\*\***

Это сообщение обычно появляется, когда вы присвоили адрес загрузки одному или более сегментам в параметрах командной строки. Удостоверьтесь, что вы назначили адреса правильно и в HEX.

**Internal Error**

Это сообщение никогда не должно возникать. Если это произойдет, пожалуйста, задокументируйте обстоятельства и сообщите о них. Используйте форму регистрации/отчета[[5]](#footnote-5) об ошибках (или копию) в этом руководстве.

Предметный указатель

А

Абсолютная основа перемещения 14

Абсолютный код 12

Абсолютный режим 14, 43

Абсолютный режим ассемблирования 14

Абсолютный файл 11

Адрес выполнения 43

Адрес перемещения 57

В

Вложенный макрос 52

Встроенные макросы 50

Встроенный макрос 47

Выражения 24

Выходные файлы PRL-типа 65

Д

Директивы ассемблера 17

З

Заголовок PRL 60

Заголовок RSX 60

И

Имена регистров 17

Индикатор основания 24

Инструкция MACRO 52

Информационные элементы 44

Исходный файл 76

К

Коды операций 16

Коды ошибок 23

Командная строка 7

Константа перемещения 43

Л

Локальные символы 49

М

Макроопределение 49

Макрос 17, 47

Макрос IRPC 51

Макросы повторения 50

Модуль 76

Н

Начальный адрес 44

О

Объект 12, 76

Объектный код 12

Объектный файл 12

Основа перемещения 13, 42, 45

Основной адрес загрузки 57

П

Параметр I 68

Параметр P 67

Переключатель 76

Перемещаемая величина 30

Перемещаемый режим 14, 44

Перемещаемый символ 30

Перемещение 43

Поле комментария 19

Поле метки 16

Поле оператора 16

Поле OPA 17

Поле OPR 17

Поток битов REL 80

Правила приоритета 28

Примеры командной строки 9

Программирование 11

Процесс макрорасширения 53

Псевдо-операции 17

Пустое пространство 76

С

Сегмент кода (CSEG) 58

Сегмент COMMON 66

СИМВОЛ 26

Символьная константа 25

Сохраняемый макрос 47

Строка заключенная в кавычки 18, 19

Счетчик адреса 26, 42

Счетчик команд 26, 42

Т

Текст ASCII 18

Ф

Файл

CFG 69

COM 12

ERR 22

HEX 12, 13

PRN 20, 76

REL 13

Файлы заголовков 61

Фактические параметры 48

Фактический параметр 53

Фиктивные параметры 48

Фиктивный параметр 53

Флаг ошибки программы 8

Формат Microsoft REL 80

Э

Элементы кода 44

A

ACCEPT 31

ASEG 14, 35

ASET 35

C

COMMENT 32

COMMON 35

CSEG 14, 36

D

DB 36

DC 36

DEFB 36

DEFC 36

DEFL 36

DEFM 36

DEFS 36

DEFW 36

DEFZ 36

DEPHASE 32

DS 36

DSEG 37

DW 37

DZ 37

E

ELSE 37

END 37

ENDIF 37

ENDM 37, 49

ENT 37

ENTRY 38

EQU 38

EXITM 38, 49

EXT 38

EXTERNAL 38

EXTRN 38

G

GLOBAL 39

H

**HD64** 32

HELP.COM 83

I

IF 39

IF1 32, 39

IF16 32

INCLUDE 40

IRP 40, 49

IRPC 40, 49

L

**LALL** 32

**LCTL** 32

**LFCOND** 33

**LIST** 33

LOCAL 40

M

MACLIB 40

MACRO 40

N

NAME 40

O

ORG 35, 40

P

PAGE 41

PHASE 33

PRINT 33

PRINTX 33

PUBLIC 41

R

REPT 41, 49

REQUEST 33

RLIST 33

S

SALL 34

SBTTL 34

SET 41

SFCOND 34

SLIST 34

SUBTTL 41

T

TITLE 34, 41

X

XALL 34

XCTL 34

XLIST 35

Z

Z80 35

Z80/Z180/HD64180 83

ZML 55

1. Термин СИМВОЛ иногда называют *Символическим именем* или *Идентификатором*. [↑](#footnote-ref-1)
2. ZMAC v1.7 снимает это ограничение [↑](#footnote-ref-2)
3. Накладываются друг на друга [↑](#footnote-ref-3)
4. В ZMAC версии 1.7 Ошибка 'K' больше не существует. [↑](#footnote-ref-4)
5. Форма регистрации не включена в переведенную версию этого руководства [↑](#footnote-ref-5)