**Программное обеспечение ПК "Орион-ПРО".**

**Система программирования на Ассемблере.**

**Руководство программиста.**

**(С) 1996 Орионсофт**

**Содержание**

|  |  |
| --- | --- |
| 1. Введение | 4 |
| 2. Обзор | 4 |
| 3. Структура исходных файлов | 5 |
| 3.1. Форматы | 5 |
| 3.1.1. Формат исходных строк | 5 |
| 3.1.2. Комментарий | 5 |
| 3.2. Метки | 6 |
| 3.2.1. Локальная метка | 6 |
| 3.2.2. Глобальная метка | 7 |
| 3.2.3. Внешняя метка | 7 |
| 3.2.4. Режим счетчика адресов памяти | 8 |
| 3.3. Коды операций и псевдооперации | 10 |
| 3.4. Аргументы (выражения) | 10 |
| 3.4.1. Операнды | 11 |
| 3.4.2. Операторы | 14 |
| 4. Характеристики Ассемблера | 17 |
| 4.1. Псевдооперации с отдельными функциями | 17 |
| 4.1.1. Псевдооперации выбора набора команд | 17 |
| 4.1.2. Псевдооперация определения данных и меток | 18 |
| 4.1.3. Псевдооперации задания режима присваивания адресов памяти | 22 |
| 4.1.4. Служебные псевдооперации | 27 |
| 4.1.5. Псевдооперации управления печатью листинга | 30 |
| 4.2. Макросредства | 36 |
| 4.2.1. Псевдооперации макроопределений | 37 |
| 4.2.2. Псевдооперации повторения | 39 |
| 4.2.3. Псевдооперации завершения | 42 |
| 4.2.4. Псевдооперация макрометок | 43 |
| 4.2.5. Специальные макрооператоры | 43 |
| 5. Псевдооперации условного ассемблирования | 45 |
| 6. Описание команд | 47 |
| 6.1. Команды загрузки | 47 |
| 6.1.1. 8-разрядные команды загрузки | 48 |
| 6.2. Косвенные операции регистров (команды PUSH и POP) | 51 |
| 6.2.1. Команды PUSH | 51 |
| 6.2.2. Команды РОР | 51 |
| 6.3. Команды смены регистров | 52 |
| 6.4. Команды передачи блоков | 53 |
| 6.5. Команды поиска в блоке | 54 |
| 6.6. Арифметические и логические операции | 55 |
| 6.6.1. 8-разрядная арифметика | 55 |
| 6.6.2. 16-Разрядная арифметика | 56 |
| 6.6.3. 8-Разрядные логические команды | 57 |
| 6.7. Команды перехода | 59 |
| 6.8. Команды сдвига | 60 |
| 6.9. Специальные команды аккумулятора и флажков | 62 |
| 6.10. Команды вызова подпрограмм | 63 |
| 6.11. Команды возврата из подпрограммы | 64 |
| 6.12. Команды управления центральным процессором | 65 |
| 6.13. Команды установки и проверки четности | 66 |
| 6.14. Команды ввода | 67 |
| 6.15. Команды вывода | 68 |
| 6.16. Перечень сокращений для описания команд | 69 |
| 6.17. Работа с битами условий (флажки) | 70 |
| 7. Перекрестные ссылки | 72 |
| 7.1. Формирование листинга перекрестных ссылок | 72 |
| 7.1.1. Формирование файла перекрестных ссылок | 73 |
| 7.1.2. Формирование листинга перекрестных ссылок | 73 |
| 7.2. Псевдооперации для управления листингами | 74 |

**1. Введение.**

В документе слово "Ассемблер", в соответствии с принятой практикой, означает как язык Ассемблера, так и транслятор с языка Ассемблера. Необходимое различие приводится по контексту.

Ассемблеру для работы требуется 19 К памяти плюс 4 К для области буферов.

Замечания по синтаксису:

[+ - Параметры в скобках задаются произвольно.

<> - Текст в угловых скобках, указанный маленькими буквами, должен быть подготовлен пользователем (к примеру, <имя файла>). Текст, указанный большими буквами, набирается клавишами (к примеру, <ет>).

() - Выбор между двумя или более вариантами.

... - Элемент может повторяться.

Все другие знаки препинания , : / = \* $ принимаются без изменений.

**2. Обзор**

а) Ассемблер поддерживает две мнемоники системы команд ПК "Орион-ПРО":

-мнемонику команд микропроцессора серии 8080;

-мнемонику команд микропроцессора типа Z80;

б) Ассемблер генерирует относительный и/или абсолютный код.

в) Ассемблер поддерживает написание макрокоманд.

Программист отводит блок для некоторого количества директив. Этому блоку присваивается имя, по которому макрокоманда вызывается. Директивы являются макроопределениями.

Если эти команды потребуются в каком-нибудь месте, то программист вызывает макрокоманду. Кроме того при макровызове Ассемблеру передаются параметры для расширения макрокоманд.

Использование макрокоманд уменьшает размер исходных программ, поскольку макроопределения могут находиться в отдельном файле на дискете и могут включаться в модуль только тогда, если к ним производится запрос во время ассемблирования.

Макрокоманды могут быть вложенными, т.е. в одной макрокоманде можно производить вызов другой. Глубина вложенности ограничивается только размером памяти.

г) условное ассемблирование.

Ассемблер поддерживает условное ассемблирование.

Программист может определить условие, при выполнении которого некоторые части программы либо ассемблируются, либо не ассемблируются.

Возможности условного ассемблирования увеличиваются с помощью полного набора условных псевдоопераций, которые включают проверку проходов Ассемблера, определений символов и параметров макрокоманд. Условия могут быть вложенными до 255 раз.

**3. Структура исходных файлов**

**3.1. Форматы**

Исходными ассемблерным файлом является некоторое количество строк текста, которые написаны на языке Ассемблера.

Последней строкой файла должна быть директива END, которая завершается символом <ВК>.

Соответствующие директивы, как например IF...ENDIF, должны находится в правильной последовательности. В других случаях строки могут появляться в программе в произвольной последовательности.

**3.1.1. Формат исходных строк**

Исходный файл ввода для ассемблера состоит из строк директив, которые разделены на части или поля:

*Метка Операция Аргумент Комментарий*

где:

*Метка* - Это поле может содержать метку одного из трех типов, после которых следует двоеточие, за исключением, когда метка является частью директивы SET, EQU или MACRO

- локальная метка;

- глобальная метка;

- внешняя метка.

*Операция* - Это поле содержит код операции, псевдооперацию, имя макрокоманды или какое-либо выражение.

*Аргумент* - Это поле содержит выражения, переменные, имена регистров, операнды и операторы.

*Комментарий* - Это поле содержит текст комментария, который всегда должен начинаться с точки с запятой.

Заполняются все поля. Допустимыми являются также и символы пробелов.

Директивы начинаются с любого столбца. Несколько символов пробелов или знаки табуляции могут быть вставлены между полями для лучшей читаемости, но между двумя полями необходимым является наличие по меньшей мере одного символа пробела или знака табуляции.

**3.1.2. Комментарий**

Комментарий всегда начинается точкой с запятой и завершается концом строки.

Для длинных комментариев может быть использована псевдооперация COMMENT (см. п. 4.1.4.), чтобы избежать вставки точек с запятыми на каждой строке комментария.

**3.2. Метки**

Метками являются имена элементарных функций или величин. Имена меток определяются программистом. Метки для этого пакета программ относятся к одному из трех типов согласно их функции.

Эти три типа следующие:

- локальный:

- глобальный;

- внешний.

Все три типа имеют атрибут режима, который устанавливается с сегментом памяти, представляемым символом (см. П. 3.2.4.).

Все три типа меток имеют следующие характеристики:

1. Метки могут быть любой длины, но количество значащих символов в ней, которые передаются сборщику, варьируется в зависимости от типа метки:

а) в локальной метке значащими являются первые 16 символов.

б) в метках глобального и внешнего типов сборщику передаются первые шесть символов.

Дополнительные символы отсекаются "внутри".

2. Действительная метка может содержать следующие символы:

A-Z 0-9 $ . ? @ +

3. Метка не должна начинаться с цифры или со знака подчеркивания.

4. При считывании какого-либо символа маленькие буквы преобразуется в большие. Таким образом, в метке могут вперемежку встречаться и маленькие и большие буквы.

**3.2.1. Локальная метка**

Локальной меткой является точка ссылки на директиву внутри программного модуля, в котором эта метка определена.

Локальная метка определяет значение своего адреса как адреса данных, которые поставлены в соответствие этой метке.

Пример: BUFF: DS 1000H

BUFF равен первому адресу зарезервированной области байтов 1000H.

Если какая-либо локальная метка определена один раз, то она может быть использована в записи в поле аргумента.

Какая-либо директива с локальной меткой в своем поле аргумента ссылается на строку директивы, в которой эта метка определена в поле метки.

Пример: LD A,(BUFF)

При такой записи содержимое регистра A переписывается в ячейку памяти, определяемой локальной меткой BUFF.

Локальная метка может состоять из любых действительных символов в количестве до 16.

Если требуется определить какую-либо локальную метку, то она должна стоять первой в строке директивы.

В мнемонике команд как микропроцессора типа Z80, так и микропроцессора серии 8080 после локальной метки сразу же должно следовать двоеточие (не символ пробела или знак табуляции), но не тогда, когда метка является частью директивы SET или EQU (тогда двоеточия нет).

Если за меткой следует два двоеточия, то она определяется как глобальная метка.

**3.2.2. Глобальная метка**

Глобальная метка определяется в поле метки. Ее различие с локальной, состоит в том, что глобальная доступна как точка ссылки из других программ.

Имеется две возможности для объявления метки глобальной:

1. Два двоеточия ставятся после имени, к примеру:

MARKE::RET.

2.Использование одной из псевдоопераций PUBLIC, ENTRY или CLOBAL, к примеру: PUBLIC MARKE.

Пример: результат обоих способов идентичен, запись:

MARKE::RET

эквивалентна последовательности директив:

PUBLIC MARKE.

.

.

.

MARKE::RET

**3.2.3. Внешняя метка**

Внешняя метка определена вне текущего программного модуля в каком-либо другом отдельном модуле как глобальная метка.

Присваивание значения ей производится при компоновке программы. При этом внешней метке передается значение глобальной метки из другого программного модуля.

Пример:

MODUL1

MARKE:: DB 7 ; PUBLIC MARKE=7

MODUL2

BYTE EXT MARKE ; EXTERNAL MARKE

Во время сборки программы для EXTERNAL MARKE используется значение PUBLIC MARKE.

Метка об'является внешней с помощью следующих трех способов:

1. Указание двух "решеток" (##) перед именем символа.

Пример: CALL MARKE##

Здесь MARKE объявляется как метка 2-байтной адресной величины, которая определена в каком-либо другом модуле.

2. Псевдооперации с 2-байтными величинами EXT, EXTRN или EXTERNAL.

Пример: EXT MARKE

Здесь MARKE объявляется как метка 2-байтной величины, которая определена в каком-либо другом модуле.

3. Псевдооперации с 1-байтными величинами BYTE EXT, BYTE EXTRN или BYTE EXTERNAL.

Пример: BYTE EXT MARKE

Здесь MARKE объявляется как метка 1-байтной величины, которая определена в каком-либо другом модуле.

Результат всех трех способов согласования один и тот же, запись:

CALL MARKE##

эквивалентна последовательности директив:

EXT MARKE

CALL MARKE

**3.2.4. Режим счетчика адресов памяти**

При появлении имени метки в списке аргументов производится ссылка на соответствующую строку директивы.

Значением метки является адрес директивы, которая помечена этой меткой. Если эта метка появляется в поле аргумента, то она заменяется своим значением и используется в операции.

Значение метки определяется согласно режиму счетчика адресов памяти (РС).

Посредством задания режима счетчика адресов памяти становится возможным деление памяти на четыре сегмента. Режим определяет, в какой сегмент загружается данная часть программы. Эти 4 сегмента следующие:

-абсолютный сегмент;

-сегмент относительно данных;

-сегмент относительно общей области.

Режим счетчика адресов памяти устанавливает, что некоторая часть программы загружается:

-в память по абсолютному адресу, заданному программистом (абсолютный режим);

-по относительным адресам, которые меняются в зависимости от размеров и числа программ (режим относительно кодов) и объема данных (режим относительно данных);

-по адресам, которые представляют собой общую область с другими программами (режим относительно общей области).

Если ничего не задается, то имеет место режим относительных кодов (стандартный).

Одна программа может содержать свои части в различных сегментах памяти.

**Абсолютный режим**

В абсолютном режиме генерируются неперемещаемые в памяти коды. Программист выбирает абсолютный режим, если некоторый программный блок всегда требуется загружать по фиксированным абсолютным адресам независимо от остальных типов загружаемых сегментов (DSEG, CSEG, COMMON).

**Режим относительных данных**

В режиме относительных данных генерируется перемещаемые в памяти коды той части программы, которая может содержать изменяемые коды и, следовательно, должна загружаться в область памяти с произвольным доступом (RAM). Особенно это относится к областям данных программ. Метки в режиме относительных данных являются перемещаемыми.

**Режим относительно кодов**

В режиме относительно кодов генерируются коды для перемещаемой в памяти части программы, коды которой не изменяются. Части программы, предназначенные для хранения в постоянной памяти (PROM), должны генерироваться в режиме относительно кодов

**Режим относительно общей области**

В режиме относительно общей области генерируются коды, которые загружаются в заранее определенную общую область данных. Это дает возможность разделить программный модуль на блок памяти и общие параметры.

Для изменения режима в какой-либо строке директивы используется псевдооперация режима счетчика адресов памяти.

ASEG -абсолютный режим

DSEG -режим относительно данных

CSEG -режим относительно кодов (режим присваивания)

COMMON -режим относительно общей области

эти псевдооперации детально описаны в п. 4.1.3.

Возможность задания режима счетчика адресов памяти в ассемблере позволяет программисту разрабатывать перемещаемые ассемблерные программы.

Перемещаемость означает, что программный модуль может загружаться по любым адресам имеющейся в наличии памяти и выполняться (при использовании ключа /P и /D сборщика).

**3.3. Коды операций и псевдооперации**

Коды операций - это мнемонические имена машинных команд (команд центрального процессора).

Псевдооперации - это инструкции для транслятора с Ассемблера, а не для микропроцессора.

Коды операций и псевдооперации заносятся (часто) в поле операции строки директивы.

Операцией может быть:

-любая мнемоническая команда для микропроцессора типа Z80 или серии 8080;

-псевдооперация Ассемблера;

-макровызов;

-выражение.

Записи в операционном поле анализируются транслятором с Ассемблера в следующей последовательности:

1. Макровызов.

2. Код операции / псевдооперация.

3. Выражение.

Ассемблер сравнивает запись в поле операции с некоторым внутренним списком макроимен. Если имя найдено, то производится раскрутка макрокоманды и соответствующие директивы включаются в модуль (смотри для этого также п. 4.2.).

Если запись не является макрокомандой, то Ассемблер пытается рассматривать ее как код операции.

Если запись не является кодом операции, то Ассемблер пытается рассматривать ее как псевдооперацию. Если запись не является и псевдооперацией, то Ассемблер рассматривает ее как выражение.

Если выражение было написано, как строка директивы без кода операции, псевдооперации или макроимени впереди, то Ассемблер об ошибке не сообщает. Более того, он предполагает, что определенный байт (псевдооперация) относится к месту перед выражением

и транслирует эту строку.

По причине определенной последовательности анализа некоторая макрокоманда, имя которой идентично какому-либо коду операции, не позволяет его использование этого кода. Это имя далее используется в качестве макровызова. Если, к примеру, какой-либо макрокоманде присвоено имя ADD, то в этой программе ADD не может использоваться как код операции!

**3.4. Аргументы (выражения)**

Аргументы для кодов операций и псевдоопераций обычно обозначаются как выражения, т.к. они напоминают математические выражения, как например: 5+4\*3.

Части выражения называются операндами (в данном математическом это 5,4 и 3) и операторами (в примере это + и \*). Выражения могут содержать один или несколько операндов.

Выражения с операндами являются наиболее часто используемыми аргументами.

Если какое-либо выражение имеет больше чем один операнд, то операнды связаны между собой оператором.

Пример:

Операнды в Ассемблере - это числовые величины, представленные числами, знаками, символами или кодами операций микропроцессора серии 8080. Операторы могут быть арифметическими или логическими.

В следующих разделах дается определение допустимые в ассемблере операнды и операторы.

**3.4.1. Операнды**

Операнды могут быть:

- числа;

- знаки;

- символы;

- коды операций микропроцессора серии 8080.

**Числа**

При работе с Ассемблером можно использовать различные системы счисления.

Принятой базой для чисел является десятичная (стандарт). Эта база может быть изменена при помощи псевдооперации .RADIX . Может быть выбрана любая база от 2 (двоичная) до 16 (шестнадцатеричная). Если базовое число больше чем 10, то в качестве цифр, следующих за цифрой 9, используются буквы от а до F. Если первый знак нечисловой, то впереди должен быть поставлен 0 (к примеру, 0F8).

Какое-либо число анализируется относительно текущего базового числа, если не используется ни одно из следующих обозначений:

NNNNB двоичное число

NNNND десятичное число

NNNNо восьмеричное число

NNNNо шестнадцатеричное число

XNNNN шестнадцатеричное число

Числами являются беззнаковые 16-битовые величины (в диапазоне значений 0 ... 65535). Переполнение числа при переходе за пределы двух байтов игнорируется, а результат образуется шестнадцатью младшими битами.

**Символьные цепочки кода КОИ-7**

Символьные цепочки образуются из символов в количестве 0 или больше и ограничиваются кавычками (") или апострофами (').

Пример: DB "авс"

В данной записи код символа, а запоминается по первому адресу, код символа в - по этому адресу +1, а код символа с - по этому адресу +2.

Сами ограничительные символы могут быть использованы просто как знаки, когда они задаются два раза подряд.

Пример: "сегодня ""прекрасный"" день"

Данная запись означает цепочку символов:

сегодня "прекрасный" день

Если между кавычками не стоят никакие символы, то эта цепочка рассматривается как пустая цепочка (пустая символьная цепочка).

**Символьные константы**

Символьные константы - это цепочки, которые составляются из символов КОИ-7 в количестве нуль, один или два и ограничиваются кавычками (") или апострофами ('). Если ограничительный символ должен входить в состав цепочки, то в этом месте он должен быть

записан дважды.

Различия между символьными константами и цепочками следующие:

1. Символьная константа состоит только из символов в количестве нуль, один или два.

2. Записанные символы только тогда являются символьной константой, когда выражение имеет больше чем один операнд. Если символы встречаются как отдельный операнд, то они рассматриваются и запоминаются как цепочка.

Пример: 'а' +1 - это символьная константа, но 'A' - это символьная цепочка.

3. Значение некоторой символьной константы вычисляются, а результат запоминается: младший байт по первому адресу, а старший байт по второму адресу.

Пример: DW 'AB' +0

При данной записи воспринимается число 4142Н, число 42Н запоминается по первому адресу, а 41Н - по второму адресу.

Символьной константе, которая содержит только один символ, присваивается код символа. Это значит, что старший байт ее значения равен нулю, а младший байт равен коду символа.

Пример: значение символьной константы 'A' - это 4100Н.

Символьная константа, которая состоит из двух символов, в качестве своего значения имеет код КОИ-7 первого символа в старшем байте и код КОИ-7 второго символа в младшем байте.

К примеру, значение символьной константы 'AB' +0 равно 41H\*256+42Н+0.

**Метки в выражениях**

Метка может быть использована в каком-либо выражении в качестве операнда. Метка анализируется и ее значение подставляется вместо символа.

**Инструкции по применению внешних меток в выражениях**

1. Внешние метки могут быть использованы в выражениях только со следующими операторами:

+ - \* / MOD HIGH LOW

2. Если в каком-либо выражении была использована внешняя метка, то результат выражения всегда будет внешнего типа.

**Инструкции по режимам для символов в выражениях**

1. При всех операциях, кроме AND, OR и XOR, операнды могут иметь любые режимы.

2. При операторах AND, OR, XOR, SHL и SHR оба операнда должны быть абсолютными и внутреннего типа.

3. Если какое-либо выражение содержит один абсолютный операнд и один операнд в другом режиме, то результат выражения будет всегда другой, отличный от абсолютного режима.

4. При перемножении двух операндов различных режимов результат будет абсолютным. В остальных случаях результат будет типа операндов.

5. Если складываются метка в режиме относительно данных и метка в режиме относительно кодов, то результат будет неопределенным. Транслятор с Ассемблера передает это выражение компоновщику программ как неопределенное и тот разрешает его.

**Текущий счетчик адресов памяти**

Дополнительным символом для поля аргументов является счетчик адресов памяти.

Значением счетчика адреса памяти является адрес следующей транслируемой директивы.

Вместо того, чтобы запоминать или вычислять текущий адрес программы, программист может использовать определенный символ, который сообщает транслятору с Ассемблера актуальное значение счетчика адресов памяти.

Текущим символом счетчика адресов памяти является $.

**Коды операций микропроцессора серии 8080 как операнды.**

Коды операций микропроцессора серии 8080 могут быть использованы как операнды.

Во время трансляции вычисляются шестнадцатеричные значения кодов операций, которые используются как операнды.

При использовании кодов операций микропроцессора серии 8080 в качестве операндов с самого начала должна быть выполнена псевдооперация .8080 (См. раздел 4.1.3.).

Только первый байт вычисленной шестнадцатеричной величины является действительным операндом.

Использование круглых значений указывает транслятору с ассемблера на то, что для кода операции следует генерировать один байт, где в обычном случае генерируется больше чем один байт.

Пример: MVI A, (SMP)

ADI (CPI)

MVI B, (RNZ)

CPI (INX H)

ACI (LXI B)

Ассемблер сообщает об ошибке, когда вычисленная шестнадцатеричная величина (внутри круглых скобок) получается больше чем один байт, как например, (CPI 5), (LXI B, LABEL1) или (JMP LABEL2).

Коды операций, которые обычно генерируют один байт, могут быть использованы в качестве операндов без заключения в круглые скобки.

**3.4.2. Операторы**

Ассемблер позволяет использование как арифметических, так и логических операторов.

Операторы, которые в качестве результата выдают "истиность" или "ложность" условия, возвращают значение "истина", если результат не равен 0, и "ложь", если результат равен 0.

В качестве значения "истина" передается 0FFFFH, а в качестве значения "ложь" - 0000н.

В выражениях разрешается использование следующих арифметических и логических операторов, при этом выражения в угловых скобках <ехр> являются числовыми выражениями.

|  |  |
| --- | --- |
| Оператор | Определение |
| NUL | - возвращает значение "истина", если аргумент параметр равен нулю. |
| ........................\* В качестве аргумента .......\* |
| Условие IF NUL аргумент является ложным, если первый знак аргумента любой кроме точки с запятой или ET.  Примечание: операторы IFB и IFNB выполняют ту же функцию, но их использование проще (см. П. 4.1.5.). |
| TYPE | - оператор TYPE возвращает байт, который описывает характеристики его аргумента:  - ...........\* Не .........\*  Аргументом оператора TYPE может быть любое выражение.  Если выражение недействительно, то оператор TYPE возвращает 0. Этот байт, который возвращается оператором, имеет следующую структуру:  .................\*:  0 0 Абсолютный режим  0 1 режим относительно кодов  1 0 режим относительно данных  1 1 режим относительно общей области  бит 7 - признак внешнего типа:  1 выражение внешнего типа  0 выражение локальное  бит 5 - признак определения выражения:  1 выражение определено локально  0 выражение не определено или внешнего типа  Оператор TYPE используется по меньшей мере внутри макрокоманды, где должен проверяться режим какого-либо аргумента для выработки решения относительно дальнейшего выполнения программы, к примеру, когда имеет место условное ассемблирование.  Пример: MARKE MAKRO X  LOKAL Z  Z SET TYPE X  IF Z...  Здесь оператор TYPE проверяет режим  и присваивание X. В зависимости от  результата анализа X блок кодов, начиная  с IF Z... Либо транслируется, либо нет. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
| LOW <EXP> | - изолирование (выделение) восьми младших битов (младшей части) абсолютной 16-битной величины. |
|  |
|  |
| HIGH <EXP> | - изолирование (выделение) восьми старших битов (старшей части) абсолютной 16-битной величины. |
|  |
|  |
| <EXP1> \* <EXP2> | - умножение. |
|  |  |
| <EXP1> / <EXP2> | - деление. |
|  |  |
| <EXP1> MOD <EXP2> | - модуль (остаток от деления).  Деление выражения <EXP1> на <EXP2> и возврат остатка в качестве результата (модуль). |
|  |
|  |
|  |
| <EXP1> SHR <EXP2> | - сдвиг вправо.  Выражение <EXP1> сдвигается вправо на число битовых позиций, заданных выражением <EXP2>. |
|  |
|  |
|  |
| <EXP1> SHL <EXP2> | - сдвиг влево. |
|  | Выражение <EXP1> сдвигается влево на число битовых позиций, заданных выражением <EXP2>. |
|  |
|  |
| -<EXP> | - присваивание отрицательного арифметического знака выражению <EXP> (отрицательное целое число). |
|  |
|  |
|  |  |
| EXP1 + EXP2 | - сложение. |
|  |  |
| <EXP1> - <EXP2> | - вычитание. |
|  |  |
| <EXP1> EQ <EXP2> | - равенство.  Возвращает значение "истина", если выражения<EXP1> и <EXP2> равны. |
|  |
|  |
|  |  |
| <EXP1> NE <EXP2> | - неравенство.  Возвращает значение "истина", если выражения <EXP1> и <EXP2> неравны. |
|  |
|  |
|  |  |
| <EXP1> LT <EXP2> | - меньше чем.  Возвращает значение "истина", если выражения<EXP1> меньше чем выражение <EXP2>. |
|  |
|  |
|  |
| <EXP1> LE <EXP2> | - меньше чем или равно.  Возвращает значение "истина", если выражения<EXP1> меньше или равно выражению <EXP2>. |
|  |
|  |
|  |
| <EXP1> GT <EXP2> | - больше чем. |
|  | Возвращает значение "истина", если выражения<EXP1> больше чем выражение <EXP2>. |
|  |
|  |
| <EXP1> GE <EXP2> | - больше чем или равно.  Возвращает значение "истина", если выражения<EXP1> больше или равно выражению <EXP2>. |
|  |
|  |
|  |
| NOT <EXP> | - инверсия <EXP>. |
|  |  |
| <EXP1> AND <EXP2> | - логическое и.  Возвращает значение "истина", если выражения <EXP1> и <EXP2> истины; возвращает значение "ложь", если одно из  обоих или оба выражения ложны. |
|  |
|  |
|  |
|  |
| <EXP1> OR <EXP2> | - логическое или.  Возвращает значение "истина", если одно из выражений или оба истины; возвращает значение "ложь", если оба выражения ложны. |
|  |
|  |
|  |
|  |
| <EXP1> XOR <EXP2> | - исключающее или. |
|  | Возвращает значение "истина", если выражение <EXP1> или <EXP2> истинно, а другое ложно.  Возвращает значение "ложь", если выражения <EXP1> и <EXP2> оба истины, или оба ложны. |
|  |
|  |
|  |
|  |
|  |

Последовательность обработки этих операторов следующая:

NUL, TYPE

LOW, HIGH

\*, /, MOD, SHR, SHL

Присваивание знака минус

+, -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Если подвыражения содержат операторы более высокого порядка, то тогда сначала вычисляется это выражение.

Последовательность обработки может быть изменена посредством использования круглых скобок вокруг части выражения, которое должно получить более высокий порядок.

Все операторы кроме +,-,\*, и / должны отделятся от своих операторов по меньшей мере одним знаком пробела. Операторы выделения байтов (HIGH и LOW) отделяют старшие или младшие 8 битов от 16-битной величины одним знаком пробела.

**4. Характеристики Ассемблера**

Ассемблер имеет три важных преимущества:

- псевдооперации с отдельными функциями;

- возможность макропрограммирования;

- условное ассемблирование.

**4.1. Псевдооперации с отдельными функциями**

Псевдооперации с отдельными функциями содержат только одну строку директивы, они задают транслятору с Ассемблера выполнение только одной функции (макрокоманды и условные команды содержат больше чем одну строку; их можно рассматривать как блок псевдоопераций).

Псевдооперации с отдельными функциями подразделяются на 5 групп:

- выбора набора команд;

- определения данных и символов;

- режима счетчика адресов памяти;

- псевдоопераций над данными;

- управления таблицами.

4**.1.1. Псевдооперации выбора набора команд**

Мнемоника команд микропроцессора серии 8080 включена в Ассемблер как стандарт. Если задана неверная псевдооперация выбора набора команд, то Ассемблер сообщает о серьезной ошибке в тех кодах операций, которые для текущего набора команд недействительны. Это означает, что при директиве .Z80 транслируются коды операций микропроцессора типа Z80, а при директиве .8080 - Коды операций микропроцессора серии 8080.

Поэтому в программах, написанных в мнемонике микропроцессора типа Z80, должна быть псевдооперация .Z80 для выбора соответствующего набора команд.

Все псевдооперации, описанные в данной главе, обрабатываются в рамках обоих наборов команд, если исключения специально не оговариваются.

**Псевдооперация .Z80**

Псевдооперация .Z80 аргументов не имеет. Она задает Ассемблеру режим трансляции кодов операций микропроцессора типа Z80.

**Псевдооперация .8080**

Псевдооперация .8080 Аргументов не имеет. Она задает Ассемблеру режим трансляции кодов операций микропроцессора серии 8080. Выбор этого набора команд стандартный.

Все коды операций, которые следуют за псевдооперацией выбора набора команд, транслируются соответствующим образом до тех пор, пока не встретится альтернативная псевдооперация.

Если встречается код операции, который не принадлежит выбранному набору команд, то транслятор с Ассемблера сообщает об ошибке типа "U".

**4.1.2. Псевдооперация определения данных и меток**

Все программные операции для определения данных и символов поддерживаются в рамках обоих наборов команд. Единственным исключением является псевдооперация SET, которая недопустима для набора команд микропроцессора типа Z80.

Определение байта

DB <EXP>[,<EXP>...+

DEFB <EXP>[,<EXP>...+

DEFM <EXP>[,<EXP>...+

Примечание: псевдооперация DB используется как пример, поясняющий все возможные псевдооперации определения байтов.

Аргументы <EXP> в псевдооперациях типа DB являются либо выражениями, либо символьными цепочками. Цепочки должны быть заключены в одинарные или двойные кавычки.

Псевдооперация DB используется для того, чтобы занести некоторые значения (цепочек или числовые) в некоторое место памяти, начиная с текущего адреса в счетчике адресов памяти.

Байты выражений должны проверяться. Если старший байт получается равным 0 или 255, то ошибки не возникает. В противном случае генерируется ошибка типа "А".

Цепочки с тремя и более символами в выражениях использоваться не должны (они должны разрываться запятой или символом конца строки).

Символы цепочек при обоих наборах команд микропроцессоров серии 8080 и типа Z80 запоминаются в последовательности их появления, каждый как однобайтная величина, старшие биты сброшены в 0.

Пример:

DB 'AB'

DB 'AB' AND 0FN

DB 'ABC', 3

Генерируются в:

0000' 41 42 DB 'AB'

0002' 02 DB 'AB' AND 0FH

0003' 41 42 43 03 DB 'ABC', 3

**Определение символов**

DC <строка>[, <строка> ...+

Псевдооперация DC заносит символы цепочки, указанные в строках, в ячейки памяти, следующие друг за другом, каждый в виде однобайтной величины, старшие биты сброшены в 0. Однако, в последнем символе каждой цепочки старшие биты устанавливаются в 1.

Если аргумент псевдооперации DC является пустой цепочкой, то генерируется ошибка.

Пример:

MARKE: DC "ABC", 'DEF'

Генерируется в:

0000' 41 42 с3 64 MARKE: DC "ABC", 'DEF'

0004 65 е6

Определение области памяти

DS <EXP1>[,<EXP2> ...+

DEFS <EXP1>[,<EXP2> ...+

Псевдооперацией определения области памяти резервируется определенная область в памяти. Значение выражения <EXP1> задает количество резервируемых байтов.

Для инициализации зарезервированной области используется выражение <EXP2>, которое должно иметь требуемое значение.

Если выражение <EXP2> опущено, то зарезервированная память не инициализируется. Зарезервированная область памяти автоматически не инициализируется (не заполняется) значениями 00н.

Такая возможность инициализации области памяти значениями 00н предоставляется при использовании ключа /м во время трансляции (см. п. 5.4.2.).

Все имена, которые используются в выражениях, должны быть предварительно определены (все имена должны быть известны к этому моменту в первом проходе).

В противном случае во время первой фазы генерируется ошибка типа "V"; ошибка типа "U" может генерироваться во втором проходе; ошибка в фазе получается, вероятно, из-за того, что в первом проходе псевдооперацией определения области памяти не было сгенерировано никакого кода.

Пример: DS 100H

Здесь резервируются 100 байтов памяти, которые не инициализируются, т.е. Там остаются те значения, которые были на их месте до загрузки программы.

Если область памяти требуется инициализировать значениями 00н, то во время трансляции должен быть использован ключ /М. Если зарезервированную область памяти требуется заполнить значениями 02, то используется директива:

DS 100H,2

Здесь резервируются 100 байтов, каждый из которых инициализируется значением 02н.

**Определение слова**

DW <EXP> [,<EXP> ...+

DEFW <EXP> [,<EXP> ...+

Псевдооперация определения слова записывает значение выражения в ячейки памяти следующие друг за другом, начиная с текущего адреса в счетчике адресов памяти. Выражения являются двухбайтными значениями (слово).

Сначала в памяти расположен младший байт, а потом следует старшие (в отличии от псевдооперации DB).

Пример:

MARKE: DW 1234H

Генерируется в:

0000'1234 MARKE: DW 1234н

Примечание: на листинге байты отображаются в порядке их появления, не в той последовательности, в которой они запоминаются в памяти.

Эквивалентность

<Имя> EQU <EXP>

Псевдооперация EQU присваивает заданному в ней имени значение выражения <EXP>. Параметр <имя> может быть именем метки любого типа или именем переменной и использоваться в дальнейшем в выражениях.

После параметра <имя> двоеточия стоять не должны.

Если выражение <EXP> является внешним, то генерируется ошибка.

Если параметр <имя> уже имеет какое-нибудь другое значение, то генерируется ошибка типа "М".

Если и в дальнейшем в программе параметр имя должен изменять свое значение, то вместо псевдооперации EQU для этого следует использовать псевдооперацию SET или ASET.

Пример: BUF EQU 0F3H

**Определение внешней метки**

EXT <имя>[,<имя>...+

EXTRN <имя>[,<имя>...+

EXTERNAL <имя>[,<имя>...+

BYTE EXT <имя>

BYTE EXTRN <имя>

BYTE EXTERNAL <имя>

Этой псевдооперацией объявляется, что имя (имена) в списке аргументов является (являются) внешним (внешними), т.е. они определены в каком-либо другом модуле.

Если одно из имен в списке аргументов уже определено в текущей программе, то генерируется ошибка типа "М".

Если в транслируемой программе за именем непосредственно следуют две "решетки" (к примеру, NAME##), то это имя также объявляется типа EXTERNAL (внешним).

Внешние имена можно анализировать по одному или по обоим байтам. Для всех внешних имен сборщику передаются только первые шесть их символов.

Дополнительные символы отсекаются.

Пример: EXTRN MARKE

При трансляции модуля для этой строки директивы Ассемблер не генерирует никакой код.8

Если MARKEX используется в качестве аргумента в директиве CALL, то генерируется код только для директивы CALL, а для MARKEX заносится значение 0.

Сборщик просматривает все модули, включаемые в результирующую программу, на предмет наличия в них директивы PUBLIC MARKEX и использует в директиве CALL MARKEX то определение, которое он нашел для MARKEX.

**Определение глобальной метки (PUBLIC)**

ENTRY <имя>[,<имя>...+

GLOBAL <имя>[,<имя>...+

PUBLIC <имя>[,<имя>...+

Псевдооперацией PUBLIC каждое имя в списке ее аргументов объявляется внутренним для использования в данной программе и в других, которые связываются вместе сборщиком.

Если имя типа EXTERNAL или имя блока общей памяти, то генерируется ошибка типа "М".

Если в транслируемой программе непосредственно за именем следуют два двоеточия (к примеру, NAME::), то оно также объявляется как глобальное (PUBLIC).

Сборщику только первые шесть символов имени глобальной метки.

Дополнительные символы отсекаются.

Пример:

PUBLIC MARKEX

MARKEX: LD HL, BER1

Здесь Ассемблер транслирует директиву LD как обычно, но для директивы PUBLIC MARKEX он не генерирует никакого кода.

Если сборщик находит в другом модуле директиву EXTRN MARKEX, то он знает, что должен искать, пока он не обнаружит эту директиву PUBLIC MARKEX. После этого сборщик связывает значение адреса строки директивы MARKEX: LD HL,BER1с директивой директивами CALL MARKEX в другом (других) модуле (модулях).

Директива SET

<имя> SET <EXP> (не для режима .Z80)

<имя> DEFL <EXP>

<имя> ASET <EXP>

Псевдооперацией SET параметру <имя> присваивается значение выражения <EXP>. Параметр <имя> может быть меткой любого типа и переменной и использоваться в дальнейшем в выражениях.

После параметра <имя> двоеточия стоять не должны. Если выражение <EXP> внешнее ((EXTERNAL), то генерируется ошибка.

Псевдооперация SET в режиме .Z80 использована быть не может, поскольку SET является кодом операции набора команд микропроцессора типа .Z80.

Псевдооперации ASET и DEFL могут быть использованы в обоих наборах команд.

Если параметр <имя> в дальнейшем должен быть определен заново, то вместо псевдооперации EQU должна быть использована одна из псевдоопераций типа SET.

Для повторного определения параметра <имя> может быть использована любая псевдооперация типа SET независимо от того, с помощью какой псевдооперации было произведено первоначальное определение (запрет на использование SET в режиме .Z80 этим не затрагивается ). Это есть противоположность псевдооперации EQU.

Пример: MARKE ASET BER+1000H

Когда имя MARKE используется в качестве выражения (операнда), Ассемблер анализирует выражение BER+1000H и подставляет его значение в качестве значения имени MARKE. В дальнейшем, если имя MARKE должно иметь какое-либо другое значение, то используется просто директива MARKE ASET с некоторым другим выражением.

MARKE ASET BER+1000H

MARKE ASET 3000TH

MARKE DEFL 6CDEH

**4.1.3. Псевдооперации задания режима присваивания адресов памяти**

Многие псевдооперации ссылаются на текущее значение счетчика адресов памяти.

Текущим значением счетчика адресов памяти является адрес следующего байта, используемого транслятором для генерации кода.

В Ассемблере метки и выражения получают присвоенный им режим.

Имеются 4 режима:

-абсолютный режим;

-режим относительно данных;

-режим относительно кодов;

-режим относительно общей области.

Если режим счетчика адресов памяти абсолютный, то счетчик содержит абсолютный адрес.

Если режим счетчика адресов памяти относительный, то счетчик содержит относительный адрес и может задействовать некоторое смещение для образования абсолютного стартового адреса перемещаемого сегмента, который может загружаться сборщиком.

Псевдооперация определения режима присваивания адресов памяти используется для типа части ассемблируемой программы.

**Определение абсолютного сегмента**

ASEG

Псевдооперация ASEG никогда не имеет операндов. При ней генерируются абсолютные неперемещаемые коды.

Псевдооперация ASEG устанавливает значение счетчика адресов памяти равное абсолютному адресу сегмента памяти текущему адресу. Стандартным значением является 0.

После псевдооперации ASEG директива ORG должна использоваться с аргументом 10зн или больше.

**Определение сегмента относительно кодов**

CSEG

Псевдооперация CSEG никогда не имеет операндов.

Коды транслируются в режиме относительно кодов (помечается апострофом (') после адреса) и могут быть загружены в постоянную память (ROM) или память с произвольным доступом (PROM).

Псевдооперация CSEG сбрасывает значение счетчика адресов памяти на сегмент памяти относительно кодов.

Присваивается то значение, которое было последним в режиме CSEG (стандарт 0), если после не использовалась директива ORG, которое изменяет присвоенные значения.

Однако, нужно обратить внимание на то, что директива ORG в режиме CSEG не устанавливает абсолютного адреса.

Директива ORG в режиме CSEG инициирует транслятор с ассемблера на прибавление к последнему значению адреса в режиме CSEG числа байтов, устанавливаемое аргументов директивы ORG.

Если, к примеру, задается директива ORG 50, то транслятор с Ассемблера прибавляет 50 байтов к текущему присвоенному адресу счетчика адресов памяти в режиме CSEG.

Действие следующей за псевдооперацией CSEG (или DSEG) директивы ORG таково, что этой директивой задается некоторое смещение.

Директива ORG не устанавливает абсолютного адреса для режима CSEG, и даже наоборот, режим CSEG сохраняет свое свойство перемещаемости.

Если требуется в режиме CSEG установить абсолютный адрес, то в сборщике используется ключ /Р.

Режим CSEG является стандартным режимом работы Ассемблера. Ассемблер начинает работу в этом режиме автоматически и устанавливает значение счетчика адресов памяти на ячейку 0 сегмента памяти относительно кодов. Все последующие директивы транслируются в сегмент памяти относительно кодов до тех пор, пока не будет отработана псевдооперация ASEG, DSEG или COMMON.

Псевдооперация CSEG далее применяется для того, чтобы Ассемблер мог возвратиться в режим относительно кодов, значение счетчика адресов памяти восстанавливается тогда в точку следующей свободной ячейки сегмента относительно кодов.

**Определение сегмента относительно данных**

DSEG

Псевдооперация DSEG никогда не имеет операндов.

Псевдооперацией DSEG задаются сегменты транслируемых и перемещаемых кодов (отмечаются кавычками (") после адреса), которые в дальнейшем могут быть загружены в память с произвольным доступом (RAM).

Псевдооперация DSEG устанавливает значение счетчика адресов памяти на сегмент памяти относительно данных.

Присваивается то значение, которое было последним в режиме DSEG (стандарт 0), если после не использовалась директива ORG, которая изменяет присвоенное значение.

Однако, нужно обратить внимание на то, что директива ORG в режиме DSEG не устанавливает абсолютного адреса.

Директива ORG в режиме DSEG инициирует Ассемблер на прибавление к последнему значению адреса в режиме DSEG числа байтов, установленное аргументом директивы ORG.

Если, к примеру, задается директива ORG 50, то Ассемблер прибавляет 50 байтов к текущему присвоенному адресу счетчика адресов памяти в режиме, которое будет его новым присвоенным значением в том же режиме.

Действие следующей за псевдооперацией DSEG (или CSEG) директивы ORG таково, что этой директивой задается некоторое смещение.

Директива ORG не устанавливает абсолютного адреса для режима DSEG, и даже наоборот, режим DSEG сохраняет свое свойство перемещаемости.

Если в режиме DSEG требуется установить абсолютный адрес, то в сборщике используется ключ /D.

**Определение блока общей области**

COMMON /[<имя блока>+/

Аргументом псевдооперации COMMON является имя общей области. Псевдооперация COMMON отводит некоторую общую область данных для всех блоков COMMON, которые известны программе.

Если <имя блока> опускается или состоит из символов пробела, то блок рассматривается как пустой блок.

Директивы COMMON являются неисполняемыми директивами резервирования памяти.

Директива COMMON присваивает некоторой общей области (отмеченной восклицательным знаком (!) После адреса) переменные, массивы и данные. Это дает возможность разделения различными программными модулями одной и той же области памяти.

Директивы, которые следуют за директивой COMMON, транслируются в область общей памяти под именем блока.

Длина общей области образуется из количества байтов, которое необходимо, чтобы поместить в этом блоке общей области определяемые переменные, массивы и данные. Блок COMMON завершается, если встречается какая-либо другая псевдооперация установки режима счетчика адресов памяти.

Общие блоки одного и того же имени могут иметь различные длины.

Если длины различны, то программный модуль с общим блоком наибольшей длины должен загружаться первым, т.е. Его имя должно быть первым в командной строке компоновщика программ.

Подробнее об этом смотрите в описании сборщика.

Директива COMMON устанавливает значение счетчика адресов памяти на выбранный общий блок памяти.

Пример:

COMMON /DATBIN/

BER2 EQU 100H

DB 0FFH

DW 1234H

DC 'WERK'

CSEG

**Установка абсолютного адреса**

ORG <EXP>

Значение счетчика адресов памяти может быть изменено в любой момент времени посредством использования директивы ORG. В режиме ASEG значение счетчика адресов памяти устанавливается такое, какое задается выражением <EXP>, и транслятор с Ассемблера присваивает генерируемые коды адресам, начиная с этого значения.

В режимах CSEG, DSEG и COMMON счетчик адресов памяти увеличивается в рамках текущего сегмента на величину, задаваемую выражением <EXP>, и Ассемблер присваивает генерируемые коды адресам, начиная с последнего присвоенного адреса сегмента плюс значение выражения <EXP>.

Все имена, использованные в выражении <EXP>, должны быть известны еще в первом проходе и его значение должно быть либо абсолютным, либо в той же области адресов, как и счетчик адресов памяти.

Пример: DSEG

ORG 50

Здесь счетчик адресов памяти в режиме относительно данных устанавливается на значение 50, относительно точки запуска сегмента памяти в режиме относительно данных. Этот метод дает свойство перемещаемости. Директива ORG <EXP> не задает в режимах CSEG и DSEG жесткого адреса, и более того, сборщик настраивает сегмент на перемещаемый адрес , который используется для всех включаемых в результирующую программу модулей.

С другой стороны какая-либо программа, начинающаяся с директив:

ASEG

ORG 800H

транслируется целиком в абсолютном режиме и загружается всегда с начального адреса 800Н. Но это не так, если директива ORG будет изменена в исходном файле.

Это означает, что следующая за ASEG директива ORG <EXP> настраивает сегмент на фиксированный адрес (в данном примере на абсолютный), который определяется выражением <EXP>.

Та же самая программа, оттранслированная в режиме относительно кодов и без директивы ORG, может быть загружена по любому адресу, если к цепочке команд сборщика добавляется ключ /P:<адрес>.

Смещение

.PHASE <EXP>

.DEPHASE

Псевдооперация .PHASE дает возможность загрузить код в некоторую область, но только в другой области отработать с указанного выражением <EXP> адреса.

Выражение <EXP> должно иметь абсолютное значение. Псевдооперация .DEPHASE используется для того, чтобы обозначить конец смещенного блока кодов.

Модуль внутри некоторого смещенного блока является абсолютным. Код загружается в эту абсолютную область по достижении

директивы .PHASE.

Код в пределах блока транспортируется после для выполнения по адресу, указанному выражением <EXP>.

Пример

.PHASE 100H

MARKE: CALL UPI

JP MARKEI

UPI: RET

.DERHASE

MARKE: JP 5

генерируется в:

.PHASE 100H

0100 CD 0106 MARKE: CALL UPI

0103 C3 0007 JP MARKEI

0106 C9 UPI: RET

.DERHASE

0007' C3 0005 MARKEI: JP 5

Смещаемый блок в пределах директив .PHASE ... DEPHASE представляет возможность выполнения блока кодов с некоторого заданного абсолютного адреса.

**4.1.4. Служебные псевдооперации**

Служебные псевдооперации выполняют следующие функции:

- вставку длинных комментариев;

- присваивание имени модулю;

- отметку окончания модуля;

- перенос других файлов в текущую программу.

**Директива комментария**

.COMMENT <огр> <текст> <огр>

Первый непустой символ, который находится после директивы .COMMENT , воспринимается как ограничитель. Следующий за ограничителем текст становится комментарием, который продолжается до следующего появления ограничителя <огр>. Текст может занимать много строк.

Директива .COMMENT используется для вставки длинных комментариев. Тогда нет необходимости ставить впереди точку с запятой для обозначения комментария.

Во время трансляции блок директивы .COMMENT игнорируется и не транслируется.

Пример

.COMMENT (здесь может быть вставлен любой многострочный

текст)

; (возврат к нормальной трансляции)

Конец программы

END [<EXP>+

Директива END отмечает конец программы и должна завершаться символом <ет>. Если директива END отсутствует, то выдается следующее предупреждающее сообщение:

"%NO END STATEMENT" (отсутствует директива END)

Выражение <ехр> может быть меткой любого типа, числом или любым другим действительным аргументом, который может быть обработан сборщиком как точка запуска программы.

Если выражение <ехр> было задано, то сборщик записывает по адресу 100н команду перехода на адрес, заданный выражением <ехр>.

Если выражение <ехр> не было задано, то сборщику для этой программы не будет передан никакой адрес запуска, выполнение будет начинаться с первого обработанного модуля.

**Директива INCLUDE (включение)**

INCLUDE <имя файла>

$INCLUDE <имя файла>

MACLIB <имя файла>

Все три псевдооперации родственны по смыслу.

Во время трансляции псевдооперациями INCLUDE добавляют исходные коды из других каких-либо исходных файлов в текущий транслируемый файл.

Посредством использования псевдооперации INCLUDE становится необязательным повторное записывание часто используемых последовательностей директив в текущей программе.

Параметром <имя файла> является любая действительная в операционной системе спецификация файла, имя файла должно записываться большими буквами.

Включаемый файл (INCLUDE) открывается и транслируется в текущий файл непосредственно за директивой INCLUDE. Если достигнут конец файла, то Ассемблер продолжает трансляцию с директивы, следующей за директивой INCLUDE.

Вложенные директивы INCLUDE не разрешены. В случае, если это имеет место, то генерируется синтаксическая ошибка объектного кода типа "0".

Файл, указанный в поле операнда директивы INCLUDE, должен существовать. В противном случае генерируется ошибка типа "V" и директива INCLUDE игнорируется.

В листинге протокола трансляции между оттранслированным кодом и исходной строкой печатается буква "С" (см. п. 5.3.).

Имя модуля

NAME ('<имя модуля>')

Параметр <имя модуля> определяет имя модуля. Круглые скобки и апострофы, в которые заключается имя модуля, обязательны.

Значащими являются только первые шесть символов имени модуля.

Имя модуля может быть также определено при помощи псевдо операции TITLE.

Если нет ни одной из псевдоопераций NAME или TITLE, то имя модуля генерируется из имени исходного файла.

Директива .RADIX (определение базы чисел)

.RADIX <EXP>

Выражение <ехр> в директиве .RADIX всегда является десятичной числовой константой, независимо от текущей базы чисел. Директива .RADIX позволяет заменить стандартную базу чисел на любую базу от 2 до 16.

Директива .RADIX не изменяет базу чисел в листинге, а только дает возможность ввода числовых величин в требуемой базе чисел без специального способа записи. (Величины с другими базами чисел требуют специального способа записи, это поясняется в разделе "операнды").

Генерируемые коды всегда представлены в шестнадцатеричном виде.

Пример:

DEC: DB 20

.RADIX 2

BIN DB 00011110

.RADIX 16

HEX DB 0CF

.RADIX 8

OCT: DB 73

.RADIX 10

DESI: DB 16

HEXA: DB 0CH

генерируется в:

0000' 14 DEC: DB 20

0002 .RADIX 2

0001' 1E BIN: DB 00011110

0010 .RADIX 16

0002' CF HEX: DB 0CF

0008 .RADIX 8

0003' 3B OCT: DB 73

000A .RADIX 10

0004' 10 DECI: DB 16

0005' 0C HEXA: DB 0CH

**Запрос**

.REQUEST <имя файла>[,<имя файла>...+

Директива .REQUEST выставляет к сборщику требование на просмотр находящихся в таблице имен файлов на предмет наличия неопределенных внешних меток (внешних меток, для которых до настоящего момента не была обработана ссылочная метка типа PUBLIC).

Найденные сборщиком неопределенные метки требуются для того, чтобы можно было загрузить один или несколько программных модулей для полной отработки сборщика.

Имена файлов в таблице должны состоять из действительных символов и не содержать информации о типе файла и назначении устройств.

Сборщик предполагает стандартный тип файла .REL и текущий дисковод.

Пример:

.

.

.REQUEST SUBR1

.

.

Здесь сборщик просматривает SUBR1 на предмет наличия внешних меток, для которых не определена ни одна ссылочная метка типа PUBLIC в уже обработанных сборщиком модулях.

**4.1.5. Псевдооперации управления печатью листинга**

Псевдооперации управления печатью листинга выполняют две общие функции:

- управление форматом;

- управление печатью листинга.

Псевдооперации управления форматом позволяют программисту производить смену страниц и вставку заголовков.

Протоколирование общих ассемблерных файлов или их частей включается и выключается с помощью псевдоопераций управления печатью листинга.

**Псевдооперации управления форматом**

Эти псевдооперации позволяют включать перевод страницы и указание титульных и подтитульных строк в листинге транслятора с Ассемблера.

Перевод страницы

\*EJECT [<EXP>+

PAGE [<EXP>+

OEJECT [<EXP>+

Псевдооперация перевода страницы задает Ассемблеру вывод с новой страницы.

Ассемблер вставляет в файл листинга в конец страницы управляющие символы перевода страницы.

Значение выражения <ехр>, если оно задано, определяет размер новой страницы (измеряемый в числе строк на страницу), который должен быть в пределах между 10 и 255. Стандартное число строк в странице равно 50.

Директива \*EJECT должна начинаться с первой колонки строки.

Пример: \*EJECT 72

Здесь Ассемблер инициирует принтер начать печатать с новой страницы и на каждой странице печатать72 строки программы.

(Этот параметр следует выбирать при длине бумаги 12 дюймов (1 дюйм = 2,54 см) и печати 6 строк на дюйм).

Заголовок

TITLE <текст>

Директива TITLE специфицирует строку заголовка, которая печатается на каждой странице первой.

Если задается более одной директивы TITLE, то генерируется ошибка типа "Q".

Если не было использовано директивы NAME, то первые шесть символов заголовка используются в качестве имени модуля.

Если нет ни директивы NAME, ни директивы TITLE, то имя модуля берется из имени исходного файла.

Пример: TITLE PROG1

Здесь модуль получает имя PROG1. Под этим именем он может быть вызван и PROG1 печатается в качестве заголовка на каждой странице листинга.

Подзаголовок

SUBTTL <текст>

Директива SUBTTL специфицирует подзаголовок, который должен печататься на каждой странице листинга в строке под заголовком.

Последовательность символов <текст> обрывается, если ее длина достигает 60 символов.

В каждой программе может быть задано любое количество подзаголовков.

Если Ассемблер находит директиву SUBTTL, то старый текст подзаголовка заменяется новым.

Чтобы выключить действие директивы SUBTTL на некоторое время вывода части листинга, нужно указать директиву SUBTTL с цепочкой пробелов в качестве текста.

Пример:

SUBTTL подпрограммы ввода/вывода

.

.

SUBTTL

.

.

Здесь первая директива SUBTTL инициирует печать фразы "подпрограммы ввода/вывода в начале каждой страницы. Вторая директива SUBTTL выключает печать подзаголовка.

**Общие псевдооперации управления печатью листинга**

.LIST печать всех строк листинга с их кодами

.XLIST подавление печати строк листинга

Стандартным является действие директивы .LIST.

Если в командной строке Ассемблера был указан файл листинга, то этот файл заполняется.

Если в исходном файле встречается директива .XLIST, то в файл листинга не передаются ни исходные , ни объектные коды.

Действие директивы .XLIST продолжается до появления директивы .LIST.

Директива .XLIST отменяет действие всех остальных псевдоопераций управления печатью листинга (кроме директивы .LIST).

Пример:

.

.

.XLIST ; здесь печать строк обрывается

.

.

.LIST ; здесь печать строк возобновляется

.

**Вывод сообщения на экран дисплея**

.PRINTX огр текст огр

Первый символ после директивы .PRINTX, не являющийся пробелом, - это ограничитель <огр>.

Текст, следующий от одного ограничителя до другого, соответственно до символа конца строки, выводится во время трансляции на экран дисплея.

Директива .PRINTX полезна для отслеживания процесса ассемблирования больших программ или для индикации значений ключей условного ассемблирования.

Директива .PRINTX действует в обоих проходах транслятора.

Если требуется вывод информации на экран дисплея только в одном из проходов Ассемблера, то для определения номера прохода используют псевдооперацию IF1 или иF2 (см. п. 4.3.).

Пример: . PRINTX \*UEBERSETZUNG ZUR HAELFTE FERTIG

(трансляция наполовину завершена)

Здесь Ассемблер выдает это сообщение дважды на экран дисплея при появлении директивы в проходе 1 и 2.

IF1

.PRINTX \*PASS 1 BEENDET\* ; MELDUNG NUR IM PASS 1

ENDIF (сообщение только в

IF2 проходе 1)

.PRINTX \*PASS 2 BEENDET\* ; MELDUNG NUR IM RASS 2

ENDIF (сообщение только в

проходе 2)

Здесь в обоих проходах каждый раз выводится одно соответствующее сообщение.

**Псевдооперации условного ассемблирования**

С помощью этих псевдоопераций возможно управление печатью в листинге условно ассемблируемых частей программы, при которых условие не выполнено.

В этих частях программы в фазах REL объектный код не генерируется. Для этого существует внутренний флаг условия FCOND.

Если этот флаг имеет значение ON, то данные части программы в листинге пропускают, иначе (значение равно OFF) пропуск этих частей программы подавляется.

Для управления этим флагом FCOND могут быть использованы три следующие псевдооперации:

.SFCOND

.LFCOND

.TFCOND

смотри также описание ключа /Х в разделе "ключи"(п. 5.2.4.).

**Подавление неверных условий**

.SFCOND

Псевдооперацией .SFCOND флаг FCOND принудительно устанавливается в значение ON.

**Печать**

.LFCOND

Псевдооперацией .LFCOND флаг FCOND принудительно устанавливается в значение OFF.

Смена неверных условий управления печатью листинга

.TFCOND

Псевдооперацией .TFCOND флаг FCOND устанавливается в зависимости от стандартного флага условий STFCOND. .TFCOND меняет значение флага STFCOND и присваивает его новое значение флагу FCOND.

Индикация флага FCOND в начале программы производится посредством присваивания ему значения флага STFCOND.

Значение инициации флага STFCOND зависит от способа вызова Ассемблера. обычно это значение равно ON. при указании в вызове ключа /Х это значение равно OFF.

|  |  |  |
| --- | --- | --- |
| Псевдооперация | Нет /Х | Есть /Х |
|  | ON | OFF |
| .SFCOND | OFF | OFF |
| .LFCOND | ON | ON |
| .TFCOND | OFF | ON |
| .TFCOND | ON | OFF |
| .SFCOND | OFF | OFF |
| .TFCOND | OFF | ON |
| .TFCOND | ON | OFF |
| .TFCOND | OFF | ON |

**Псевдооперации управления печатью макрорасширений**

Псевдооперации управления печатью макрорасширений управляют печатью строк макрокоманд или псевдоопераций повторения (REPT, IRP, IRPC) и могут быть использованы только внутри какой-либо макрокоманды или повторяемого блока.

**Исключение макрострок, не генерирующих код**

.XALL

Директива .XALL является стандартной. при директиве .XALL печатаются все исходные и объектные коды, которые генерируются макрокомандой.

Исходные строки, которые не генерируют кода, не печатаются.

**Печать текста макрокоманд**

.LALL

При задании директивы .LALL печатается полный текст макрорасширения включая строки, которые не генерируют никакого кода.

**Подавление печати текста макрокоманд**

.SALL

При задании директивы .SALL печать полного текста макрокоманды и генерируемого ей кода подавляется.

Для этих трех псевдоопераций следующие примеры:

MAC MACRO X,Y

Y1 DEFL Y

; строка комментария

LD X,Y1

ENDM

Здесь при вызове MAC B,4 и при использовании соответствующих псевдоопераций генерируется следующий текст и объектные коды:

.XALL

MAC B,4

0000' 06 04 + LD B,Y1

.LALL

MAC B,4

0004 + Y1 DEFL 4

; строка комментария

0002' 06 04 + LD B,Y1

.SALL

+ MAC B,4

**Псевдооперации управления перекрестными ссылками**

Имеется возможность генерации перекрестных ссылок только для некоторой части программы, а не для всей программы. Для этого используются псевдооперации .CREF и .XCREF в исходном файле для транслятора с Ассемблера.

Обе эти псевдооперации могут быть использованы в поле операции на любом месте программы.

Как и все псевдооперации управления печатью листинга они аргументов не имеют.

**Подавление перекрестных ссылок**

.XCREF

Директива .XCREF выключает действие директивы .CREF, являющееся стандартной установкой. Действие директивы .XCREF продолжается до тех пор, пока транслятор с Ассемблера не встретит директиву .CREF.

Директива .XCREF используется для того, чтобы подавить в некоторой выбранной части файлов генерацию перекрестных ссылок.

Поскольку не действуют ни директива .CREF ни директива .XCREF, если в командной строке транслятора с Ассемблера не задается ключ /С, то директива .XCREF не должна быть использована, если требуется нормальный листинг (без перекрестных ссылок). то же самое достигается и тогда, когда в командной строке ключ /с опускается.

**Генерация перекрестных ссылок**

.CREF

Действие директивы .CREF является стандартным.

Директива .CREF используется для того, чтобы возобновить генерацию перекрестных ссылок после того, как она была заблокирована директивой .XCREF. директива .CREF действует до тех пор, пока не встретится директива .XCREF.

Директива .CREF только тогда действует, если в командной строке Ассемблера указан ключ /С.

**4.2. Макросредства**

Макросредства позволяют записывать блоки и директивы, которые должны быть использованы повторно, без повторной их записи.

Эти блоки директив начинаются либо с псевдооперации макросредств, либо с псевдооперации повторения. Они завершаются псевдооперацией ENDM .3 в пределах блока макрокоманды могут быть использованы все псевдооперации макрокоманд.

Глубина вложения макрокоманд ограничивается только размерами памяти.

Макросредства Ассемблера включают следующие псевдооперации, которые описываются ниже:

- макроопределение:

MACRO

- повторения:

REPT (повторение)

IPR (неопределенное повторение)

IPRC (неопределенное повторение с символом)

- завершение:

ENDM

EXITM

-метки, встречающиеся внутри блока макрокоманды только

один раз:

LOCAL

Макросредства поддерживаются, кроме того, и некоторыми следующими специальными макрооперациями:

& ;; ! %

**4.2.1. Псевдооперации макроопределений**

<имя> MACRO <мним.пар.> [,<мним. пар.> ... ╜

.

.

.

.

ENDM

Блок директив в пределах от MACRO до ENDM образует ядро макрокоманды или макроопределение.

При этом параметр <имя> является меткой и подчиняется всем правилам образования меток. параметр <имя> может иметь любую длину, но сборщику передаются только первые 16 символов.

После того, как макрокоманда определена, параметр <имя> может быть использован для ее вызова.

Параметр <мним. пар.> является символическим параметром, который заменяется реальным параметром посредством замены его текста при вызове макрокоманды.

Каждый мнимый параметр <мним. пар.> может иметь длину 32 символа, количество символических параметров ограничено только длиной строки. символические параметры отделяются друг от друга запятыми.

Ассемблер интерпретирует все символы между двумя запятыми как один символический параметр.

Примечание:

Некоторый символический параметр распознается исключительно как таковой.

Если, к примеру, какое-либо имя регистра а или в было использовано как символический параметр, то во время раскрутки макрокоманды оно будет заменено параметром.

Блок макрокоманд будет транслироваться не при его появлении, а только тогда, когда он вызывается, транслятор с Ассемблера раскручивает вызов макрокоманды в блок директив.

Если для части программы, где появляется блок макрокоманды, используются псевдооперации TITLE, SUBTTL или NAME, то следует быть осторожным при написании этой строки директивы.

Пример: SUBTTL MACRO DEFINITION

Здесь Ассемблер транслирует директиву как макроопределение с именем SUBTTL, а DEFINITION как символический параметр. Чтобы избежать этого, можно было бы видоизменить слово MACRO.

**Макровызов**

Если требуется использовать какую-либо макрокоманду, то для этого нужно записать директиву макровызова.

<имя> <параметр>[,<параметр>...

Параметр имя такой же, что и в блоке макрокоманды. Параметры в макровызове заменяют символические параметры в макроопределении. Количество параметров ограничивается только длиной строки. Параметры должны отделяться друг от друга запятыми.

Если параметры, разделенные запятыми, заключены в угловые скобки, то Ассемблер рассматривает все, что стоит в этих скобках, как один отдельный параметр.

Пример:

Мас 1,2,3,4,5

Здесь в макрокоманду передаются 5 параметров.

Мас <1,2,3,4,5>

Здесь в макрокоманду передается только один параметр.

Количество параметров в директиве макровызова не обязательно должно совпадать с количеством символических параметров в макроопределении.

Если макровызов содержит больше параметров, чем предусмотрено в макроопределении, то лишние параметры игнорируются. Если их меньше, то остальные параметры устанавливаются в нуль.

При каждой директиве макровызова блок макрокоманды вставляется в транслируемый текст.

Пример:

EXCHNG MACRO X,Y

PUSH X

RUSH Y

POP X

POP Y

ENDM

В следующей программе используется макровызов:

LD A,(2FH)

LD H,A

LD A,(3FH)

LD D,A

EXCHNG HL,DE

Транслятор из этой программы генерирует следующий код:

0000' 3A 002F LD A,(2FH)

0003' 67 LD H,A

0004' 3A 003F LD A,(3FH)

0007' 57 LD D,A

EXCHNG HL,DE

0008' E5 + PUSH HL

0009' D5 + PUSH DE

000A' E1 + POP HL

000B' D1 + POP DE

**4.2.2. Псевдооперации повторения**

Эти псевдооперации позволяют повторять операции в одном кодовом блоке столько раз, сколько указано.

Главные различия между псевдооперациями повторения и псевдооперациями макросредств следующие:

1. Директива MACRO задает блоку имя, под которым он может быть включен в результирующий код при макровызове. Макрокоманда может быть использована во многих различных программах с помощью простой подстановки директивы макровызова.

2. Директива MACRO дает возможность передачи параметров блоку макрокоманды, когда он вызывается; эти параметры могут изменяться.

Псевдооперации повторения должны принадлежать кодовому блоку в качестве его составной части.

Использование псевдоопераций повторения удобно тогда, когда параметры известны заранее и не изменяются и когда повторение должно быть реализовано при каждом выполнении программы.

С помощью псевдооперации макросредств макрокоманда вызывается каждый раз, когда это требуется!

Обратите внимание, что каждый блок повторения должен оканчиваться псевдооперацией ENDM.

Повторение

REPT <EXP>

.

.

ENDM

Такая запись задает повторение блока директив, заключенных между псевдооперациями REPT и ENDM, число раз, задаваемое выражение <ехр>, причем <ехр> представляет собой беззнаковое 16-разрядное число.

Если выражение <ехр> содержит внешнюю метку или неопределенные операнды, то генерируется ошибка.

Пример:

X DEFL 0

REPT 10

X DEFL X+1

DB X

ENDM

Генерируется в следующий листинг ассемблирования:

0000 X DEFL 0

REPT 10

X DEFL X+1

REPT 10

ENDM

0000' 01 + DB X

0001' 02 + DB X

0002' 03 + DB X

0003' 04 + DB X

0004' 05 + DB X

0005' 06 + DB X

0006' 07 + DB X

0007' 08 + DB X

0008' 09 + DB X

0009' 0A + DB X

Неопределенное повторение

IRP <мним. пар.>,<параметр>

.

.

ENDM

Параметры должны быть заключены в угловые скобки!

Параметрами могут быть действительные метки, символьные цепочки, числа или символьные константы.

Блок директив повторяется для каждого параметра. В каждом повторении символический параметр заменяется очередным следующим параметром в блоке.

Если какой-либо параметр равен нулю , то блок воспроизводится один раз с этим параметром.

Пример:

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

DB X

ENDM

В данном примере генерируется тот же код (DB 1 -:- DB 10), что и в предыдущем при использовании директивы REPT.

Если директива IRP используется внутри блока макроопределения, то угловые скобки, в которые заключаются параметры, в директиве макровызова устраняются перед тем, как эти параметры будут переданы в блок макрокоманды.

Пример, в котором генерируется тот же код, что и выше, наглядно демонстрирует устранение скобок при параметрах:

MAC MACRO X

IRP Y,<X>

DB Y

ENDM

Если транслируется директива макровызова:

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

то макрорасширение выглядит следующим образом:

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

DB Y

ENDM

Угловые скобки при параметрах здесь устранены и содержимое передается как один единственный параметр.

**Неопределенное повторение с символами**

IRPC мним. пар. , строка

.

.

ENDM

Директивы блока повторяются для каждого символа символьной цепочки.

В каждом повторении символический параметр в блоке заменяется очередным символом символьной цепочки.

Пример:

IRPC X,0123456789

DB X+1

ENDM

В этом примере генерируется тот же самый код (DB 1 -:- DB 10), что и в двух предыдущих примерах.

**4.2.3. Псевдооперации завершения**

**Конец макрокоманды**

ENDM

Директива ENDM сообщает Ассемблеру, что блок макрокоманды или повторения завершен.

Каждая псевдооперация MACRO, REPT, IRP, IRPC должна завершаться соответствующей псевдооперацией ENDM. Если это не имеет место, то в конце каждого прохода Ассемблером генерируется сообщение:

UNTERMINATED REPT/IRP/IRPC/MACRO

(нет завершения REPT/IRP/IRPC/MACRO)

Лишняя директива ENDM является причиной ошибки типа "О".

Если требуется произвести выход из макрокоманды или блока повторения до завершения макрорасширения, то для этого используется псевдооперация EXITM.

**Выход из макрокоманды**

EXITM

Директива EXITM используется внутри блока макрокоманды или повторения для досрочного завершения макрорасширения, когда выполнение какого-либо условия делает дальнейшую генерацию макрорасширения ненужной или нежелательной. Чаще всего директива EXITM используется в сочетании с какой-нибудь условной псевдооперацией.

Когда транслируется директива EXITM, то генерация макрорасширения прекращается. остаток макрорасширения или повторения не генерируется.

Если блок, который содержит директиву EXITM, вложен в какой-либо другой блок, то генерация макрорасширения продолжается на внешнем уровне.

Пример:

MAC MACRO X

Y DEFL 0

REPT X

Y DEFL Y+1

IFE Y-0FFH ; TEST Y

EXITM ; WENN 0, VERLASSEN REPT

ENDIF

DB Y

ENDM

ENDM

**4.2.4. Псевдооперация макрометок**

LOCAL <мним. пар.>[,<мним. пар.> ... ╜

Псевдооперация LOCAL используется только в пределах блока макроопределения.

Если директива LOCAL используется, то Макроассемблер создает однозначную метку для каждого параметра мним. пар. и заменяет каждый символический параметр в макрорасширении на эту метку.

Эти однозначные метки чаще всего используются для определения меток внутри макрокоманд, таким образом избегается многократное определение меток при многократном использовании макрокоманды.

Создаваемые Ассемблером метки находятся в диапазоне от ..0001 до ..FFFF. пользователю следует избегать применения меток вида ..NNNN ! Директива LOCAL должна предшествовать всем остальным директивам в макроопределении.

Пример:

MAC MACRO NUM,Y

LOCAL A,B,C,D,E

A: DB 7

B: DB 8

C: DB Y

D: DB Y+1

E: DW NUM+1

JP A

ENDM

Вызов MAC 0C00H,0BEH генерирует следующий листинг ассемблирования:

0000' 07 + ..0000: DB 7

0001' 08 + ..0001: DB 8

0002' BE + ..0002: DB 0BEH

0003' BF + ..0003: DB 0BEH+1

0004' 0C01 + ..0004: DW 0C00H+1

0006' C3 0000' + JP ..0000

**4.2.5. Специальные макрооператоры**

Специальные операторы могут быть использованы в блоке макрокоманды для выбора дополнительных функций Ассемблера.

Амперсанд &

Амперсанд связывает текст или метки в цепочку. Этот символ нельзя использовать в директиве макровызова.

Символический параметр в макрорасширении не заменяется реальным параметром, если перед ним не стоит символ &.

Метка образуется из текста метки в макроопределении и символического параметра, между которыми ставится символ &.

Пример:

ERRGEN MACRO X

ERROR&X: PUSH BC

LD B,'&X'

JP ERROR&X

ENDM

Вызов ERRGEN A генерирует в:

0000' C5 + ERROR&A: PUSH BC

0001' 06 41 + LD B,'A'

0003' C3 0000' + JP ERROR&A

**Двойная точка с запятой ;;**

Комментарий, которому предшествуют две точки с запятой, не содержится в блоке как часть макрорасширения (при задании директивы .LALL в листинге он не печатается).

Комментарий после точки с запятой входит в состав макрорасширения и печатается.

**Восклицательный знак !**

Восклицательный знак может быть записан в каком-либо аргументе, чтобы указать, что следующий по порядку символ должен рассматриваться как литерал. Поэтому символы !; означают то же самое, что и <;>.

**Процент %**

Символ процента используется только в макроаргументах для преобразования выражения, следующего за символом % (чаще всего это метка), в некоторое число по текущей базе чисел (установленной директивой .RADIX).

Во время генерации макрорасширения это число заменяет символический параметр.

Использование оператора % допускается в макровызове вместе с каким-либо числом.

Часто макровызов бывает вызов со ссылкой, при этом текст макроаргумента однозначно заменяет символический параметр.

Выражение, следующее за символом % должно соответствовать тем же правилам, что и при псевдооперации DS. Это значит, что требуется действительное выражение, результатом которого будет абсолютная (неперемещаемая) константа.

Пример:

PRINTE MACRO MSG,N

.PRINTX \*MSG,N\*

ENDM

SYM1 EQU 100

SYM2 EQU 200

PRINTE SYM1+SYM2= ,% SYM1+SYM2

В обычной ситуации при макровызове символический параметр N был бы заменен на цепочку (SYM1+SYM2).Результат был бы следующим:

PRINTX\*SYM1+SYM2=,(SYM1+SYM2)\*

Если перед параметром стоит символ % , то генерируется следующее:

PRINTX\*SYM1+SYM2=,300\*

**4.3. Псевдооперации условного ассемблирования**

Псевдооперации условного ассемблирования дают пользователю возможность создавать кодовые блоки, в которых проверяются специальные условия и которые генерируются в соответствии с этими условиями.

Все псевдооперации условного ассемблирования имеют следующий формат:

|  |  |
| --- | --- |
| IFXXXX[аргумент] | COND [аргумент] |
| . | . |
| . | . |
| . | . |
| [ELSE] | [ELSE] |
| . | . |
| . | . |
| . | . |
| ENDIF | ENDC |

Для завершения условия каждой директиве IFXXXX должна соответствовать одна директива ENDIF.

Каждое условие по директиве COND должно завершаться соответствующей директивой ENDC. В противном случае в конце каждого прохода будет генерироваться сообщение:

UNTERMINATED CONDITIONAL

(нет завершения условия)

Директива ENDIF без соответствующей директивы IF или директива ENDC без соответствующей директивы COND вызывают ошибку типа "с".

Ассемблер анализирует директиву условия на предмет выполнения условия "истина", (значение равно FFFFH, или -1, или любое значение, неравное нулю) или на предмет его невыполнения "ложь",(значение равно нулю).

Если значение соответствует условию, которое определено в директиве условия, то код внутри блока условия транслируется.

Если значение условию не соответствует, то Ассемблер игнорирует блок условия либо полностью, либо транслирует только ту его часть, которая содержится под альтернативной директивой ELSE (иначе).

Условия могут быть вложенными до 255 раз. Каждый аргумент условия должен быть известен в первом проходе, чтобы избежать ошибок типа "V" или некорректного анализа. выражение для директив IF/IFT/COND и IFF/IFE должно иметь значение, которое было ранее определено и которое абсолютно.

Если после IFDEF или IFNDEF определено имя, то Ассемблер в первом проходе рассматривает его как неопределенное, но оно определяется во втором проходе.

Каждый блок условия может содержать альтернативную псевдооперацию ELSE, которая дает возможность генерации альтернативного кода, если выполняется противоположное условие.

Для директив IFXXXX/COND разрешается использование только одной директивы ELSE.

Директива ELSE всегда связана с последней открытой директивой IF. Условие с числом директив ELSE, большим чем одна, или директива ELSE без какого-либо условия вызывает ошибку типа "С".

**Псевдооперации условий**

|  |  |
| --- | --- |
| IF <EXP> | - директивы внутри блока условия транслируются, если результат выражения <ехр> не равен нулю. |
| IFT <EXP> |
| COND <EXP> |
| IFE <EXP> | - директивы внутри блока условия транслируются, если результат выражения <ехр> равен нулю. |
| IFF <EXP> |
|  |
| IF1 | - директивы внутри блока условия транслируются, если Ассемблер находится в первом проходе. |
|  |
|  |
| IF2 | - директивы внутри блока условия транслируются, если Ассемблер находится во втором проходе. |
|  |
|  |
| IFDEF <метка> | - директивы внутри блока условия транслируются, если метка определена или она объявлена как  внешняя (EXTERNAL). |
|  |
|  |
|  |
| IFNDEF <метка> | - директивы внутри блока условия транслируются, если метка не определена или не объявлена как внешняя (EXTERNAL). |
| IFB <Арг> | - угловые скобки при <Арг> обязательны. директивы внутри блока условия транслируются, если аргумент пустой (не задан) или нулевой (<>). |
|  |
|  |
|  |
|  |
| IFNB <Арг> | - угловые скобки при <Арг> обязательны. Директивы внутри блока условия транслируются, если аргумент не пустой. Такая запись не используется для проверки символических пара метров. |
|  |
|  |
|  |
|  |
|  |
|  |
| IFIDN <<Арг1>>,<<Арг2>> | - угловые скобки при <Арг1> и <Арг2> обязательны. Директивы внутри блока условия транслируются, если цепочка <Арг1> идентична цепочке <Арг2>. |
|  |
|  |
|  |
|  |
| IFDIF <<Арг1>>,<<Арг2>> | - угловые скобки при <Арг1> и <Арг2> обязательны. Директивы внутри блока условия транслируются, если цепочка <Арг1> отлична от цепочки <Арг2>. |
|  |
|  |
|  |
|  |
| ELSE | - директива ELSE предоставляет возможность генерации альтернативного кода, если выполняется  противоположное условие. Директива ELSE может быть использована с любой псевдооперацией условия. |
|  |
|  |
|  |
|  |
|  |
|  |  |
| ENDIF  ENDC | - эти псевдооперации завершают блок условия. они должны быть использованы в каждой псевдооперации условия.  Директива ENDIF должна иметь соответствующую директиву IFXXXX, а директива ENDC - директиву COND. |
|  |
|  |
|  |
|  |

**6. Описание команд**

В настоящей главе описывается действие отдельных команд центрального процессора.

Раздел 6.16. Содержит перечень сокращений. В разделе 6.17. Описана работа с битами условия (флажки).

В описании принципа действия команд действительно следующее:

(HL) : Ячейка памяти, адресуемая парой регистров HL. При

этом регистр L содержит восемь младших битов, а

регистр н содержит восемь старших битов адреса.

(IX+D) : Ячейка памяти, адресуемая индексным регистром IX

плюс сдвиг D.

(IY+D) : Ячейка памяти, адресуемая индексным регистром IY

плюс сдвиг D.

**6.1. Команды загрузки**

Команды загрузки осуществляют внутреннюю передачу данных между регистрами центрального процессора или между регистрами центрального процессора и памятью записи/считывания (RAM - память с произвольной выборкой). Команды должны содержать исходный адрес, с которого передаются данные, и выходной адрес.

Исходная ячейка памяти не изменяется в результате команды загрузки.

**6.1.1. 8-разрядные команды загрузки**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| MOV R,R' | LD R,R' | Содержимое регистра R' перезаписывается в регистр R. |
| MVI R,N | LD R,N | Прямой операнд N загружается в регистр R. |
| MOV R,M | LD R,(HL) | Содержимое ячейки памяти м или (HL) |
|  |  | загружается в регистр R. |
|  |  |  |
| --- | LD R,(IX+D) | Содержимое ячейки памяти (IX+D) заг- |
|  |  | ружается в регистр R. |
|  |  |  |
| --- | LD R,(IY+D) | Содержимое регистра памяти (IY+D) |
|  |  | загружается в регистр R. |
|  |  |  |
| MOV M,R | LD (HL),R | Переносит один байт из регистра R |
|  |  | в ячейку памяти м или (HL). |
|  |  |  |
| --- | LD (IX+D),R | Переносит один байт из регистра R |
|  |  | в ячейку памяти (IX+D). |
|  |  |  |
| --- | LD (IY+D) | Переносит один байт из регистра |
|  |  | в ячейку памяти (IY+D). |
|  |  |  |
| MVI M,N | LD (HL),N | Вызывает передачу прямых операндов, |
|  |  | определенных параметром N, в ячей- |
|  |  | ку памяти м или (HL). |
|  |  |  |
| --- | LD (IX+D),N | Вызывает передачу прямых операндов, |
|  |  | определенных параметром N, в ячей- |
|  |  | ку памяти (IX+D). |
|  |  |  |
| --- | LD (IY+D),N | Вызывает передачу прямых операндов, |
|  |  | определенных параметром N, в ячей- |
|  |  | ку памяти (IY+D) |
| LDAX B | LD A,(BC) | Содержимое ячейки памяти, адресован- |
|  |  | ной парой регистров вс, загружается |
|  |  | в аккумулятор. |
|  |  | Регистр с содержит восемь младших |
|  |  | битов, регистр в содержит восемь |
|  |  | старших битов адреса. |
|  |  |  |
| LDAX D | LD A,(DE) | Содержимое ячейки памяти, адресован- |
|  |  | ной парой регистров DE, загружается |
|  |  | в аккумулятор. |
|  |  | Регистр E содержит восемь младших |
|  |  | битов, регистр D содержит восемь |
|  |  | старших битов адреса. |
|  |  |  |
| LDA NN | LD A,(NN- | Содержимое ячейки памяти, адресован- |
|  |  | ной в параметре NN, загружается в |
|  |  | аккумулятор. |
|  |  |  |
| STA NN | LD (NN),A | Содержимое аккумулятора загружается |
|  |  | в ячейку памяти, адресованную в па- |
|  |  | раметре NN. |
|  |  |  |
| STAX B | LD (BC),A | Содержимое аккумулятора загружается |
|  |  | в ячейку памяти, адрес которой опре- |
|  |  | делен в паре регистров ВС. |
|  |  | Регистр с содержит восемь младших |
|  |  | битов, регистр в содержит восемь |
|  |  | старших битов адреса. |
|  |  |  |
| STAX D | LD (DE),A | Содержимое аккумулятора загружается |
|  |  | в ячейку памяти, адрес которой опре- |
|  |  | делен в паре регистров DE. |
|  |  | Регистр е содержит восемь младших |
|  |  | битов, регистр D содержит восемь |
|  |  | старших битов адреса. |
|  |  |  |
| --- | LD A,I | Содержимое регистра прерываний I |
|  |  | загружается в аккумулятор. |
|  |  |  |
| --- | LD..A,R | Содержимое регистра обновления R |
|  |  | загружается в аккумулятор. |
|  |  |  |
| --- | LD I,A | Содержимое аккумулятора загружается |
|  |  | в регистр прерываний I. |
|  |  |  |
| --- | LD R,A | Содержимое аккумулятора загружается |
|  |  | в регистр обновления R. |
|  |  |  |
| LXI DD,NN | LD DD,NN | 16-Разрядный прямой операнд загружа- |
|  |  | ется в двойной регистр DD. |
|  |  |  |
| --- | LD IX,NN | 16-Разрядный прямой операнд загружа- |
|  |  | ется в индексный регистр IX. |
|  |  |  |
| --- | LD IY,NN | 16-Разрядный прямой операнд загру- |
|  |  | жается в индексный регистр IY. |
|  |  |  |
| LHLD NN | LD HL,(NN- | Содержимое ячеек памяти, адресован- |
|  |  | ных в параметрах NN и NN+1, заг- |
|  |  | ружается в двойной регистр HL: |
|  |  | содержимое NN+1 ---> регистр Н, |
|  |  | содержимое NN ---> регистр L. |
|  |  |  |
| --- | LD DD,(NN) | Содержимое ячеек памяти, адресован- |
|  |  | ных в параметрах NN и NN+1, заг- |
|  |  | ружается в двойной регистр DD: |
|  |  | Содержимое NN+1 ---> старший ре- |
|  |  | гистр |
|  |  | (B,D,SPH); |
|  |  | Содержимое NN ---> младший ре- |
|  |  | гистр |
|  |  | (C,E,SPL). |
| --- | LD IX,(NN) | Содержимое ячеек памяти, адресован- |
|  |  | ных в параметрах NN и NN+1, за- |
|  |  | гружается в индексный регистр IX: |
|  |  | Содержимое NN+1 --> регистр IXH, |
|  |  | Содержимое NN --> регистр IXL. |
|  |  |  |
| --- | LD IY,(NN) | Содержимое ячеек памяти, адресован- |
|  |  | ных в параметрах NN и NN+1, за- |
|  |  | гружается в индексный регистр IY: |
|  |  | Содержимое NN+1 --> регистр IYH; |
|  |  | Содержимое NN --> регистр IYL. |
|  |  |  |
| SHLD NN | LD (NN),HL | Содержимое двойного регистра HL |
|  |  | передается на адреса NN и NN+1: |
|  |  |  |
|  |  | Содержимое регистра н --> содержи- |
|  |  | мое адреса NN+1; |
|  |  | Содержимое регистра L --> содержи- |
|  |  | мое адреса NN. |
|  |  |  |
| --- | LD (NN),DD | Содержимое пары регистров передает- |
|  |  | ся на адреса NN и NN+1: |
|  |  | Содержимое старшего регистра (В,D |
|  |  | SPH) ---> адрес NN+1, |
|  |  | Содержимое младшего регистра (С,Е |
|  |  | SPL) ---> адрес NN. |
|  |  |  |
| --- | LD (NN),IX | Содержимое индексного регистра IX |
|  |  | передается на адреса NN+1 и NN: |
|  |  | Содержимое IXH ---> адрес NN+1; |
|  |  | Содержимое IXL ---> адрес NN. |
|  |  |  |
| --- | LD (NN),IY | Содержимое индексного регистра IY |
|  |  | передается на адреса NN+1 и NN: |
|  |  | Содержимое IYH ---> адрес NN+1; |
|  |  | Содержимое IYL ---> адрес NN. |
|  |  |  |
| SPHL | LD SP,HL | Содержимое двойного регистра HL пе- |
|  |  | реносится в указатель стека: |
|  |  | Содержимое регистра Н ---> SPH; |
|  |  | Содержимое регистра L ---> SPL. |
|  |  |  |
| --- | LD SP,IX | Содержимое индексного регистра IX |
|  |  | переносится в указатель стека: |
|  |  | Содержимое регистра IXH -->SPH; |
|  |  | Содержимое регистра IXL -->SPL. |
|  |  |  |
| --- | LD SP,IY | Содержимое индексного регистра IY |
|  |  | переносится в указатель стека: |
|  |  | Содержимое регистра IYH -->SPH; |
|  |  | Содержимое регистра IYL -->SPL. |

Примечание

Байт, следующий в данном случае непосредственно за кодом операции, является младшим байтом 16-разрядного слова.

**6.2. Косвенные операции регистров (команды PUSH и POP)**

**6.2.1. Команды PUSH**

При использовании команд PUSH содержимое пары регистров GG или регистра IX, а также IY переносится во внешний стек памяти с произвольной выборкой, организованный как файл LIFO (LIFO - последняя запись считывается первой).

При этом указатель стека постоянно содержит фактический 6-разрядный адрес, соответствующий действительно низшему адресу области стека.

Команда PUSH вычитает 1 из содержимого указателя стека и загружает старший байт пары регистров или регистра IX либо IY в ячейку памяти, адресованную содержимым указателя стека SP.

После этого содержимое указателя стека уменьшается еще раз. Теперь младший байт переносится в ячейку памяти, адресованную содержимым указателя стека.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| PUSH PSW | PUSH AF | Уменьшение SP |
|  |  | (SP-1) <-- A . |
|  |  | (SP-2) <-- F |
|  |  |  |
| PUSH B | PUSH BC | Аналогично PUSH AF. |
|  |  |  |
| PUSH D | PUSH DE | Аналогично PUSH AF. |
|  |  |  |
| PUSH H | PUSH HL | Аналогично PUSH AF. |
|  |  |  |
| --- | PUSH IX | (SP-2) <-- IXL. |
|  |  | (SP-1) <-- IXXH. |
|  |  |  |
| --- | PUSH IY | Аналогично PUSH IX. |

**6.2.2. Команды РОР**

При использовании команд РОР содержимое двух байтов внешнего стека, адресованных указателем стека SP и (SP+1). Переносится в пару регистров GG или в регистр IX либо IY.

Сначала команда РОР передает содержимое ячейки памяти, адресованной фактическим значением указателя стека, в раздел младших разрядов пары регистров или регистра IX либо IY.

После этого указатель стека уменьшается, и содержимое адресованной теперь ячейки памяти передается в раздел старших разрядов пары регистров или регистра IX либо IY. По окончании указатель стека вновь увеличивается.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| POP PSW | POP AF | F <-- (SP) |
|  |  | Увеличение SP |
|  |  | A <-- (SP+1) |
|  |  | Увеличение SP |
|  |  |  |
| POP B | POP BC | Аналогично POP AF |
|  |  |  |
| POP D | POP DE | Аналогично POP AF |
|  |  |  |
| POP H | POP HL | Аналогично POP AF |
|  |  |  |
| --- | POP IX | IXH <-- (SP+1) |
|  |  | IXL --- (SP) |
|  |  |  |
| --- | POP IY | Аналогично POP IX |

Пример:

Сохранение регистров в подпрограмме

BEISP: PUSH HL

PUSH DE

PUSH BC

. ) Команды обработки подпрограммы

. )

. )

POP BC

POP DE

POP HL

RET

После обработки подпрограммы регистры ( HL, DE, BC ) вновь имеют то же содержимое, что и перед вызовом.

**6.3. Команды смены регистров**

Длина команд смены регистров составляет один байт, за исключением команд между (SP) и соответственно базисным регистром, длина команд которых составляет 2 байта.

Благодаря небольшой длине команд возможны короткие времена ответа на прерывание. Для смены регистров в распоряжении имеется набор регистров центрального процессора второго уровня (называют также 2-м набором регистров).

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| XCHG | EX DE,HL | Осуществляется смена 16-разрядного |
|  |  | Содержимого пар регистров DE и HL: |
|  |  | DE <--> HL |
|  |  |  |
| --- | EX AF,AF' | Осуществляется смена 16-разрядного |
|  |  | Содержимого пар регистров AF и AF'. |
|  |  | AF' состоит из регистров A' и F'. |
|  |  | AF <--> AF' |
|  |  |  |
| --- | EXX | Смена 16-ти разрядного содержимого |
|  |  | Следующих регистров осуществляется |
|  |  | Следующим образом: |
|  |  | BC <--> BC' |
|  |  | DE <--> DE' |
|  |  | HL <--> HL' |
|  |  |  |
| XTHL | EX (SP),HL | Содержимое регистра L заменяется |
|  |  | содержимым ячейки памяти, адресован- |
|  |  | ной содержимым указателя стека SP. |
|  |  | Содержимое регистра н заменяется со- |
|  |  | держимым ячейки памяти, адресованной |
|  |  | содержимым указателя стека SP плюс 1 |
|  |  | Н --- (SP+1) |
|  |  | L --- (SP) |
|  |  |  |
| --- | EX (SP),IX | Раздел младших разрядов регистра IX |
|  |  | заменяется содержимым ячейки памяти, |
|  |  | адресованной содержимым указателя |
|  |  | стека SP. |
|  |  | Раздел старших разрядов регистра IX |
|  |  | заменяется содержимым ячейки памяти, |
|  |  | адресованной содержимым указателя |
|  |  | стека SP плюс 1. |
|  |  | IXH <--> (SP+1) |
|  |  | IXL <--> (SP) |
|  |  |  |
| --- | EX (SP),IY | Аналогично EX (SP),IX |
|  |  | IYH <--> (SP+1) |
|  |  | IYL <--> (SP) |

**6.4. Команды передачи блоков**

При помощи одной единственной команды блок любой величины можно передать из одной ячейки памяти в другую ячейку памяти.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| --- | LDIR | Передача нескольких байтов данных |
|  |  | из ячейки памяти, адресуемой парой |
|  |  | регистров HL, в ячейку памяти, |
|  |  | адресуемую парой регистров DE. Ко- |
|  |  | личество байтов содержится в паре |
|  |  | регистров ВС. |
|  |  | После каждой передачи байтов содер- |
|  |  | жимое HL и DE увеличивается на 1, |
|  |  | а регистра ВС уменьшается на 1. |
|  |  | Передача заканчивается, если (ВС)=0. |
|  |  | Пример: |
|  |  | LD,NL,DATA ; начало исходной области |
|  |  | LD DE,PUF ; начало выходной области |
|  |  | LD BC,737 ; длина цепочки данных |
|  |  | LDIR ; передача цепочки данных |
|  |  | ; в выходную область |
|  |  |  |
| --- | LDI | Передача байта данных из ячейки па- |
|  |  | мяти, адресуемой регистром HL, в |
|  |  | ячейку памяти, адресуемую регистром |
|  |  | DE. |
|  |  | Регистры DE и HL увеличиваются на |
|  |  | 1, Регистр ВС уменьшается на 1. |
|  |  | Пример: |
|  |  | LD HL,DATA ; начало исходной области |
|  |  | LD DE,PUF ; начало выходной области |
|  |  | LD BC,132 ; макс. Длина цепочки ВС |
|  |  | LD A,'K' ; признак конца в регистре А |
|  |  | LOOP:CMP(HL);сравнение содержимого |
|  |  | ;памяти с признаком |
|  |  | ; Конца |
|  |  | JR Z,END- ; переход на END, если |
|  |  | ; символы одинаковые |
|  |  | LDI ; передача символов (HL) |
|  |  | ; в (DE). |
|  |  | ; Увеличение HL и DE, |
|  |  | ; уменьшение ВС |
|  |  | JP PE,LOOP ; переход на LOOP, если |
|  |  | ; Должны передаваться |
|  |  | ; еще символы, в ином |
|  |  | ; случае продолжение |
|  |  | ; Действий |
|  |  | ;P/V -флажок = 1, пока |
|  |  | ; ВС не станет = 0 |
|  |  | END: JP HALT; Останов |
|  |  |  |
| --- | LDDR | Команда действует как LDIR, только |
|  |  | здесь содержимое регистров DE и HL |
|  |  | уменьшается на 1. |
|  |  |  |
| --- | LDD | Команда действует как LDI, только |
|  |  | здесь содержимое регистров DE и HL |
|  |  | уменьшается на 1. |

**6.5. Команды поиска в блоке**

Эти команды предназначены для обработки больших массивов данных. С помощью одной единственной команды в блоке памяти любой величины можно отыскать определенный 8-разрядный символ. Команда автоматически завершается после нахождения символа.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| --- | CPI | Сравнение содержимого ячейки памяти, |
|  |  | адресованной регистром HL, с содер- |
|  |  | жимым аккумулятора (регистр А). Ре- |
|  |  | гистр ВС может работать как счетчик |
|  |  | байтов. Содержимое пары регистров ВС |
|  |  | уменьшается на 1. |
|  |  |  |
| --- | CPIR | Сравнение содержимого аккумулятора |
|  |  | с содержимым адресованной области |
|  |  | памяти. Начальный адрес области со- |
|  |  | держится в паре регистров HL, длина |
|  |  | области содержится в паре регистров |
|  |  | ВС. Искомая константа находится в |
|  |  | аккумуляторе. Сравнение завершает- |
|  |  | ся, если аккумулятор = (HL) или ес- |
|  |  | ли ВС = 0. Команда осуществляет по- |
|  |  | иск, при этом содержимое регистра |
|  |  | HL увеличивается, а содержимое ре- |
|  |  | гистра BC уменьшается на 1. |
|  |  |  |
| --- | CPD | Команда действует как CPI, только |
|  |  | здесь уменьшается содержимое ре- |
|  |  | гистра HL. |
|  |  |  |
| --- | CPDR | Команда действует как CPIR, только |
|  |  | здесь уменьшается содержимое ре- |
|  |  | гистра HL. |

**6.6. Арифметические и логические операции**

Арифметические и логические команды работают с данными, находящимися в аккумуляторе (регистр А) и в других универсальных регистрах центрального процессора или в ячейках памяти.

Результаты после выполнения этих операций находятся в аккумуляторе. Флажки устанавливаются в соответствии с результатом операции.

**6.6.1. 8-разрядная арифметика**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| ADD R | ADD A,R | Содержимое регистра R прибавляется |
|  |  | к содержимому аккумулятора. |
|  |  |  |
| ADD M | ADD A,(HL) | Содержимое ячейки памяти м или (HL) |
|  |  | прибавляется к содержимому аккуму- |
|  |  | лятора. |
|  |  |  |
| ADI N | ADD A,N | Прямой операнд N прибавляется к |
|  |  | содержимому аккумулятора. |
|  |  |  |
| --- | ADD A,(IX+D) | Содержимое ячейки памяти (IX+D) при- |
|  |  | бавляется к содержимому аккумулятора |
|  |  |  |
| --- | ADD A,(IY+D) | Содержимое ячейки памяти (IY+D) при- |
|  |  | бавляется к содержимому аккумулятора |
| ADC R | ADC A,R | содержимое регистра R и флажок пере- |
|  |  | носа (CY) прибавляются к содержимому |
|  |  | аккумулятора. |
|  |  |  |
| ADC M | ADC A,(HL) | Содержимое ячейки памяти м или (HL) |
|  |  | и флажок переноса (CY) прибавляются |
|  |  | к содержимому аккумулятора. |
|  |  |  |
| ACI N | ADC A,N | Прямой операнд N и CY прибавляяются |
|  |  | к содержимому аккумулятора. |
|  |  |  |
| --- | ADC A,(IX+D) | Содержимое ячейки памяти (IX+D) и |
|  |  | CY прибавляются к содержимому аккумуля- |
|  |  | тора |
|  |  |  |
| --- | ADC A,(IY+D) | Содержимое ячейки памяти (IY+D) и |
|  |  | CY прибавляются к содержимому акку- |
|  |  | мулятора. |
|  |  |  |
| SUB R | SUB R | Содержимое регистра R вычитается из |
|  |  | содержимого аккумулятора. |
|  |  |  |
| SUB M | SUB (HL) | Содержимое ячейки памяти M или (IL) |
|  |  | вычитается из содержимого аккумулятора |
|  |  |  |
| SUI N | SUB N | Прямой операнд N вычитается из со- |
|  |  | держимого аккумулятора. |
|  |  |  |
| --- | SUB (IX+D) | Содержимое ячейки памяти (IX+D) вы- |
|  |  | читается из содержимого аккумулятора |
|  |  |  |
| --- | SUB (IY+D) | Содержимое ячейки памяти (IY+D) вы- |
|  |  | читается из содержимого аккумулятора |
|  |  |  |
| SBB R | SBC A,R | Содержимое регистра R и флажок пе- |
|  |  | реноса (CY) вычитаются из содержимо- |
|  |  | го аккумулятора |
|  |  |  |
| SBB M | SBC A,(HL) | Содержимое ячейки памяти м или (HL) |
|  |  | и флажок переносA (CY) вычитаются из |
|  |  | содержимого аккумулятора. |
|  |  |  |
| SBI N | SBC A,N | Прямой операнд N и флажок переноса |
|  |  | (CY) вычитаются из содержимого акку- |
|  |  | мулятора. |
|  |  |  |
| --- | SBC A,(IX+D) | Содержимое ячейки памяти (IX+D) и |
|  |  | флажок переноса (CY) вычитаются из |
|  |  | содержимого аккумулятора. |
|  |  |  |
| --- | SBC A,(IY+D) | Содержимое ячейки памяти (IY+D) и |
|  |  | флажок переноса (CY) вычитаются из |
|  |  | содержимого аккумулятора. |
|  |  |  |
| INR R | INC R | Содержимое регистра R увеличивается |
|  |  | на 1. |
|  |  |  |
| INR M | INC (HL) | Содержимое ячейки памяти м или (HL) |
|  |  | увеличивается на 1. |
|  |  |  |
| --- | INC (IX+D) | Содержимое ячейки памяти (IX+D) |
|  |  | увеличивается на 1. |
|  |  |  |
| --- | INC (IY+D) | Содержимое ячейки памяти (IY+D) |
|  |  | увеличивается на 1. |
|  |  |  |
| DCR R | DEC R | Содержимое регистра R уменьшается |
|  |  | на 1. |
|  |  |  |
| DCR M | DEC (HL) | Содержимое ячейки памяти м или (HL) |
|  |  | уменьшается на 1. |
|  |  |  |
| --- | DEC (IX+D) | Содержимое ячейки памяти (IX+D) |
|  |  | уменьшается на 1. |
|  |  |  |
| --- | DEC (IX+D) | Содержимое ячейки памяти (IY+D) |
|  |  | уменьшается на 1. |

**6.6.2. 16-Разрядная арифметика**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| DAD DD | ADD HL,DD | Содержимое DD прибавляется к содер- |
|  |  | жимому пары регистров HL. |
|  |  |  |
| --- | ADD IX,IX | Содержимое регистра IX прибавляется |
|  |  | к самому себе. Это удвоение равно- |
|  |  | значно смещению 16 разрядов влево |
|  |  | на одну позицию. |
|  |  |  |
|  |  |  |
| --- | ADD IY,IY | Содержимое регистра IY прибавляется |
|  |  | к самому себе. Это удвоение равно- |
|  |  | значно смещению 16 разрядов влево |
|  |  | на одну позицию. |
|  |  |  |
|  |  |  |
| --- | ADD IX,PP | Содержимое PP прибавляется к содер- |
|  |  | жимому 16-разрядного регистра IX. |
|  |  |  |
|  |  |  |
| --- | ADD IY,PP | Содержимое PP прибавляется к содер- |
|  |  | жимому 16-разрядного регистра IY. |
|  |  |  |
|  |  |  |
| --- | ADC HL,DD | Содержимое двойного регистра DD и |
|  |  | флажок переноса (CY) прибавляются к |
|  |  | содержимому пары регистров HL. |
|  |  |  |
|  |  |  |
| --- | SBC HL,DD | Содержимое двойного регистра DD и |
|  |  | флажок переноса (CY) вычитаются из |
|  |  | содержимого пары регистров HL. |
|  |  |  |
| INX DD | INC BB | Содержимое двойного регистра DD или |
|  |  | BB увеличивается на 1. |
|  |  |  |
| DCX DD | DEC BB | Содержимое двойного регистра DD или |
|  |  | BB уменьшается на 1. |

**6.6.3. 8-Разрядные логические команды**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| ANA R | AND S | Логическое "и" регистра, байта памяти |
| ANA M |  | или прямо адресуемой величины с ак- |
| ANI N | S = R,N,(HL), | кумулятором. Специфицированный байт |
|  | (IX+D),(IY+D) | побитно связывается кон'юктивно с ак- |
|  |  | кумулятором.Логическое "и" двух битов |
|  |  | равняется 1 только в том случае, если |
|  |  | оба бита равны 1. |
|  |  | Пример: |
|  |  | Аккумулятор:FCH 1111 1100 |
|  |  | Регистр: 0FH 0000 1111 |
|  |  | --------------- |
|  |  | Результат: 0CH 0000 1100 |
|  |  |  |
| ORA R | OR S | Логическое "или" регистра, байта памя- |
| ORA M |  | ти или прямо адресуемой величины с |
| ORI N | S = R,N,(HL), | аккумулятором. Специфицированный |
|  | (IX+D),(IY+D) | байт побитно связывается с содержи- |
|  |  | мым аккумулятора диз'юнктивно. Логи- |
|  |  | ческое или двух битов равняется 0 |
|  |  | только в том случае, если оба бита |
|  |  | равны 0. |
|  |  | Пример: |
|  |  | Аккумулятор: FCH 1111 1100 |
|  |  | Регистр: F1H 1111 0001 |
|  |  | --------------- |
|  |  | Результат: FDH 1111 1101 |
|  |  |  |
| XRA R | XOR S | Исключающее "или" регистра, байта па- |
| XRA M |  | мяти или прямо адресуемой величины |
| XRI N | S = R,N,(HL), | с аккумулятором. Специфицированный |
|  | (IX+D),(IY+D) | байт побитно связывается искючающе |
|  |  | с содержимым аккумулятора. Исключаю- |
|  |  | щее или равно 1 в том случае, если |
|  |  | один бит = 1 и второй бит = 0. |
|  |  | Пример: |
|  |  | Аккумулятор: FCH 1111 1100 |
|  |  | Регистр: F1H 1111 0001 |
|  |  | --------------- |
|  |  | Результат: 0DH 0000 1101 |
|  |  |  |
|  |  |  |
| CMP R | CP S | Содержимое S сравнивается с аккуму- |
| CMP M |  | лятором. Первоначальное содержимое |
| CPI N | S = R,N,(HL), | А сохраняется. Результат сравнения |
|  | (IX+D),(IY+D) | идентифицируется по флажкам. |

**6.7. Команды перехода**

Следует различать условные и безусловные переходы. Кроме того, возможны относительные переходы, при которых для образования адресов требуется только один байт вместо двух.

При условных переходах проверяются условия переходов. Эти условия содержатся во флажковом регистре F.

В зависимости от флажков условий условия переходов могут быть выполнены или не выполнены.

В случае выполненного условия перехода счетчик назначения памяти изменяется в соответствии с заданием адреса в команде перехода. Если условие перехода не выполнено, то команда перехода игнорируется.

При относительных переходах цель перехода определяется по значению Е. Величина перехода Е прибавляется к фактическому состоянию счетчика назначения памяти (в двоичном дополнении) и позволяет осуществлять переход в диапазоне от -128 до 127 байтов.

При символической адресации в относительных переходах ассемблер автоматически вычисляет счетчик назначения памяти.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| JMP NN | JP NN | Условный переход по адресу NN. |
|  |  |  |
| JNZ NN | JP NZ,NN | Переход поH адресу NN, если флажок Z |
|  |  | равен 0. |
|  |  |  |
| JZ NN | JZ,NN | Переход по адресу NN, если флажок Z |
|  |  | равен 1. |
|  |  |  |
| JNC NN | JP NC,NN | Переход по адресу NN, если флажок C |
|  |  | равен 0. |
|  |  |  |
| JC NN | JP C,NN | Переход по адресу NN, если флажок |
|  |  | равен 1. |
|  |  |  |
| JPO NN | JP PO,NN | Переход по адресу NN, если флажок |
|  |  | P/V равен 0. |
|  |  |  |
| JPе NN | JP Pе,NN | Переход по адресу NN, если флажок |
|  |  | P/V равен 1. |
|  |  |  |
| JP NN | JP P,NN | Переход по адресу NN, если флажок C |
|  |  | равен 0. |
|  |  |  |
| JM NN | JP MM,NN | Переход по адресу NN, если флажок S |
|  |  | равен 1. |
|  |  |  |
| --- | JR E | Безусловный относительный переход. |
|  |  |  |
| --- | JR NZ,E | Относительный переход на смещение Е, |
|  |  | если флажок Z равен 0. |
|  |  |  |
| --- | JR Z,E | Относительный переход на смещение Е, |
|  |  | если флажок Z равен 1. |
|  |  |  |
| --- | JR NC,E | Относительный переход на смещение Е, |
|  |  | если флажок с равен 0. |
|  |  |  |
| --- | JR C,E | Относительный переход на смещение Е, |
|  |  | если флажок с равен 1. |
|  |  |  |
| PCHL | JP (HL) | Безусловный переход по адресу, нахо- |
|  |  | дящемуся в регистре HL. |
|  |  |  |
| --- | JP (IX) | Безусловный переход по адресу, нахо- |
|  |  | дящемуся в регистре IX. |
|  |  |  |
| --- | JP (IY) | Безусловный переход по адресу, нахо- |
|  |  | дящемуся в регистре IY. |
|  |  |  |
| --- | DJNZ E | Содержимое регистра в уменьшается |
|  |  | на 1. Команда условного относитель- |
|  |  | ного перехода на смещение Е, если |
|  |  | содержимое регистра в = 0 |

**6.8. Команды сдвига**

При помощи этих команд можно просто или циклически сдвигать данные в аккумуляторе, универсальном регистре или ячейке памяти. Эти операции можно применять в большой внешней области, включая целочисленное умножение и деление.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| RLC | RLCA | Вращение содержимого аккумулятора |
|  |  | влево. Содержимое аккумулятора сдви- |
|  |  | гается на одну позицию влево. Стар- |
|  |  | ший бит 7 становится содержимым |
|  |  | младшего бита 0 и флажка переноса. |
|  |  | CY <------ 7 <-- 0 <-- |
|  |  |  |
| RRC | RRCA | Вращение вправо содержимого аккуму- |
|  |  | лятора. Содержимое аккумулятора |
|  |  | сдвигается на одну позицию вправо. |
|  |  | Младший бит 0 становится содержимым |
|  |  | старшего бита 7 и флажка переноса. |
|  |  | ---> 7 --> 0 ----> CY |
|  |  |  |
| RAL | RLA | Вращение влево содержимого аккумму- |
|  |  | лятора через CY. Содержимое аккуму- |
|  |  | лятора сдвигается влево на одну по- |
|  |  | зицию. Старший бит 7 заменяет флажок |
|  |  | переноса, а флажок переноса заменяет |
|  |  | младший бит 0 аккумулятора. |
|  |  | --> CY <--- 7 <-- 0 -- |
|  |  |  |
| RAR | RRA | Вращение вправо содержимого аккуму- |
|  |  | лятора через CY. Содержимое аккуму- |
|  |  | лятора сдвигает вправо на одну по- |
|  |  | зицию. Младший бит 0 заменяет флажок |
|  |  | переноса, а флажок переноса заменяет |
|  |  | старший бит 7. |
|  |  | --> 7 --> 0 --> CY-- |
|  |  |  |
| --- | RLS T | Вращение влево T аналогично команде |
|  | T=R,(HL), | RLCA. |
|  | (IH+D),(IY+D) |  |
|  |  |  |
| --- | RRC T | Вращение вправо T аналогично команде |
|  |  | RRCA. |
|  |  |  |
| --- | RL T | Вращение влево T через CY аналогично |
|  |  | команде RLA. |
|  |  |  |
| --- | RR T | Вращение вправо T через CY аналогич- |
|  |  | но команде RRA. |
|  |  |  |
| --- | SLA T | Смещение влево T на один бит через |
|  |  | CY. Mладший бит регистра 0 стано- |
|  |  | вится 0. |
|  |  | CY <--- 7 <--- 0 <--- 0 |
|  |  |  |
| --- | SRL T | Смещение вправо T на один бит через |
|  |  | CY. Старший бит регистра 7 становит- |
|  |  | ся 0. |
|  |  | 0 ---> 7 ---> 0---> CY |
|  |  |  |
| --- | SRA T | Смещение вправо т на один бит через |
|  |  | CY. Содержимое бита 7 сохраняется. |
|  |  | --- 7--->0 -----> CY |
|  |  |  |
| --- | RLD | Циклический сдвиг влево между акку- |
|  |  | мулятором и содержимым ячейки памя- |
|  |  | ти, адресованной регистром HL. |
|  |  | A (HL) |
|  |  | 1101 1111 0000 1111 |
|  |  | Перед командой |
|  |  | 0101 0000 1111 1111 |
|  |  | После команды |
|  |  | 4 Младших бита ячейки памяти, адре- |
|  |  | сованной HL, перемещаются в 4 млад- |
|  |  | ших разряда аккумулятора. Младшие 4 |
|  |  | бита аккумулятора перемещаются в |
|  |  | младшие 4 бита ячейки памяти. |
|  |  |  |
| --- | RRD | Циклический сдвиг вправо между акку- |
|  |  | мулятором и содержимым ячейки памя- |
|  |  | ти, адресованной HL. |
|  |  | A (HL) |
|  |  | 0101 1111 0000 1111 |
|  |  | Перед командой |
|  |  | 0101 1111 1111 0000 |
|  |  | После команды |
|  |  | Младшие 4 бита ячейки памяти, адре- |
|  |  | сованной HL, переносятся в младшие |
|  |  | 4 Разряда аккумулятора, которые, в |
|  |  | свою очередь, перемещаются в старшие |
|  |  | 4 Разряда ячейки памяти, адресован- |
|  |  | ной HL. Старшие 4 бита из ячейки па- |
|  |  | мяти, адресованной HL, переносятся в |
|  |  | младшие 4 разряда. |

**6.9. Специальные команды аккумулятора и флажков**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| DAA | DAA | После сложения / вычитания двух |
|  |  | упакованных двоично-кодированных |
|  |  | десятичных чисел исправляет содер- |
|  |  | жимое аккумулятора таким образом, |
|  |  | что в аккумуляторе вновь достигает- |
|  |  | ся упакованное двоично-кодированное |
|  |  | представление десятичных чисел. |
|  |  | Пример |
|  |  |  |
|  |  | 65 0110 0101 |
|  |  | +57 0101 0111 |
|  |  | --------- |
|  |  | 1011 1100 |
|  |  | После этого команда DAA выполняет |
|  |  | необходимое исправление. В зависи- |
|  |  | мости от значения двух полубайтов |
|  |  | прибавляется определенный байт кор- |
|  |  | рекции (в имеющемся примере J66). |
|  |  | CY 0110 0110 |
|  |  | +++++++++++ |
|  |  | Y |
|  |  | L 0010 0010 = 122 |
| CMA | CPL | Побитное отрицание (дополнение) со- |
|  |  | содержмого аккумулятора. |
|  |  |  |
| --- | NEG | Вычитание содержимого аккумулятора |
|  |  | из 0. По значениям соответствует |
|  |  | двоичному дополнению. |
|  |  |  |
| CMC | CCF | Дополнение флажка переноса. |
|  |  |  |
| STC | SCF | Установка флажка переноса. |

**6.10. Команды вызова подпрограмм**

Эти команды представляют собой особую форму команд перехода. Следует различать безусловные и условные вызовы подпрограмм. При безусловном вызове подпрограммы состояние счетчика назначения ячеек памяти, следующее за вызовом, запоминается в стеке для сохранения.

Заданный в команде начальный адрес подпрограммы NN вновь загружается счетчиком назначения ячеек памяти с адресом перехода из стека.

При условных вызовах подпрограмм в случае выполненного условия перехода принцип действия аналогичен действиям при безусловном вызове подпрограммы. Если условие перехода не выполнено, команда игнорируется.

Часть старших адресов в счетчике назначения ячеек памяти переносится на адрес стека минус 1, часть младших адресов - на адрес стека минус 2.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| CALL NN | CALL NN | Безусловный вызов подпрограммы |
|  |  | (SP-1) <-- PCH |
|  |  | (SP-2) <-- PCL |
|  |  | PC <-- NN |
|  |  |  |
| CNZ NN | CALL NZ,NN | Вызов подпрограммы, если флажок Z |
|  |  | равен 0. |
|  |  |  |
| CZ NN | CAL Z, NN | Вызов подпрограммы, если флажок Z |
|  |  | равен 1. |
|  |  |  |
| CNC NN | CALL NC,NN | Вызов подпрограммы, если флажок C |
|  |  | равен 0. |
|  |  |  |
| CC NN | CALL C,NN | Вызов подпрограммы, если флажок C |
|  |  | равен 1. |
|  |  |  |
| CPO NN | CALL PO,NN | Вызов подпрограммы, если флажок P/V |
|  |  | равен 0. |
|  |  |  |
| CPE NN | CALL PE,NN | Вызов подпрограммы, если флажок P/V |
|  |  | равен 1. |
|  |  |  |
| CP NN | CAL P,NN | Вызов подпрограммы, если флажок S |
|  |  | равен 0. |
|  |  |  |
| См NN | CALL M,NN | Вызов подпрограммы, если флажок S |
|  |  | равен 1. |
|  |  |  |
| RST K | RST P | Команда RST является специальным |
|  |  | Вызовом подпрограммы. Допустимы |
|  |  | Следующие 8 адресов RST: P =( 00н; |
|  |  | 08Н;10н; 18н; 28н; 30н; 38н ). |
|  |  | При этом часть старших адресов рав- |
|  |  | на 0. |
|  |  | По последующим действиям команда |
|  |  | RCT соответствует безусловному |
|  |  | вызову подпрограммы. |

**6.11. Команды возврата из подпрограммы**

Команда возврата завершает подпрограмму. Следует различать безусловный возврат, условные возвраты и возвраты из подпрограмм обработки прерываний.

В случае безусловного возврата и при выполненном условии перехода в случае условных возвратов содержимое счетчика назначения ячеек памяти, сохраненное в стеке при вызове подпрограммы, вновь переносится в счетчик назначения памяти.

PCL <-- (SP)

PCH <-- (SP+1)

SP <-- (SP+2)

Если условие перехода не выполнено, выполняется команда, следующая за командой возврата.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| RET | RET | Безусловный возврат. |
|  |  |  |
| RNZ | RET NZ | Возврат из подпрограммы, если |
|  |  | флажок Z равен 0. |
| RZ | RET Z | Возврат из подпрограммы, если |
|  |  | флажок Z равен 1. |
|  |  |  |
| RNC | RET NC | Возврат из подпрограммы, если |
|  |  | флажок с равен 0. |
|  |  |  |
| RC | RET C | Возврат из подпрограммы, если |
|  |  | флажок с равен 1. |
|  |  |  |
| RPO | RET PO | Возврат из подпрограммы, если |
|  |  | флажок P/V равен 0. |
|  |  |  |
| RPE | RET PE | Возврат из подпрограммы, если |
|  |  | флажок P/V равен 1. |
|  |  |  |
| RP | RET P | Возврат из подпрограммы, если |
|  |  | флажок S равен 0. |
|  |  |  |
| BM | RET M | Возврат из подпрограммы, если |
|  |  | флажок S равен 1. |
|  |  |  |
| --- | RETI | Происходит возврат из подпрограммы |
|  |  | обработки прерываний. Внешний мо- |
|  |  | дуль заявивший прерывание, извещает |
|  |  | об окончании его программы. |
|  |  | После этого модуль деблокирует за- |
|  |  | блокированную им цепочку прерываний |
|  |  | (DAISY-SHAIN),что позволяет обраба- |
|  |  | тывать прерывания нижнего уровня. |
|  |  | Содержимое IFF2 переносится в IFF1. |
|  |  | В результате оператора RETI маски- |
|  |  | руемое прерывание не деблокируется. |
|  |  | Как правило, перед каждым операто- |
|  |  | ром RETI должна стоять команда EI, |
|  |  | позволяющая принимать в дальнейшем |
|  |  | последующие запросы на прерывание. |
|  |  |  |
| --- | RETN | Происходит возврат из подпрограммы |
|  |  | обработки прерываний, запущенной в |
|  |  | результате немаскируемого прерывания |
|  |  | (NMI). Сначала оператор действует |
|  |  | как оператор RET. Дополнительно со- |
|  |  | держимое IFF2 переносится в IFF1, |
|  |  | так что обработка запросов маскируе- |
|  |  | мого прерывания деблокирована непос- |
|  |  | редственно после выполнения команды |
|  |  | RETN, если эта обработка была дебло- |
|  |  | кирована перед запросом NMI. |

**6.12. Команды управления центральным процессором**

Команды управления центральным процессором вызывают различные условия и режимы работы.

Эта группа содержит также такие команды, как команда включения и выключения триггера приема прерывания или команда установки режима прерываний.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| NOP | NOP | Центральный процессор не выполняет |
|  |  | операции. Формируются циклы обнов- |
|  |  | ления. |
| HLT | HALT | Центральный процессор до тех пор |
|  |  | выполняет последовательность ко- |
|  |  | манд NOP, пока в центральном про- |
|  |  | цессоре не будет активировано пре- |
|  |  | рывание или вход RESET (Сброс). |
|  |  | формируются циклы обновления. |
|  |  |  |
| DI | DI | Маскируемое прерывание блокируется |
|  |  | посредством возврата в исходное по- |
|  |  | ложение триггера деблокировки пре- |
|  |  | рываний IFF1 или IFF2 центрального |
|  |  | процессора. |
|  |  | Немаскируемые прерывания признаются. |
|  |  |  |
| EI | EI | Маскируемое прерывание деблокируется |
|  |  | посредством установки триггера де- |
|  |  | блокировки прерываний IFF1 или IFF2 |
|  |  | центрального процессора. |
|  |  | Во время выполнения команды цент- |
|  |  | ральный процессор не принимает зап- |
|  |  | росы на прерывание. |
|  |  |  |
| --- | IM 0 | Команда устанавливает центральный |
|  |  | процессор в режим прерывания 0. |
|  |  |  |
| --- | IM 1 | Команда устанавливает центральный |
|  |  | процессор в режим прерывания 1. |
|  |  |  |
| --- | IM 2 | Команда устанавливает центральный |
|  |  | процессор в режим прерывания 2. |

**6.13. Команды установки и проверки четности**

**(действия с битами)**

Команды действий с битами позволяют устанавливать, сбрасывать и проверять на четность биты в регистре или в ячейке памяти.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| --- | SET B,R | Бит, обозначенный B, устанавливается |
|  |  | в регистре R. |
|  |  |  |
| --- | SET B,(HL) | Бит, обозначенный B, устанавливается |
|  |  | в регистре (HL). |
|  |  |  |
| --- | SET B,(IX+D) | Бит, обозначенный B, устанавливается |
|  |  | в регистре (IX+D) |
|  |  |  |
| --- | SET B,(IY+D) | Бит, обозначенный B, устанавливается |
|  |  | В регистре (IY+D) |
|  |  |  |
| --- | RES B,T T=R, | Бит, обозначенный B,сбрасывается в T |
|  | (IX+D),(IY+D) |  |
|  |  |  |
| --- | BIT B,T | Бит, обозначенный B, проверяется на |
|  |  | четность в T. Дополнение бита, про- |
|  |  | веряемого на четность, загружается |
|  |  | во флажок Z. |

**6.14. Команды ввода**

Группа команд ввода-вывода расширяет область применения передачи данных между ячейками памяти или регистрами общего назначения центрального процессора и внешними устройствами ввода-вывода.

Команды ввода автоматически устанавливают флажковый регистр, так что нет необходимости в дополнительных командах для определения состояния входных данных.

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоническое обозначение для  Z80 | Принцип действия команд |
| IN N | IN A,(N) | Aдрес канала устанавливается посред- |
|  |  | ством прямого операнда. |
|  |  | Выходным регистром является аккуму- |
|  |  | лятор а <-- (N). |
|  |  |  |
| --- | IN R,(C) | Адрес канала устанавливается косвен- |
|  |  | но через регистр С. |
|  |  | R <-- (C). |
|  |  |  |
| --- | INI | Адрес канала устанавливается косвен- |
|  |  | но через регистр С. Выходной адрес |
|  |  | устанавливается через регистр (HL). |
|  |  | В может работать как счетчик байтов. |
|  |  | B уменьшается, HL увеличивается. |
|  |  | (HL) <-- (C) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL+1 |
|  |  |  |
| --- | INIR | Адрес канала устанавливается косвен- |
|  |  | но через регистр с, выходной адрес - |
|  |  | через регистр HL. В работает как |
|  |  | счетчик байтов. В уменьшается, HL |
|  |  | увеличивается. Выполняется передача |
|  |  | блока, пока в не станет = 0. |
|  |  | (HL) <-- (C) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL+1 |
|  |  | Повторение до В=0. |
|  |  |  |
| --- | IND | Адрес канала устанавливается косвен- |
|  |  | но через регистр С. Выходной адрес |
|  |  | устанавливается через регистр HL. |
|  |  | Mв может работать как счетчик байтов. |
|  |  | B и HL уменьшаются. |
|  |  | (HL) <-- (C) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL-1 |
|  |  |  |
| --- | INDR | Адрес канала устанавливается косвен- |
|  |  | Но через регистр с, выходной адрес |
|  |  | - Через регистр HL. В работает как |
|  |  | счетчик байтов. B и HL уменьшаются. |
|  |  | Передача блока выполняется до тех |
|  |  | Пор, пока в = 0. |
|  |  | (HL) <-- (C) |
|  |  | B <-- B-1 |
|  |  | NL <-- HL-1 |
|  |  | Повторение до В = 0. |

Адрес канала располагается в нижней половине шины адресов А0 - А7. В верхней половине шины адресов (А8 - А15) при IN A,(N) располагается содержимое аккумулятора, в случае остальных команд - содержимое регистра В.

**6.15. Команды вывода**

|  |  |  |
| --- | --- | --- |
| Мнемоническое обозначение для  8080 | Мнемоничес кое обозначение для  Z80 | Принцип действия команд |
| OUT N | OUT (N),A | Адрес канала устанавливается прямым |
|  |  | операндом. Исходным регистром явля- |
|  |  | ется аккумулятор: (N) <-- A. |
|  |  |  |
| --- | OUT C,R | Адрес канала устанавливается косвен- |
|  |  | но через регистр С. Исходным регист- |
|  |  | ром является R: (C) <-- R. |
|  |  |  |
| --- | OUTI | Адрес канала устанавливается косвен- |
|  |  | но через регистр С, исходный адрес - |
|  |  | через регистр HL. В может работать |
|  |  | как счетчик байтов. В уменьшается, |
|  |  | HL увеличивается. |
|  |  | (C) <-- (HL) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL+1 |
|  |  |  |
| --- | OTIR | Адрес канала устанавливается косвен- |
|  |  | но через регистр С, исходный адрес - |
|  |  | через регистр Н. |
|  |  | В работает как счетчик байтов. В |
|  |  | уменьшается, HL увеличивается. Вы- |
|  |  | полняется передача блока, пока в |
|  |  | не станет = 0. |
|  |  | (С) <-- (HL) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL+1 |
|  |  | Повторение до В = 0. |
| --- | OUTD | Адрес канала устанавливается косвен- |
|  |  | но через регистр С, исходный адрес - |
|  |  | через регистр HL. B может работать |
|  |  | как счетчик байтов. |
|  |  | B и HL уменьшаются. |
|  |  | (С) <-- (HL) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL-1 |
| --- | OTDR | Адрес канала устанавливается косвен- |
|  |  | но через регистр С, исходный адрес - |
|  |  | через регистр HL. B работает как |
|  |  | счетчик байтов. В HL уменьшаются. |
|  |  | Выполняется передача блока, пока в |
|  |  | Не станет = 0. |
|  |  | (С) <--(HL) |
|  |  | B <-- B-1 |
|  |  | HL <-- HL-1 |
|  |  | Повторение до В = 0. |

**6.16. Перечень сокращений для описания команд**

R....: Один из регистров A, B, C, D, E, H или L.

R' : один из регистров..A', B', C', D', E', H' или L'.

DD : Oдин из двойных регистров BC, DE, HL или SP.

GG : один из двойных регистров AF, BC, DE или HL.

PP : один из двойных регистров BC, DE или SP.

BB : один из двойных регистров BC, DE, HL, SP, IX или IY.

S : разрешены R, N, (HL), (IX+D) и (IY+D).

T : возможны R, (HL), (IX+D) и (IY+D).

N : 8-разрядный прямой операнд.

NN : 16-разрядный прямой операнд.

D : сдвиг при адресации индексным регистром, допустимо в диапазоне от -128 <= D <= 127.

Границы области Ассемблером не проверяются!

E : относительный адрес перехода, значения допустимы в диапазоне от -128 <= E <= 127.

Границы области Ассемблером не проверяются!

B : бит, который должен обрабатываться в командах действий с

битами 0 <= B < =7.

М : содержимое ячейки памяти, адресованной HL.

K : допустимы значения 0, 1, 2, 3, 4, 5, 6, 7.

P : допустимы значения 00H, 08H, 10H, 18H, 28H, 30H, 38H.

CY : флажок переноса.

Примечание

В мнемонических обозначениях для модели 8080 для операндов пишется только первая буква:

H для HL

D для DE

В для ВС

Однако остается SP, и для AF применяется обозначение PSW.

Для М нельзя писать (HL).

**6.17. Работа с битами условий (флажки)**

Флажковый регистр F представляет данные о результате последней операции центрального процессора.

Он предназначен в основном для выполнения условных программных переходов, а также условных вызовов подпрограмм или возвратов из подпрограмм.

Флажковый регистр

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| S | Z | X | H | X | P/V | N | CY |

S -бит знака числа (SIGN-FLAG)

Z -нулевой бит (ZERO-FLAG)

H -бит переноса полубайта (HALF-CARRY-FLAG)

N -бит сложения/вычитания

P/V -бит четности/переполнения (PARITY-OVERFLOW-FLAG)

CY -бит переноса (CARRY-FLAG)

х -не занят

**Бит знака числа (флажок S)**

При определенных командах загружается старший бит аккумулятора.

При выполнении арифметических команд с числами, имеющими знак числа, положительное число обозначается 0, отрицательное число обозначается 1 в старшем двоичном разряде.

**Бит нуля (флажок Z)**

Бит устанавливается при арифметических и логических однобайтных операциях, если байт результата аккумулятора равен 0. В ином случае байт результата сбрасывается.

При командах сравнения и поиска устанавливается флажок Z, как только результат сравнения положительный.

При командах операций с битами флажок Z загружается с дополнительным значением проверяемого бита.

При передаче байта между ячейкой памяти и интерфейсом ввода-вывода (INI, IND, OUTI, OUTD) флажок Z устанавливается в 1.

Если значение регистра счета становится = 0.

При IN R,(C) устанавливается флажок Z, если выданные или находящиеся в вентиле ввода-вывода данные имеют значение 0.

**Бит переноса полубайта (флажок н)**

В соответствии с результатом передачи между битами 3 и 4 арифметической однобайтной операции флажок н устанавливается (если передача произведена) или сбрасывается (если передача не произведена). Бит применяется при команде DAA для коррекции результата сложения или вычитания упакованных двоично-кодированных десятичных чисел.

|  |  |  |
| --- | --- | --- |
| Н | Сложение | Вычитание |
| 1 | Перенос бита 3 к биту 4 | Отрицательный пере нос бита 4 |
| 0 | Перенос бита 3 к биту 4 не производится | Отрицательный пере нос бита 4 не производится |

**Бит четности/переполнения (флажок P/V)**

Бит применяют различным образом.

При арифметических командах флажок PO/V устанавливается, если в результате устанавливается старший бит аккумулятора.

При логических операциях и командах сдвига флажок служит для проверки четности результата. Если количество установленных битов в проверяемом байте четное (0, 2, 4, 6, 8), то устанавливается флажок P/V, в ином случае бит сбрасывается.

При командах передачи блока (LDI, LDIR, LDD, LDDR) и поиска в блоке (CPI, CPIR, CPD, CPDR) флажок N информирует о состоянии счетчика байтов. P/V сбрасывается, если после уменьшения содержимого счетчика байтов (=<ВС>) в качестве результата появляется 0. Во всех остальных случаях флажок P/V остается равным 1

**Бит сложения/вычитания (флажок N)**

Бит используется при команде DAA для различия команд сложения и вычитания. При всех командах сложения флажок N сбрасывается. Команды вычитания устанавливают флажок N.

**Бит переноса (флажок CY)**

Флажок CY устанавливается и сбрасывается в зависимости от выполняемой операции.

Флажок CY устанавливается, если:

- при командах сложения возникает перенос и

- при командах вычитания возникает отрицательный перенос.

Флажок CY сбрасывается:

- при командах сложения, не формирующих перенос и

- при командах вычитания, не формирующих отрицательный перенос.

При командах сдвига (RLA, RLC, RR, RL) флажок CY используется в качестве промежуточной памяти для передачи младших и старших битов регистра центрального процессора или ячейки памяти.

При командах RLCA, RLC и SLA флажок CY содержит значение старшего бита, полученного в результате команды из обрабатываемого регистра или ячейки памяти.

При командах RRCA, RRC, SRA, SRL флажок CY аналогично содержит значение младшего бита.

Логические команды AND, OR, XOR, как правило сбрасывают флажок CY.

Команды SCF (установка флажка CY) и CCF (дополнение флажка CY), предназначенные специально для флажка CY, позволяют осуществлять установку и дополнение флажка CY.

**7. Перекрестные ссылки**

Программа перекрестных ссылок обрабатывает специально оттранслированный файл вывода на печать в листинг всех адресных ссылок внутри модуля, включая ячейки памяти, в которых они были определены.

Перекрестные ссылки можно использовать в качестве вспомогательного средства при программной отладке.

Программа перекрестных ссылок позволяет программисту обрабатывать файл перекрестных ссылок, сформированный Ассемблером.

Этот файл содержит управляющие символы, которые использовались во время трансляции Ассемблером.

Программа перекрестных ссылок формирует листинг, аналогичный файлу .PRN, но имеющий два дополнительных признака.

1. Каждый исходный оператор снабжается номером перекрестной ссылки.

2. В конце листинга располагаются имена переменных в алфавитной последовательности.

Каждому имени назначаются номера строк в порядке возрастания, в которых это имя появляется. Номер строки, в которой имя определено, обозначается #.

Листинг перекрестных ссылок заменяет файл. PRN Ассемблера и получает также тип файла .PRN.

**7.1. Формирование листинга перекрестных ссылок**

Формирование листинга перекрестных ссылок происходит двумя этапами.

1. Формирование файла перекрестных ссылок (.CRF).

2. Формирование листинга перекрестных ссылок (.PRN).

Первый этап выполняется в Ассемблере, второй - в программе перекрестных ссылок.

**7.1.1. Формирование файла перекрестных ссылок**

Для формирования файла перекрестных ссылок в строке команды Ассемблера должен быть установлен переключатель /С

Пример: ASM = PROG/C

В следствии этого транслируется PROG.MAC и формируются PROG.REL (объектный файл) и PROG.CRE (файл перекрестных ссылок).

**7.1.2. Формирование листинга перекрестных ссылок**

Листинг перекрестных ссылок формируется программой перекрестных ссылок при прогоне файла .CRF.

Вызов осуществляется при помощи команды REF.

Программа заявляет о себе символом "звездочка"(\*). После этого должно быть введено имя файла .CRF:

Пример: REF =PROG

Формируется файл PROG.PRN.

Этот файл .PRN при использовании команд операционной системы может быть выдан на печать или на дисплей.

Кроме того, REF может использоваться на тех же устройствах вывода, с которыми работает Ассемблер.

Пример: REF LST:=PROG

выдает листинг на печатающее устройство. Файл на дискете не формируется.

REF TTY:=PROG

вывод осуществляется только на дисплей.

Для вывода можно задать также другое устройство.

Пример: REF B:=A:PROG

Дискета с PROG.CRF находится на устройстве А; PROG.PRN должен быть переписан на дискету, находящуюся на устройстве В.

После завершения REF заявляет о себе символ "звездочка"(\*) теперь можно вводить новое имя файла.

Возврат в операционную систему происходит с помощью ^C.

Файл вывода на печать может также получить другое имя и другой тип файла, в этом случае их следует задать.

Пример:

REF PROG.CRL=PROG

или

REF PROGREF=PROG

Верхняя строка команды формирует файл вывода на печать с именем PROG.CRL; нижняя строка формирует файл с обозначением PROG.PRN.

Типы файлов можно задавать, чтобы отличать листинг перекрестных ссылок от файла Ассемблера, выдаваемого на печать.

**7.2. Псевдооперации для управления листингами**

Дополнение для формирования перекрестной ссылки можно применять для программных разделов, но не для всей программы.

Для управления составлением списка и исключением из него перекрестных ссылок в исходном файле следует задавать псевдооперации .CREF и .XCREF. Они могут быть записаны в поле операций в любом месте программы.

.CREF и .XCREF не имеют аргументов.

.CREF формирование перекрестной ссылки

.CREF является стандартным условием: .CREF можно применять, если после .XCREF вновь должно быть разрешено формирование перекрестной ссылки.

Примечание

.CREF действует только в том случае, если в строке команды Ассемблера установлен переключатель /с.

.XCREF стирание перекрестной ссылки

.XCREF исключает действие .CREF .XCREF стирает перекрестную ссылку для программных разделов.

Поскольку ни .CREF ни .XCREF не действует, если в строке команды Ассемблера не установлен переключатель /С, то .XCREF применять нельзя, если желателен нормальный листинг (без перекрестной ссылки). Этого можно достичь также простым образом путем исключения переключателя /С из строки команды.