О Г Л А В Л Е Н И Е

1. АРХИТЕКТУРА ЦЕНТРАЛЬНОГО ПРОЦЕССОРА ПЭВМ

Основными аппаратными компонентами персональной ЭВМ являются:
- системный блок;
- клавиатура;
- монитор с дисплеем;
- накопители на магнитных дисках;
- принтер.

Кроме того, компьютер может быть укомплектован средствами асинхронной связи, графопостроителями, сканерами и т.п.

В системный блок входят:
- системная плата;
- блок питания;
- гнезда расширения для дополнительных плат.

Главным компонентом персонального компьютера ПК является центральный микропроцессор ЦП, который выполняет все операции обработки команд и данных. Центральный процессор ПК IBM и совместимых с ними может быть реализован на микросхемах фирмы Intel 8080, 8086, 80286 или 80386.

Центральный процессор может быть разделен на два логических блока (см. рис.1.1):
- блок исполнения EU;
- блок итерфейса шин BIU.

Итерфейс - это совокупность средств, обеспечивающих сопря(interface)жение устройств и программных модулей как на физическом, так и на логичес ком уровнях. Иногда этот термин употребляют так же и для характеристики средств доступа пользователя к вычислительной системе - "интерфейс пользователя". Интерфейс состоит из нес кольких контроллеров ввода/вывода,каждый из которых может содержать в себе одну или более микросхем СБИС (Chips).

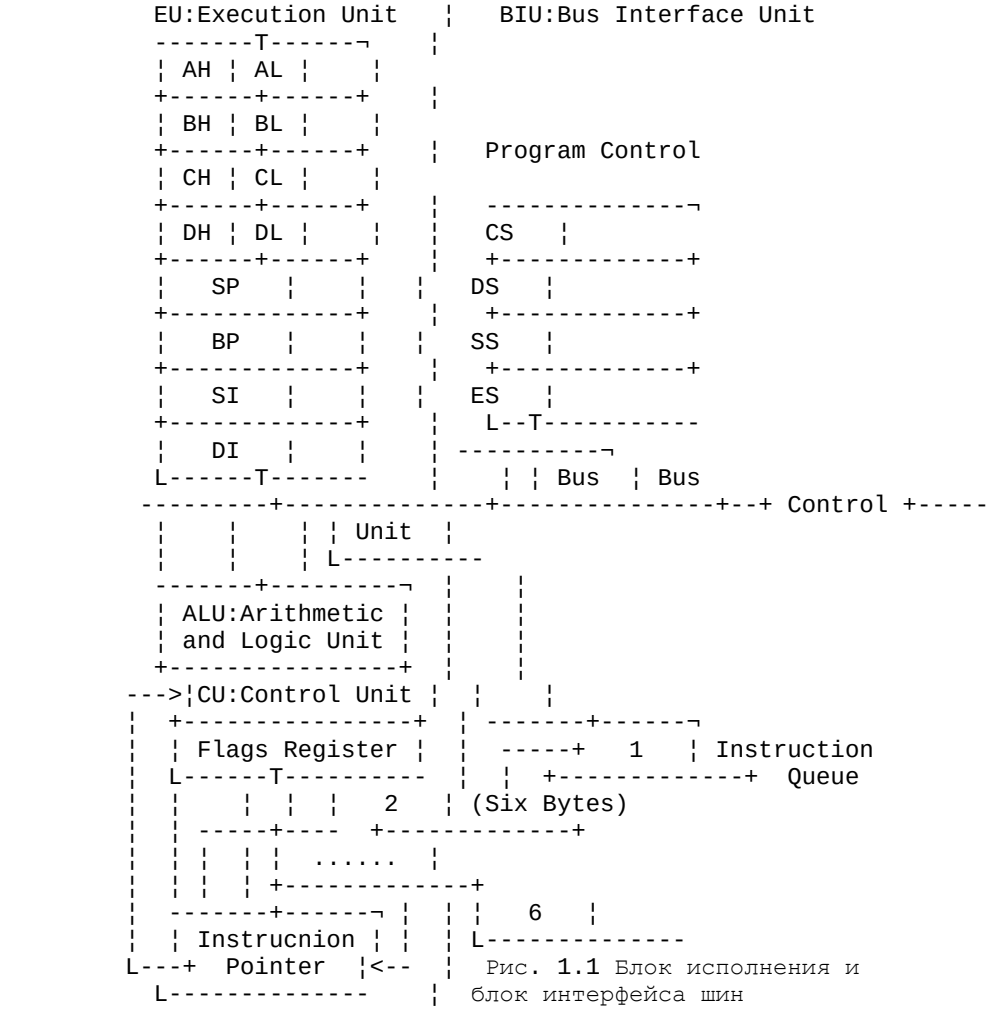

Рис. 1.1 Блок исполнения и блок интерфейса шин

Системная шина - это совокупность проводников, соединяющих(bus) ЦП с памятью и устройствами ввода/вывода.
Проводники шины объединены в три группы:
- линии данных;
- линии адреса;
- линии управления.

Взаимодействие между интерфейсом ввода/выводда и шиной осуществляется через регистры, называемые портами ввода/вывода.
Назначение блока EU - исполнение команд, а блок BIU выполняет передачу команд и данных в EU. В состав EU входят арифметико-логическое устройство ALU, устройство управления CU и десять регистров. Устройства блока EU обеспечивают обработку команд, выполнение арифметических и логических операций.Три части блока BIU - устройство управления шинами, блок очереди команд и регистры сегментов - предназначены для выполнения следующих функций:
- управление обменом данными с EU, памятью и внешними устройствами ввода/вывода;
- адресация 1 Мбайт памяти;
- выборка команд. Выборка команд осуществляется с помощью блока очереди команд Queue (шесть байтов), который позволяет выбирать команды с упреждением.

ПК имеет два вида внутренней памяти:
- постоянное запоминающее устройство ПЗУ (ROM - read only memory);
- оперативное запоминающее устройство ОЗУ (RAM - random acces memory).

Одна из основных задач ПЗУ обеспечить процедуру старта. В ПЗУ хранятся базовая система ввода/вывода BIOS, интерпретатор BASIC и некоторые служебные программы и таблицы, например, на чальный загрузчик, программы тестирования и т.п. ОЗУ используется для хранения программы и необходимых ей областей данных. Характерным для ПК является тот факт, что при выключении электропитания содержимое ОЗУ утрачивается. Наименьшей единицей данных, с которой работает компьютер, является бит (bit). Значением бита может быть либо ноль, либо единица. Группа из восьми битов называется байтом (Byte) и представляет собой наименьшую адресуемую единицу - ячейку. Биты в байте нумеруют справа налево цифрами 0...7.

```
  ----T---T---T---T---T---T---T---¬
  BYTE  ¦ 7 ¦ 6 ¦ 5 ¦ 4 ¦ 3 ¦ 2 ¦ 1 ¦ 0 ¦
  L---+---+---+---+---+---+---+---  .
```

Каждому из байтов присвоен уникальный адрес памяти, начиная с нулевого (самый младший адрес).Двухбайтовое поле образует шестнадцатиразрядное машинное слово (Word), биты в котором нумеруются от 0 до 15 справа налево.

```
    WORD
 ----T---T---T---T---T---T---T---T---T---T---T---T---T---T---T---¬
 ¦15 ¦14 ¦13 ¦ 12¦11 ¦10 ¦ 9 ¦ 8 ¦ 7 ¦ 6 ¦ 5 ¦ 4 ¦ 3 ¦ 2 ¦ 1 ¦ 0 ¦
 L---+---+---+---+---+---+---+---¦---+---+---+---+---+---+---+----
```

Байт с меньшим адресом считается младшим.В ПК принята двоичная система представления данных. Символьная информация кодируется в соответствии с кодом ASCII (Американский стандартный код для обмена информацией). Некоторые из его 256 символов заменяются на символы русского алфавита(кириллица). Числовые данные кодируются в соответствии с двоичной арифметикой. Отрицательные числа представляются в дополнительном коде. Для удобства представления данных используется шестнадцатиричная система счисления.

Принято двоичные числа сопровождать латинской буквой B или b, например, 101B, а шестнадцатиричные - буквой H или h на конце. Если число начинается с буквы, то обязательной является постановка нуля впереди, например, 0BA8H.

Регистры

Процессоры 8086/8088 имеют 14 шестнадцатиразрядных регистров, которые используются для управления исполнением команд, адресации и выполнения арифметических операций. Регистр, содержащий одно слово, адресуется по имени. Регистры сегмента CS, DS, SS и ES

Регистр CS - содержит начальный адрес сегмента кода. Этот адрес, суммированный со значением смещения в указателе команд (IP), определяет очередную выполняемую команду. Регистр DS - содержит начальный адрес сегмента данных. В простейшем случае этот адрес плюс смещение, указанное в инструкции, задает адрес в сегменте данных. Регистр SS - содержит начальный адрес регистра стека. Регистр ES - используется для адресации памяти в строковых операциях. И этом плане он похож на регистр DI. Регистр ES ини циализируется программой на ассемблере.

Регистры общего назначения AX, BX, CX и DX

Регистры общего назначения являются основными рабочими регистрами ассемблерных программ. Их отличает то, что к ним можно адресоваться одним словом или однобайтовым кодом. Левый байт считается старшим, а правый - младшим. Регистр AX - первичный аккумулятор, используется во всех операциях ввода/вывода, в некоторых операциях со строками и в некоторых арифметических операциях. Регистр BX - базовый регистр, единственный из регистров общего назначения,

используемый в индексной адресации. Кроме того, регистр BX используется при вычислениях.Регистр DX - регистр данных. Используется в некоторых операциях ввода/вывода, в операциях умножения и деления больших чисел совместно с регистром AX.

Любой из регистров общего назначения может быть использован для суммирования или вычитания 8- или 16-разрядных величин.

Регистры указателя SP и BP

Регистры указателя используются для обращения к данным в сегменте стека. Регистр SP - указатель стека. Используется для временного хранения адресов и иногда данных. Адресует стек аналогично регистру SS.Регистр BP - указатель базы. Обеспечивает ссылки на параметры (данные и адреса, передаваемые через стек).

Индексные регистры SI и DI

Индексные регистры используются для адресации, а также для выполнения операций сложения и вычитания.Регистр SI - индекс источника. Используется в некоторых операциях со строками или символами, аналогичен регистру DS. Регистр DI - индекс приемника. Используется в тех же операциях, что и регистр SI. Аналогичен регистру ES.

Регистр указателя команд IP

Регистр IP используется для выборки очередной команды программы с целью ее исполнения.

Регистр флагов Flags

Регистр Flags содержит девять активных битов (из 16), которые отражают состояние машины и результаты выполнения машинных команд.

```
Биты  : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Регистр :   OF DF IF TF SF ZF  AF  PF  CF

------------------------ПРИЗНАКИ---------------------------
```

OF (переполнения) - равен 1, если возникает арифметическое переполнение, т.е. когда об'ем результата превышает размер ячейки назначения

DF (направления) - устанавливается в 1 для автоматического декремента в командах обработки строк, и в 0 для инкремента

IF (разрешения прерывания) - прерывания разрешены, если IF=1.

Если IF=0, то распознаются лишь немаскированные прерывания

TF (трассировки) - если TF=1, то процессор переходит в состояние прерывания INT 3 после выполнения каждой команды

SF (знака) - SF=1, когда старший бит результата равен 1. Иными словами, SF=0 для положительных чисел, и SF=1 для отрицательных чисел

ZF (нулевого результата) - ZF=1, если результат равен нулю

AF (дополнительный признак переноса) - этот признак устанавливается в 1 во время выполнения команд десятичного сложения и вычитания при необходимости выполнения переноса или заема между полубайтами

PF (четности) - этот признак устанавливается в 1,если результат имеет четное число единиц

CF (переноса) - этот признак устанавливается в 1,если имеет место перенос или заем из старшего бита результата; он полезен для произведения операций над числами длиной в несколько слов, которые сопряжены с переносами и заемами из слова в слово.

Сегменты

Сегментом называется область памяти, которая начинается на границе параграфа, то есть в любой точке, адрес которой кратен 16 (восемь младших битов равны нулю). Существуют три основных типа сегментов:

- сегмент кода - содержит машинные команды, адресуется регистром CS;
- сегмент данных - содержит данные, то есть константы и рабочие области, необходимые программе. Адресуется регистром DS;
- сегмент стека - содержит адреса возврата в точку вызова подпрограмм.Адресуется регистром SS.

Каждый из упомянутых регистров содержит адрес начала сегмента (базовый адрес). В программе все адреса записаны относительно начала сегмента, и они определяются как смещение (offset) от начала сегмента. Двухбайтовое смещение (одно слово) может принимать значение от 0000 до 0FFFFh. Для того, чтобы выполнить обращение по любому адресу процессор выполняет суммирование адреса, записанного в регистре сегмента, со смещением. При этом, содержимое регистра сдвигается на четыре двоичных разряда влево. Результирующий адрес занимает 20 позиций, что и позволяет адресовать 1 Мбайт памяти.

Пример.

```
Содержимое DS 045F
+
Смещение    0032
------
```


```
    Исполнительный 04622
    адрес (EA)
    Примечание:    Адреса шестнадцатиричные.

     РЕЖИМЫ АДРЕСАЦИИ
     Стандартный сег-
  Режим адресации    Формат адреса    ментный регистр
  ===============    =============    =================
  Регистровая   регистр    Нет
  Непосредственная данные Нет
  Косвенная регистровая  [BX]  DS
    [BP]  SS
    [DI]  DS
    [SI]  DS
  По базе со смещением метка [BX]    DS
  метка [BP]    SS
  Прямая метка [DI]    DS
  с индексированием   метка [SI]    DS
  По базе   метка [BX + SI]  DS
  с индексированием  метка [BX + DI]  DS
     метка [BP + SI]  SS
     метка [BP + DI]  SS
  Строковые команды  исходный адрес  DS:SI
     место назначения ES:DI
 * Метка [...] может быть заменена на [смещение + ...]. Следовательно, запись [24 + BX]
будет означать адрес 24+BX.
    ============================================================

    Примечание:  Многие строковые команды используют ES:DI как место назначения, а DS:SI как
адрес источника.
    Эта таблица приводит количество временных тактов, требуемых для вычисления
исполнительного адреса на микропроцессоре 8088. Микропроцессор 80*8* производит эти
вычисления быстрее, так что эта таблица содержит "самые медленные" данные.
 ------------------------------------------------------------
  Способы адресации Такты для 8088   Пример
 ------------------------------------------------------------
  Смещение  6  MOV AX,ADDR
  Косвенная регистровая  5  MOV AX,[BX]
   BX, SI, DI
  По базе или с индек-   9  MOV AX,ADDR[BP]
  сированием
  + смещение
 BX+смещение, BP+смещение
 SI+смещение, DI+смещение

 По базе или с индексированием (без смещения)
   BP+DI,  BX+SI    7  MOV AX,[BP+DI]
   BP+SI,  BX+DI    8  MOV AX,[BX+DI]

 По базе с индексированием + смещение
   BP+DI+смещение    11 MOV AX,ADDR[BP+DI]
   BX+SI+смещение

   BP+SI+смещение    12 MOV AX,ADDR[BP+SI]
   BX+DI+смещение
 ------------------------------------------------------------

 Примечание:
Прибавьте 2 такта в случае вычисления исполнительного адреса из другого сегмента.
Каждое обращение к памяти занимает дополнительные 4 такта.Поле обращений в описаниях команд
содержит информацию о количестве обращений к памяти для каждой команды.

     Стеки
Во многих случаях программе требуется временно запомнить
```

информацию, а затем считывать ее в обратном порядке. Эта проблема в ПК решена посредством реализации стека LIFO ("последний пришел - первый ушел"), называемого также стеком включения/извлечения (stack - кипа, например, бумаг). Наиболее важное использование стека связано с процедурами. Стек обычно рассчитан на косвенную адресацию через регистр SP - указатель стека. При включении элементов в стек производится автоматический декремент указателя стека, а при извлечении - инкремент, то есть стек всегда "растет" в сторону меньших адресов памяти. Адрес последнего включенного в стек элемента называется вершиной стека (TOS).
Физический адрес стека формируется из SP и SS или BP и SS,причем SP служит неявным указателем стека для всех операций включения и извлечения, а SS - сегментным регистром стека. Содержимое SS является самым младшим адресом (границей) области стека и называется базой стека. Первоначальное содержимое SP считается наибольшим смещением, которого может достигать стек. Регистр BP предназначен, главным образом, для произвольных обращений к стеку.

Прерывания

Иногда необходимо выполнить одну из набора специальных процедур, если в системе или в программе возникают определенные условия, например, нажата клавиша на клавиатуре. Действие, стимулирующее выполнение одной из таких процедур, называется прерыванием, поскольку основной процесс при этом приостанавливается на время выполнения этой процедуры. Существует два общих класса прерываний: внутренние и внешние. Первые инициируются состоянием ЦП или командой, а вторые - сигналом, подаваемым от других компонентов системы. Типичные внутренние прирывания: деление на нуль, переполнение и т.п., а типичные внешние - это запрос на обслуживание со стороны какого-либо устройства ввода/вывода. Переход к процедуре прерывания осуществляется из любой программы, а после выполнения процедуры прерывания обязательно происходит возврат в прерванную программу. Перед обращением к процедуре прерывания должно быть сохранено состояние всех регистров и флагов, используемых процедурой прерывания, а после окончания прерывания эти регистры должны быть восстановлены.Некоторыми видами прерываний управляют флажки IF и TF, которые для восприятия прерываний должны быть правильно установлены. Если условия для прерывания удовлетворяются и необходимые флажки установлены, то микропроцессор завершает текущую команду, а затем реализует последовательность прерывания:

- текущее значение регистра Flags включается в стек (эквивалентно команде pushf);
- текущее значение кодового сегмента CS включается в стек(эквивалентно команде push CS);
- текущее значение указателя инструкции IP включается в стек (эквивалентно команде push IP);
- сбрасываются флажки IF и TF. Новое содержимое IP и CS определяет начальный адрес выполняемой процедуры прерывания (обслуживание прерывания). Возврат в прерванную программу осуществляется командой, которая извлекает из стека содержимое для:
- IP (эквивалентно pop IP);
- CS (эквивалентно pop CS);
- Flags (эквивалентно popf).

Двойное слово, в котором находится новое содержимое IP и CS, называется указателем прерывания, или вектором. Каждому типу прерывания назначено число из диапазона 0...255, и адрес указателя прерывания находится путем умножения номера типа на четыре.

2. СИСТЕМА КОМАНД ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Центральный процессор содержит следующий набор команд.

ПЕРЕСЫЛКА ДАННЫХ

```
MOV   PUSH   POP   XCHG   OUT   IN
XLAT  LEA    LDS   LES    LAHF  SAHF
PUSHF POPF
```

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

```
ADD   ADC   INC   SUB    SBB   DEC
CMP   MUL   IMUL  DIV    IDIV  NEG
AAA   DAA   AAS   DAS    AAM   AAD
CBW   CWD
```

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

```
NOT   SHL/SAL  SHR   SAR    ROL    ROR
RCL   RCR    AND   TEST   OR XOR
```

ОБРАБОТКА БЛОКОВ ДАННЫХ

```
REP   MOVS   CMPS   SCAS   LODS   STOS
CMPB  CMPSW   LODSB  LODSW  MOVSB   MOVSB
MOVSW  REPE   REPNZ  SCASB  SCASW   STOSB
STOSW
```

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ
CALL JMP RET

КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА
JA JLE JNL JS
JAE JNA JNLE JZ
JB JNAE JNO LOOP
JBE JNB JNP LOOPE
JC JNBE JNS LOOPNE
JCXZ JNC JNZ LOOPNZ
JE JNE JO LOOPZ
JG JNE JP JGE
JL JPE JNGE JPO

КОМАНДЫ ПРЕРЫВАНИЯ
INT INTO IRET

УПРАВЛЕНИЕ СОСТОЯНИЕМ ПРОЦЕССОРА
CLC CMC STC CLD STD CLI
STI HLT WAIT ESC LOCK NOP

Подробное описание каждой из команд приводится ниже. Каждая запись этого списка содержит информацию о том, какие признаки из регистра FLAGS процессора 8088 и как изменяются. Поскольку регистр FLAGS содержит всего 9 признаков, эту информацию можно выдать в компактной форме, например:

Признаки: O D I T S Z A P C
0 * * ? * 0 , где приняты следующие обозначения признаков:
? - неопределен после операции;
* - изменился в зависимости от результатов выполнения команды;
0 - всегда сброшен;
1 - всегда установлен.

Диаграмма времени иллюстрирует синхронизацию для 8088. Т.к. процессоры 80*8* выполняют команды за меньшее число тактов, чем 8088, эта диаграмма отражает наиболее медленный вариант. Операнды В этом поле приводится список возможных операндов и способы адресации для каждой команды.

Такты Число временных циклов (тактов), необходимых для выполнения команды на 8088. Вычисление исполнительного адреса (EA)требует дополнительного времени, как показано в таблице EA. Обращения Число обращений к памяти. Каждое обращение к памяти длится 4 такта. Байты Число байтов в команде.

Примечание:

Дополнительное время, требуемое для изменения указателя команд и выборки следующей команды после команд, передающих управление (таких как JMP или CALL), уже учтено при составлении диаграммы времени. В случае команд, осуществляющих условную передачу управления (таких как JZ) приводится две диаграммы времени; меньшее число тактов соответствует случаю, когда переход не осуществляется.

2.1 Команды пересылки данных

2.1.1 IN Ввод байта или слова

Признаки не меняются.
Команда: IN accumulator,port.
Логика : accumulator = (port).

IN передает байт или слово из заданного порта port в AL или AX. Адрес порта может определяться как непосредственным байтовым значением (в диапазоне 0-255), так и с использованием косвенной адресации по регистру DX.

Операнды байт(слово)	ТактыОбращения	Байты	Пример
accumulator,непоср.8	10(14)	1 2	IN AL,45h
accumulator,DX	8(12)	1 1	IN AX,DX

Примечания :

Следует указать на то, что аппаратная часть не использует порты от F8h до FFh для ввода/вывода, поскольку они зарезервированы для контроля за внешним процессором и для других возможных расширений процессора в будущем.

2.1.2 LAHF Загрузка AH из регистра флагов

Признаки не меняются.

Команда: LAHF .

Логика : биты регистра AH : 7 6 4 2 0

биты регистра признаков FLAGS : S Z A P C .

Команда LAHF копирует пять признаков процессора 8080/8086(признаки знака, нулевого результата, вспомогательного переноса, четности и переноса) в биты регистра AH с номерами 7, 6, 4, 2, 0 соответственно. Сами признаки при выполнении этой команды не меняются.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	4	-	1	LAHF

Примечания :

Эта команда используется, в основном, в целях обеспечения совместимости микропроцессоров семейств 8080/8085 и 8086. После выполнения этой команды значения битов регистра AH с номерами 1, 3 и 5 не определены.

2.1.3 LDS Загрузка указателя с использованием DS

Признаки не меняются.

Команда: LDS destination,source.

Логика : DS = (source)

destination = (source + 2) .

Команда LDS загружает в два регистра 32-битный указатель, расположенный в памяти по адресу source. При этом старшее слово заносится в сегментный регистр DS, а младшее слово - в базовый регистр destination. В качестве операнда destination может выступать любой 16-битный регистр, кроме сегментных.

Операнды	Такты	Обращения	Байты	Пример
регистр16,память	32	24+EA	2	2-4 LDS DI,32_POINTER

Примечания :

Команда LES, загрузка указателя с использованием ES, выполняет те же действия, что и LDS, но использует при этом вместо регистра DS регистр ES.

2.1.4 LEA Загрузка исполнительного адреса

Признаки не меняются.

Команда: LEA destination,source.

Логика : destination = Addr(source).

Команда LEA присваивает значение смещения (offset) операнда source (а не его значение !) операнду destination. Операнд source должен быть ссылкой на память, а в качестве операнда destination может выступать любой 16-битный регистр, кроме сегментных.

Операнды	Такты	Обращения	Байты	Пример
регистр16,память	32	2+EA	-	2-4 LEA BX,MEM_ADDR

Примечания :

Эта команда имеет то преимущество по сравнению с использованием оператора OFFSET в команде MOV, что операнду source можно иметь индексы. Например, следущая строчка не содержит ошибок :

LEA BX,TABLE[SI] в то время, как строка MOV BX,OFFSET TABLE[SI] ошибочна, так как оператор OFFSET вычисляется во время ассемблирования, а указанный адрес не будет известен до тех пор, пока программа не будет запущена на счет.

2.1.5 LES Загрузка указателя с использованием ES

Признаки не меняются

Команда: LES destination,source.

Логика : ES = (source)

destination = (source + 2).

Команда LES загружает в два регистра 32-битный указатель, расположенный в памяти по адресу source. При этом высшее слово заносится в сегментный регистр ES, а низшее слово - в базовый регистр destination. В качестве операнда destination может выступать любой 16-битный регистр, кроме сегментных.

Операнды	Такты	Обращения	Байты	Пример
регистр16,память	32	24+EA	2	2-4 LES DI,STR_ADDR

Примечания :

Команда LDS, загрузка указателя с использованием DS, вы полняет те же действия, что и LES, но использует при этом вместо регистра ES регистр DS.

2.1.6 MOV Пересылка (байта или слова)

Признаки не меняются.

Команда: MOV destination,source.

Логика: destination = source .

MOV пересылает по адресу destination байт или слово, находящееся по адресу source.

Операнды	Такт	Обраще- ния	Байты	Пример
регистр,регистр	2	-	2	MOV BX,SI
регистр,непоср.операнд	4	-	2-3	MOV CX,128
аккумулятор,память	10(14)	1	3	MOV AL,MEM_SOURCE
регистр,память	8(12)+EA	1	2-4	MOV DI,[DX]
память,регистр	9(13)+EA	1	2-4	MOV BETA,DI
память,непоср.операнд	10(14)+EA	1	3-6	MOV GAMMA,16h
память,аккумулятор	10(14)	1	3	MOV MEM_DEST,AX
сегм.регистр,регистр16	2	-	2	MOV DS,BX
сегм.регистр,память16	8(12)+EA	1	2-4	MOV DS,SEGMENT_VAL
регистр16,сегм.регистр	2	-	2	MOV BP,SS
память,сегм.регистр	9(13)+EA	1	2-4	MOV SEGMENT_VAL,DS

2.1.7 OUT загрузка в порт

Признаки не меняются.

Команда: OUT port,accumulator.

Логика : (port) = accumulator.

OUT передает байт или слово из AL или AX в заданный порт. Адрес порта может определяться как непосредственным байтовым значением (в диапазоне 0-255), так и с использованием косвенной адресации по регистру DX.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
непоср.8,accumulator	10(14)	1	2	OUT 254,AX
DX,accumulator	8(12)	1	1	OUT DX,AL

Примечания :

Следует указать на то, что аппаратная часть не использует порты от F8h до FFh для ввода/вывода, поскольку они зарезервированы для контроля за внешним процессором и для других возможных расширений процессора в будущем.

2.1.8 POP выборка слова из стека

Признаки не меняются.

Команда: POP destination.

Логика : destination = (SP)

SP = SP + 2 .

Команда POP пересылает слово из верхушки стека по адресу destination, затем увеличивает указатель стека SP на 2, чтобы он указывал на новую верхушку стека.

Операнды	Такты	Обращения	Байты	Пример
регистр	12	1	1	POP CX
сегм.регистр(кроме CS)	12	1	1	POP ES
память	25 + EA	2	2-4	POP VALUE

```
 ------------------------------------------------------------
    2.1.9  POPF пересылка слова из стека в регистр FLAGS
 Признаки: O D I T S Z A P C
      r r r r r r r r r
 Команда: POPF .
 Логика : flag-register = (SP)
  SP = SP + 2 .
```

Команда POPF пересылает слово из верхушки стека в регистр FLAGS, изменяя значения всех признаков, затем увеличивает указатель стека SP на 2, чтобы он указывал на новую верхушку стека.

```
 ------------------------------------------------------------
 Операнды     Такты   Обращения   Байты   Пример
 нет операндов   12      1      1    POPF
 ------------------------------------------------------------

     2.1.10  PUSH загрузка слова в стек
 Признаки не меняются.
 Команда: PUSH source.
 Логика : SP = SP - 2
  (SP) = source .
```

Команда PUSH уменьшает значение указателя стека SP на 2,затем пересылает операнд в новую верхушку стека. Операндом source не может быть 8-битный регистр.

```
 ------------------------------------------------------------------
 Операнды         Такты  Обращения Байты  Пример
 регистр          15     1    1   PUSH BX
 сегм.регистр(кроме CS) 14     1    1   PUSH ES
 память         24 + EA   2    2-4   PUSH PARAMETERS
 ------------------------------------------------------------------
```

Примечание :

Даже если source указывает на байт, в стек пересылается целое слово. Микропроцессоры 80286 и 80786 перешлют в стек не те же значения, что микропроцессоры 8086/8088, если использовать команду PUSH SP. Микропроцессоры 80286 и 80386 перешлют старое значение SP, а 8086/8088 - новое значение SP в верхушку стека. Поэтому, в целях получения одинаковых результатов для всех микропроцессоров, используйте следующую последовательность команд:

```
   PUSH  BP
   MOV   BP, SP
   XCHG  BP, [SP]
```

Эта последовательность команд соответствует выполнению команды PUSH SP на микропроцессорах 8088/8086.

```
  2.1.11  PUSHF  загрузка содержимого регистра FLAGS в стек
 Признаки не меняются.
 Команда: PUSHF .
 Логика : SP = SP - 2
  (SP) = flag-register .
```

Команда PUSHF уменьшает значение указателя стека SP на 2, затем пересылает слово из регистра FLAGS в верхушку стека.

```
 ------------------------------------------------------------
 Операнды     Такты   Обращения   Байты   Пример
 нет операндов   14      1      1    PUSHF
 ------------------------------------------------------------
    2.1.12  SAHF загрузка регистра AH в регистр флагов
 Признаки: O D I T S Z A P C .
 Команда: SAHF .
 Логика : биты регистра признаков FLAGS : S Z A P C
  биты регистра AH :          7 6 4 2 0 .
```

Команда SAHF копирует биты регистра AH с номерами 7, 6, 4, 2 и 0 в регистр FLAGS, заменяя текущие значения признаков знака, нулевого результата, вспомогательного признака переноса,четности и переноса.

```
 ------------------------------------------------------------
 Операнды    Такты   Обращения   Байты   Пример
 нет операндов   4      -      1    SAHF
 ------------------------------------------------------------
```

Примечания :

Эта команда используется, в основном, в целях обеспечения совместимости микропроцессоров семейств 8080/8085 и 8086. После выполнения этой команды признаки переполнения, направления, прерывания и трассировки не изменяются.

2.1.13 XCHG обмен значениями

Признаки не меняются.
Команда: XCHG destination,source .
Логика: destination <--> source .

Команда XCHG обменивает значения своих операндов, которые могут быть байтами или словами.

Операнды	Такты	Обращения	Байты	Пример
байты (слова)				
память,регистр	17(25)+EA	2	2-4	LOCK XCHG SEM,DX
регистр,регистр	4	-	2	XCHG CL,DL
аккумулятор,регистр	16	3 -	1	XCHG AX,SI

Примечание:
Эта команда в паре с префиксом LOCK полезна, в частности,при реализации семафоров для управления разделенными ресурсами.

2.1.14 XLAT кодирование AL по таблице

Признаки не меняются.
Команда: XLAT translate-table .
Логика: AL = (BX + AL) .
Команда XLAT переводит байт, согласно таблице преобразований. Указатель 256-байтовой таблицы преобразований находится в BX. Байт, который нужно перевести, расположен в AL. После выполнения команды XLAT байт в AL заменяется на байт, смещенный на AL байтов от начала таблицы преобразований.

Операнды	Такты	Обращения	Байты	Пример
translate-table	11	1	1	XLAT SINE_TABLE

Примечания:
Таблица преобразований может содержать менее 256 байтов.
Операнд, т.е. translate-table, является необязательным, поскольку указатель таблицы должен быть загружен в BX еще до начала выполнения команды.
Следующий пример иллюстрирует перевод десятичного числа (от 0 до 15) в соответствующую "цифру" шестнадцатеричной системы счисления:

```
LEA  BX,HEX_TABLE    ;указатель таблицы засылаем в BX,
MOV  AL,DECIMAL_DIGIT  ;а переводимую цифру - в AL
XLAT  HEX_TABLE  ;переводим
  .    ;теперь в AL находится ASCII-код
  .    ;соответствующей цифры
  .    ;шестнадцатеричной системы
HEX_TABLE   DB   '0123456789ABCDEF'
```

2.2 Арифметические операции

2.2.1 AAA ASCII-коррекция при сложении

Команда: AAA .
Признаки: O D I T S Z A P C
? ? ? * ? * ? * .
Логика: if (AL & 0Fh) > 9 or (AF = 1) then
AL = AL + 6
AH = AH + 1
AF = 1; CF = 1
else
AF = 0; CF = 0
AL = AL & 0Fh .
Переводит число, записанное в младшем полубайте аккумулятора AL в число, представленное в неупакованном формате в двоично-десятичном коде (старший полубайт AL содержит нули).

Операнды	Такты	Обращения	Байты	Пример
нет операндов	4	-	1	AAA

```
 ---------------------------------------------------------------
```

Если младший полубайт в AL больше, чем 9 или дополнительный признак переноса AF установлен (=1), то эта команда преобразовывает содержимое AL в его неупакованный двоично-десятичный код путем прибавления числа 6 к AL, увеличения AH на 1 и установки признака переноса CF и дополнительного признака переноса AF. Старший полубайт AL сбрасывается.

Примечания:

В неупакованном двоично-десятичном коде каждому байту соответствует одна цифра, и AH содержит более значащую цифру, а AL - менее значащую.

2.2.2 AAD ASCII-коррекция при делении

```
Признаки: O D I T S Z A P C
          ?   * * ? * ?
Команда:    AAD .
Логика: AL = AH * 10 + AL
 AH = 0 .
```

AAD переводит двухзначное число, представленное в неупакованном формате в регистре AX, из двоично-десятичного кода в двоичный, готовя число к выполнению операций деления DIV или IDIV, которые обрабатывают двоичные числа быстрее.

```
 ---------------------------------------------------------------
 Операнды    Такты    Обращения    Байты    Пример
 нет операндов  60       -       2     AAD
 ---------------------------------------------------------------
```

AAD преобразует числитель в AL таким образом, чтобы результат деления был представлен числом в двоично-десятичном коде. Для того, чтобы последующая операция деления DIV давала правильный результат, необходимо, чтобы AH=0. После деления частное заносится в AL, а остаток - в AH.

Примечания:

В неупакованном двоично-десятичном коде каждому байту соответствует одна цифра, и AH содержит более значащую цифру, а AL - менее значащую.

2.2.3 AAM ASCII-коррекция при умножении

```
Признаки: O D I T S Z A P C
          ?   * * ? * ? .
Команда: AAM .
Логика: AH = AL / 10
 AL = AL MOD 10 .
```

Эта команда корректирует результат предшествующего умножения двух операндов, представленных в неупакованном двоично-десятичном коде. Двузначное неупакованное число берется из AX,проводится корректировка, и результат возвращается в AX. Для
того, чтобы эта команда дала верный результат, необходимо, чтобы старшие полубайты обоих сомножителей были равны нулю.

```
 ---------------------------------------------------------------
 Операнды    Такты    Обращения    Байты    Пример
 нет операндов  83       -      1     AAM
 ---------------------------------------------------------------
```

Примечание: В неупакованном двоично-десятичном коде каждому байту соответствует одна цифра, и AH содержит более значащую цифру, а AL - менее значащую.

2.2.4 AAS ASCII-коррекция при вычитании

```
Признаки: O D I T S Z A P C
          ?   ? ? * ? *
Команда:    AAS .
Логика: if (AL & 0Fh) > 9 or (AF = 1) then
   AL = AL - 6
   AH = AH - 1
   AF = 1;  CF = 1
 else
   AF = 0;  CF = 0
   AL = AL & 0Fh .
```

Эта команда корректирует результат предшествующего вычитания двух операндов, представленных в неупакованном двоично-десятичном коде, засчет перевода содержимого AL в двоично-десятичный код. Операнд назначения (destination) команды вычитания должен быть специфицирован так же, как AL. Старший полубайт AL всегда равен нулю.

```
 ---------------------------------------------------------------
 Операнды    Такты    Обращения    Байты    Пример
```

нет операндов 4 - 1 AAS

Примечание: В неупакованном двоично-десятичном коде каждому байту соответствует одна цифра, и AH содержит более значащую цифру, а AL - менее значащую.

2.2.5 ADC Сложение с переносом

Признаки: O D I T S Z A P C
* * * * * .

Команда: ADC destination,source.

Логика: destination = destination + source + CF.

ADC складывает операнды, прибавляет единицу, если признак переноса CF установлен (CF=1), и засылает сумму по назначению (destination). Оба операнда могут быть байтами или словами, и оба операнда могут быть двоичными числами со знаком или без знака.

Операнды	Такты	Обращения	Байты	Пример
(байты) (слова)				
регистр,регистр	3	-	2	ADC BX,SI
регистр,непоср.операнд	4	-	3-4	ADC CX,128
аккумулятор,непоср.оп.	4	-	2-3	ADC AL,10
регистр,память	9(13)+EA	1	2-4	ADC DX,RESULT
память,регистр	16(24)+EA	2	2-4	ADC BETA,DI
память,непоср.операнд	17(25)+EA	2	3-6	ADC GAMMA,16h

Примечание: Команда ADC полезна при сложении чисел,которые занимают больше 16 бит, т.к. она прибавляет перенос от предыдущей операции.

2.2.6 ADD Сложение

Признаки: O D I T S Z A P C
* * * * * .

Команда: ADD destination,source .

Логика: destination = destination + source .

ADD складывает операнды и засылает сумму по назначению (destination). Оба операнда могут быть байтами или словами, и оба операнда могут быть двоичными числами со знаком или без знака.

Операнды	Такты	Обращения	Байты	Пример
(байты) (слова)				
регистр,регистр	3	-	2	ADD BX,SI
регистр,непоср.операнд	4	-	3-4	ADD CX,128
аккумулятор,непоср.оп.	4	-	2-3	ADD AL,10
регистр,память	9(13)+EA	1	2-4	ADD DI,[DX]
память,регистр	16(24)+EA	2	2-4	ADD BETA,DI
память,непоср.операнд	17(25)+EA	2	3-6	ADD GAMMA,16h

Примечания:

При сложении чисел, занимающих более 16 бит, полезна команда ADC, т.к. она прибавляет перенос от предыдущей операции.

2.2.7 CBW Преобразование байта в слово

Признаки не меняются.

Команда: CBW.

Логика:
```
if (AL < 80h) then
  AH = 0
   else
  AH = FFh .
```

CBW расширяет бит знака регистра AL в регистр AH. Эта команда переводит байтовую величину со знаком в эквивалентное ей слово со знаком.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	CBW

Примечания :

Эта команда положит AH равным 0FFh, если бит знака регистра AL (т.е. седьмой бит) установлен; если же седьмой бит AL не установлен, то в AH заносятся нули. Эта команда полезна для преобразования байта в слово, в первую очередь, с целью выполнения операции деления байтов.

2.2.8 CMP Сравнение
Признаки: O D I T S Z A P C
* * * * * * .
Команда: CMP destination,source .
Логика: Установка признаков в соответствии с результатом.
(destination - source)
CMP сравнивает два числа, вычитая операнд source из операнда destination, и изменяет значения признаков. CMP не изменяет сами операнды. Операндами могут быть байты или слова .

Операнды	Такты	Обращения	Байты	Пример
байты (слова)				
регистр,регистр	3	-	2	CMP BX,SI
регистр,непоср.операнд	4	-	3-4	CMP CX,128
аккумулятор,непоср.оп.	4	-	2-3	CMP AL,02h
регистр,память	9(13)+EA	1	2-4	CMP DI,[DX]
память,регистр	9(13)+EA	1	2-4	CMP BETA,DI
память,непоср.операнд	10(14)+EA	1	3-6	CMP GAMMA,16h

2.2.9 CWD Преобразование слова в двойное слово
Признаки не меняются.
Команда: CWD .
Логика:
```
if (AX < 8000h) then
 DX = 0
  else
 DX = FFFFh .
```
CWD расширяет бит знака регистра AX на весь регистр DX. Эта команда генерирует двойное слово, эквивалентное числу со знаком, находящемуся в регистре AX.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	5	-	1	CWD

Примечания:
Эта команда положит DX равным 0FFFFh, если бит знака(15-ый бит) регистра AX установлен, и равным 0, если бит знака AX сброшен.

2.2.10 DAA Десятичная коррекция при сложении
Признаки: O D I T S Z A P C
? * * * * *
Команда: DAA .
Логика:
```
if (AL & 0Fh) > 9 or (AF = 1) then
 AL = AL + 6
 AF = 1
   else AF = 0
   if (AL > 9Fh) or (CF = 1) then
 AL = AL + 60h
 CF = 1
   else CF = 0 .
```
Команда DAA корректирует результат предшествующего ей сложения двух упакованных десятичных операндов (заметьте, что результат должен находиться в AL). Эта команда изменяет содержимое AL так, чтобы AL содержал пару упакованных десятичных цифр.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	4	-	1	DAA

Примечания:
В упакованном двоично-десятичном коде каждому полубайту соответствует одна цифра; менее значащую цифру содержит младший полубайт. После деления или умножения чисел, записанных в упакованном двоично-десятичном коде, производить коррекцию нельзя.Поэтому, если Вы хотите воспользоваться операцией деления или умножения, то лучше использовать числа в неупакованном двоично десятичном коде.См., например, описание команды AAM
(ASCII-коррекция при умножении).

2.2.11 DAS десятичная коррекция при вычитании
Признаки: O D I T S Z A P C
? * * * * * .


```
Команда:  DAS .
Логика:   if (AL & 0Fh) > 9 or (AF = 1) then
 AL = AL - 6
 AF = 1
   else AF = 0
   if (AL > 9Fh) or (CF = 1) then
 AL = AL - 60h
 CF = 1
   else CF = 0 .
```

Команда DAS корректирует результат предшествующего ей вычитания двух упакованных десятичных операндов (заметьте, что результат должен находиться в AL). Эта команда изменяет содержимое AL так, чтобы AL содержал пару упакованных десятичных цифр.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	4	-	1	DAS

Примечания:

В упакованном двоично-десятичном коде каждому полубайту соответствует одна цифра; менее значащую цифру содержит младший полубайт. После деления или умножения чисел, записанных в упакованном двоично-десятичном коде, производить коррекцию нельзя. Поэтому, если Вы хотите воспользоваться операцией деления или умножения, то лучше использовать числа в неупакованном двоично-десятичном коде.См., например, описание команды AAM (ASCII-коррекция при умножении).

2.2.12 DEC Декремент

```
Признаки: O D I T S Z A P C
          *   * * * * * .
Команда: DEC destination.
Логика:  destination = destination - 1
```

Эта команда отнимает от операнда destination единицу. Операнд destination, который может быть словом или байтом, интерпретируется как двоичное число без знака.

Операнды	Такты	Обращения	Байты	Пример
Байт(слово)				
Регистр16	2	1		DEC BX
Регистр8	3	2		DEC BL
память	15(23)+EA	2	2-4	DEC MATRIX[SI]

Примечания :

При выполнении этой команды признак переноса CF не изменяется, поэтому если Вы хотите декрементировать число, записанное несколькими словами, то лучше воспользоваться командами SUB и SBB.

2.2.13 DIV Деление без учета знака

```
Признаки: O D I T S Z A P C
          ?   ? ? ? ? ? .
Команда:   DIV source .
Логика:  AL = AX / source  ;операнд source - байт
  AH = remainder
or
  AX = DX:AX / source ;операнд source - слово
  DX = remainder .
```

Эта команда выполняет деление без учета знака. Если операнд source является байтом, то DIV делит значение слова в AX на операнд source, засылая частное в AL и остаток (remainder) в AH. Если же операнд source является словом, то DIV делит значение двойного слова из DX:AX на операнд source, засылая частное в AX и остаток в DX.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
регистр8	0-90	-	2	DIV BL
регистр16	4-162	-	2	DIV BX
память8	(86-96)+EA	1	2-4	DIV VYUP
память16	(154-172)+EA	1	2-4	DIV NCONQUER[SI]

Примечания:

Если результат слишком велик и не умещается в AL (соотв.AX), то генерируется прерывание INT 0 (деление на ноль), и тогда частное с остатком не определены. Когда генерируется

прерывание INT 0, то для процессоров 80286 и 80386 запоминаемое значение CS:IP указывает на неудавшуюся команду (т.е. на команду DIV). Для процессоров 8088/8086 CS:IP указывает, однако, на команду, следующую за неудавшейся командой DIV.

2.2.14 IDIV Деление с учетом знака

```
Признаки: O D I T S Z A P C
          ?   ? ? ? ? ? .
Команда: IDIV source .
Логика:  AL = AX / source  ;операнд source - байт
  AH = remainder
or
  AX = DX:AX / source ;операнд source - слово
  DX = remainder .
```

Эта команда выполняет деление с учетом знака. Если операнд source является байтом, то IDIV делит значение слова в AX на операнд source, засылая частное в AL и остаток (remainder) в AH. Если же операнд source является словом, то IDIV делит значение двойного слова из DX:AX на операнд source, засылая частное в AX и остаток в DX.

Операнды байт(слово)	Такты	Обращения	Байты	Пример
регистр8	101-112	-	2	IDIV CL
регистр16	165-184	-	2	IDIV CX
память8	(107-118)+EA	1	2-4	IDIV BYTE[SI]
память16	(175-194)+EA	1	2-4	IDIV [BX].WORD_ARRAY

Примечания:

Если результат слишком велик и не умещается в AL (соотв.AX), то генерируется прерывание INT 0 (деление на ноль), и тогда частное с остатком не определены. Микропроцессоры 80286 и 80386 могут в качестве частного после выполнения этой команды генерировать наибольшее (по абсолютной величине) негативное число (80h или 8000h), однако, 8088/8086 сгенерируют в такой ситуации прерывание INT 0. Когда генерируется прерывание INT 0,то для процессоров 80286 и 80386 запоминаемое значение CS:IP указывает на неудавшуюся команду (т.е. на команду IDIV).Для процессоров 8088/8086 CS:IP указывает, однако, на команду, следующую за неудавшейся командой IDIV.

2.2.15 IMUL Умножение с учетом знака

```
Признаки: O D I T S Z A P C
          *   ? ? ? ? * .
Команда: IMUL source .
Логика:  AX = AL * source  ;операнд source - байт
or
  DX:AX = AX * source ;операнд source - слово.
```

Эта команда выполняет умножение с учетом знака. Если операнд source является байтом, то IMUL умножает операнд source на AL, засылая произведение в AX. Если же операнд source является словом, то IMUL умножает операнд source на AX, засылая произведение в DX:AX. Признаки переноса и переполнения CF и OF устанавливаются (=1), если старшая половина результата (т.е. AH для случая, когда source - байт, и DX, когда source - слово) содержит какую-либо значащую цифру произведения, иначе они сбрасываются (=0).

Операнды байт(слово)	Такты	Обращения	Байты	Пример
регистр8	80-98	-	2	IMUL CL
регистр16	128-154	-	2	IMUL BX
память8	(86-104)+EA	1	2-4	IMUL BYTE
память16	(138-164)+EA	1	2-4	IMUL WORD[BP][DI]

2.2.16 INC Инкремент

```
Признаки: O D I T S Z A P C
          *   * * * * .
Команда: INC destination .
Логика:  destination = destination + 1 .
```

Эта команда прибавляет к операнду destination единицу. Операнд destination, который может быть словом или байтом, интерпретируется как двоичное число без знака.


```
 Операнды      Такты     Обращения  Байты  Пример
 байт(слово)
 регистр16       2         1        INC BX
 регистр8       3         2        INC BL
 память       15(23)+EA     2     2-4  INC MATRIX[SI]

 ------------------------------------------------------------
```

Примечания :

При выполнении этой команды признак переноса CF не изменяется, поэтому если Вы хотите инкрементировать число, записанное несколькими словами, то лучше воспользоваться командами ADD и ADC.

2.2.17 MUL Умножение без учета знака

```
Признаки: O D I T S Z A P C
      *    ? ? ? ? * .
Команда: MUL source .
Логика:  AX = AL * source  ;операнд source - байт
or
   DX:AX = AX * source ;операнд source - слово .
```

Эта команда выполняет умножение без учета знака. Если операнд source является байтом, то MUL умножает операнд source на AL, засылая произведение в AX. Если же операнд source является словом, то MUL умножает операнд source на AX, засылая произведение в DX:AX. Признаки переноса и переполнения CF и OF устанавливаются (=1), если старшая половина результата (т.е. AH для случая, когда source - байт, и DX, когда source - слово) содержит какую-либо значащую цифру произведения, иначе они сбрасываются (=0).

```
 ------------------------------------------------------------
 Операнды     Такты     Обращения  Байты  Пример
 байт(слово)
 регистр,8     70-77       -      2   MUL CH
 регистр,16    118-133      -      2   MUL BX
 память,8     (76-83)+EA    1     2-4  MUL A_BYTE
 память,16     (128-143)+EA   1    2-4  MUL A_WORD

 ------------------------------------------------------------
```

2.2.18 NEG Получение дополнительного кода

```
Признаки: O D I T S Z A P C
      *    * * * * * .
Команда: NEG destination .
```

Логика: destination = -destination; дополнительный код. Команда NEG вычитает операнд destinstion из 0 и засылает результат обратно в destination. Эта команда является реализацией выполнения операции нахождения дополнительного кода операнда. Операндом может быть как байт, так и слово.

```
 ------------------------------------------------------------
 Операнды     Такты     Обращения  Байты   Пример
 байты(слова)
 регистр       3        -      2    NEG DL
 память     16(24)+EA      2    2-4   NEG COEFFICIENT
 ------------------------------------------------------------
```

Примечания:

Если операнд равен нулю, то признак переноса CF сбрасывается (=0); во всех остальных случаях он устанавливается (=1).Попытка применения операции NEG к байту -128 или слову -32,768 не приводит к изменению операнда, а только устанавливает признак переполнения (OF=1).

2.2.19 SBB Вычитание с заемом

```
Признаки: O D I T S Z A P C
      *    * * * * * .
Команда: SBB destination,source.
Логика:   destination = destination - source - CF.
```

Команда SBB вычитает операнд source из операнда destination, вычитает 1 из результата, если признак переноса установлен (т.е. если CF = 1), и засылает результат по адресу destination. Оба операнда могут быть байтами или словами, и оба операнда могут быть двоичными числами со знаком или без знака.

```
 ------------------------------------------------------------
 Операнды          Такты  Обращения Байты  Пример
  байты
     (слова)
 регистр,регистр       3      -     2   SBB DX,AX
```


```
регистр,непоср.операнд  4      -     3-4   SBB BH,4
аккумулятор,непоср.оп.  4      -     2-3   SBB AX,8
регистр,память      9(13)+EA  1    2-4   SBB DX,FEE
память,регистр      16(24)+EA  2    2-4   SBB SIGH,SI
память,непоср.операнд 17(25)+EA  2    3-6   SBB TOTAL,10
-------------------------------------------------------------
```

Примечания:

Команда SBB полезна для вычитания чисел, которые длиннее 16 битов, поскольку она вычитает заем (находящийся в CF) от предыдущего вычитания.Вы можете вычитать непосредственный операнд размером в байт из операнда destination, даже если он размером в слово; в этом случае перед вычитанием байт растягивается до 16 битов, занося в новые биты значение бита знака.

2.2.20 SUB Вычитание

```
Признаки: O D I T S Z A P C
     *    * * * * * .
```

Команда: SUB destination,source .

Логика: destination = destination - source.

Команда SUB вычитает операнд source из операнда destination и засылает результат по адресу destination. Оба операнда могут быть байтами или словами, и оба операнда могут быть двоичными числами со знаком или без знака.

```
-------------------------------------------------------------
Операнды        Такты Обращения Байты  Пример
 байты
 (слова)
регистр,регистр      3    -     2   SUB DX,BX
регистр,непоср.операнд  4    -     3-4   SUB DX,5280
аккумулятор,непоср.оп.  4    -     2-3   SUB AH,25
регистр,память      9(13)+EA 1    2-4   SUB DX,TOTAL
память,регистр      16(24)+EA 2    2-4   SUB RATE,CL
память,непоср.операнд 17(25)+EA 2    3-6   SUB TOTAL,10
-------------------------------------------------------------
```

Примечания:

Если Вы хотите произвести вычитание чисел,которые длиннее 16 битов, то Вы можете воспользоваться командой SBB, которая вычитает также заем от предыдущего вычитания.
Вы можете вычитать непосредственный операнд размером в байт из операнда destination, даже если он размером в слово; в этом случае перед вычитанием байт растягивается до 16 битов,занося в новые биты значение бита знака.

2.3 Логические операции

2.3.1 AND Логическое умножение

```
Признаки: O D I T S Z A P C
     0    * * ? * 0 .
```

Команда: AND destination,source .

Логика: destination = destination AND source .AND выполняет логическое умножение побитно над своими операндами и засылает результат по назначению (destination).Оба операнда могут быть словами или байтами.

Логика команды AND

```
Destination Source Результат
0 0 0
0 1 0
1 0 0
1 1 1
```

Команда AND устанавливает каждый бит результата в 1, если соответствующие биты операндов равны 1.

```
-------------------------------------------------------------
Операнды        Такты   Обращения Байты  Пример
 байты
(слова)
регистр,регистр      3      -     2   AND AL,DL
регистр,непоср.операнд  4      -     3-4  AND CX,0FFh
аккумулятор,непоср.оп.  4      -     2-3  AND AX,01000010b
регистр,память      9(13)+EA   1    2-4  AND CX,MASK
память,регистр      16(24)+EA   2    2-4  AND VALUE,BL
память,непоср.операнд  17(25)+EA   2    3-6  AND GAMMA,01h
-------------------------------------------------------------
```

2.3.2 NOT Логическое отрицание
Признаки не меняются .
Команда: NOT destination .
Логика: destination = NOT(destination); обратный код .
Логика команды NOT

destination	результат
0	1
1	0

Операнды байты(слова)	Такты	Обращения	Байты	Пример
регистр	3	-	2	NOT DX
память	16(24)+EA	2	2-4	NOT MASK

2.3.3 OR Логическое сложение
Признаки: O D I T S Z A P C
0 * * ? * 0 .
Команда: OR destination,source.
Логика: destination = destination OR source .OR выполняет логическое сложение побитно над своими операндами и засылает результат по назначению (destination).Оба операнда могут быть словами или байтами.
Логика команды OR

Destination	Source	Результат
0	0	0
0	1	1
1	0	1
1	1	1

Команда OR устанавливает каждый бит результата в 1,если хотя бы один из соответствующих битов операндов равен 1.

Операнды (слова) байты	Такты	Обращения	Байты	Пример
регистр,регистр	3	-	2	OR CH,DL
регистр,непоср.операнд	4	-	3-4	OR CX,00FFh
аккумулятор,непоср.оп.	4	-	2-3	OR AL,01000010b
регистр,память	9(13)+EA	1	2-4	OR CX,MASK
память,регистр	16(24)+EA	2	2-4	OR MASK,CX
память,непоср.операнд	17(25)+EA	2	3-6	OR GAMMA,01h

2.3.4 RCL Циклический сдвиг влево через CF
Признаки: O D I T S Z A P C
* * .
Команда: RCL destination,count.
Команда RCL сдвигает слово или байт, находящийся по адресу destination, влево на число битовых позиций, определяемое вторым операндом, COUNT. Бит, который выскакивает за левый предел операнда destination, заносится в признак переноса CF, а старое значение CF осуществляет ротацию, в том смысле, что оно заносится в освободившийся крайний правый бит операнда destination.Число таких "битовых ротаций" определяется операндом COUNT. Если COUNT не равен 1, то признак переполнения OF не определен.Если же COUNT равен 1, тогда в OF заносится результат выполнения операции исключающего или, примененной к 2 старшим битам исходного значения операнда destination.

Операнды байт(слово)	Такты	Обращения	Байты	Пример
регистр, 1	2	-	2	RCL CX,1
регистр, CL	8 + 4/бит	-	2	RCL BL,CL
память, 1	15(23) + EA	2	2-4	RCL MULTIPL,1
память, CL	20(28)+EA+4/бит	2	2-4	RCL MOVE_AR,CL

Примечания:
В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере. Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы

получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды. Несколько команд RCL, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти, чем одна команда RCL,использующая CL в качестве операнда COUNT. Признак переполнения не определен, если операнд COUNT больше 1.

2.3.5 RCR Циклический сдвиг вправо через CF

Признаки: O D I T S Z A P C
* * .

Команда: RCR destination,count .

Команда RCR сдвигает слово или байт, находящийся по адресу destination, вправо на число битовых позиций, определяемое вторым операндом, COUNT. Бит, который выскакивает за правый предел операнда destination, заносится в признак переноса CF, а старое значение CF осуществляет ротацию, в том смысле, что оно заносится в освободившийся крайний левый бит операнда destination.Число таких "битовых ротаций" определяется операндом COUNT.Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1,тогда в OF заносится результат выполнения операции исключающего или, примененной к 2 старшим битам результата.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
регистр, 1	2	-	2	RCR CX,1
регистр, CL	8 + 4/бит	-	2	RCR DL,CL
память, 1	15(23) + EA		2	2-4 RCR DIVIDE,1
память, CL	20(28)+EA+4/бит		2	2-4 RCR MOVE_AR,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд RCR, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти,чем одна команда RCR, использующая CL в качестве операнда COUNT. Признак переполнения не определен, если операнд COUNT больше 1.

2.3.6 ROL Циклический сдвиг влево

Признаки: O D I T S Z A P C
* * .

Команда: ROL destination,count.

Команда ROL сдвигает слово или байт, находящийся по адресу destination, влево на число битовых позиций, определяемое вторым операндом, COUNT. Бит, который выскакивает за левый предел операнда destination, заносится в него снова с правого края.Значение CF совпадает со значением бита, который последним был вытеснен за левый край операнда.

Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1, тогда в OF заносится результат выполнения операции исключающего или, примененной к 2 старшим битам исходного значения операнда destination.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
регистр, 1	2	-	2	ROL DI,1
регистр, CL	8 + 4/бит	-	2	ROL BX,CL
память, 1	15(23) + EA		2	2-4 ROL BYTE,1
память, CL	20(28)+EA+4/бит		2	2-4 ROL WORD,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд ROL, использующих 1 в качестве второго операнда,

выполняются быстрее и требуют меньшего об'ема памяти,чем одна команда ROL, использующая CL в качестве операнда COUNT.
Признак переполнения не определен, если операнд COUN больше 1.

2.3.7 ROR Циклический сдвиг вправо

Признаки: O D I T S Z A P C
* *

Команда: ROR destination,count .

Команда ROR сдвигает слово или байт, находящийся по адресу destination, вправо на число битовых позиций, определяемое вторым операндом, COUNT. Бит, который выскакивает за правый предел операнда destination, заносится в него снова с левого края. Значение CF совпадает со значением бита, который последним был вытеснен за правый край операнда. Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1, тогда в OF заносится результат выполнения операции исключающего или, примененной к 2 старшим битам результата.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
регистр, 1	2	-	2	ROR BL,1
регистр, CL	8 + 4/бит	-	2	ROR AX,CL
память, 1	15(23) + EA	2	2-4	ROR WORD,1
память, CL	20(28)+EA+4/бит	2	2-4	ROR BYTE,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32,чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд ROR, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти, чем одна команда ROR, использующая CL в качестве операнда COUNT. Признак переполнения не определен, если операнд COUNT больше 1.

2.3.8 SAL Арифметический сдвиг влево

Признаки: O D I T S Z A P C
* * ? * * .

Команда: SAL destination,count .

Команда SAL сдвигает слово или байт, находящийся по адресу destination, влево на число битовых позиций, определяемое вторым операндом, COUNT. По мере вытеснения битов за левый край операнда destination, справа в освободившиеся биты заносятся нули. Значение CF совпадает со значением бита, который последним был вытеснен за левый край операнда.Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1, тогда OF=0, если 2 старших бита исходного значения операнда destination совпадали, иначе OF=1.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
регистр, 1	2	-	2	SAL AL,1
регистр, CL	8 + 4/бит	-	2	SAL SI,CL
память, 1	15(23) + EA	2	2-4	SAL WORD,1
память, CL	20(28)+EA+4/бит	2	2-4	SAL BYTE,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере. Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32,чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд SAL, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти, чем одна команда SAL, использующая CL в качестве операнда COUNT.Признак переполнения не определен, если операнд COUNT больше 1.
Команда SHL, логический сдвиг влево, - это та же команда, что и SAL.

2.3.9 SAR Арифметический сдвиг вправо

Признаки: O D I T S Z A P C
* * ? * * .

Команда: SAR destination,count .

Команда SAR сдвигает слово или байт, находящийся по адресу destination, вправо на число битовых позиций, определяемое вторым операндом, COUNT. По мере вытеснения битов за правый край операнда destination, слева в освободившиеся биты заносятся биты, совпадающие с битом знака исходного значения операнда destination. Значение CF совпадает со значением бита, который последним был вытеснен за правый край операнда. Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1, то OF=0.

Операнды байт(слово)	Такты	Обращения	Байты	Пример
регистр, 1	2	-	2	SAR DX,1
регистр, CL	8 + 4/бит	-	2	SAR DI,CL
память, 1	15(23) + EA	2	2-4	SAR WORD,1
память, CL	20(28)+EA+4/бит	2	2-4	SAR BYTE,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд SAR, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти, чем одна команда SAR, использующая CL в качестве операнда COUNT.

Признак переполнения не определен, если операнд COUNT больше 1.

2.3.10 SHL Логический сдвиг влево

Признаки: O D I T S Z A P C
* * ? * * .

Команда: SHL destination,count .

Команда SHL сдвигает слово или байт, находящийся по адресу destination, влево на число битовых позиций, определяемое вторым операндом, COUNT. По мере вытеснения битов за левый край операнда destination, справа в освободившиеся биты заносятся нули. Значение CF совпадает со значением бита, который последним был вытеснен за левый край операнда. Если COUNT не равен 1, то признак переполнения OF не определен. Если же COUNT равен 1, тогда OF=0, если 2 старших бита исходного значения операнда destination совпадали, иначе OF=1.

Операнды байт(слово)	Такты	Обращения	Байты	Пример
регистр, 1	2	-	2	SHL AL,1
регистр, CL	8 + 4/бит	-	2	SHL SI,CL
память, 1	15(23) + EA	2	2-4	SHL WORD,1
память, CL	20(28)+EA+4/бит	2	2-4	SHL BYTE,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако,Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1,как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот пери од времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд SHL, использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти,чем одна команда SHL, использующая CL в качестве операнда COUNT.Признак переполнения не определен, если операнд COUNT больше 1.Команда SAL, арифметический сдвиг влево, - это та же команда, что и SHL.

2.3.11 SHR Логический сдвиг вправо

Признаки: O D I T S Z A P C
* * ? * * .

Команда: SHR destination,count

Команда SHR сдвигает слово или байт, находящийся по адресу destination, вправо на число битовых позиций, определяемое вторым операндом, COUNT, или содержимым CL, если второй операнд опущен. По мере вытеснения битов за правый край операнда destination, слева в освободившиеся биты заносятся нули. Если бит знака сохраняет свое значение, то признак переполнения OF равен 0, иначе он равен 1. Значение CF совпадает со значением бита, который последним был вытеснен за правый край операнда.Если COUNT не равен 1, то признак

переполнения OF не определен. Если же COUNT равен 1, то OF равен значению старшего бита исходного операнда.

Операнды	Такты	Обращения	Байты	Пример
регистр, 1	2	-	2	SHR SI,1
регистр, CL	8 + 4/бит	-	2	SHR SI,CL
память, 1	15 + EA	2	2-4	SHR ID_BYTE[SI][BX],1
память, CL	20 + EA + 4/бит	2	2-4	SHR INPUT_WORD,CL

Примечания:

В качестве операнда COUNT берется обычно значение в регистре CL. Если, однако, Вы хотите осуществить сдвиг лишь на одну позицию, то замените второй операнд (CL) на число 1, как показано выше в первом примере.Микропроцессоры 80286 и 80386 ограничивают значение COUNT числом 31. Если COUNT больше, чем 31, то эти микропроцессоры используют COUNT MOD 32, чтобы получить новый COUNT в пределах 0-31. Эта верхняя граница имеет своей целью сократить тот период времени, на который будет задерживаться ответ на прерывание из-за ожидания конца выполнения команды.Несколько команд SHR,использующих 1 в качестве второго операнда, выполняются быстрее и требуют меньшего об'ема памяти,чем одна команда SHR, использующая CL в качестве операнда COUNT.

Признак переполнения не определен, если операнд COUNT больше 1.

2.3.12 TEST Тест

Признаки: O D I T S Z A P C
0 * * ? * 0 .

Команда: TEST destination,source .

Логика: (destination AND source); Только изменение признаков
CF = 0
OF = 0 .

Команда TEST выполняет операцию AND над своими операндами и меняет значения признаков. Сами операнды не изменяются.

Операнды	Такты	Обращения	Байты	Пример
байты (слова)				
регистр,регистр	3	-	2	TEST AL,DL
регистр,непоср.операнд	5	-	3-4	TEST CX,0FFh
аккумулятор,непоср.оп.	4	-	2-3	TEST AX,01000010b
регистр,память	9(13)+EA	1	2-4	TEST CX,MASK
память,непоср.операнд	11+EA	-	3-6	TEST GAMMA,01h

Команда TEST полезна при проверке значения конкретного бита. Например, следующий кусок программы передает управление в ONE_FIVE_OFF, если биты 1 и 5 регистра AL сброшены. Значения остальных битов во внимание не принимаются.

```
TEST   AL,00100010b ;Маскируем все биты, кроме 1 и 5
JZ ONE_FIVE_OFF ;Если хотя бы один установлен,
  ;то ZF=1
NOT_BOTH:    .
 .
ONE_FIVE_OFF:  .    ;Биты 1 и 5 сброшены
 .
 .
```

2.3.13 XOR Исключающее ИЛИ

Признаки: O D I T S Z A P C
0 * * * * 0 .

Команда: XOR destination,source .

Логика: destination = destination XOR source .

XOR выполняет операцию исключающего или побитно над своими операндами и засылает результат по назначению (destination).Оба операнда могут быть словами или байтами.

Логика команды XOR

Destination	Source	Результат
0	0	0
0	1	1
1	0	1
1	1	0

Команда XOR устанавливает каждый бит результата в 1, если в точности один из соответствующих битов операндов равен 1.


```
 ---------------------------------------------------------------
 Операнды                Такты Обращения Байты Пример
  байты
  (слова)
 регистр,регистр          3         -      2   XOR CX,BX
 регистр,непоср.операнд   4         -     3-4  XOR SI,00C2h
 аккумулятор,непоср.оп.   4         -     2-3  XOR AL,01000010b
 регистр,память          9(13)+EA   1     2-4  XOR CL,MASK
 память,регистр         16(24)+EA   2     2-4  XOR MASK,DX
 память,непоср.операнд 17(25)+EA    2     3-6  XOR GAMMA,01h
 ---------------------------------------------------------------
  2.4. Обработка блоков данных
   2.4.1  CMPS Сравнение строк
 Признаки: O D I T S Z A P C
             *    * * * * * .

 Команда: CMPS destination-string,source_string .
 Логика: CMP (DS:SI), (ES:DI) ; только устанавливает признаки
  if DF = 0
    SI = SI + n   ; n = 1 для байта, 2 для слова
    DI = DI + n
  else
    SI = SI - n
    DI = DI - n .
```

Эта команда сравнивает два значения, вычитая байт или слово, на которое указывает ES:DI, из байта или слова, на которое указывает DS:SI, и устанавливает признаки в соответствии с результатами сравнения. Сами операнды не изменяются. После сравнения, SI и DI увеличиваются на 1 (для байтов) или 2 (для слов), если признак направления сброшен, или уменьшаются на 1 или 2, если признак направления установлен. Тем самым подготавливаются к сравнению следующие элементы обеих строк.

```
 ---------------------------------------------------------------
 Операнды             Такты                Обращения Байты Пример
  байты
 (слова)
 dest,source          22(30)                  2      1   CMPS STR1,STR2
 (повтор) dest,source 9+22(30)/rep           2/rep   1   REPE CMPS STR1,STR2
 ---------------------------------------------------------------
 Примечания:
```

Эта команда всегда переводится ассемблером или в CMPSB,сравнение строк из байтов, или в CMPSW, сравнение строк из слов, в зависимости от того, являются ли операнды строками из байтов или из слов. В обоих случаях Вы должны в явном виде загрузить в регистры SI и DI смещения обеих строк.Пусть имеют место следующие определения :

```
 buffer1 db   100 dup (?)
 buffer2 db   100 dup (?) .
```

Тогда следующий пример выполняет сравнение элементов строк BUFFER1 и BUFFER2 до первого их несовпадения (mismatch) :

```
 cld   ;сканируя в прямом направлении
 mov   cx, 100 ;100 байтов (CX используется
       ; в REPE),
 lea   si, buffer1  ;начиная с 1го элемента BUFFER1
 lea   di, buffer2  ;и с 1го элемента BUFFER2,
 repe  cmps  buffer1,buffer2;сравниваем их.
 jne   mismatch  ;признак ZF = 0, если сравниваемые
    ;строки не совпадают (mismatch)
 match: .  ;если мы попали сюда, значит, они
  .  ;совпадают (match)
  .
 mismatch:   .
 dec   si ;если мы попали сюда, то мы нашли
 dec   di ;несовпадение, и возвращаем указатели
  .  ;SI и DI обратно, чтобы они указывали
  .  ;на первые несовпадающие элементы.
```

После выхода из цикла REPE CMPS признак ZF будет сброшен,если несовпадение было найдено, и установлен в противоположном случае. Если несовпадение было найдено, то DI и SI будут указывать на байты, следующие непосредственно за байтами, которые не совпали; DEC DI и DEC

SI уменьшают значения в этих регистрах таким образом, чтобы они указывали на сами несовпадающие байты.

2.4.2 CMPSB Сравнение строк из байтов

```
Признаки: O D I T S Z A P C
             *    * * * * * .
Команда: CMPSB.
Логика: CMP (DS:SI), (ES:DI) ; только устанавливает признаки
 if DF = 0
   SI = SI + 1
   DI = DI + 1
 else
   SI = SI - 1
   DI = DI - 1 .
```

Эта команда сравнивает два значения, вычитая байт, на который указывает ES:DI, из байта, на который указывает DS:SI, и устанавливает признаки в соответствии с результатами сравнения.Сами операнды не изменяются. После сравнения, SI и DI увеличиваются на 1, если признак направления сброшен, или уменьшаются на 1, если признак направления установлен. Тем самым подготавливаются к сравнению следующие элементы обеих строк.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	22	2	1	CMPSB
(повтор)	9+22/rep	2/rep	1	REPE CMPSB

Пусть имеют место следующие определения: buffer1 db 100 dup (?) buffer2 db 100 dup(?) .
Тогда следующий пример выполняет сравнение элементов строк BUFFER1 и BUFFER2 до первого их несовпадения (mismatch):

```
cld ;сканируя в прямом направлении
mov  cx, 100   ;100 байтов (CX используется в REPE),
lea  si, buffer1 ;начиная с 1го элемента BUFFER1
lea  di, buffer2 ;и с 1го элемента BUFFER2,
repe  cmpsb    ;сравниваем их.
jne  mismatch  ;признак ZF = 0, если сравниваемые
   ;строки не совпадают (mismatch)
match: .  ;если мы попали сюда, значит, они
     .  ;совпадают (match)
     .
mismatch:   .
dec  si  ;если мы попали сюда, то мы нашли
dec  di  ;несовпадение, и возвращаем указатели
     .  ;SI и DI обратно, чтобы они указывали
     .  ;на первые несовпадающие байты.
```

После выхода из цикла REPE CMPSB признак ZF будет сброшен, если несовпадение было найдено, и установлен в противоположном случае. Если несовпадение было найдено, то DI и SI будут указывать на байты, следующие непосредственно за байтами, которые не совпали; DEC DI и DEC SI уменьшают значения в этих регистрах таким образом, чтобы они указывали на сами несовпадающие байты.

2.4.3 CMPSW Сравнение строк из слов

```
Признаки: O D I T S Z A P C
             *    * * * * * .

Команда: CMPSW .
Логика: CMP (DS:SI), (ES:DI) ; только устанавливает признаки
 if DF = 0
   SI = SI + 2
   DI = DI + 2
 else
   SI = SI - 2
   DI = DI - 2 .
```

Эта команда сравнивает два значения, вычитая слово, на которое указывает ES:DI, из слова, на которое указывает DS:SI, и устанавливает признаки в соответствии с результатами сравнения.Сами операнды не изменяются. После сравнения, SI и DI увеличиваются на 2, если признак направления сброшен, или уменьшаются на 2, если признак направления установлен. Тем самым подготавливаются к сравнению следующие элементы обеих строк.


```
Операнды        Такты   Обращения Байты Пример
нет операндов   30          2       1   CMPSW
(повтор)        9+30/rep  2/rep     1   REPE CMPSW
-----------------------------------------------------------
```

Пусть имеют место следующие определения :

```
buffer1 dw   50 dup (?)
buffer2 dw   50 dup (?)
```

Тогда следующий пример выполняет сравнение элементов строк BUFFER1 и BUFFER2 до первого их несовпадения (mismatch) :

```
cld    ;сканируя в прямом направлении
mov  cx, 50   ;50 слов (CX используется в REPE),
lea  si, buffer1 ;начиная с 1го элемента BUFFER1
lea  di, buffer2 ;и с 1го элемента BUFFER2,
repe  cmpsw   ;сравниваем их.
jne  mismatch  ;признак ZF = 0, если сравниваемые
  ;строки не совпадают (mismatch)
match:    .  ;если мы попали сюда, значит, они
   .  ;совпадают (match)
   .
mismatch:  .
dec  si  ;если мы попали сюда, то мы нашли
dec  si  ;несовпадение, и возвращаем указатели
dec  di  ;SI и DI обратно, чтобы они указывали
dec  di  ;на первые несовпадающие слова.
```

После выхода из цикла REPE CMPSW признак ZF будет сброшен, если несовпадение было найдено, и установлен в противоположном случае. Если несовпадение было найдено, то DI и SI будут указы вать на слова (по два байта), следующие непосредственно за словами, которые не совпали; пары команд DEC DI и DEC SI уменьшают значения в этих регистрах таким образом, чтобы они указывали на сами несовпадающие слова.

2.4.4 LODS Загрузка строки (из байтов или слов)

Признаки не меняются.

Команда: LODS source-str .

Логика: Accumulator = (DS:SI)

```
   if DF = 0
 SI = SI + n  ; n = 1 для байта, 2 - для слова
   else
 SI = SI - n .
```

Команда LODS передает байт или слово, расположенное по адресу DS:SI в AX или AL, а также инкрементирует или декрементирует SI (в зависимости от состояния признака направления DF),чтобы указатель переместился на следующий элемент.

```
-------------------------------------------------------------
Операнды            Такты        Обращения Байты Пример
 байты
(слова)
source-str          12(16)           -       1   LODS LIST
(повтор) source-str  9+13(17)/rep   1/rep    1   REP LODS LIST
-------------------------------------------------------------
```

Примечания:

Эта команда всегда ассемблируется или как LODSB, загрузка строки из байтов, или как LODSW, загрузка строки из слов, в зависимости от того, указывает ли source-str на строку байтов или на строку слов. Однако, в обоих случаях Вы должны в явном виде загрузить в регистр SI смещение строки.Хотя и разрешется использовать эту команду в повторном режиме, это почти никогда не делается, т.к. это привело бы к постоянному изменению значения в AL.

Следующий пример иллюстрирует пересылку восьми байтов из INIT_PORT в порт 250. (Не пробуйте делать этого на вашей машине, если Вы не знаете о назначении порта 250.)

```
INIT_PORT:
 DB  '$CMD0000'  ;Строка, которую мы хотим переслать
   .
   .
 CLD ;Будем передвигаться в прямом
   ;направлении
 LEA SI,INIT_PORT ;Засылаем в SI стартовый адрес
   ;строки
 MOV CX,8  ;CX является счетчиком
   ;для команды LOOP
```


```
AGAIN: LODS INIT_PORT   ;В имени INIT_PORT ассемблер
 OUT 250,AL ;нуждается только для того, чтобы
   ; определить, имеет ли он дело
 LOOP AGAIN ; с байтами или со словами
```

2.4.5 LODSB Загрузка строки из байтов

Признаки не меняются .

```
Команда: LODSB .
Логика:  AL = (DS:SI)
   if DF = 0
 SI = SI + 1
   else
 SI = SI - 1 .
```

Команда LODSB передает байт, расположенный по адресу DS:SI в AL, а также инкрементирует или декрементирует SI (в зависимости от состояния признака направления DF), чтобы указатель переместился на следующий байт строки.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	12	-	1	LODSB
(повтор)	9+13/rep	1/rep	1	REP LODSB

Примечания:

Хотя и разрешется использовать эту команду в повторном режиме, это почти никогда не делается, т.к. это привело бы к постоянному изменению значения в AL.

Следующий пример иллюстрирует пересылку восьми байтов из INIT_PORT в порт 250. (Не пробуйте делать этого на вашей машине, если Вы не знаете о назначении порта 250.)

```
INIT_PORT:
 DB   '$CMD0000'   ;Строка, которую мы хотим
 .    ;переслать
 .
 CLD   ;Будем передвигаться
  ;в прямом направлении
 LEA  SI,INIT_PORT  ;Засылаем в SI стартовый
  ;адрес строки
 MOV  CX,8   ;CX является счетчиком
  ;для команды LOOP
AGAIN: LODSB  ;Загружаем байт в AL...
 OUT  250,AL  ; ...и высылаем его в порт
 LOOP  AGAIN
```

2.4.6 LODSW Загрузка строки из слов

Признаки не меняются .

```
Команда: LODSW .
Логика:  AX = (DS:SI)
   if DF = 0
 SI = SI + 2
   else
 SI = SI - 2 .
```

Команда LODSW передает слово, расположенное по адресу DS:SI в AX, а также инкрементирует или декрементирует SI (в зависимости от состояния признака направления DF), чтобы указатель переместился на следующее слово строки.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	16	-	1	LODSW
(повтор)	9+17/rep	1/rep	1	REP LODSW

Примечания:

Хотя и разрешется использовать эту команду в повторном режиме, это почти никогда не делается, т.к. это привело бы к постоянному изменению значения в AX.

Следующий пример иллюстрирует пересылку восьми байтов из INIT_PORT в порт 250. (Не пробуйте делать этого на вашей машине, если Вы не знаете о назначении порта 250.)

```
INIT_PORT:
 DB   '$CMD0000'   ;Строка, которую мы хотим
  ;переслать
 .
 CLD   ;Будем передвигаться в прямом
```


```
   ;направлении
  LEA  SI,INIT_PORT  ;Засылаем в SI стартовый
   ;адрес строки
  MOV  CX,4   ;Будем пересылать 4 слова
   ;(8 байтов)
 AGAIN: LODSW  ;Загружаем слово в AX...
  OUT  250,AX  ; ...и высылаем его в порт
  LOOP  AGAIN
```

2.4.7 MOVS Пересылка строки (из байтов или слов)

Признаки не меняются .

Команда: MOVS destination,source .

Логика:

```
   (ES:DI) = (DS:SI)
    if DF = 0
  SI = SI + n ; n = 1 для байта, 2 - для слова
  DI = DI + n
    else
  SI = SI - n
  DI = DI - n .
```

Эта команда пересылает байт или слово,расположенное по адресу DS:SI, по адресу ES:DI. После пересылки SI и DI инкрементируются (если признак направления сброшен) или декрементируются (если признак направления установлен),чтобы указатель переместился на следующий элемент строки.

Операнды	Такты	Обращения	Байты	Пример
байты (слова)				
dest,source	18(26)	2	1	MOVS WD_BF,INPUT
(повтор) dest,source	9+17(25)/rep	2/rep	1	REP MOVS W,I

Примечания:

Эта команда всегда ассемблируется или как MOVSB, пересылка строки из байтов, или как MOVSW, пересылка строки из слов, в зависимости от того, является ли source ссылкой на строку из байтов или строку из слов. В обоих случаях Вы должны в явном виде загрузить в регистры SI и DI смещения строк source и destination.

Пример.

Предположим, что строка BUFFER1 была где-то в программе описана следующим образом:

```
 BUFFER1    DB   100 DUP (?)
```

Тогда следующий пример описывает пересылку 100 байтов из BUFFER1 в BUFFER2:

```
    CLD  ;Двигаемся в прямом направлении
    LEA SI,BUFFER1   ;Адрес исходной строки засылаем в SI
    LEA DI,BUFFER2   ;Адрес строки назначения засылаем в DI
    MOV CX,100 ;CX используем в префиксе REP
 REP MOVS BUFFER1,BUFFER2;Производим пересылку
```

2.4.8 MOVSB Пересылка строки из байтов

Признаки не меняются .

Команда: MOVSB .

Логика:

```
   (ES:DI) = (DS:SI)
    if DF = 0
  SI = SI + 1
  DI = DI + 1
    else
  SI = SI - 1
  DI = DI - 1 .
```

Эта команда пересылает байт, расположенный по адресу DS:SI, по адресу ES:DI. После пересылки SI и DI инкрементируются (если признак направления сброшен) или декрементируются (если признак направления установлен),чтобы указатель переместился на следующий байт.

Операнды	Такты	Обращения	Байты	Пример
байты(слова)				
нет операндов	18	2	1	MOVSB
(повтор)	9+17/rep	2/rep	1	REP MOVSB

Пример.

Предположим, что строка BUFFER1 была где-то в программе описана следующим образом:
```
BUFFER1    DB   100 DUP (?)
```
Тогда следующий пример описывает пересылку 100 байтов из BUFFER1 в BUFFER2:
```
   CLD  ;Двигаемся в прямом направлении
   LEA   SI,BUFFER1 ;Адрес исходной строки засылаем в SI
   LEA   DI,BUFFER2 ;Адрес строки назначения засылаем в DI
   MOV   CX,100   ;CX используем в префиксе REP
REP MOVSB    ;Производим пересылку
```

2.4.9 MOVSW Пересылка строки из слов

Признаки не меняются .
Команда: MOVSW .
Логика: (ES:DI) = (DS:SI)
```
   if DF = 0
 SI = SI + 2
 DI = DI + 2
   else
 SI = SI - 2
 DI = DI - 2 .
```
Эта команда пересылает слово, расположенное по адресу DS:SI, по адресу ES:DI. После пересылки SI и DI инкрементируются (если признак направления сброшен) или декрементируются (если признак направления установлен),чтобы указатель переместился на следующее слово.

Операнды байты(слова)	Такты	Обращения	Байты	Пример
нет операндов	26	2	1	MOVSW
(повтор)	9+25/rep	2/rep	1	REP MOVSW

Пример.
Предположим, что строка BUFFER1 была где-то в программе описана следующим образом:
```
BUFFER1    DB   100 DUP (?) .
```
Тогда следующий пример описывает пересылку 50 слов (100 байтов) из BUFFER1 в BUFFER2:
```
   CLD ;Двигаемся в прямом направлении
   LEA   SI,BUFFER1 ;Адрес исходной строки засылаем в SI
   LEA   DI,BUFFER2 ;Адрес строки назначения засылаем в DI
   MOV   CX,50   ;CX используем в префиксе REP
  ; пересылаем 50 слов
REP MOVSW    ;Производим пересылку
```

2.4.10 REP Повтор

Признаки не меняются .
Команда: REP КОС (команда обработки строк) .
Логика: while CX <> 0 ; для КОС MOVS, LODS или STOS выполнить КОС CX = CX - 1

```
while CX <> 0 ; для КОС CMPS или SCAS
  CX = CX - 1
  if ZF = 0 то заканчиваем цикл
```
REP - это префикс, который может быть употреблен перед любой КОС (CMPS, LODS, MOVS, SCAS и STOS). Префикс REP заставляет выполняться следующую за ним КОС в повторном режиме до тех пор, пока CX не станет равным 0; CX уменьшается на 1 после каждого выполнения КОС. (Для КОС CMPS и SCAS циклический повтор прерывается также, если признак нулевого результата ZF оказывается сброшенным после очередного выполнения КОС.)

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	REP MOVS TO,FROM

Примечания:
Если CX с самого начала равно 0, то КОС не выполняется ни разу. Проверка о равенстве CX нулю проводится перед выполнением КОС. Проверка о равенстве ZF нулю проводится только для команд CMPS и SCAS, причем лишь после очередного выполнения КОС.Префиксы REP, REPE (повтор пока равно) и REPZ (повтор пока ноль) - все являются синонимами одного и того же префикса.Префикс REPNZ (повтор пока не ноль) похож на REP и отличается лишь тем, что для команд CMPS и SCAS повтор прекращается,когда ZF установлен, а не когда сброшен (как в REP).Префикс REP используется обычно с КОС MOVS (пересылка строки) и STOS (запись строки), его можно интерпретировать как "повторяй, пока не кончится строка".

У Вас нет необходимости инициализировать ZF перед использованием повторяющихся КОС.

Повторяющаяся КОС, которая была прервана между повторами,будет корректно возобновлена после возврата из прерывания. Однако, если перед КОС находятся также другие префиксы (например,LOCK) в добавление к REP, то все префиксы, кроме того, который непосредственно предшествует команде, будут потеряны. Поэтому,если Вам нужно использовать команду с несколькими префиксами одновременно, то Вам следует запретить прерывания на время выполнения команды (и снова разрешить из после ее выполнения). Обратите Ваше внимание на то, что даже такая мера предосторожности не предохраняет от немаскированных прерываний и что обработка длинных строк может существенно задерживать обработку прерываний.Следующий пример иллюстрирует пересылку 100 байтов из

```
BUFFER1 в BUFFER2 :
   CLD    ;двигаться будем в прямом направлении
   LEA SI, BUFFER1 ;засылаем в SI и DI стартовые адреса
   LEA DI, BUFFER2 ;строк (исходной и назначения)
   MOV CX, 100   ;префикс REP использует CX как счетчик
   REP MOVSB    ;осуществляем пересылку
```

2.4.11 REPE Повторять пока равно

Признаки не меняются .

Команда: REPE .

Эта команда является синонимом REP. Смотрите описание REP.

2.4.12 REPNE Повторять пока не равно

Признаки не меняются.

Команда: REPNE КОС (команда обработки строк).

Логика: while CX <> 0 ; для КОС MOVS, LODS или STOS выполнить КОС CX = CX - 1

```
  while CX <> 0 ; для КОС CMPS или SCAS
CX = CX - 1
if ZF <> 0 то заканчиваем цикл ; это единственное
   ; отличие от REP
```

REPNE - это префикс, который может быть употреблен перед любой КОС (CMPS, LODS, MOVS, SCAS и STOS). Префикс REPNE заставляет выполняться следующую за ним КОС в повторном режиме до тех пор, пока CX не станет равным 0; CX уменьшается на 1 после каждого выполнения КОС. (Для КОС CMPS и SCAS циклический повтор прерывается также, если признак нулевого результата ZF оказывается установленным после очередного выполнения КОС. Ср. с префиксом REP, при котором циклический повтор прерывается, если ZF оказывается, наоборот, сброшенным.)

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	REPNE SCASB

Примечания:

Если CX с самого начала равно 0, то КОС не выполняется ни разу. Проверка о равенстве CX нулю проводится перед выполнением КОС. Проверка о равенстве ZF единице проводится только для команд CMPS и SCAS, причем лишь после очередного выполнения КОС.

Префиксы REPNE и REPNZ являются синонимами одного и того же префикса.У Вас нет необходимости инициализировать ZF перед использованием повторяющихся КОС. Повторяющаяся КОС, которая была прервана между повторами,будет корректно возобновлена после возврата из прерывания. Од-
нако, если перед КОС находятся также другие префиксы (например,LOCK) в добавление к REPNE, то все префиксы, кроме того, который непосредственно предшествует команде, будут потеряны. Поэтому, если Вам нужно использовать команду с несколькими префиксами одновременно, то Вам следует запретить прерывания на время выполнения команды (и снова разрешить из после ее выполнения).Обратите Ваше внимание на то, что даже такая мера предосторожности не предохраняет от немаскированных прерываний и что обработка длинных строк может существенно задерживать обработку прерываний.

Следующий пример иллюстрирует поиск первого байта, равного 'A' в стобайтовой строке STRING :

```
   CLD    ;будем двигаться в прямом направлении
   MOV  AL,'A'   ;будем сравнивать с 'A'
   LEA  DI, STRING ;засылаем в DI стартовый адрес строки
   MOV  CX,100   ;сравнивать будем 100 байтов
REPNE SCASB   ;сравниваем 'A' с очередным байтом
   DEC  DI ;возвращаем указатель DI на 'A'
```

После окончания выполнения команды REPNE SCASB, CX будет равен нулю, если байтовое значение 'A' не было обнаружено в строке STRING, и CX будет больше 0, в противном случае.

2.4.13 REPNZ Повторять пока не ноль

Признаки не меняются.

Команда: REPNZ .

Эта команда является синонимом REPNE. Смотрите описание REPNE.

2.4.14 SCAS Просмотр строки (из байтов или слов)

Признаки: O D I T S Z A P C

* * * * * * .

Команда: SCAS destination-string .

Логика: CMP Accumulator,(ES:DI) ; только устанавливает признаки

```
if DF = 0
  DI = DI + n   ; n = 1 для байта, 2 для слова
else
  DI = DI - n .
```

Эта команда сравнивает аккумулятор (AL или AX) с байтом или словом, на которое указывает ES:DI, и устанавливает признаки в соответствии с результатами сравнения. Сами операнды не изменяются. После сравнения, DI увеличивается на 1 (для байтов) или 2 (для слов), если признак направления сброшен, или уменьшается на 1 или 2, если признак направления установлен. Тем самым подготавливается к сравнению следующий элемент строки.

Операнды байты(слова)	Такты	Обращения	Байты	Пример
dest-str	15(19)	1	1	SCAS WORD_TABLE
dest-str (повтор)	9+15(19)/rep	1/rep	1	REPNE SCAS BYTE_TABLE

Примечания:

Эта команда всегда ассемблируется или как SCASB, просмотр строки из байтов, или как SCASW, просмотр строки из слов, в зависимости от того, является ли операнд строкой из байтов или из слов. В обоих случаях Вы должны в явном виде загрузить в регистр DI смещение строки.Команда SCAS полезна в тех случаях, когда требуется найти ячейку с заданным байтом или словом. Если Вы хотите сравнить две строки из памяти поэлементно, то используйте команду CMPS.

Пример.

Пусть имеет место следующее определение :

```
LOST_A  DB  100 dup (?) .
```

Тогда следующий пример описывает поиск символа 'A' в блоке памяти длиной 100 байтов, начинающемся с LOST_A :

```
 MOV AX,DS
 MOV ES,AX   ;SCAS использует ES:DI,
  ;поэтому копируем DS в ES
 CLD   ;сканировать будем в прямом направлении
 MOV AL,'A'  ;ищем "потерянное" 'A'
 MOV CX,100  ;сканировать будем 100 байтов
  ;(CX используется в REPNE)
 LEA DI,LOST_A ;засылаем стартовый адрес в DI
 REPNE SCAS LOST_A  ;ищем 'A'
 JE  FOUND   ;признак ZF равен 1, если мы нашли 'A'
NOTFOUND:   .  ;если мы попали сюда, то 'A' не обнаружено
    .
    .
  FOUND:DEC DI ;возвращаем указатель DI на первое
    . ;обнаруженное вхождение 'A'
    .
```

После выхода из цикла REPNE SCAS, ZF=1, если вхождение 'A' было обнаружено, и ZF=0, иначе. В первом случае DI указывает на байт, следующий за байтом, где находится 'A', поэтому мы производим коррекцию указателя DI с помощью DEC DI.

2.4.15 SCASB Просмотр строки из байтов

Признаки: O D I T S Z A P C

* * * * * * .

Команда: SCASB .

Логика: CMP AL, (ES:DI) ;только устанавливает признаки

```
if DF = 0
  DI = DI + 1
else
  DI = DI - 1 .
```

Эта команда сравнивает AL с байтом на который указывает ES:DI, и устанавливает признаки в соответствии с результатами сравнения. Сами операнды не изменяются. После сравнения, DI увеличивается на 1, если признак направления сброшен, или уменьшается на 1, если признак направления установлен. Тем самым подготавливается к сравнению следующий байт.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	15	1	1	SCASB
(повтор)	9+15/rep	1/rep	1	REPNE SCASB

Примечание : Команда SCASB полезна в тех случаях, когда требуется найти ячейку с заданным байтом. Если Вы хотите сравнить две строки из памяти поэлементно, то используйте команду CMPSB.

Пример.

Пусть имеет место следующее определение :

```
LOST_A  DB  100 dup (?) .
```

Тогда следующий пример описывает поиск символа 'A' в блоке памяти длиной 100 байтов, начинающемся с LOST_A :

```
  MOV AX,DS
  MOV ES,AX   ;SCASB использует ES:DI,
   ;поэтому копируем DS в ES
  CLD   ;сканировать будем в прямом направлении
  MOV AL,'A'  ;ищем "потерянное" 'A'
  MOV CX,100  ;сканировать будем 100 байтов
   ;(CX используется в REPNE)
  LEA DI,LOST_A ;засылаем стартовый адрес в DI
  REPNE SCASB  ;ищем 'A'
  JE  FOUND   ;признак ZF равен 1, если мы нашли 'A'
 NOTFOUND:   . ;если мы попали сюда,
     . ;то 'A' не обнаружено
     .
   FOUND:DEC DI ;возвращаем указатель DI на первое
     . ;обнаруженное вхождение 'A'
     .
```

После выхода из цикла REPNE SCASB, ZF=1, если вхождение 'A' было обнаружено, и ZF=0, иначе. В первом случае DI указывает на байт, следующий за байтом, где находится 'A', поэтому мы производим коррекцию указателя DI с помощью DEC DI.

2.4.16 SCASW просмотр строки из слов

```
Признаки: O D I T S Z A P C
            *   * * * * * .
```

Команда: SCASW .

```
Логика: CMP AX, (ES:DI) ;только устанавливает признаки
 if DF = 0
   DI = DI + 2
 else
   DI = DI - 2 .
```

Эта команда сравнивает AX со словом на которое указывает ES:DI, и устанавливает признаки в соответствии с результатами сравнения. Сами операнды не изменяются. После сравнения, DI увеличивается на 2, если признак направления сброшен,или уменьшается на 2, если признак направления установлен. Тем самым подготавливается к сравнению следующее слово.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	19	1	1	SCASW
(повтор)	9+19/rep	1/rep	1	REPNE SCASW

Примечания:

Команда SCASW полезна в тех случаях, когда требуется найти ячейку с заданным словом. Если Вы хотите сравнить две строки из памяти поэлементно, то используйте команду CMPSW.

Пример.

Пусть имеет место следующее определение :

```
LOST_A  DB  100 dup (?) .
```

Тогда следующий пример описывает поиск символа 'A' в блоке памяти длиной 100 байтов, начинающемся с LOST_A :

```
  MOV AX,DS
  MOV ES,AX   ;SCASW использует ES:DI,
```


```
  ;поэтому копируем DS в ES
 CLD   ;сканировать будем в прямом направлении
 MOV AL,'A'  ;ищем "потерянное" 'A'
 MOV CX,50   ;сканировать будем 50 слов
  ;(CX используется в REPNE)
 LEA DI,LOST_A ;засылаем стартовый адрес в DI
 REPNE SCASW  ;ищем 'A'
 JE  FOUND   ;признак ZF равен 1, если мы нашли 'A'
NOTFOUND:   . ;если мы попали сюда,
    . ;то 'A' не обнаружено
    .
  FOUND:DEC DI ;возвращаем указатель DI
  ;на первое обнаруженное
 DEC DI ;вхождение 'A'
    .
    .
```

После выхода из цикла REPNE SCASW, ZF=1, если вхождение 'A' было обнаружено, и ZF=0, иначе. В первом случае DI указывает на слово, следующее за словом, где находится 'A', поэтому мы производим коррекцию указателя DI с помощью двух DEC DI.

2.4.17 STOS Запись в строку (из байтов или слов)

Признаки не меняются.

Команда: STOS destination-string.

Логика: (ES:DI) = Accumulator

```
   if DF = 0
 DI = DI + n  ; n = 1 для байта, 2 - для слова
   else
 DI = DI - n .
```

Команда STOS копирует байт или слово, расположенное в AL или AX, в место памяти, на которое указывает (ES:DI), а также инкрементирует или декрементирует DI (в зависимости от состояния признака направления DF), чтобы подготовиться к копированию аккумулятора в следующую ячейку (байт или слово) памяти.

Операнды байты(слова)	Такты	Обращения	Байты	Пример
dest-str	11(15)	1	1	STOS WORD_ARRAY
dest-str (повтор)	9+10(14)/rep	1/rep	1	REP STOS WORD_ARRAY

Примечания:

Эта команда всегда ассемблируется или как STOSB, запись в строку из байтов, или как STOSW, запись в строку из слов, в зависимости от того, указывает ли destination-string на строку байтов или на строку слов. Однако, в обоих случаях Вы должны в явном виде загрузить в регистр DI смещение строки.

Пример.

Если команду записи в строку использовать в сочетании с префиксом REP, то такая команда будет полезна для инициализации блока памяти; следующий пример иллюстрирует инициализацию сто байтового блока памяти, расположенного по адресу BUFFER, в 0 :

```
    MOV AL,0   ;значение,которое присваиваем
    ;при инициализации
    LEA DI,BUFFER ;загружаем стартовый адрес блока памяти
    MOV CX,100  ;размер блока памяти
    CLD   ;будем двигаться в прямом направлении
REP STOS BUFFER  ;сравните эту строку с ПРИМЕРом для STOSB
```

2.4.18 STOSB Запись в строку из байтов

Признаки не меняются.

Команда: STOSB

логика: (ES:DI) = AL

```
   if DF = 0
 DI = DI + 1
   else
 DI = DI - 1 .
```

Команда STOSB копирует байт, расположенный в AL, в место памяти, на которое указывает ES:DI, а также инкрементирует или декрементирует DI (в зависимости от состояния признака направ ления DF), чтобы подготовиться к копированию AL в следующий байт памяти.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	11	1	1	STOSB
(повтор)	9+10/rep	1/rep	1	REP STOSB

Пример.

Если команду записи в строку использовать в сочетании с префиксом REP, то такая команда будет полезна для инициализации блока памяти; следующий пример иллюстрирует инициализацию стобайтового блока памяти, расположенного по адресу BUFFER, в 0 :

```
    MOV AL,0   ;значение,которое присваиваем
    ;при инициализации
    LEA DI,BUFFER ;загружаем стартовый адрес блока памяти
    MOV CX,100  ;размер блока памяти
    CLD   ;будем двигаться в прямом направлении
REP STOSB  ;сравните эту строку с ПРИМЕРом для STOS
```

2.4.19 STOSW Запись в строку из слов

Признаки не меняются.

Команда: STOSW.

логика: (ES:DI) = AX

```
   if DF = 0
 DI = DI + 2
   else
 DI = DI - 2 .
```

Команда STOSW копирует слово, расположенное в AX, в место памяти, на которое указывает ES:DI, а также инкрементирует или декрементирует DI (в зависимости от состояния признака направления DF), чтобы подготовиться к копированию AX в следующее слово памяти.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	15	1	1	STOSW
(повтор)	9+14/rep	1/rep	1	REP STOSW

Пример.

Если команду записи в строку использовать в сочетании с префиксом REP, то такая команда будет полезна для инициализации блока памяти; следующий пример иллюстрирует инициализацию стобайтового блока памяти, расположенного по адресу BUFFER, в 0 :

```
    MOV AL,0   ;значение,которое присваиваем
    ;при инициализации
    LEA DI,BUFFER ;загружаем стартовый адрес блока памяти
    MOV CX,50   ;размер блока памяти (в словах)
    CLD   ;будем двигаться в прямом направлении
REP STOSW  ;сравните эту строку с ПРИМЕРом для STOS
```

2.5 Команды передачи управления

2.5.1 CALL Вызов подпрограммы

Признаки не меняются.

Команда: CALL procedure_name .

Логика: if FAR CALL (внешний сегмент)

```
  PUSH CS
  CS = dest_seg
   PUSH IP
   IP = dest_offset .
```

CALL передает управление подпрограмме, которая может находиться как внутри текущего сегмента (NEAR-proc), так и вне его (FAR-proc). Этим двум типам CALL соответствуют две различные машинные команды, и команда RET возврата из подпрограммы должна соответствовать типу команды CALL (потенциальная возможность несоответствия возникает, когда подпрограмма и команда CALL ассемблируются раздельно).

Операнды	Такты	Обращения	Байты	Пример
байты				
(слова)				

near-proc	19(23)	1	3	CALL NEAR_PROC
far-proc	28(36)	2	5	CALL FAR_PROC
память-указатель16	21(29)+EA	2	2-4	CALL PROC[SI]
регистр-указатель16	16(24)	1	2	CALL AX
память-указатель32	37(57)+EA	4	2-4	CALL [BX].ROUTINE

Примечания:

Если подпрограмма находится в другом сегменте, то процессор засылает в стек сначала текущее значение CS, затем текущее значение IP (IP указывает на команду, следующую за командой CALL), а затем передает управление в подпрограмму. Если же подпрограмма находится в том же сегменте, то процессор засылает в стек сначала текущее значение IP (которое опять же указывает на команду, следующую за командой CALL), а затем передает управление в подпрограмму. CALL также может считать адрес подпрограммы из регистра или из памяти. Эта форма команды CALL называется косвенным вызовом.

2.5.2 JMP Безусловный переход

Признаки не меняются .

Команда: JMP target .

Условие перехода: переход осуществляется всегда.

Команда JMP всегда передает управление в место, определяемое операндом target. В отличие от команды CALL, JMP не запоминает значение IP, т.к. появление команды возврата RET не ожидается. Переход внутри сегмента может быть задан как операндом типа память, так и через 16-битный регистр. Переход во внешний сегмент может быть задан только через операнд типа память.

Операнды	Такты	Обращения	Байты	Пример
short-label	15	-	2	JMP ROPE_NEAR
near-label	15	-	3	JMP SAME_SEGMENT
far-label	15	-	5	JMP FAR_LABEL
пам.-указатель16	18 + EA	-	2-4	JMP SAME_SEG
рег.-указатель16	11	-	2	JMP BX
пам.-указатель32	24 + EA	-	2-4	JMP NEXT_SEG

Примечания:

Если ассемблер может определить, что в случае перехода внутри сегмента цель перехода находится в пределах 127 байтов от места расположения текущей команды, то ассемблер автоматически сгенерирует двухбайтовую команду (короткий переход); в противном случае сгенерируется трехбайтовый NEAR JMP. В целях генерации двухбайтовой команды Вы можете сделать "подсказку" ассемблеру, используя специальный оператор "short" :

```
JMP short near_by
```

2.5.3 RET Возврат из подпрограммы

Признаки не меняются .

Команда: RET optional-pop-value .

Логика:

```
if FAR RETURN (внешний сегмент)
  POP CS
SP = SP + optional-pop-value (если оно имеется) .
```

Команда RET передает управление из вызванной подпрограммы команде, следовавшей непосредственно за CALL, и делает это следующим образом :

- пересылает слово из верхушки стека в IP;
- если возврат осуществляется во внешний сегмент, то пересылает слово из новой верхушки стека в CS;
- увеличивает SP на значение операнда optional-pop-value, если оно задано.

Ассемблер сгенерирует возврат во внутренний сегмент, если подпрограмма, содержащая RET, будет обозначена программистом как NEAR, и возврат во внешний сегмент, - если как FAR. Операнд optional-pop-value определяет значение, которое надо прибавить к SP, что имеет смысл "освобождения" стека от "лишних" байтов (например, от входных параметров, когда они передаются подпрограмме через стек).

Операнды	Такты	Обращения	Байты	Пример
(внутр.сегм.,без pop)	20	1	1	RET
(внутр.сегм., с pop)	24	1	3	RET 4
(внешн.сегм.,без pop)	32	2	1	RET

(внешн.сегм., с pop)	31	2	3	RET 2

2.6. Команды условного перехода

2.6.1 JA Переход если выше

Признаки не меняются .
Команда: JA short-label .
Условие перехода: Jump if CF = 0 and ZF = 0 .

Команда JA используется после команд CMP и SUB и передает управление по метке short-label, если первый операнд (который должен быть числом без знака) был больше, чем второй операнд (также без знака). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JA ABOVE

Примечания:
Команда JNBE, переход если не ниже и не равно, - это та же команда, что и JA.
Команду JA, переход если выше, следует использовать при сравнении чисел без знака.
Команду JG, переход если больше, следует использовать при сравнении чисел со знаком.

2.6.2 JAE Переход если выше или равно

Признаки не меняются .
Команда: JAE short-label .
Условие перехода: Jump if CF = 0

Команда JAE используется после команд CMP или SUB и передает управление по метке short-label, если первый операнд был больше или равен второму. (Оба операнда рассматриваются как числа без знака.) Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JAE ABOVE_EQUAL

Примечания:
Команда JNB, переход если не ниже, - это та же команда, что и JAE.Команду JAE, переход если выше или равно, следует использовать при сравнении чисел без знака. Команду JGE, переход если больше или равно, следует использовать при сравнении чисел со знаком.

2.6.3 JB Переход если ниже

Признаки не меняются.
Команда: JB short-label.
Условие перехода: Jump if CF = 1 .

Команда JB используется после команд CMP и SUB и передает управление по метке short-label, если первый операнд был меньше, чем второй. (Оба операнда рассматриваются как числа без знака.) Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JB BELOW

Примечания:
Команды JC (переход если перенос), JB и JNAE (переход если не выше и не равно) все являются синонимами одной и той же команды.Команду JB, переход если ниже, следует использовать при сравнении чисел без знака.
Команду JL, переход если меньше, следует использовать при сравнении чисел со знаком.

2.6.4 JBE Переход если ниже или равно

Признаки не меняются.
Команда: JBE short-label.
Условие перехода: Jump if CF = 1 or ZF = 1 .

Команда JBE используется после команд CMP и SUB и передает управление по метке short-label, если первый операнд был меньше или равен второму. (Оба операнда рассматриваются как числа без знака.) Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JBE NOT_ABOVE

Примечания:

Команда JNA, переход если не выше, - это та же команда, что и JBE.Команду JBE, переход если ниже или равно, следует использовать при сравнении чисел без знака.

Команду JLE, переход если меньше или равно, следует использовать при сравнении чисел со знаком.

2.6.5 JC Переход если перенос

Признаки не меняются .

Команда: JC short-label .

Условие перехода: Jump if CF = 1 .

Команда JC передает управление по метке short-label, если признак переноса CF установлен (т.е. =1). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JC CARRY_SET

Примечания:

Команды JB (переход если ниже), JC и JNAE (переход если не выше и не равно) все являются синонимами одной и той же команды.Пользуйтесь командой JNC, переход если нет переноса, для перехода в том случае, когда признак переноса CF сброшен (т.е.=0).

2.6.6 JCXZ Переход если CX = 0

Признаки не меняются .

Команда: JCXZ short-label .

Условие перехода: Jump if CX = 0 .

Команда JCXZ передает управление по метке short-label, если регистр CX равен 0. Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	18 или 6	-	2	JCXZ COUNT_DOWN

Примечание: Эта команда обычно применяется в начале цикла, чтобы пропустить тело цикла, когда переменная счетчика (CX) равна нулю.

2.6.7 JE Переход если равно

Признаки не меняются .

Команда: JE short-label.

Условие перехода: Jump if ZF = 1 .

Команда JE используется после команд CMP и SUB и передает управление по метке short-label, если первый операнд был равен второму. Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JE ZERO

Примечания:

Команда JZ, переход если ноль, - это та же команда, что и JE.

2.6.8 JG Переход если больше

Признаки не меняются.

Команда: JG short-label .

Условие перехода: Jump if ZF = 0 and SF = OF .

Команда JG используется после команд CMP или SUB и передает управление по метке short-label, если первый операнд был больше, чем второй. (Оба операнда рассматриваются как числа со знаком.) Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JG GREATER

```
 ---------------------------------------------------------------
    Примечания:
    Команда JNLE, переход если не меньше и не равно, - это та же команда, что и JG.
    Команду JA,  переход если выше, следует использовать при сравнении чисел без знака.
    Команду JG, переход если больше, следует использовать при сравнении чисел со знаком.

    2.6.9  JGE Переход если больше или равно
 Признаки не меняются .
 Команда: JGE short-label .
 Условие перехода: Jump if SF = OF .
    Команда JGE используется после команд CMP или SUB и передает управление по метке short-
label, если первый операнд был больше или равен второму.
 (Оба операнда рассматриваются как числа со  знаком.)  Цель перехода должна лежать в
пределах от -128 до 127 байтов от следующей команды.
 ---------------------------------------------------------------
 Операнды      Такты    Обращения  Байты    Пример
 short-label  16 или 4      -        2      JGE GREATER_EQUAL
 ---------------------------------------------------------------
    Примечания:
Команда JNL,переход если не меньше, - это та же команда, что и JGE. Команду JAE, переход
если выше или равно, следует использовать при сравнении чисел без знака.     Команду JGE,
переход если больше или равно, следует использовать при сравнении чисел со знаком.

2.6.10  JL Переход если меньше
 Признаки не меняются.
 Команда: JL short-label .
 Условие перехода: Jump if SF <> OF .
    Команда JL используется после команд CMP или SUB и передает управление по метке short-
label, если первый операнд был меньше, чем второй. (Оба операнда рассматриваются как числа
со знаком.) Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей
команды.
 ---------------------------------------------------------------
 Операнды       Такты    Обращения    Байты    Пример
 short-label  16 или 4       -          2     JL LESS
 ---------------------------------------------------------------
    Примечания:
    Команда JNGE, переход если не больше и не равно, - это та же команда, что и JL.
    Команду JB,  переход если ниже, следует использовать при сравнении чисел без знака.
    Команду JL, переход если меньше, следует использовать при сравнении чисел со знаком.

   2.6.11  JLE Переход если меньше или равно
 Признаки не меняются .
 Команда: JLE short-label .
 Условие перехода: Jump if SF <> OF or ZF = 1 .
    Команда JLE используется после команд CMP или SUB и передает управление по метке short-
label, если первый операнд был меньше или равен второму. (Оба операнда рассматриваются как
числа со  знаком.)  Цель перехода должна лежать в пределах от -128 до 127 байтов от
следующей команды.
 ---------------------------------------------------------------
 Операнды       Такты   Обращения  Байты  Пример
 short-label  16 или 4      -        2   JLE NOT_GREATER
 ---------------------------------------------------------------
    Примечания:
Команда JNG, переход если не больше, - это та же команда, что и JLE.Команду JBE, переход
если ниже или равно, следует использовать при сравнении чисел без знака. Команду JLE,
переход если меньше или равно, следует использовать при сравнении чисел со знаком.

     2.6.12  JNA Переход если не выше
 Признаки не меняются .
 Команда: JNA  short-label .
 JNA - синоним JBE. См. описание JBE.

  2.6.13  JNAE Переход если не выше и не равно
 Признаки не меняются .
 Команда: JNAE  short-label .
```

JNAE - синоним JB. См. описание JB.

2.6.14 JNB Переход если не ниже
Признаки не меняются.
Команда: JNB short-label .
JNB - синоним JAE. См. описание JAE.

2.6.15 JNBE Переход если не ниже и не равно
Признаки не меняются.
Команда: JNBE short-label .
JNBE - синоним JA. См. описание JA.

2.6.16 JNC Переход если нет переноса
Признаки не меняются.
Команда: JNC short-label .
Условие перехода: Jump if CF = 0 .

Команда JNC передает управление по метке short-label, если признак переноса CF сброшен (т.е. =0). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JNC CARRY_CLEAR

Примечание:
Пользуйтесь командой JC, переход если перенос, для перехода в том случае, когда признак переноса CF установлен (т.е. =1).

2.6.17 JNE Переход если не равно
Признаки не меняются .
Команда: JNE short-label .
Условие перехода: Jump if ZF = 0
Команда JNE используется после команд CMP и SUB и передает управление по метке short-label, если первый операнд не был равен второму. Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JNE NOT_EQUAL

Примечание:
Команда JNZ, переход если не ноль, - это та же команда, что и JNE.

2.6.18 JNG Переход если не больше
Признаки не меняются .
Команда: JNG short-label .
JNG - синоним JLE. См. описание JLE.

2.6.19 JNGE Переход если не больше и не равно
Признаки не меняются .
JNGE short-label .
JNGE - синоним JL. См. описание JL.

2.6.20 JNL Переход если не меньше
Признаки не меняются .
Команда: JNL short-label .
JNL - синоним JGE. См. описание JGE.

2.6.21 JNLE переход если не меньше и не равно
Признаки не меняются .
Команда: JNLE short-label .
JNLE - синоним JG. См. описание JG.

2.6.22 JNO Переход если нет переполнения
Признаки не меняются.
Команда: JNO short-label .

Условие перехода: Jump if OF = 0 .

Команда JNO передает управление по метке short-label, если признак переполнения OF сброшен (т.е. =0). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JNO NO_OVERFLOW

Примечание:

Пользуйтесь командой JO, переход если переполнение, для перехода в том случае, когда признак переполнения OF установлен (т.е. =1).

2.6.23 JNP Переход если нечетно

Признаки не меняются .

Команда: JNP short-label .

Условие перехода: Jump if PF = 0 .

Команда JNP передает управление по метке short-label, если признак четности PF сброшен (т.е. =0). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JNP ODD_PARITY

Примечание: Команда JPO, переход если нечетно, - это та же команда, что и JNP. Пользуйтесь командой JP, переход если четно, для перехода в том случае, когда признак четности PF установлен (т.е. =1).

2.6.24 JNS Переход если положительный результат

Признаки не меняются.

Команда: JNS short-label .

Условие перехода: Jump if SF = 0 .

Команда JNS передает управление по метке short-label, если признак знака SF сброшен (т.е. =0). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JNS AQUARIUS

Примечание:

Пользуйтесь командой JS, переход если отрицательный результат, для перехода в том случае, когда признак знака SF установлен (т.е. =1).

2.6.25 JNZ Переход если не ноль

Признаки не меняются .

Команда: JNZ short-label .

JNZ - синоним JNE. См. описание JNE.

2.6.26 JO Переход если есть переполнение

Признаки не меняются .

Команда: JO short-label .

Условие перехода: Jump if OF = 1 .

Команда JO передает управление по метке short-label, если признак переполнения OF установлен (т.е. =1). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JO SIGNED_OVERFLOW

Примечание:

Пользуйтесь командой JNO, переход если нет переполнения,для перехода в том случае, когда признак переполнения OF сброшен (т.е. =0).

2.6.27 JP Переход если четно

Признаки не меняются .

Команда: JP short-label .

Условие перехода: Jump if PF = 1 .

Команда JP передает управление по метке short-label, если признак четности PF установлен (т.е. =1). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JP EVEN_PARITY

Примечание:
Команда JPE, переход если четно, - это та же команда, что и JP.Пользуйтесь командой JNP, переход если нечетно, для перехода в том случае, когда признак четности PF сброшен (т.е. =0).

2.6.28 JPE Переход если четно

Признаки не меняются .
Команда: JPE short-label .
JPE - синоним JP. См. описание JP.

2.6.29 JPO Переход если нечетно

Признаки не меняются .
Команда: JPO short-label .
JPO - синоним JNP. См. описание JNP.

2.6.30 JS Переход если отрицательный результат

Признаки не меняются .
Команда: JS short-label .
Условие перехода: Jump if SF = 1 .
Команда JS передает управление по метке short-label, если признак знака SF установлен (т.е. =1). Цель перехода должна лежать в пределах от -128 до 127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	16 или 4	-	2	JS NEGATIVE

Примечания:
Пользуйтесь командой JNS, переход если положительный результат, для перехода в том случае, когда признак знака SF сброшен (т.е. =0).

2.6.31 JZ Переход если ноль

Признаки не меняются .
Команда: JZ short-label .
JZ - синоним JE. См. описание JE.

2.6.32 LOOP Переход по счетчику

Признаки не меняются .
LOOP short-label .
логика: CX = CX - 1
if (CX <> 0)
JMP short-label .
Команда LOOP уменьшает CX на 1, затем передает управление по метке short-label, если CX не равно 0. Операнд short-label должен находиться в пределах от -128 до +127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	17/5	-	2	LOOP AGAIN

2.6.33 LOOPE Переход пока равно

Признаки не меняются.
Команда: LOOPE short-label .
Логика: CX = CX - 1
if (CX <> 0) and (ZF = 1)
JMP short-label .
Команда LOOPE используется после команд CMP или SUB.Она уменьшает CX на 1, затем передает управление по метке short-label, если CX не равно нулю и если первый операнд команд CMP или SUB был равен второму операнду. Операнд short-label должен находиться в пределах от -128 до +127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример

Операнды	Такты	Обращения	Байты	Пример
short-label	18 или 6	-	2	LOOPE AGAIN

Примечание: Команда LOOPZ, переход пока ноль, - это та же команда, что и LOOPE.

2.6.34 LOOPNE Переход пока не равно

Признаки не меняются .

Команда: LOOPNE short-label .

Логика:
```
CX = CX - 1
if (CX <> 0) and (ZF = 0)
 JMP short-label .
```

Команда LOOPNE используется после команд CMP или SUB. Она уменьшает CX на 1, затем передает управление по метке short-label, если CX не равно нулю и если первый операнд команд CMP или SUB не равен второму операнду. Операнд short-label должен находиться в пределах от -128 до +127 байтов от следующей команды.

Операнды	Такты	Обращения	Байты	Пример
short-label	18 или 6	-	2	LOOPNE AGAIN

Примечания:
Команда LOOPNZ, переход пока не ноль, - это та же команда, что и LOOPNE.

2.6.35 LOOPNZ Переход пока не ноль

Признаки не меняются .

Команда: LOOPNZ short-label .

LOOPNZ - синоним LOOPNE. См. описание LOOPNE.

2.6.36 LOOPZ Переход пока ноль

Признаки не меняются .

Команда: LOOPZ short-label .

LOOPZ - синоним LOOPE. См. описание LOOPE.

2.7 Команды прерывания

2.7.1 INT Прерывание

Признаки: O D I T S Z A P C
0 0 .

Команда: INT interrupt-num .

Логика :
```
PUSHF   ;загрузка регистра FLAGS в стек
 TF = 0  ;сброс разряда трассировки
 IF = 0  ;запрещаем прерывания
 CALL FAR (INT*4) ;вызываем обработчик прерываний
```

Команда INT загружает регистр FLAGS в стек, сбрасывает признаки трассировки и разрешения прерывания, загружает CS и IP в стек, затем передает управление обработчику прерываний, который определяется по значению операнда interrupt-num. Если обработчик прерываний производит возврат по команде IRET, то исходное значение регистра FLAGS восстанавливается.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
непоср.8 (тип=3)	52	5	1	INT 3
непоср.8 (тип<>3)	51	5	2	INT 21

Примечания :Регистр FLAGS хранится в том же формате, который используется в команде PUSHF. Адрес вектора прерывания определяется умножением операнда interrupt-num на 4. Первое слово, находящееся по полученному адресу, загружается в IP, а второе слово - в CS. Все номера interrupt-num, кроме типа 3, вырабатывают двухбайтовый код операции; interrupt-num, равный 3, вырабатывает однобайтовую команду, называемую прерыванием по контрольной точке (Breakpoint interrupt).

2.7.2 INTO Прерывание по переполнению

Признаки: O D I T S Z A P C
0 0 .

Команда: INTO.

Логика:
```
if (OF = 1)
PUSHF   ;загрузка регистра FLAGS в стек
TF = 0  ;сброс разряда трассировки
```


```
  IF = 0  ;запрещаем прерывания
  CALL FAR (10h)  ;вектор прерывания INTO расположен
    ;по адресу 0000:0010h
```

Команда INTO активизирует прерывание типа 4, если признак переполнения OF равен 1; если OF = 0, то эта команда не выполняет никаких действий. Если OF = 1, то прерывание выполняется аналогично команде INT 4; в этом случае INTO загружает регистр FLAGS в стек, сбрасывает признаки трассировки и разрешения прерывания, загружает CS и IP в стек, затем передает управление обработчику прерываний, соответствующему типу 4 и на который
указывает вектор по адресу 10h. Если обработчик прерываний производит возврат по команде IRET, то исходное значение регистра FLAGS восстанавливается.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
нет операндов	53 или 4	5	1	INTO

Примечание : Регистр FLAGS хранится в том же формате, который используется в команде PUSHF. INTO может быть использован после операции, которая могла бы привести к переполнению, в целях вызова подпрограммы восстановления.

2.7.3 IRET Возврат после обработки прерывания

```
Признаки: O D I T S Z A P C
          r r r r r r r r .
Команда: IRET .
Логика : POP IP
  POP CS
  POPF  ;пересылка слова из стека в регистр FLAGS
```

Команда IRET передает управление из подпрограммы обработки прерываний в место возникновения прерывания, восстанавливая из стека значения регистров IP, CS и FLAGS.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	32	3	1	IRET

2.8 Управление состоянием процессора

2.7.1 CLC Сброс признака переноса

```
Признаки: O D I T S Z A P C
                        0 .
Команда: CLC .
Логика:   CF = 0 .
```

CLC сбрасывает признак переноса. Другие признаки не меняются.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	CLC

2.7.2 CLD Сброс признака направления

```
Признаки: O D I T S Z A P C
                    0    .
Команда: CLD .
```

Логика: DF = 0 (Разрешает инкремент в командах обработки строк).CLD сбрасывает (устанавливает равным нулю) признак направления. Другие признаки не меняются. Сброшенный признак направления влечет увеличение SI и DI на единицу в командах обработки строк.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	CLD

Примечание: Команды обработки строк увеличивают SI и DI на единицу, когда DF=0.

2.7.3 CLI Сброс признака разрешения прерывания

```
Признаки: O D I T S Z A P C
                    0    .
Команда: CLI .
```

Логика: IF = 0 .

CLI сбрасывает (устанавливает равным нулю) признак разрешения прерывания, вследствие чего процессор не распознает замаскированные прерывания. Другие признаки не меняются. (Немаскированные прерывания распознаются процессором всегда, независимо от значения признака IF.)

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	CLI

2.7.4 CMC Инвертирование признака переноса

Признаки: O D I T S Z A P C
* .

Команда: CMC .

Логика: CF = -CF .

CMC меняет текущее значение признака переноса на противоположное.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	CMC

2.7.5 ESC Выборка кода операции и операнда

Признаки не меняются .

Команда: ESC opcode,source .

Команда ESC используется для передачи управления от микропроцессора внешнему процессору, такому как 8087 или 80287. В ответ на ESC микропроцессор выбирает код операции для внешнего процессора (opcode) и код операнда source и помещает их в шину BUS. Внешний процессор поджидает команду ESC и выполняет команду, размещенную в шине, используя исполнительный адрес source.

Операнды	Такты	Обращения	Байты	Пример
байт(слово)				
непоср.,память	8(12)+EA	1	2-4	ESC 6,ADR[SI]
непоср.,регистр	2	-	2	ESC OPCODE,AH

Примечания:

В целях синхронизации с внешним процессором программист должен перед каждой командой ESC помещать команду WAIT. Процессоры 80288 и 80386 обладают средствами автоматической синхронизации команд, поэтому для них команды WAIT можно опустить.

2.7.6 HLT Останов

Признаки не меняются .

Команда: HLT .

Эта команда производит останов ЦП и переводит его в состояние ожидания сигнала сброса или сигнала немаскированного прерывания.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	HLT

2.7.7 LOCK Блокирование шины BUS

Признаки не меняются.

Команда: LOCK .

LOCK - это однобайтный префикс, который может предшествовать любой команде. LOCK заставляет процессор выработать сигнал блокировки шины на время выполнения последующей команды. Использование сигнала блокировки делает шину недоступной для любого внешнего устройства или события, включая прерывания и передачу данных.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	LOCK XCHG FLAG,AL

Примечание:

Эта команда была предусмотрена для поддержки мультипроцессорных систем с разделенными ресурсами. В такого рода системах доступ к этим ресурсам контролируется обычно через аппаратное и программное обеспечение с использованием семафоров.

Эту команду следует использовать только с целью предотвращения прерывания операций по пересылке данных.Поэтому ее следует употреблять только перед командами XCHG,MOV и MOVS.

2.7.8 NOP Нет операции

Признаки не меняются.
Команда: NOP .
Логика: нет .
Команда NOP является пустым оператором. Она часто используется в целях подгонки времени, для выравнивания памяти и как "держатель места".

Операнды	Такты	Обращения	Байты	Пример
нет операндов	3	-	1	NOP

2.7.9 STC Установка признака переноса

Признаки: O D I T S Z A P C
1 .
Команда: STC .
Логика: CF = 1
STC устанавливает признак переноса CF в единицу. Другие признаки не меняются.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	STC

2.7.10 STD Установка признака направления

Признаки: O D I T S Z A P C
1 .
Команда: STD .
Логика: DF = 1 (декремент в командах обработки строк)
STD устанавливает признак направления DF в единицу.Другие признаки не меняются. Установление DF в 1 влечет изменение SI и DI в сторону уменьшения в командах обработки строк.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	STD

2.7.11 STI Установка признака разрешения прерывания

Признаки: O D I T S Z A P C
1 .
Команда: STI .
Логика: IF = 1 .
STI устанавливает признак разрешения прерывания в единицу, разрешая процессору распознавать маскированные прерывания. Другие признаки не меняются. (Немаскированные прерывания распознаются процессором всегда, независимо от значения признака IF.)

Операнды	Такты	Обращения	Байты	Пример
нет операндов	2	-	1	STI

Примечание:
Ожидающее прерывание не будет распознано, пока не выполнится команда, следующая сразу за STI.

2.7.12 WAIT Ожидание

Признаки не меняются .
Команда: WAIT .
Команда WAIT переводит процессор в состояние ожидания. Процессор будет оставаться в неактивном состоянии, пока на входной линии TEST микропроцессора не появится сигнал.

Операнды	Такты	Обращения	Байты	Пример
нет операндов	3+5n	-	1	WAIT

Примечание:

Эта команда позволяет "синхронизировать" микропроцессор внешними сигналами без использования запросов на прерывание.

3. ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ

3.1 Общие сведения

Текст исходной программы состоит из операторов ассемблера, каждый из которых занимает отдельную строку этого текста. Различают два типа операторов: инструкции и директивы.Первые при трансляции преобразуются в команды процессора, которые исполняются после загрузки в память загрузочного модуля программы, имеющего расширение .COM или .EXE . Операторы второго типа управляют процессом ассемблирования - преобразования текста исходной программы в коды объектного модуля (расширение .OBJ).Ассемблер интерпретирует и обрабатывает операторы один за другим, генерируя последовательность из команд процессора и байтов данных.

Особо следует остановиться на использовании макрокоманд.При программировании на макроассемблере можно формировать обращение к часто повторяющейся последовательности команд при помощи одного оператора. Этот прием несколько напоминает вызов подпрограмм в языках высокого уровня, но между ними лежит значительное различие, заключающееся в том, что подпрограмма, занимающая некоторый участок памяти, может быть исполнена неограниченное число раз путем передачи ей управления из вызывающей программы, в которую подпрограмма сама затем возвращает управление. В ассемблере используются макровызовы макроопределений. Макроопределение - это последовательность операторов, которые могут содержать формальные параметры. Макроопределение и команда обращения к макроопределению (макровызов) образуют макрокоманду. Макровызов - это оператор вызова макроопределения. Если макроопределение содержит формальные параметры, то макровызов обязан содержать фактические значения этих параметров, которые будут подставлены вместо соответствующих формальных. В результате макровызовва формируется реальная последовательность команд - макрорасширение. Макрорасширение вставляется в исходный текст программы на место оператора макровызова. Таким образом,в исходный текст программы макрорасширение одного и того же макроопределения может быть вставлено несколько раз, по числу макровызовов. Каждое макрорасширение после трансляции естественно занимает свой участок памяти.

В ассемблере имеется три вида вызова подпрограмм.

- CALL NEAR (короткий вызов);
- CALL FAR (длинный вызов);
- INT (прерывание),

но ни одн из них не содержит явного механизма передачи параметров. Этот механизм определяется внутренней организацией вызываемой подпрограммы.

Ниже приведен список слов зарезервированных для ассемблера, использование которых в иных целях запрещено.

```
 .186 DI .ERRNZ    LENGTH    .SALL
 .286c DL ES  .LFCOND   SEG
 .286p DQ EVEN .LIST SEGMENT
 .287 DS EXITM LOCAL .SFCOND
 .8086 DT EXTRN LOW  SHL
 .8087 DW FAR  LT  SHORT
 =   DWORD   GE  MACRO SHR
 AH  DX GROUP MASK SI
 AL  ELSE    GT  MOD  SIZE
 AND  END HIGH NAME SP
 ASSUME    ENDIF   IF  NE  SS
 AX  ENDM    IF1  NEAR STRUC
 BH  ENDP    IF2  NOT  SUBTTL
 BL  ENDS    IFB  OFFSET    TBYTE
 BP  EQ IFDEF OR  .TFCOND
 BX  EQU IFDIF ORG  THIS
 BYTE .ERR    IFE  %OUT TITLE
 CH  .ERR1   IFIDN PAGE TYPE
 CL  .ERR2   IFNB PROC .TYPE
 COMMENT   .ERRB   IFNDEF    PTR  WIDTH
 .CREF .ERRDEF  INCLUDE   PUBLIC    WORD
 CS  .ERRDIF  IRP  PURGE .XALL
 CX  .ERRE   IRPC QWORD .XCREF
 DB  .ERRIDN  LABEL .RADIX    .XLIST
 DD  .ERRNB   .LALL RECORD    XOR
 DH  .ERRNDEF  LE  REPT
```

Примечания:

Данный список не включает мнемонических обозначений, используемых для набора команд. Данные слова не зависят от типа букв, это означает, что и PURGE, и purge являются зарезервированными словами.

Общий формат оператора ассемблера имеет следующий вид:
[Метка:]Код_операции[Операнд1[,Операнд2]][;Комментарий],где элементы, указанные в квадратных скобках, могут отсутствовать.Пробелы вводятся произвольно, но минимум один пробел должен быть после кода операции.
Метка - это идентификатор, присваевыемый первому байту того оператора, в котором она появляется.
Код_операции - это мнемоническое обозначение соответствующих команд процессора.
Операнды оператора ассемблера описываются выражениями. Выражения конструируются на основе операций над числовыми и текстовыми константами, метками и идентификаторами переменных с использованием знаков операций и некоторых зарезервированных слов.
Ниже приведены все определенные в ассемблере операции.
Порядок старшинства операций от высшей к низшей:

```
LENGTH, SIZE, WIDTH, MASK, (), [], <>
.
:
PTR, OFFSET, SEG, TYPE, THIS
HIGH, LOW
+ (unary), - (unary)
*, /, MOD, SHL, SHR
+, -
EQ, NE, LT, LE, GT, GE
NOT
AND
OR, XOR
SHORT, .TYPE
```

Старшинство операций определяет порядок, по которому будет вычисляться выражение. Более старшие операции будут производиться раньше операций, имеющих меньшее старшинство.
Примечания:
Операции, стоящие в одной строке, имеют равный приоритет.
Операции равного старшинства вычисляются слева направо.
Операции, стоящие в скобках, выполняются первыми.

Пример оператора ассемблера:
`l0c_1: mov ax,(DAT_1+4) SHR 4,`
здесь использованы следующие операции ассемблера: ( ), + и SHR.

3.2 Арифметические операторы

3.2.1 + Сложение или унарный плюс

`expression1 + expression2` (сложение)
или
`+ expression` (унарный плюс)

Бинарный "+" суммирует значения двух выражений. Унарный "+" сохраняет знак и значение выражения.
Примечания:
Оператор сложения ('+') может использоваться для прибавления целого числа к операнду, перемещаемому в памяти. Операндом, перемещаемым в памяти, может быть только одно из выражений. Оба выражения могут быть целыми числами. Унарная операция '+' обладает более высоким приоритетом, чем оператор сложения. Смотри 'Старшинство операций', где описан порядок старшинства операций.

3.2.2 - Вычитание или унарный минус

`expression1 - expression2` (вычитание)
или
`- expression` (унарный минус).
Бинарный "-" вычитает одно выражение из другого.Унарный "-" изменяет знак выражения.
Примечания:
Операндами оператора вычитания могут быть целые числа или операнды, перемещаемые в памяти. Если оба операнда являются адресами памяти, то они должны располагаться в одном и том же сегменте.

Унарная операция '-' обладает более высоким приоритетом, чем оператор вычитания. Смотри 'Старшинство операций', где описан порядок старшинства операций.

3.2.3 * Умножение

expression1 * expression2 .

Перемножает значения двух выражений.

Примечания:

Выражения должны быть целыми числами. Они не могут быть адресами, перемещаемыми в памяти.

3.2.4 / Деление

expression1 / expression2

Делит одно выражение на другое.

Примечания:

Выражения должны быть целыми числами. Они не могут быть адресами, перемещаемыми в памяти.

3.2.5 MOD Деление по модулю

выражение1 MOD выражение2

Выдает остаток от деления.

Примечания:

Оба выражения должны быть целыми числами. Они не должны быть настраиваемыми адресами.

Например, 14 MOD 4 = 2, т.к. 14 / 4 дает остаток 2.

3.3 . Оператор доступа к полю структуры

structvariable.field .

Используется для доступа к полю переменной структурного типа.

Примечания:

Structvariable - это имя переменной, описанной ранее как переменная структурного типа, а field - это имя поля внутри структуры.

Исполнительный адрес, формируемый данным оператором (.), является суммой смещения переменной structvariable и смещения поля field внутри структуры. Этот адрес является относительным внутри той группы или сегмента, где определена переменная structvariable.

Этот оператор эквивалентен использованию оператора сложения (+) с индексными или базовыми операндами.

3.4 [] Оператор индексации

expression1[expression2]

Прибавляет значение выражения expression1 к значению выражения expression2. Этот оператор такой же, как и '+', за тем исключением, что здесь выражение expression1 не является обязательным.

Примечания:

Если выражение expression1 задано, оно может быть целым значением, перемещаемым операндом или абсолютным символом. Expression2 может быть целым значением или абсолютным символом, а также перемещаемым операндом, в том случае, когда выражение expression1 не задано.

Этот оператор используется как правило для доступа к отдельным элементам массива. Например, в следующей команде в регистр AL помещается 3-й байт строки символов:

```
MOV   AL, STRING [2]
```

3.5 Операторы сдвига

3.5.1 SHL Сдвиг влево

выражение SHL count

Сдвигает выражение влево на count битов.

Примечания:

Биты, сдвинутые за конец, теряются. При сдвиге вводятся нули. Если count больше или равен 16, результат равен 0.Если сдвигаемое значение - слово, все 16 бит будут сдвинуты. Однако, если значение - байт, сдвигаются только 8 бит.

3.5.2 SHR Сдвиг вправо

выражение SHR count

Сдвигает выражение вправо на count битов.

Примечания:

Биты, сдвинутые за конец, теряются. При сдвиге вводятся нули. Если count больше или равен 16, результат равен 0.Если сдвигаемое значение - слово, все 16 бит будут сдвинуты. Однако, если значение - байт, сдвигаются только 8 бит.

3.6 Побитовые логические операции

3.6.1 NOT Побитовое отрицание

NOT выражение

Производит над выражением побитовую операцию отрицания(инвертирования). Следующая таблица показывает результат применения операции NOT к одному биту:

Бит	Результат
0	1
1	0

3.6.2 AND Побитное логическое "И"

expression1 AND expression2 .

Производит по битам операцию логического умножения над операндами expression1 и expression2. Следующая таблица показывает результаты применения операции AND к двум битам:

Бит	Бит	Результат
0	0	0
0	1	0
1	0	0
1	1	1

3.6.3 OR Побитовая логическая операция "ИЛИ"

expression1 OR expression2

Производит по битам операцию логического ИЛИ над операндами expression1 и expression2. Следующая таблица показывает результаты применения операции OR к двум битам:

Бит	Бит	Результат
0	0	0
0	1	1
1	0	1
1	1	1

3.6.4 XOR Побитовое логическое "исключающее ИЛИ"

expression1 XOR expression2

Производит по битам операцию логического исключающего ИЛИ над операндами expression1 и expression2. Следующая таблица показывает результаты применения операции XOR к двум битам:

Бит	Бит	Результат
0	0	0
0	1	1
1	0	1
1	1	0

3.7 Операторы отношений

3.7.1 EQ Оператор отношения "равно"

expression1 EQ expression2

Возвращает значение истина (0FFFFh), если выражения expression1 и expression2 равны, иначе возвращает ложь (0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 16-битные числа. Выражения, у которых 16-ый бит равен 1, являются отрицательными. Так, -1 EQ 0FFFFh есть истина.

3.7.2 NE Операция отношения "не равно"

expression1 NE expression2

Возвращает значение истина (0FFFFh), если выражение expression1 не равно выражению expression2, иначе возвращает ложь (0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 16-битные числа. Выражения с ненулевым 16-м битом считаются отрицательными. Так -1 NE 0FFFFh дает ложь.

3.7.3 LT Операция отношения "меньше чем"

expression1 LT expression2

Возвращает значение истина (0FFFFh), если выражение expression1 меньше выражения expression2, иначе возвращает ложь (0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 17-битные числа. 17-й бит задает знак операнда. Так что максимально возможное значение операнда есть 0FFFFh (65535).

3.7.4 GT Оператор отношения "больше" expression1 GT expression2

Возвращает значение истина (0FFFFh), если выражение expression1 больше выражения expression2, иначе возвращает ложь(0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 17-битные числа. 17-й бит задает знак операнда. Так что максимально возможное значение операнда есть 0FFFFh (65535).

3.7.5 LE Оператор отношения "меньше или равно"

expression1 LE expression2

Возвращает значение истина (0FFFFh), если выражение expression1 меньше или равно выражению expression2, иначе возвращает ложь (0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 17-битные числа. 17-й бит задает знак операнда. Так что максимально возможное значение операнда есть 0FFFFh (65535).

3.7.6 GE Оператор отношения "больше или равно"

expression1 GE expression2

Возвращает значение истина (0FFFFh), если выражение expression1 больше или равно выражению expression2, иначе возвращает ложь (0000h).

Примечания:

Оба выражения должны приводиться к абсолютным значениям.Этот оператор трактует операнды, как 17-битные числа. 17-й бит задает знак операнда. Так что максимально возможное значение операнда есть 0FFFFh (65535).

3.8 Оператор явного задания сегмента

: Оператор явного задания сегмента

Оператор MASM

сегментный_регистр:выражение

или

имя_сегмента:выражение

или

имя_группы:выражение .

Этот оператор заставляет вычислять метку или адрес переменной, используя начало сегмента, задаваемого сегментным регистром, сегментным именем или именем группы.

Примечания:

Сегментным регистром может быть один из регистров CS, SS, ES или DS.

Имя сегмента и имя группы должны быть определены командами SEGMENT и GROUP. Они должны быть также присвоены какому-либо сегментному регистру командой ASSUME.

Выражение может быть как перемещаемым операндом, так и абсолютным символом.

В зависимости от команды и от типов операндов, используемый адрес операнда вычисляется относительно одного из регистров ES, DS и SS. Когда же регистр задается в явном виде оператором ':', то эти стандарты не принимаются во внимание.

3.9 Операторы типа

3.9.1 PTR Изменение типа переменной

type PTR выражение .

Временно изменяет тип выражения (которое может быть меткой или переменной) с его стандартного типа на тип type.Параметр type может иметь одно из следующих значений или имен:

Имя	Значение
BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10
NEAR	0FFFFh
FAR	0FFFEh

BYTE (байт), WORD (слово), DWORD (двойное слово) могут использовать только с операндами из памяти. NEAR и FAR могут использоваться только с метками.

Примечания:

Оператор PTR обычно используется для способа доступа к переменной, отличного от указанного при ее определении. Например,для доступа к старшему байту переменной типа WORD (слово). PTR также используется для явного описания типа переменной или метки, являющейся ссылкой вперед.

Пример.

```
MOV  AL, BYTE PTR WORDVAR
CALL  FAR PTR ROUTINE
```

SHORT Установка метки на тип SHORT
(оператор MASM)

3.9.2 SHORT метка

Устанавливает тип метки SHORT (короткий).

Примечания:

Если расстояние между меткой и переходом, ссылающимся на нее, меньше 128 байтов, метка может быть объявлена как метка типа SHORT. Команды, использующие метки SHORT, содержат в себе на 1 байт меньше, чем команды, использующие метки NEAR.

3.9.3 THIS Создание операнда по текущей позиции

THIS type .

Создает операнд типа type, с адресом (сегмент и смещение),равным текущему указателю на размещение в памяти.

Параметр type может быть следующим:

```
BYTE
WORD
DWORD
QWORD
TBYTE
NEAR
FAR  .
```

Этот оператор используется для создания меток и переменных, использующих операцию EQU или знак равенства '=', также как и команда LABEL.

Пример.

```
byteLabel equ THIS BYTE
```

аналогично

```
byteLabel LABEL BYTE.
```

3.9.4. HIGH Возврат старших 8 бит

HIGH expression .

Этот оператор возвращает старшие 8 бит выражения expression.

Пример.

```
MOV   BL, HIGH 1234h
```

Эта команда поместит 12h в BL.

3.9.5 LOW Получение восьми младших битов

LOW expression .

Этот оператор выдает младший байт (8 бит) выражения expression.

Пример.

```
MOV   BL, LOW 1234h
```

Эта команда поместит 34h в BL.

3.9.6 SEG Выдача значения сегмента

SEG выражение .

Выдает значение сегмента, в котором расположено выражение.

Примечания:

Выражение может быть переменной, меткой, именем сегмента,именем группы либо другим символом.

3.9.7 OFFSET Смещение выражения

OFFSET выражение .

Выдает число байт между выражением и началом сегмента, в котором оно определено.

Примечания:

Выражение может быть меткой, именем сегмента или переменной.

Для получения правильного смещения при доступе к переменным в сегментах или в группах, используйте префикс переопределения сегмента ':' для того, чтобы команда OFFSET вычисляла смещение относительно начала группы или сегмента.

3.9.8 .TYPE Выдача режима и контекста для выражения .TYPE выражение .
Выдает байт, описывающий состояние и область доступности(видимости) выражения.
Примечания:
Байт выдается в следующем формате:

Биты
```
  7 6 5 4 3 2 1 0
  % . . . . . . . 1=External;0=Local или Public
  . % . . . . . . Всегда 0
  . . % . . . . . Определено
  . . . % % % . . Всегда 0
  . . . . . . % . В зависимости от данных
  . . . . . . . % В зависимости от программы
```
Если выражение неправильное, весь байт равен 0. Этот оператор обычно используется с условными командами, в которых для управления программой используется состояние пере менной.

3.9.9 TYPE Получение размера типа
TYPE выражение .
Выдает число байт, необходимых для хранения переменной того типа, каким является выражение; для метки NEAR выдает 0FFFFh, а для метки FAR - 0FFFEh.

Примечания:
Выдаваемые значения аналогичны приведенным в описании оператора PTR. Заметьте, что число выдаваемых байт равно числу байт, занимаемых типом выражения. Следовательно, переменная типа слово всегда дает 2, даже если она повторяется 10 раз.

3.9.10 LENGTH Возврат длины переменной
LENGTH переменная .
Возвращает число единиц типа BYTE, WORD, DWORD, QWORD или TBYTE, занимаемых переменной.
Примечания:
Тип переменной определяет, в каких единицах измерения возвращается ее длина.
Значение, возвращаемое этим оператором, есть число, предшествующее оператору DUP в описании переменной. Поэтому, только для переменных, описанных с помощью оператора DUP, будет возвращено значение, отличное от 1. Строковые константы также возвратят 1.

3.9.11 SIZE Выдача количества байт,используемых под переменную
SIZE переменная
Выдает число байт, занимаемых переменной.
Примечания:
Значение, возвращаемое этим оператором, равно длине (LENGTH) переменной, повторенной TYPE (тип) раз:
SIZE переменная=(LENGTH переменная)*(TYPE переменная)

3.10 Использование специальных операторов макрокоманд
3.10.1 & Оператор подстановки
```
&dummyparameter
```
или
```
dummyparameter& .
```
Заставляет ассемблер заменить подставной параметр dummyparameter на значение фактического параметра в тексте макроопределения.
Примечания:
Рассмотрим смысл этого оператора на следующем примере:
```
newlabel   MACRO  x
label&x    db     &x
ENDM
newlabel   0
newlabel   1 .
```

3.10.2 <> Оператор буквального прочтения текста
<текст> .
Трактует текст как единое целое, независимо от того, содержит ли он пробелы, запятые или другие разделители.

Примечания:

Этот оператор используется в макросах или в повторных блоках, чтобы быть уверенным в том, что вызов макрокоманды будет трактоваться как единый параметр.

Этот оператор может также применяться в целях использования спецсимволов, таких как точка с запятой, в качестве литер.Например, точка с запятой сама по себе обозначает комментарий, а <;> обозначает сам символ "точка с запятой". Каждый раз, когда макрокоманде передается параметр, макроассемблер снимает один слой угловых скобок. При использовании вложенных макросов нужно убедиться в наличии достаточного количества угловых скобок.

3.10.3. ! Оператор буквальной интерпретации символа

!символ

Заставляет ассемблер интерпретировать символ буквально.

Примечания:

Этот оператор может быть использован только в макроопределениях (включая повторные блоки) или с командами условного ассемблирования. !символ эквивалентно <символ>; позволяет Вам использовать специальные символы, имеющие особое значение, такие как ';' и '&', просто как некоторые абстрактные символы, равноправные со всеми остальными символами.

3.10.4 % Оператор преобразования в выражение

%текст .

Интерпретирует текст как выражение. Макроассемблер вычислит значение этого выражения в текущей системе счисления и заменит текст на это значение.

Примечания:

Этот оператор редко используется просто для подсчета значения некоторого непосредственно используемого выражения;обычно он используется при вызове макросов, когда передаваемое макрокоманде значение является результатом вычисления значения выражения.

Текст должен быть правильно заданным выражением в текущей системе счисления.

3.10.5 ;; Макрокомментарий

;;текст_комментария .

Позволяет включать комментарий в макроопределение таким образом, что он удаляется при макрорасширении.

3.11 Размещение сегментов, имеющих одинаковые имена в области памяти. Комбинирование сегментов

3.11.1 PUBLIC Соединение одноименных сегментов

segname SEGMENT PUBLIC .

Заставляет компоновщик соединить все сегменты с одинаковым именем сегмента.

Примечания:

Новый соединенный сегмент будет целым и непрерывным. Все команды и адреса данных в этом сегменте будут вычислены относительно начала этого нового сегмента.Если не указано никакого комбинированного типа,сегмент при загрузке в память получит свой собственный физический сегмент.

3.11.2 STACK Определение стекового сегмента

segname SEGMENT STACK .

Заставляет компоновщик соединять все одноименные сегменты и вычислять все адреса в этих сегментах относительно регистра SS.

Примечания:

Комбинированный тип STACK (стек) аналогичен комбинированному типу PUBLIC, за исключением того, что регистр SS является стандартным сегментным регистром для сегментов типа STACK. Регистр SP устанавливается на конец данного объединенного сегмента типа STACK.Если не указано ни одного сегмента типа STACK, компоновщик выдаст предупреждение, что стековый сегмент не найден.Если стековый сегмент создан, а комбинированный тип STACK не используется, пользователь должен явно загрузить в регистр SS адрес сегмента.

Если комбинированный тип не указан, сегмент при загрузке в память получает свой собственный физический сегмент.

3.11.3 COMMON Определение совмещаемых сегментов

segname SEGMENT COMMON .

Заставляет редактор связей (linker) помещать все сегменты, имена которых совпадают, по одному и тому же адресу, так что адреса данных и команд в каждом из таких сегментов являются смещениями по отношению к общему стартовому адресу.

Примечания:

Для совмещенных структур, получающихся вследствие описания комбинированного типа COMMON, справедливо: если данные описываются в нескольких местах под общим именем, то самое последнее описание "забивает" все предыдущие описания.Длина нового совмещенного сегмента равна длине максимального из сегментов комбинированного типа COMMON.Если этот комбинированный тип не был задан, то сегмент не совмещается ни с каким другим сегментом.

3.11.4 MEMORY Размещает сегмент как последний возможный

segname SEGMENT MEMORY.

Заставляет компоновщик (Linker) Microsoft воспринимать этот сегмент как PUBLIC. Компоновщик Microsoft не поддерживает смешанного типа MEMORY в той форме, как отмечено в описании смешанного типа MEMORY фирмы Intel. Вместо этого все сегменты MEMORY воспринимаются как PUBLIC.

Примечания:

В соответствии с положениями фирмы Intel, комбинированный тип памяти предлагался для того, чтобы компоновщик помещал его после всех связывающих сегментов. Первый сегмент MEMORY будет размещен в памяти как наивысший сегмент. Последующие сегменты, содержащие сегмент MEMORY, будут восприниматься как сегменты COMMON.

Если смешанный (комбинированный) тип не указан, сегмент не смешивается с другими сегментами и при загрузке в память ему отводится отдельный физический сегмент.

3.11.5 AT Определение абсолютного сегмента

segname SEGMENT AT address .

Задает абсолютный стартовый адрес сегмента. Все метки и адреса в сегменте являются относительными по отношению к заданному адресу.

Примечания:

Параметр address является 16-тибитным адресом начала параграфа для сегмента. Он может также являться выражением, но не должен содержать ссылок на последующие строки.

Этот комбинированный тип используется обычно для данных или кодов, которые должны располагаться по адресам, не зависящим от распределения памяти; например, для областей данных ПЗУ БСВВ, векторов прерывания или буфера экрана.Комбинированный тип AT нельзя использовать для насильственной загрузки кодов или данных в заданный участок памяти.

3.12. Управление размещением сегментов в области памяти. Типы размещения

3.12.1 BYTE Располагает сегмент по адресу некоторого байта

segname SEGMENT BYTE .

Указывает, что сегмент должен начинаться на границе байта.

Примечания:

Реальный стартовый адрес не вычисляется до тех пор, пока программа не загружена.

3.12.2 WORD Выравнивание на 2-байтовую границу

segname SEGMENT WORD .

Указывает, что сегмент должен начинаться с адреса памяти, кратного 2, т.е. последний значащий бит равен 0. (Такой адрес памяти называется также границей слова).

Примечания:

Действительный начальный адрес не вычисляется до загрузки программы.

3.12.3 PARA Выравнивание на 16-байтовую границу segname SEGMENT PARA .

Указывает, что сегмент должен начинаться с адреса памяти, кратного 16, т.е. последняя шестнадцатеричная цифра должна быть 0h (Такой адрес памяти называется также границей параграфа).

Примечания:

Действительный начальный адрес не вычисляется до загрузки программы.

Если тип расположения не указан, по умолчанию принимается тип расположения PARA.

3.12.4 PAGE Выравнивание на 256-байтовую границу

segname SEGMENT PAGE .

Указывает, что сегмент должен начинаться с адреса памяти, кратного 256, т.е. две последние шестнадцатеричные цифры должны быть 00h (Такой адрес памяти называется также границей страницы).

Примечания:
Действительный начальный адрес не вычисляется до загрузки программы.

3.13 Привязка сегментов к сегментным регистрам
ASSUME сегментный_регистр:имя_сегмента,,,
или
ASSUME NOTHING.
Задает сегментный регистр, который будет использоваться для вычисления исполнительных адресов всех меток и переменных, определенных под заданным именем сегмента или группы ("имя сегмента").
Примечания:
Аргументом "сегментный_регистр" может быть CS, DS, ES или SS.
Аргументом "имя_сегмента" должно быть имя сегмента, определенное заранее директивой SEGMENT, или имя группы, определенное директивой GROUP, или ключевое слово NOTHING. Если используется ключевое слово NOTHING, то предшествующий выбор сегмента анулируется.
ASSUME NOTHING аннулирует выбор сегмента для всех 4-х сегментных регистров.
Если для формирования исполнительного адреса используется оператор задания сегмента в явном виде (:), то сегментный регистр, заданный директивой ASSUME, во внимание не принимается.

3.14 Определение меток и переменных
3.14.1 Спецификация типов данных
3.14.1.1 BYTE Тип данных для 1 байта
Используется для определения типа переменной как байт (8 бит).
Примечания:
Этот тип данных используется:
- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS.
Для описания переменной типа BYTE используется команда DB.

3.14.1.2 WORD Тип данных в 2 байта
Используется для задания переменной типа данных в виде слова (2 байта).
Примечания:
Этот тип данных используется:
- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS.
DW используется для определения переменной типа WORD.

3.14.1.3 DWORD Тип данных для 4 байтов
Используется для определения типа переменной как двойное слово(4 байта).
Примечания:
Этот тип данных используется:
- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS.
Для описания переменной типа DWORD используется команда DD.
3.14.1.4 QWORD Тип данных в 8 байт
Используется для определения типа переменной как четверное слово(8 байт).
Примечания:
Этот тип данных используется:
- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS.
Смотри DQ для определения переменной типа QWORD.

3.14.1.5 TBYTE Тип данных в 10 байтов

Используется для определения типа переменной в 10 байт.

Примечания:

Этот тип данных используется:

- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS.

Смотри DT для определения переменной типа TBYTE.

3.14.2 Спецификация типов меток

3.14.2.1 FAR Тип данных для метки из другого сегмента

Используется для определения типа метки как дальней (из другого сегмента).

Примечания:

Этот тип данных используется:

- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS;
- директива PROC.

3.14.2.2 NEAR Тип данных в том же сегменте

Используется для описания типа данных и метки как near (ближайший, тот же сегмент).

Примечания:

Этот тип данных используется:

- директивой EXTRN;
- оператором PTR;
- директивой LABEL;
- оператором THIS;
- директивой PROC.

Метка, за которой стоит двоеточие, например:

```
Address:    является меткой типа near.
```

3.14.3 $ Операнд счетчика размещения

Этот спецсимвол обозначает текущее значение счетчика размещения. Под счетчиком размещения понимается значение текущего смещения внутри текущего сегмента в процессе ассемблирования.

Примечания:

Этот операнд обладает такими же атрибутами, как и метка near label.

Счетчик размещения - это адрес, увеличивающийся в процессе ассемблирования; он отражает текущий адрес оператора исходного файла, который ассемблируется.

Пример.

```
helpMessage  DB  'This is help for the program'
helpLength   =  $ - helpMessage .
```

После ассемблирования этих двух строк значение 'helpLength' будет равно длине вспомогательного сообщения, помещенного в строке helpMessage.

3.14.4 Массивы и буферы. Оператор DUP

DUP - дублирование начального значения.

count DUP (initialvalue,,,).

Используется вместе с директивами DB, DW, DD, DQ и DT для задания нескольких одинаковых начальных значений.

Примечания:

Начальных значений initialvalue может быть одно или более, тогда их нужно отделить запятыми. Каждое начальное значение должно быть выражением, которое преобразуется в целое, в символьную константу или в другой оператор DUP. Обратите Ваше внимание на то, что начальные значения ДОЛЖНЫ быть заключены в скобки.

Аргумент count задает число раз, которое нужно продублировать начальное значение.

Пример.

```
DB 200 DUP (1) ;Определяет 200 байтов,инициализированных в
;1
DB 50 DUP (1,0)  ;Определяет 100 байтов со значениями
; 1,0,1,0 и т.д.
DB 2 DUP (3 DUP (1)) ;Определяет 6 байтов,
;инициализированных в 1 .
```

3.15 Специальные операторы для работы с записями

3.15.1 MASK Получение битовой маски

MASK recordFieldName

или

MASK record .

Выдает последовательность битов, представляющих биты, используемые в отдельных полях записи. Биты, используемые в поле, должны быть единичными, тогда неиспользуемые - нулевыми.

Примечания:

recordFieldName должно быть именем поля записи. Record должно быть именем записи.

3.15.2 WIDTH Получение ширины в битах

WIDTH recordFieldName

или

WIDTH record .

Выдает число бит, занимаемых полем записи или всей записью.

Примечания:

recordFieldName должно быть именем поля записи.

Record должно быть именем записи.

4. ДИРЕКТИВЫ АССЕМБЛЕРА

При трансляции исходного модуля программы (расширение .ASM) в объектный файл (расширение .OBJ) с помощью транслятора MASM используются приведенные ниже директивы, определяющие режимы трансляции.

4.1 .186 Разрешает команды процессора 80186

Разрешает ассемблирование команд процессоров 80186 и 8086.

Примечания:

Эту директиву следует использовать для программ, которые будут выполняться только на системах 80186.

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

4.2 .286c Разрешает команды реального режима процессора 80286

Разрешает ассемблирование команд реального режима процессора 80286, а также команд процессоров 80186 и 8086 (команды 80186 идентичны командам реального режима 80286).

Примечания:

Эту директиву следует использовать для программ, которые будут выполняться только в реальном режиме на 80286 (или 80386).

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

Смотри также директиву .286p, которая разрешает команды защищенного режима процессора 80286.

4.3 .286p Разрешает команды защищенного режима процессора 80286

Разрешает ассемблирование команд защищенного режима 80286, а также команд незащищенного (реального) режима 8086, 80186, 80286.

Примечания:

Эту директиву следует использовать для программ, которые будут выполняться только в защищенном режиме на 80286.

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

Смотри также директиву .286c, которая разрешает лишь команды реального режима процессора 80286.

4.4 .287 Разрешает команды процессора 80287

Разрешает ассемблирование команд процессора 80287.

Примечания:

Смотри описание по .8087, где содержится дополнительная информация по использованию этой директивы.Эту директиву следует использовать в программах, которые используют операции с плавающей точкой и будут выполняться только на системах 80287.

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

4.5 .8086 Разрешает команды процессора 8086

Разрешает ассемблирование команд процессора 8086(8088), при этом запрещает ассемблирование команд, которые имеются только на 80186 и 80286.

Примечания:

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

Если не задана ни одна из директив разрешающих использование команд того или иного процессора, то по умолчанию берутся директивы .8086 и .8087.

4.6 .8087 Разрешает команды процессора 8087

Разрешает ассемблирование команд 8086 и запрещает ассемблирование команд 80287.

Примечания:

При использовании директив .8087 или .287 добавляйте в командную строку макроассемблера /R или /E для задания режима обработки ассемблером команд с плавающей точкой:

/R - ассемблер генерирует действительный код команды для команд с плавающей точкой;

/E - ассемблер вырабатывает код, который может быть использован программой-эмулятором арифметики с плавающей точкой.

Помещайте все директивы, разрешающие ассемблирование тех или иных команд, в начало исходного файла, чтобы быть уверенными в том, что все команды файла проассемблировались с использованием набора команд одного и того же микропроцессора.

Если не задана ни одна из директив разрешающих использование команд того или иного процессора, то по умолчанию берутся директивы .8086 и .8087.

4.7 = Создание абсолютного символа

name=expression .

Присваивает значение выражения expression абсолютному символу name.

Примечания:

Размещения абсолютного символа в памяти не происходит. Ассемблер заменяет каждый вход абсолютного символа name на 16-тибитное числовое значение выражения.

Выражение expression может быть целым, константным выражением, адресным выражением, строковой константой, состоящей из 1 или 2 символов. Значение выражения не должно превышать 65535. Имя name должно использоваться впервые или уже быть ранее введено командой '='. Абсолютный символ МОЖЕТ быть переопределен в любое время.

4.8 COMMENT Ввод комментария в несколько строк

COMMENT разделитель

text

разделитель [text] .

Предоставляет возможность включать комментарии длиной в несколько строк в исходный текст, заключая комментарий между 2-мя разделителями (оба разделителя должны совпадать).

Примечания:

Под разделителем понимается любой символ, отличный от пробела, следующий за словом COMMENT. Весь текст, следующий за разделителем до следующего появления того же разделителя, игнорируется.

Кроме того, любой текст, следующий за вторым разделителем в той же строке, также игнорируется.

4.9 .CREF Разрешает листинг перекрестных ссылок

.CREF .

Разрешает генерацию листингов перекрестных ссылок для меток, переменных и символов.

Примечания:

Эта директива не дает указания ассемблеру составить листинг перекрестных ссылок, но, если листинг перекрестных ссылок задан, то эта директива делает возможным генерацию меток, переменных и символов с перекрестными ссылками; в то время, как после директивы .XCREF все ссылки (метки, переменные и символы) не будут перекрестными ссылками, пока снова не встретится .CREF.

4.10 DB Описание байта

[name] DB initialvalue,,, .

Размещает и инициализирует один или более байтов (по 8 бит) памяти. Аргумент initialvalue может быть задан одним из следующих способов:

- целое число (например, 12);
- строка (например, 'message');
- константное выражение (например, 2 * 5);
- оператор DUP (например, 10 DUP (?));
- знак вопроса (?) (например, 0,1,?,2).

Примечания:

Когда имеется аргумент name (он не является обязательным), то ассемблер создает переменную, значение смещения которой равно текущему значению счетчика размещения. Эта переменная будет иметь тип BYTE.

Знак вопроса ? дает указание ассемблеру оставить начальное значение неопределенным. Для того, чтобы задать более одного начального значения, отделите их запятыми. Начальное значение строковой константы может быть любой длины, при условии, что она умещается на одной строке. Строчные переменные хранятся таким образом, чтобы первому символу соответствовал меньший адрес, чем последнему.

4.11 DD Описание двойного слова

[name] DD initialvalue,,, .

Размещает и инициализирует одно или более двойных слов (по 4 байта) памяти.

Примечания:

Когда имеется аргумент name (он не является обязательным), то ассемблер создает переменную, значение смещения которой равно текущему значению счетчика размещения. Эта переменная будет иметь тип DWORD.

Аргумент initialvalue может быть задан одним из следующих способов:

- целое число, например, 1234;
- вещественное число, например, 2.3;
- строковая константа, одно- или двухсимвольная, например, 'gh';
- закодированное вещественное число, например, 2F00000r;
- константное выражение ,например, 2 * 12;
- адресное выражение, например, farArrayPtr;
- оператор DUP, например, 10 DUP (?) .

Знак вопроса ? дает указание ассемблеру оставить начальное значение неопределенным. Для того, чтобы задать более одного начального значения, отделите их запятыми.

Строчные переменные типа DWORD должны содержать не более 2-х символов. Последний символ помещается в младший байт младшего слова, а первый символ (если их 2) или 0 (если символ всего один) помещается в следующий байт. Остальные байты заполняются нулями.

4.12 DQ Описание учетверенного слова

[name] DQ initialvalue,,, .

Размещает и инициализирует одно или более учетверенных слов (по 8 байтов) памяти.

Примечания:

Когда имеется аргумент name (он не является обязательным), то ассемблер создает переменную, значение смещения которой равно текущему значению счетчика размещения. Эта переменная будет иметь тип QWORD.

Аргумент initialvalue может быть задан одним из следующих способов:

- целое число, например, 1234;
- вещественное число, например, 2.3;
- строковая константа, одно- или двухсимвольная, например, 'gh';
- закодированное вещественное число, например, 2F0000000000000r;
- константное выражение, например, 2 * 12;
- оператор DUP, например, 10 DUP (?);
- знак вопроса (?), например, 0,1,?,2.

Знак вопроса ? дает указание ассемблеру оставить начальное значение неопределенным. Для того, чтобы задать более одного начального значения, отделите их запятыми.

Строчные переменные типа QWORD должны содержать не более 2-х символов. Последний символ помещается в младший байт младшего слова, а первый символ (если их 2) или 0 (если символ всего один) помещается в следующий байт. Остальные байты заполняются нулями.

4.13 DT Описание 10-байтной единицы

[name] DT initialvalue,,, .

Размещает и инициализирует одну или более десятибайтных единиц памяти.

Примечания:

Когда имеется аргумент name (он не является обязательным), то ассемблер создает переменную, значение смещения которой равно текущему значению счетчика размещения. Эта переменная будет иметь тип TBYTE.

Аргумент initialvalue может быть задан одним из следующих способов:

- целочисленное выражение (например, 12334d);
- упакованное десятичное число (например, 0123456789);
- строковая константа, одно- или двухсимвольная, например, 'gh';
- закодированное вещественное число, например, 2F0000000000000r);
- оператор DUP, например, 10 DUP (?);
- знак вопроса (?), например, 0,1,?,2 .

Знак вопроса ? дает указание ассемблеру оставить начальное значение неопределенным. Для того, чтобы задать более одного начального значения, отделите их запятыми.

Строчные переменные типа TBYTE должны содержать не более 2 -х символов. Последний символ помещается в младший байт младшего слова, а первый символ (если их 2) или 0 (если символ всего один) помещается в следующий байт. Остальные байты заполняются нулями.

Константы с десятичными цифрами трактуются как упакованные десятичные числа, а не как целые. Чтобы с помощью DT задать целую константу, добавьте в конце числа букву, задающую систему счисления, в которой это число задается. Например, добавьте 'D' в конце десятичного числа, 'H' - в конце числа, записанного в шестнадцатеричной системе счисления.

4.14 DW Описание слова

```
[name] DW initialvalue,,, .
```

Размещает и инициализирует одно или более слов (по 2 бай- та) памяти.

Аргумент initialvalue может быть задан одним из следующих способов:

- целое число, например, 1234;
- строковая константа, одно- или двухсимвольная, например, 'gh';
- константное выражение, например, 2 * 12;
- адресное выражение, например, arrayAddress;
- оператор DUP, например, 10 DUP (?);
- знак вопроса (?), например, 0,1,?,2.

Примечания:

Когда имеется аргумент name (он не является обязательным), то ассемблер создает переменную, значение смещения которой равно текущему значению счетчика размещения. Эта переменная будет иметь тип WORD.

Знак вопроса ? дает указание ассемблеру оставить начальное значение неопределенным. Для того, чтобы задать более одного начального значения, отделите их запятыми.

Строчные переменные типа WORD должны содержать не более 2-х символов. Последний символ помещается в младший байт слова, а первый символ (если их 2) или 0 (если символ всего один) помещается в старший байт.

4.15 ELSE Ассемблирование, если условие не выполнено

```
IF условие
команды
[ELSE
команды]
ENDIF .
```

Дает указание ассемблеру сгенерировать коды одной или более команд, если условие IF не выполнено. Команды, заключенные между директивами IF и ELSE выполняются, если условие IF удовлетворено. Если же условие IF не удовлетворено, то ассемблируются команды, заключенные между директивами ELSE и ENDIF.

Примечания:

Вложенная директива ELSE соответствует ближайшей из директив IF, у которой нет директивы ELSE.

4.16 END Конец модуля

```
END [expression] .
```

Помечает конец модуля. Все команды, следующие за этой директивой, игнорируются.

Примечания:

Аргумент expression, если присутствует, задает точку вхождения программы, т.е. адрес, с которого начинается выполнение программы. Программа может содержать более одного модуля,но только в одном модуле разрешается задавать точку вхождения. Модуль, в котором задана точка вхождения, называется главным.

4.17 ENDIF Конец условного блока

```
IF... условие
команды
[ELSE
команды]
```

ENDIF .
Завершает условный блок, начинающийся одной из директив IF, IF1, IF2, IFB, IFDEF, IFDIF, IFE, IFIDN, IFNB или IFNDEF.

4.18 ENDIF Конец условного блока
IF... условие
команды
[ELSE
команды]
ENDIF .
Завершает условный блок, начинающийся одной из директив IF, IF1, IF2, IFB, IFDEF, IFDIF, IFE, IFIDN, IFNB или IFNDEF.

4.19 ENDM Конец макроопределения или повторного блока
1. имя MAKRO [формальный_параметр,,,] команды ENDM
2. REPT выражение команды ENDM
3. IRP формальное_имя,<параметр,,,> команды ENDM
4. IRPC формальное_имя,строка команды ENDM .

Завершает макроопределение (MACRO) и повторные блоки (REPT, IRP, IRPC).

4.20 ENDP Конец описания процедуры
имя PROC [расстояние]
команды
имя ENDP .
Отмечает конец процедуры.
Примечания:
Перед директивой ENDP должно стоять имя процедуры.Процедура должна содержать хотя бы одну директиву RET, поскольку в конце процедуры директива RET не вырабатывается автоматически, как в других языках программирования.Аргументом "расстояние" может быть NEAR или FAR.

4.21 ENDS Конец описания сегмента или структуры
1. имя SEGMENT [типрасположения] [типкомбинирования] ['класс'] имя ENDS
2. имя STRUC описания_полей имя ENDS .
Отмечает конец описания сегмента или структуры.

4.22 EQU Создание символа
name EQU expression .
Создает абсолютные символы (имена, которые представляют 16-битные значения), "прозвища" (имена, которые представляют другие символы) или текстовые символы (имена, которые представляют строки) путем присвоения имени name значения выражения expression.
Параметром expression может быть:
- целое число (234);
- вещественное число (3.23);
- строковая константа ('abc');
- закодированное вещественное число (2F0000000000000r);
- мнемокод команды (mov ax, 1);
- константное выражение (23*4);
- адресное выражение (arrayPtr).

Примечания:
Параметр name не должен быть ранее определен и не может быть переопределен.
Если значение выражения есть число между 0 и 65535, то ассемблер заменяет имя name на это число. Во всех остальных случаях ассемблер заменяет имя name на текст.

4.23 .ERR Симуляция ошибки
.ERR .
Симулирует фатальную ошибку во время первого или второго прохода ассемблера.

Примечания:
Эта директива может быть использована в блоках условного ассемблирования или в макросах в целях отладки.Сообщение об ошибке, генерируемое этой директивой, имеет номер 89 и гласит 'Forced error' (Принудительная ошибка).Когда встречается эта директива, то объектный модуль удаляется, и происходит выход из ассемблера с кодом выхода 7.

4.24 .ERR1 Симуляция ошибки при первом проходе

.ERR1 .

Симулирует предупреждающую ошибку во время первого прохода ассемблером программы, при условии, что был использован переключатель /D в командной строке (для выдачи листинга первого прохода).

Примечания:

Эта директива может быть использована в блоках условного ассемблирования или в макросах в целях отладки. Сообщение об ошибке, генерируемое этой директивой, имеет номер 87 и гласит 'Forced error - pass1' (Принудительная ошибка на первом проходе).Эта директива не окажет никакого воздействия на программу, если в командной строке не будет использован переключатель /D.Эта директива сгенерирует только предупреждающую ошибку, а не фатальную.

4.25 .ERR2 Симуляция ошибки при втором проходе

.ERR2 .

Если эта директива встречается во время 2-го прохода ассемблером программы, то она генерирует фатальную ошибку.

Примечания:

Эта директива может быть использована в блоках условного ассемблирования или в макросах в целях отладки.Сообщение об ошибке, генерируемое этой директивой, имеет номер 88 и гласит 'Forced error - pass2' (Принудительная ошибка на втором проходе).

Когда встречается эта директива, то объектный модуль удаляется, и происходит выход из ассемблера с кодом выхода 7.

4.26 .ERRB Ошибка, если строка пустая

.ERRB <строка> .

Симулирует фатальную ошибку, если строка пуста.

Примечания:

Обратите Ваше внимание на то, что строка должна быть заключена в угловые скобки. Строкой может быть любое имя, число или выражение.

Сообщение об ошибке, генерируемое этой директивой, имеет номер 94 и гласит 'Forced error - string blank' (Принудительная ошибка - строка пуста).Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.Директивы .ERRB и .ERRNB используются обычно для выявления наличия или отсутствия параметров в макро.

4.27 .ERRDEF Ошибка, если имя определено

.ERRDEF name .

Симулирует фатальную ошибку, если имя name определено.

Примечания:

Если имя name определено как ссылка вперед программы, то оно считается неопределенным на первом проходе и определенным на втором.Сообщение об ошибке, генерируемое этой директивой, имеет номер 93 и гласит 'Forced error - symbol defined' (Принудитель- ная ошибка - символ определен).Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.

4.28 .ERRDIF Ошибка, если строки различаются

.ERRDIF <string1>,<string2> .

Симулирует фатальную ошибку, если две строки (которые могут быть именами, числами или выражениями) не совпадают посимвольно (т.е. они различны).

Примечания:

Строчное и прописное написание одной и той же буквы воспринимаются как различные символы. Обе строки должны быть заключены в угловые скобки и отделяться друг от друга запятой.Сообщение об ошибке, генерируемое этой директивой, имеет номер 97 и гласит 'Forced error - strings different' (Принудительная ошибка - строки различаются).Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.Эта директива часто используется для тестирования параметров, переданных макроопределению.

4.29 .ERRE Ошибка, если ложь

.ERRE expression .

Симулирует фатальную ошибку, если значение выражения expression есть ложь (0).

Примечания:

Выражение expression должно приводиться к абсолютному значению и не содержать ссылок вперед программы.Сообщение об ошибке, генерируемое этой директивой, имеет номер 90 и гласит 'Forced error - expression equals 0' (Принудительная ошибка - выражение равно 0).

Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ссемблера с кодом 7.

4.30 .ERRIDN Ошибка, если строки идентичны
.ERRIDN <string1>,<string2> .
Симулирует фатальную ошибку, если две строки (которые могут быть именами, числами или выражениями) Совпадают посимвольно (т.е. они идентичны).
Примечания:
Строчное и прописное написание одной и той же буквы воспринимаются как различные символы.Сообщение об ошибке, генерируемое этой директивой, имеет номер 96 и гласит 'Forced error - strings identical' (Принудительная ошибка - строки совпадают).Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.Эта директива часто используется для тестирования параметров, переданных макроопределению.

4.31 .ERRNB Ошибка, если строка не пустая .ERRNB <строка> .
Симулирует фатальную ошибку, если строка не пуста.
Примечания:
Обратите Ваше внимание на то, что строка должна быть заключена в угловые скобки. Строкой может быть любое имя, число или выражение.
Сообщение об ошибке, генерируемое этой директивой, имеет номер 95 и гласит 'Forced error - string not blank' (Принудительная ошибка - строка не пуста).
Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ссемблера с кодом 7.Директивы .ERRB и .ERRNB используются обычно для выявления наличия или отсутствия параметров в макро.

4.32 .ERRNDEF Ошибка, если имя не определено
.ERRNDEF name .
Симулирует фатальную ошибку, если имя name не определено.
Примечания:
Если имя name определено как ссылка вперед, то оно считается неопределенным на первом проходе и определенным на втором.Сообщение об ошибке, генерируемое этой директивой, имеет номер 92 и гласит 'Forced error - symbol not defined' (Принудительная ошибка - символ не определен).Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.

4.33 .ERRNZ Ошибка, если истина.ERRNZ expression .
Симулирует фатальную ошибку, если значение выражения expression есть истина (не ноль).
Примечания:
Выражение expression должно приводиться к абсолютному значению и не содержать ссылок вперед.Сообщение об ошибке, генерируемое этой директивой, имеет номер 91 и гласит 'Forced error - expression not equal 0' (Принудительная ошибка - выражение не равно 0).
Если ошибка генерируется, то объектный модуль удаляется и происходит выход из ассемблера с кодом 7.

4.34 EVEN Располагает на границе слова EVEN .
Заставляет ассемблер разместить следующий байт (данные или команду) на границе слова.
Примечания:
Если следующий байт располагался бы по нечетному адресу, не будь этой директивы, то ассемблер генерирует команду NOP в ответ на директиву EVEN. Директива EVEN не может использоваться в сегментах, расположенных на границе байта.

4.35 EXITM Немедленный выход из макро EXITM .
Влечет немедленный выход из макро (MACRO) или повторного блока (REPT) и возвращает управление оператору, следующему за оператором вызова макро или повторного блока.
Примечания:
Если команда EXITM находится во вложенном макро или повторном блоке, то управление возвращается в блок внешнего уровня.
Директива EXITM используется большей частью вместе с условной директивой IF в целях условного включения или исключения операторов в (из) макро или повторный блок.

4.36 EXTRN Описание внешнего имени
EXTRN name:type,,, .
Описывает переменную, метку или символ как внешние. Внешней считается единица, которая описана в другом программном модуле, но используется в программном модуле, содержащем директиву EXTRN. В модуле, где единица вводится, она должна быть описана директивой PUBLIC.

Примечания:

Тип (type) имени name должен совпадать в исходном модуле, где единица описана, и в модуле, где она используется как внешняя. Допустимыми типами являются: BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR и ABS.

Если директива EXTRN расположена внутри какого-либо сегмента, то предполагается, что внешняя единица находится в том же сегменте. Директива PUBLIC, описывающая эту единицу в другом модуле, должна также находиться в этом сегменте. Если директива EXTRN находится вне всех сегментов, то соответствующая директива PUBLIC может быть в любом месте другого модуля.Все символы имени name преобразуются в заглавные при образовании объектного файла. Для сохранения строчных символов пользуйтесь переключателями макроассемблера /ML и /MX. Для символов, представляющих абсолютные числа, используется тип ABS.

4.37 GROUP Описание группы сегментов

name GROUP segmentname,,, .

Заставляет ассемблер ассоциировать имя name с одним или более сегментами так, что для всех меток и переменных, описанных в этих сегментах, смещение вычисляется по отношению к началу имени name.

Примечания:

Аргумент name должен быть единичным, а аргумент segmentname должен быть сегментом, описанным заранее директивой SEGMENT или выражением SEG. Группа сегментов не обязана быть слитной. Между сегментами группы могут располагаться сегменты, ей не принадлежащие.Единственное ограничение заключается в том, что первый байт первого сегмента группы должен быть удален от последнего байта последнего сегмента группы не более, чем на 65535 байтов. Это означает, что если сегменты группы располагаются слитно, то объем группы не может превышать 65536 байтов (64К).Аргумент name не должен встречаться больше ни в одной другой группе никакого исходного файла.Имя группы name может использоваться в директиве ASSUME и для переопределения сегмента.

4.38 IF Начало условного блока

IF выражение
команды
[ELSE
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) ELSE, если значение выражения есть истина (т.е. не равно 0).

Примечания:

Допускается вложение IF до 255 уровней.

Команды, следующие за IF и ELSE могут также являться условными блоками.Если значение выражения есть ложь (т.е. ноль) и директива ELSE включена в блок, то ассемблируются команды, заключенные между ELSE и ENDIF.Вложенная директива ELSE соответствует ближайшей из директив IF, у которой нет директивы ELSE.Выражение, стоящее в директиве IF, должно иметь абсолютное значение, и не содержать ссылок вперед программы.

4.39 IF1 Ассемблирование, если первый проход IF1

команды
[ELSE
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) ELSE, если ассемблер совершает первый проход.

Примечания:

Директива IF1 проверяет только номер прохода (первый или второй), поэтому она не сопровождается выражением.

4.40 IF2 Ассемблирование, если второй проход IF2

команды
[ELSE
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) ELSE, если ассемблер совершает второй проход.

Примечания:

Директива **IF2** проверяет только номер прохода (первый или второй), поэтому она не сопровождается выражением.

4.41 IFB Ассемблирование, если аргумент пустой

IFB <аргумент>
команды
[**ELSE**
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если аргумент отсутствует.

Примечания:

Директивы **IFB** и **IFNB** используются обычно для условного ассемблирования макросов, в зависимости от наличия или отсутствия параметров в макровызове. Обратите Ваше внимание на необходимость заключения аргумента в угловые скобки **<>**. Аргументом может быть любое имя, число или выражение.

4.42 IFDEF Ассемблирование, если имя определено

IFDEF имя
команды
[**ELSE**
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если имя является меткой, переменной или символом, которые определены.

Примечания:

Аргументом имя может являться любое разрешенное имя.Если имя определено как ссылка вперед, то на первом проходе **IFDEF** будет ложно, а на втором истинно.

4.43 IFDIF Ассемблирование, если аргументы различны

IFDIF <аргумент1>,<аргумент2>
команды
[**ELSE**
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если два аргумента (которые могут быть именами, числами или выражениями) не совпадают посимвольно (т.е. они различны).

Примечания:

Строчное и прописное написание одной и той же буквы воспринимаются как различные символы.Оба аргумента должны быть заключены в угловые скобки и отделяться друг от друга запятой.Эта директива часто используется для тестирования параметров, переданных макроопределению.

4.44 IFE Ассемблирование, если ложь

IFE выражение
команды
[**ELSE**
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если значение выражения есть ложь (т.е. равно **0**).

Примечания:

Допускается вложение **IFE** до **255** уровней.Выражение, стоящее в директиве **IFE**, должно иметь абсолютное значение, и не содержать ссылок вперед программы.

4.45 IFIDN Ассемблирование, если аргументы совпадают

IFIDN <аргумент1>,<аргумент2>
команды
[**ELSE**
команды]
ENDIF .

Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если два аргумента (которые могут быть именами, числами или выражениями) совпадают посимвольно (т.е. являются идентичными).

Примечания:
Строчное и прописное написание одной и той же буквы воспринимаются как различные символы.Оба аргумента должны быть заключены в угловые скобки и отделяться друг от друга запятой.Эта директива часто используется для тестирования параметров, переданных макроопределению.

4.46 **IFNB** Ассемблирование, если аргумент не пуст
IFNB <аргумент>
команды
[**ELSE**
команды]
ENDIF .
Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если аргумент присутствует.
Примечания:
Директивы **IFB** и **IFNB** используются обычно для условного ассемблирования макросов, в зависимости от наличия или отсутствия параметров в макровызове.Обратите Ваше внимание на необходимость заключения аргумента в угловые скобки **<>**. Аргументом может быть любое имя, число или выражение.

4.47 **IFNDEF** Ассемблирование, если имя не определено
IFNDEF имя
команды
[**ELSE**
команды]
ENDIF .
Открывает условный блок, заставляя ассемблер обрабатывать команды, предшествующие (необязательному) **ELSE**, если имя является меткой, переменной или символом, которые еще не определены.
Примечания:
Аргументом имя может являться любое разрешенное имя.Если имя определено как ссылка вперед, то на первом проходе **IFNDEF** будет истинно, а на втором ложно.

4.48 **INCLUDE** Включение кодов из внешнего файла
INCLUDE имя_файла .
Заставляет ассемблер загрузить исходный код из внешнего файла и обработать его.
Примечания:
Имя файла должно отсылать к существующему файлу, иначе ассемблер выдаст сообщение об ошибке и остановится. Имя файла может содержать полное или частичное описание маршрута с разделителями **/** или ****. Если Вы использовали в командной строке переключатель макроассемблера **/I** для задания "маршрута включаемых файлов", то в директиве **INCLUDE** можно этот маршрут не задавать. Макроассемблер проверяет сначала каждый из маршрутов, заданных в опции **/I**. Затем, если файл не найден, макроассемблер ищет в текущем каталоге. Если файл все еще не найден, то макроассемблер выдает сообщение об ошибке и останавливается.Когда ассемблер встречает директиву **INCLUDE**, он считывает и выполняет целиком весь файл; только после этого возвращается в файл, в котором содержалась команда **INCLUDE**.Коды файлов, включенных директивой **INCLUDE**, помечаются в листинге макроассемблера буквой **'C'**.

4.49 **IRP** Ассемблирование по **1** разу для каждого параметра
IRP формальное_имя,<параметр,,,>
операторы
ENDM .
Заставляет ассемблер выполнить операторы по **1** разу для каждого значения параметра, заменяя на каждой итерации все вхождения формального имени на текущее значение параметра.
Примечания:
Может быть задано неограниченное число параметров и формальное имя может встречаться в теле блока любое число раз. Если список параметров пуст, то операторы не ассемблируются. Если, однако, задан нулевой параметр (**<>**), то формальное имя заменяется на ноль.Список параметров должен быть заключен в угловые скобки и,
если задается более одного параметра, то их следует разделять запятыми.Параметром может быть любой разрешенный символ, строка,числовая или символьная константа.

4.50 IRPC Ассемблирование по 1 разудля каждого символа

IRP формальное_имя,строка
операторы
ENDM .

Заставляет ассемблер выполнить операторы по 1 разу для каждого сивола строки, заменяя на каждой итерации все вхождения формального имени на текущий символ.

Примечания:

Формальное имя может встречаться в теле блока неограниченное число раз.

Строка может состоять из букв, цифр и других символов. Если строка содержит пробелы, запятые или другие символы-разделители, то заключите ее в угловые скобки.

4.51 LABEL Создание переменной или метки

имя LABEL тип .

Создает новую переменную или метку, присваивая ей текущее значение счетчика размещения.

Примечания:

Имя не должно быть ранее определено.Типом может быть BYTE, WORD, DWORD, QWORD, TBYTE, NEAR,FAR или имя описанного структурного типа.

Пример.

```
BYTE_BUFFER  LABEL   BYTE
WORD_BUFFER  DW 512 dup (?) .
```

К буферу длиной 1024 байта (512 слов) можно обращаться как к буферу, состоящему из 512 слов, используя адрес WORD_BUFFER, или как к буферу, состоящему из 1024 байтов, используя адрес BYTE_BUFFER.

4.52 .LALL Распечатка всех макрорасширений

.LALL .

Заставляет ассемблер распечатать листинг исходных операторов всех макрорасширений, включая комментарии, которым предшествует единичные точка с запятой. Комментарии, которым предшествуют удвоенные точка с запятой (;;) удаляются из макрорасширений.

4.53 .LFCOND Выдача блоков с отрицательными условиями

.LFCOND

Заставляет макроассемблер выдавать на печать блоки с отрицательными условиями (т.е. условные операторы, для которых значение IFвыражения есть ложь).

Примечания:

Это режим работы ассемблера, который принимается по умолчанию.

4.54 .LIST Разрешение выдачи исходных кодов

.LIST .

Заставляет макроассемблер выдавать строки исходных кодов.

Примечания:

Директивы .LIST и .XLIST обычно используются для подавления распечатки отдельного блока программы.

4.55 LOCAL Объявление символа для использования в Макросе

LOCAL dummyname,,, .

Создает уникальные символьные имена, используемые в макросах.

Примечания:

Когда макрос раскрывается, dummyname (формальное имя) заменяется символом следующим образом:

??number ,где number - шестнадцатеричное число в пределах от 0 до FFFFh.

Не описывайте другие символы данным образом, т.к.макрокоманда будет генерировать метки данного типа, а повторение меток может вызвать ошибку. LOCAL может использоваться только в макросах, и при описании макроса команда LOCAL должна предшествовать другим командам (включая комментарии).Данная директива обычно используется в макросах, использующих метки. Если макрокоманда вызывается несколько раз и при всех вызовах используется одна и та же метка, то произойдет ошибка, т.к. метка будет
объявлена несколько раз. Во избежание этого используйте директиву LOCAL для того, чтобы сделать все метки типа LOCAL.

4.56 MACRO Начало описания макрокоманды

имя MACRO [dummyparameter,,,]
команды
ENDM .

Начинает описание макрокоманды, состоящей из имени и внутренних команд.
Примечания:
Имя должно быть правильнымм и уникальным. Может быть объявлено любое число формальных параметров(dummyparameter), но они должны все стоять в одной строке. Данные формальные параметры играют роль меток-заполнителей; при раскрытии макроса они заменяются на действительные параметры, передаваемые при вызове макрокоманды.
Макрокоманды могут быть вложены на любую глубину, могут вызывать другие макросы, и могут вызывать сами себя. Они также могут быть переопределены. Макросы генерируют код при их вызове, а не при их описании. Все генерируемые адреса относятся к месту вызова макроса, а не к месту, где он определяется.
Будьте осторожны при использовании слова MACRO в директивах TITLE, SUBTTL и NAME. Т.к. MACRO заменяет эти директивы, то использование MACRO в них может привести к тому, что пользователь будет иметь макросы с именами TITLE, SUBTTL и NAME.

4.57 NAME Задание имени модуля

NAME modulename .
Присваивает имя данному модулю.
Примечания:
Имя модуля (modulename) может иметь любую длину, но значение имеют только первые шесть символов. Может использоваться любая комбинация букв и цифр.
LINK использует имя модуля при сообщениях об ошибках. Если директива NAME не используется, ассемблер берет имя, состоящее из первых шести символов заголовка TITLE. Если нет директивы TITLE, то ассемблер берет по умолчанию имя A.

4.58 ORG Задание счетчика размещения в памяти

ORG выражение
Устанавливает ассемблерный счетчик размещения в соответствии с выражением.
Примечания:
Выражение должно быть преобразовано к абсолютному числу при первом проходе, следовательно, нельзя использовать ссылки вперед. Может быть использован символ счетчика размещения $.

4.59 %OUT Выдача текста при ассемблировании

%OUT текст .
Заставляет ассемблер при ассемблировании выводить текст на экран. Текст выводится, когда в исходной программе встречается директива %OUT, таким образом можно выводить информацию при длительном ассемблировании.
Примечания:
Директива %OUT будет восприниматься дважды, по разу на каждом проходе. Для вывода сообщений, соответствующих каждому выполняемому проходу, используйте IF1 и IF2.

4.60 PAGE Постраничное управление листингом

PAGE длина, ширина
PAGE +
PAGE .
Устанавливает длину и ширину страницы листинга; либо увеличивает номер секции; либо производит перевод страницы.
Примечания:
Схема нумерации страниц, используемая листингами программы, следующая:
section-page,где section - порядковый номер программной секции внутри модуля;
page - номер страницы в секции.Номера секции и страницы начинаются с 1-1.
Директива PAGE без аргументов посылает на принтер перевод страницы и генерирует строку заголовка и подзаголовка.
Директива "PAGE +" увеличивает номер секции, а номер страницы устанавливает равным 1.
Директива "PAGE длина, ширина" определяет параметры страницы для листинга. Опции длины и ширины следующие:

	Минимум	Максимум	По умолчанию
Длина	10	255	50
Ширина	60	132	80

Если ширина указана, а длина нет, то перед шириной нужно ставить запятую.

4.61 PROC Начало описания процедуры

имя PROC [расстояние]
команды
имя ENDP .

Отмечает начало процедуры.

Примечания:

Имя должно быть уникальным.

Расстояние может быть либо типа NEAR (внутри того же сегмента), либо FAR (за пределами данного сегмента). Если расстояние не указано, предполагается тип NEAR.

Допускается вложенность процедур.Процедура должна содержать хотя бы одну команду RET, поскольку в конце процедуры команда RET не вырабатывается автоматически, как в других языках программирования.Имя имеет те же возможности, что и метка, разрешается вызов, переход или цикл, использующие имя в качестве назначения.

4.62 PUBLIC Объявление символа доступным для всех модулей

PUBLIC имя,,, .

Делает метки, переменные или абсолютные символы доступными во всех модулях программы.

Примечания:

Имя должно быть определено внутри данного исходного файла.Если указаны абсолютные символы, они могут представлять только одно или двухбайтовые целые, либо строковые значения.При генерации объектного файла все символы в имени будут представлены прописными буквами. Для сохранения строчных букв могут использоваться переключатели MASM: /ML и /MX.Символы, используемые символьными отладчиками, такими как Symdeb и CodeView, должны быть объявлены PUBLIC. Для каждого символа, объявленого как EXTERN, в одном из исходных файлов программы должна стоять директива PUBLIC.

4.63 PURGE Удаление описания Макроса

PURGE macroname,,, .

Удаляет одно и более макроопределений, освобождая память.

Примечания:

Если происходит вызов макроса, который был удален командой PURGE, возникает ошибка.

Нет необходимости удалять макрос, который Вы хотите переопределить. При переопределении старое макроопределение удаляется автоматически. Макрос может удалить сам себя, если последней строкой будет директива PURGE.

Данная директива может также использоваться для удаления ненужных макросов из библиотеки.Если macroname (имя макроса) является обозначением команды, то оно восстанавливается к своему первоначальному значению.

4.64 .RADIX Установка системы счисления для ввода

.RADIX выражение .

Устанавливает систему счисления для ввода чисел,входящих в выражения.

Примечания:

Аргумент выражение рассматривается как десятичное число, независимо от текущей системы счисления. Оно может быть любым целым числом от 2 до 16.

Стандартная система счисления - десятичная.

Заметьте, что числа, вводимые в виде аргументов команд DD, DQ и DT, всегда считаются десятичными, независимо от текущей системы счисления, если они не сопровождаются указателем системы счисления.

Заметьте также, что буквы B и D, добавленные к числу, всегда считаются указателями системы счисления, даже если текущая система счисления - шестнадцатеричная. Для записи шестнадцатеричного числа, оканчивающегося буквой B и D при шестнадцатеричной системе счисления, Вы должны добавить указатель системы счисления H.

4.65 RECORD Описание типа записи

recordName RECORD fieldName:ширина [=выражение],,, .

Определяет 8-битовый либо 16-битовый тип записи с одним и более битовыми полями указанной ширины и (необязательно) с указанным начальным значением.

Примечания:

Параметр ширина указывает число битов от 1 до 16. Может быть включено любое число параметров fieldName:ширина, но общее число бит не должно превышать 16. Параметры fieldName:ширина разделяются запятыми.

Необязательный параметр "=выражение" позволяет задавать начальное значение для поля. Выражение должно приводиться к целому значению и не может включать ссылки вперед. Если поле имеет ширину по крайней мере семь битов, выражение может быть символом ASCII.

Первое объявленное поле становится старшим полем записи.Последнее объявленное поле становится младшим полем записи.Младший бит младшего поля всегда располагается в нулевом бите.Так например, если сумма длин Ваших полей равняется семи битам, они будут занимать биты с 6-го по нулевой.

Если общая ширина записи 8 и менее бит, ассемблер использует один байт. Если она больше восьми, ассемблер использует два байта. Во всех случаях неиспользуемые биты обнуляются ассемблером.

Директива RECORD сама по себе не создает данных. Она только определяет тип данных. Для создания данных этого типа используйте команду:

[имя] recordName <[начальноеЗначение,,,]> , где имя - это имя переменной, а recordName - имя типа записи, ранее определенного директивой RECORD. Для одного поля записи может использоваться одно начальное значение, для отделения начальных значений для каждого поля используют запятые. Например <,,3> инициализирует третье поле структуры.

Пример.

```
equipment   RECORD  disk:2, printer:2=1, RS232:1
PCequipment  equipment   <>
```

Эти две строки создают 8-битовую запись с именем PCequipment (оборудование) с полем RS232, занимающим бит 0 и равным 0, с полем printer, занимающим биты 2-1 и равным 1, и с полем disk, занимающим биты 4-3 и равным 0. Биты 7-5 будут инициализированы в 0.

4.66 REPT Начало повторяемого блока

REPT выражение
операторы
ENDM .

Создает повторяющийся блок, заставляя ассемблер выполнять операторы столько раз, сколько задает выражение.

Примечания:

Выражение должно давать 16-битовое беззнаковое целое и не должно содержать внешних или неопределенных имен.

4.67 .SALL Подавление листинга всех макрорасширений

.SALL .

Заставляет ассемблер подавлять макрорасширения в листинге исходных операторов. Вызов макрокоманд содержится в исходном листинге, но строки программы, генерируемые при вызове отсутствуют.

4.68 SEGMENT Начало описания сегмента

имя SEGMENT [тип_расположения][тип_комбинирования]['класс']
имя ENDS

Отмечает начало сегмента.

Примечания:

Сегмент - это набор команд или данных, адреса которых вычисляются относительно одного и того же сегментного регистра.

Ваша программа может иметь более одного описания сегмента с одним и тем же именем. Ассемблер воспринимает все одноименные сегменты за один.

Параметры типрасположения, типкомбинирования и класс, если они присутствуют, должны описываться в указанном выше порядке. Все эти параметры необязательны.

Если параметр типрасположения присутствует, он указывает ассемблеру как выбирать начальный адрес для сегмента. Для более подробного описания смотри следующие пункты: BYTE, WORD, PARA и PAGE.

Если параметр типкомбинирования присутствует, он указывает ассемблеру как обрабатывать несколько сегментов с одним именем. Для более подробного описания смотри следующие пункты: PUBLIC, STACK, COMMON, MEMORY и AT.

Если параметр 'класс' включен, он указывает компоновщику, какие сегменты загружать в последовательные блоки памяти один за другим. Имя класс должно быть заключено в одиночные кавычки. Если не используются опции /ML или /MX, то имя класс не зависит от типа букв (большие они или маленькие). Заметьте, что имя класс не может быть присвоено ни одному символу в программе.

4.69 .SFCOND Подавление листинга ложных условий

.SFCOND .

Подавляет включение в распечатку блоков с отрицательными условиями (условные операторы, у которых IF-выражение дает ложь).

4.70 STRUC Определение структурного типа

имя STRUC
field_Definitions

имя ENDS.
Начинает описание структурного типа.
Примечания:
Структура может иметь любое число полей. Каждое описание поля должно соответствовать следующему:

```
[имя] DB | DW | DD | DQ | DT initialValue,,,
```

Необязательный параметр имя описывает имя поля. DB, DW, DD, DQ и DT описывают размер поля. InitialValue используется для указания значения по умолчанию, если при объявлении значение не указано значение. Для более полной информации по использованию данных директив смотри DB, DW, DD, DQ и DT.
Директива STRUC не создает данных. Она только описывает
тип данных. Для создания данных этого типа используйте:
[имя] structure_Name <[начальное_Значение,,,]> где
имя - имя переменной;
structure_Name - имя типа структуры, ранее описанной с помощью директивы STRUC.
Для одного поля структуры может использоваться одно начальное значение, для разделения начальных значений для каждого поля используйте запятые. Например, <,,3> определит третье поле структуры.
Описание структуры может содержать только комментарии и описания полей, следовательно, вложенность структур запрещена.

4.71 SUBTTL Описание подзаголовка для листинга

SUBTTL текст .
Указывает какой текст использовать во второй строке каждой страницы листинга.
Примечания:
Внутри программы может использоваться любое число подзаголовков (subtitle), каждый новый заменяет старый. Если подзаголовок не указан, вторая строка страницы будет пустой. Используются только первые 60 символов текста.

4.72 .XALL Список макрорасширений,генерирующих коды

.XALL .
Заставляет ассемблер выдавать листинг исходных операторов раскрываемых макросов только для тех команд, которые дают данные или коды. Комментарии исключаются.
Примечания:
Этот режим листинга макрорасширений задан по умолчанию.

4.73 .XCREF Подавление формирования списка перекрестных ссылок

.XCREF [имя,,,] .
Подавляет генерацию перекрестных ссылок и символьных таблиц входов для всех
выбранных меток, переменных или символов.
Примечания:
В аргументе может стоять одно и более имен, разделенных запятыми. Если данный аргумент задан, то перекрестные ссылки и символьная таблица входов подавляются только для указанных имен.
Смотри .CREF для полного описания того, как .CREF и .XCREF действуют сообща.

4.74 .XLIST Подавление списка исходных кодов

.XLIST .
Подавляет листинг нижеследующих исходных кодовых строк до момента, когда листинг будет разрешен командой .LIST.
Примечания:
Эта команда отменяет все другие команды для листинга.