Лекция 10

2.5. Команды обработки строк (цепочки) символов

Под *строкой* символов здесь понимается последовательность байт, а *цепочка* — это более общее название для случаев, когда элементы последовательности имеют размер больше байта — слово или двойное слово.

Таким образом, цепочечные команды позволяют проводить действия над блоками памяти, представляющими собой последовательности элементов следующего размера:

- 8 бит — байт;
- 16 бит — слово;
- 32 бита — двойное слово.

Содержимое этих блоков для микропроцессора не имеет никакого значения. Это могут быть символы, числа и все что угодно. Главное, чтобы размерность элементов совпадала с одной из перечисленных и эти элементы находились в соседних ячейках памяти.

Предусмотрено **7** основных операций (примитивов):

1. пересылка элементов;
2. сравнение элементов;
3. сканирование элементов;
4. загрузка элементов;
5. хранение элементов;
6. получение элементов цепочки из порта ввода-вывода;
7. вывод элементов цепочки в порт ввода-вывода

2.5.1. Пересылка

Команда **MOVS:**
MOVS адрес_приемника, адрес_источника (MOVe String) — переслать цепочку;
MOVSB (MOVe String Byte) — переслать цепочку байт;
MOVSW (MOVe String Word) — переслать цепочку слов;
MOVSD (MOVe String Double word) — переслать цепочку двойных слов.

Команда **MOVS** копирует байт, слово или двойное слово из цепочки, адресуемой операндом **адрес_источника**, в цепочку, адресуемую операндом **адрес_приемника**.

При трансляции в зависимости от типа операндов транслятор преобразует ее в одну из трех машинных команд: **MOVSB, MOVSW** или **MOVSD.**

Если перед командой написать префикс **REP**, то одной командой можно переслать до 64 Кбайт данных (если размер адреса в сегменте 16 бит — use16) или до 4 Гбайт данных (если размер адреса в сегменте 32 бит - use32).

Алгоритм:

1. Установить значение флага **DF** в зависимости от того, в каком направлении будут обрабатываться элементы цепочки — в направлении возрастания (**DF=0**) или убывания адресов ((**DF=1**));
2. Загрузить указатели на адреса цепочек в памяти в пары регистров **DS**:**(E)SI** и **ES: (E)DI;**

3. Загрузить в регистр **ECX/CX** количество элементов, подлежащих обработке;
4. Выдать команду **MOVS** с префиксом **REP.**

Следовательно, перед выполнением команд обработки строк нужно соответственно установить состояние флага **DF** с помощью команды:

STD (set direction flag) - DF = 1 - в направлении убывания адресов.

CLD (clear direction flag) (DF = 0) - в направлении возрастания адресов.

Префиксы повторения

Вместо команд **LOOP** здесь употребляются специальные префиксы повторения. Количество повторений предварительно записывается в регистр **CX**.

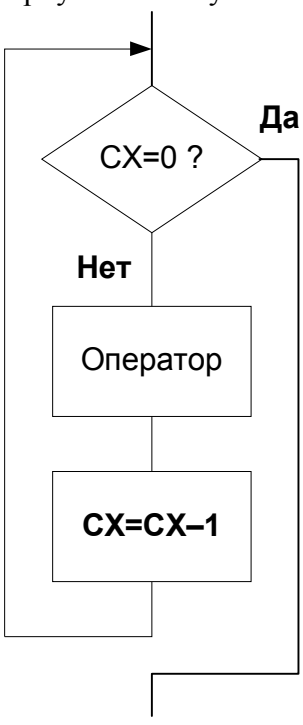
Например:

```
MOV CX, 50
REP MOVS DEST, SOURCE
```

REP используется перед строковыми командами и их краткими эквивалентами: **movs, stos, ins, outs**

Алгоритм:

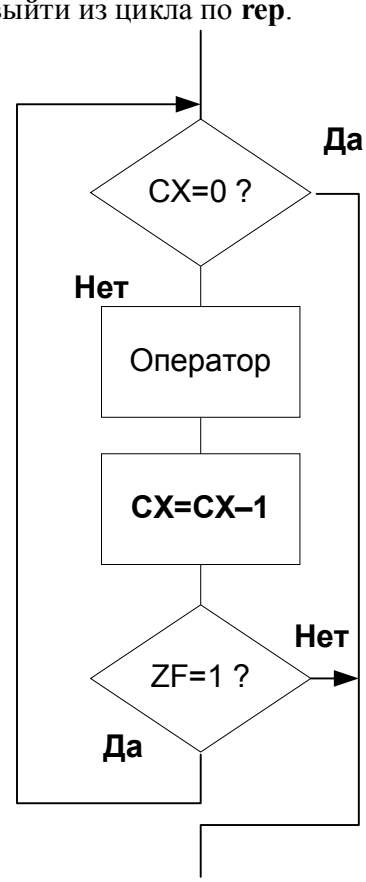
1. анализ содержимого **CX;**
2. если **CX<>0**, то выполнить строковую команду, следующую за данным префиксом и перейти к шагу 4;
3. если **CX=0**, то передать управление команде, следующей за данной строковой командой (выйти из цикла по **REP**);
4. уменьшить значение **CX=CX–1** и вернуться к шагу 1.



REPE и **REPZ** используются перед следующими цепочечными командами и их краткими эквивалентами: **cmps, scas**.

Алгоритм:

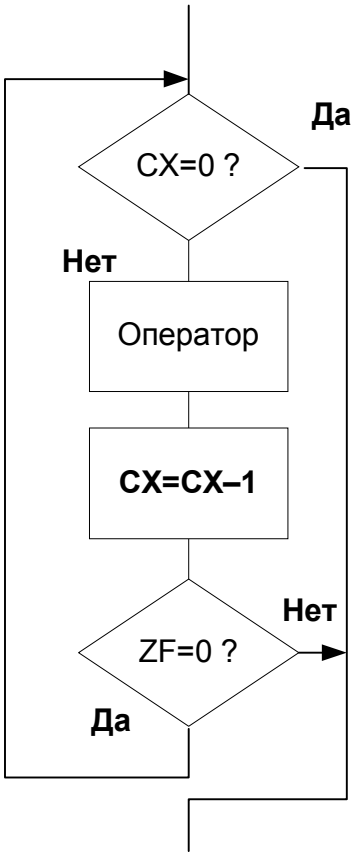
1. анализ содержимого **CX ;**
2. если **CX <>0** , то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 4;
3. если **CX =0** или **ZF=0**, то передать управление команде, следующей за данной цепочечной командой и перейти к шагу 6;
4. уменьшить значение **CX =CX -1**
5. если **ZF=1** вернуться к шагу 1
6. выйти из цикла по **rep**.



REPNE и **REPNZ** также имеют один код операции и имеют смысл при использовании перед эквивалентами: **cmps, scas**.

Алгоритм:

1. анализ содержимого **CX ;**
2. если **CX <>0** , то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 4;
3. если **CX =0** или **ZF=0**, то передать управление команде, следующей за данной цепочечной командой и перейти к шагу 6;
4. уменьшить значение **CX =CX -1**
5. если **ZF=0** вернуться к шагу 1
6. выйти из цикла по **rep**.



Фрагмент Пересылки строк командой **movs** будет иметь вид:

```
MASM
MODEL   small
STACK   256
.data
source  db      'Тестируемая строка','$'
;строка-источник
dest    db      19 DUP (' ')    ;строка-приёмник
.code
        assume  ds:@data,es:@data
main:   ;точка входа в программу
        mov     ax,@data        ; загрузка сегментных регистров
        mov     ds,ax   ; настройка регистров DS и ES       ;на адрес сегмента данных
        mov     es,ax
        cld     ; сброс флага DF — обработка строки от начала к концу
        lea     si,source       ;загрузка в si смещения строки-источника
        lea     di,dest ; загрузка в DS смещения строки-приёмника
        mov     cx,20   ; для префикса rep — счетчик повторений (длина строки)
rep     movs    dest, source    ;пересылка строки
        lea     dx, dest
        mov     ah,09h  ;вывод на экран строки-приёмника
        int     21h
exit:
        mov     ax,4c00h ; тоже самое, что и RET
        int     21h
end     main
```

2.5.2. Сравнения строк

CMPS адрес_приемника, адрес_источника - сравнивает байты или слова
CMPSB
CMPSW
CMPSD

Адрес_источника находится в сегменте данных и адресуется с помощью регистров **DS** и **SI**. **Адрес_приемника** - в дополнительном сегменте и адресуется с помощью регистров **ES** и **DI.**

Выполняется как вычитание, но в отличие от **CMP**, ***от источника отнимается приемник***. Это необходимо учитывать для следующих команд условного перехода.

Алгоритм:

1. Загрузить адрес источника — в пару регистров **DS:ESI/SI**;
2. Загрузить адрес назначения — в пару регистров **ES:EDI/DI**;
3. Выполнить вычитание элементов (источник – приемник);
4. В зависимости от состояния флага **DF** изменить значение регистров **ESI/SI** и **EDI/DI**:
5. Если **DF**=0, то увеличить содержимое этих регистров на длину элемента последовательности;
6. Если **DF**=1, то уменьшить содержимое этих регистров на длину элемента последовательности;
7. В зависимости от результата вычитания установить флаги:
8. Если очередные элементы цепочек не равны, то **CF**=1, **ZF**=0;
9. Если очередные элементы цепочек или цепочки в целом равны, то **CF**=0, **ZF**=1;
10. При наличии префикса выполнить определяемые им действия (см. команды REPE/REPNE).

Чаще с данной командой для повторения употребляется префикс **REPE** (до первого отличия) или **REPNE** (до первого совпадания).

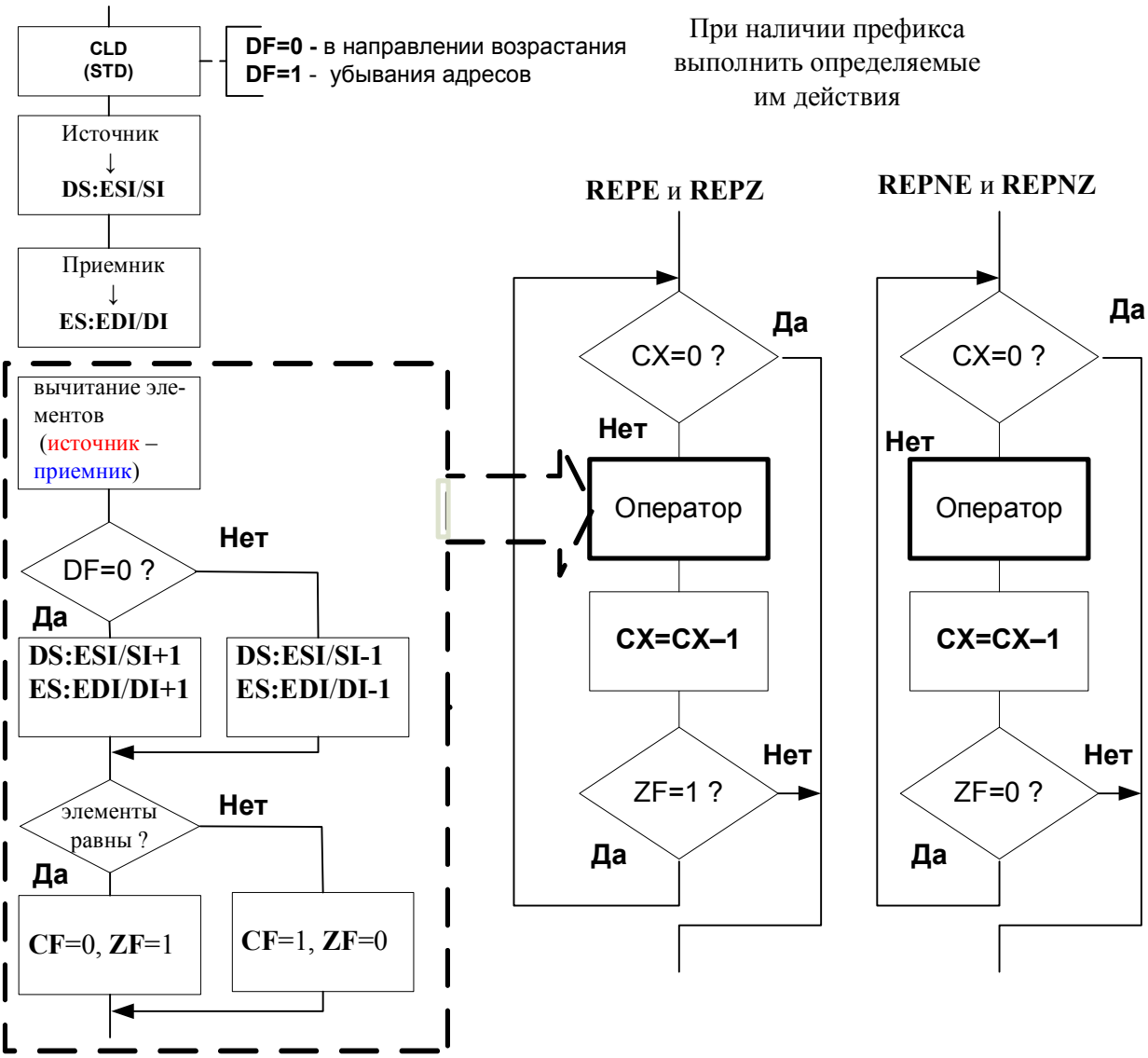
```
CLD
MOV CX,50
REPE CMPS DEST, SOURCE; до первого отличия
```

Выход из цикла происходит по двум причинам - массив пересмотрен полностью, или есть различия. Поэтому после **CMPS** необходимо реагировать на соответствующую причину, используя команды условного перехода.

Следовательно, после приведенного фрагмента нужно поставить переход на метку:

```
JNE FOUND; отличие найдено
……
FOUND:      ; да,.продлить обработку
```

Команда **CMPS** превращается транслятором: на **CMPSB** или на **CMPSW**, аналогично команде **MOVS**.



Например, можно определить число разных пар в строках:

```
MOV AX, 0; количество различных пар
CLD
LEA SI, SOURCE
LEA DI, ES:DEST
MOV CX, N
COMP: REPE CMPSB
JE FIN; переход на FIN, если строки равны
INC AX; следующая несовпадающая пара
CMP CX, 0; остались символы?
JNE COMP; продолжить, если строки не пусты
FIN:
```

2.5.3. Сканирования строк

SCAS адрес_приемника - позволяет найти значение (байт или слово) в строке-приемнике, который находится в дополнительном сегменте, начало которого фиксирует регистр ES.

SCASB
SCASW
SCASD

При этом заданное значение должно находиться в регистре **AX** (при поиске слова) или в **AL** (при поиске символа).

То есть, это фактически команды сравнения с содержанием аккумулятора.

Например, найти первую точку и заменить на '*':

```
CLD
LEA DI, ES: STRING
MOV AL, '.'
MOV CX, 50
REPNE SCAS STRING ; или SCAS B
JNE FIN ; точки нет
MOV BYTE PTR ES:[DI -1], '*'
FIN:
```

Это будет поиск в строке STRING первой точки. При выходе из цикла в регистре **DI** будет адрес следующего за точкой '.' байта.

2.5.4. Загрузка строки

После поиска слова или символа с помощью сканирования с ним что-то нужно сделать.

LODS адрес_источника - команда загрузки строки

LODSB
LODSW
LODSD

Пересылается операнд строка_источник, который адресован регистром **SI** из сегмента данных в регистр **AL** (при пересылке байта) или в регистр **AX** (пересылка слова), а затем изменяет регистр **SI** так, чтобы он показывал на следующий элемент строки. То есть, регистр **SI** увеличивается на 1 или 2, когда **DF** = 0 и уменьшается, когда **DF** = 1.

Следовательно, команды **LODS** эквивалентные двум:

```
MOV AL, [SI]
INC SI
```

Пример:

```
CLD
LEA DI, ES DEST ; смещение адреса DEST
LEA SI, SOURCE ; смещение адреса SOURCE
MOV CX, 100
REPE CMPSB; проверка отмены
        JCXZ MATCH; несовпадения нет?
        DEC SI; скорректировать SI
        LODS SOURCE; загрузить отличный элемент в регистр AL
MATCH; несовпадения нет
```

2.5.5. Команда сохранения строки

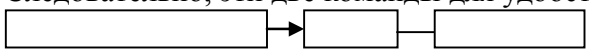
STOS адрес_приемника - противоположная к **LODS,** пересылает байт из **AL** или слово с **AX** в строку-приемник, которая находится в дополнительном сегменте и адресуется регистром **DI**. После этого изменяет регистр **DI** так, что он показывает на следующий элемент строки.

STOSB
STOSW
STOSD

Командой удобно заполнять строки определенными значениями. Например, в строке MY_STRING осуществляется поиск среди 100 слов первого ненулевого элемента. Если такой элемент найден, то следующие за ним 5 слов заполняются нулями.

```
CLD
LEA DI, ES: MY_STRING ; адрес строки
MOV AX, 0;
MOV CX, 100  ; счетчик поиска
REPNE SCASW ; сканирование строки
JCXZ NOT_SCAN  ; найдено ненулевое слово?
MOV CX, 5  ; да!
REP STOS MY_STRING
NOT SCAN
```

Следовательно, эти две команды для удобства, когда нашли что-то, то:



SOURCE AX(AL) DEST

2.5.6. Получение элементов цепочки из порта ввода-вывода

INS адрес_приемника, номер_порта - ввести элементы из порта ввода-вывода в цепочку.

INSB
INSW
INSD

Эта команда вводит элемент из порта, номер которого находится в регистре **dx**, в элемент цепочки, адрес которого определяется операндом **адрес_приемника**.

Пример: Введем 10 байт из порта 5000h в область памяти pole.

```
.data
pole    db      10 dup (‘ ‘)
.code
...
        push    ds
        pop     es      ;настройка es на ds
        mov     dx, 5000h
        lea     di, pole
        mov     cx,10
rep     insb
...
```

2.5.7. Вывод элемента цепочки в порт ввода-вывода

OUTS номер_порта, адрес_источника (Output String) — вывести элементы из цепочки в порт ввода-вывода.
OUTSB
OUTSW
OUTSD

Эта команда выводит элемент цепочки в порт, номер которого находится в регистре **dx**. Адрес элемента цепочки определяется операндом **адрес_источника**. Несмотря на то, что цепочка, из которой выводится элемент, адресуется указанием этого операнда, значение адреса должно быть явно сформировано в паре регистров **ds:esi/si**.

В качестве примера рассмотрим фрагмент программы, которая выводит последовательность символов в порт ввода-вывода, соответствующего принтеру (**номер 378 (lpt1)).**

```
.data
str_pech        db      'Текст для печати'
.code
...
        mov     dx,378h
        lea     di,str_pech
        mov     cx,16
rep     outsb
...
```

Команды загрузки адресных пар в регистры

Использование строчных команд нуждается в определенном количестве установочных команд. Их можно сократить с помощью двух команд, которые устанавливают пары регистров **DS:SI** и **ES:SI** на обработку строк.

Это команды: **LDS, LES, LFS, LGS, LSS**

LDS приемник, источник - получение полного указателя в виде сегментной составляющей и смещения.

Приемник - имя регистра, а **источник** - память - адрес двойного слова памяти, которое задает абсолютный адрес. Команда записывает в регистр смещения **ofs**, а в регистр **DS** - номер этого сегмента.

Алгоритм:

Алгоритм работы команды зависит от действующего режима адресации (use16 или use32):

- если **use16**, то загрузить первые ***два*** байта из ячейки памяти источник в 16-разрядный регистр, указанный операндом **приемник**. Следующие два байта в области **источник** должны содержать сегментную составляющую некоторого адреса; они загружаются в регистр **DS/ES/FS/GS/SS –** в зависимости от команды;
- если use32, то загрузить первые ***четыре*** байта из ячейки памяти источник в 32-разрядный регистр, указанный операндом **приемник**. Следующие два байта в области источник должны содержать сегментную составляющую, или селектор, некоторого адреса; они загружаются в регистр **DS/ES/FS/GS/SS**.

Пример:

```
ADR DD ALPHA; [ADR] → ofs [ADR + 2]:SEG
LDS SI, ADR;  SI - ofs, DS → SEG
```

Вторая команда:

LES приемник, источник - (Load pointer using **ES**) аналогичная к первой, только номер сегмента загружается в регистра **ES**.

Например:

```
DATA1 SEGMENT
        S1 DB 200DUP(?)
        AS DD S2
DATA1 ENDS
DATA2 SEGMENT
        S2 DB 200 DUP(?)
DATA2 ENDS
; копировать массив S1 в S2
CODE SEGMENT
         ASSUME CS: CODE, DS:DATA1
; пусть в этот момент DS = DATA1
        CLD
         LEA SI, S1 ; DS:SI = начало S1
         LES DI, AS ; ES:DI = начало S2
        MOV CX, 200
        REP MOVSB; копировать S1 в S2
```

2.6. Команды управления микропроцессором

1. команды управления флагами;
2. команды внешней синхронизации;

3. команды холостого хода.

2.6.1. Команды управления флагами

Позволяют влиять на состояние трех флагов регистра флагов: **CF** (carry flag) - переноса, **DF** (direction) – направления и прерывания **IF** (interupt F).

Структура команд очень простая: установить бит в единицу (Set), сбросить в 0 (Clear). Соответствующие две буквы является начальными для команды. Третья буква определяет, какой именно бит нужно обработать. Поэтому, имеем 6 команд:

STC	**STD**	**STI**
CLC	**CLD**	**CLI**

И седьмая команда - изменить значение биту переноса на противоположное:

CMC (CoMpliment Carr Flag).

Команды управления битом Carry употребляются перед выполнением команд циклического сдвига с переносом **RCR** (**RLC)** - операция циклического сдвига операнда вправо (влево) через флаг переноса **CF** .

Команды управления битом направления, как видно, используются перед командами обработки строк и задают направление модификации индексных регистров **DI** и **SI** (**DF** = 0 - в сторону увеличения, **DF** = 1 - в сторону уменьшения).

Команды управления битом **IF** используются, например, при обработке маскируемых прерываний. Если во время работы программы никакое маскируемое прерывание не позволяется, то нужно установить **IF** в 0. Напомним, что при этом немаскированные прерывания позволяются.

2.6.2. Команды внешней синхронизации

Используются для синхронизации работы МП с внешними событиями.

Команда **HLT** (halt -- остановить) - останавливает работу микропроцессора, он переходит на холостой ход и не выполняет никаких команд. Из этого состояния его можно вывести сигналами на входах **RESET**, **NMI**, **INTR**. Если для возобновления работы микропроцессора используется прерывание, то сохраненное значение пары **cs:eip/ip** указывает на команду, следующую за **hlt**. В микропроцессоре не предусмотрено специальных средств для подобного переключения. Сброс микропроцессора можно инициировать, если вывести байт со значением **0feh** в порт клавиатуры **64h**. После этого микропроцессор переходит в реальный режим и управление получает программа **BIOS**, которая анализирует байт отключения в **CMOS**-памяти по адресу **0fh**. Для нас интерес представляют два значения этого байта — **5h** и **0ah**:

5h — сброс микропроцессора инициирует инициализацию программируемого контроллера прерываний на значение базового вектора **08h**. Далее управление передается по адресу, который находится в ячейке области данных **BIOS 0040:0067**;

0ah — сброс микропроцессора инициирует непосредственно передачу управления по адресу в ячейке области данных **BIOS 0040:0067** (то есть без перепрограммирования контроллера прерываний).

Таким образом, если вы не используете прерываний, то достаточно установить байт **0fh** в **CMOS**-памяти в **0ah**. Предварительно, конечно, вы должны инициализировать ячейку области данных **BIOS 0040:0067** значением адреса, по которому необходимо передать управление после сброса. Для программирования **CMOS**-памяти используются номера портов **070h** и **071h**. Вначале в порт **070h** заносится нужный номер ячейки **CMOS**-памяти, а затем в порт **071h** — новое значение этой ячейки.

Если прерывания заблокированы во время останова, ЭВМ полностью "замирает". В этой ситуации единственная возможность запустить ЭВМ заново - выключить питание и включить его снова. Однако, если прерывания были разрешены в момент останова микропроцессора, они продолжают восприниматься и управление будет передаваться обработчику прерываний. После выполнения команды **IRET** в обработчике программа продолжает выполнение с ячейки, следующей за командой **HLT**. Команду **HLT** можно использовать в мультизадачных системах, чтобы завершить текущую активную задачу, но это не всегда лучший способ такого завершения. Разработчики персональной ЭВМ используют команду останова только тогда, когда возникает катастрофическая ошибка оборудования и дальнейшая работа бессмысленна.

Пример:

```
;работаем в реальном режиме, готовимся к переходу в защищенный режим:
      push   es
      mov    ax,40h
      mov    es,ax
      mov    word ptr es:[67h],offset ret_real
;ret_real — метка в программе, с которой должно
;начаться выполнение программы после сброса
      mov    es:[69h],cs
      mov    al,0fh  ;будем обращаться к ячейке 0fh в CMOS
      out    70h,al
      jmp    $+2     ;чуть задержимся, чтобы аппаратура отработала
;сброс без перепрограммирования контроллера
      mov    al,0ah
      out    71h,al
;переходим в защищенный режим установкой бита 0 cr0 в 1
;работаем в защищенном режиме готовимся перейти обратно в реальный режим
      mov    al,01fch
      out    64h,al  ;сброс микропроцессора  hlt
;остановка до физического окончания процесса сброса
      ret_real:      ...     ;метка, на которую будет передано управление после сброса
```

После этого МП переходит выполнять команду, следующую за **HLT**.

Команда **WAIT** (wait -- ожидать) - переводит МП на холостой ход, но при этом через каждые 5 тактов проверяется активность входной линии **TEST**. Если этот вывод активен во время выполнения **WAIT**, то остановка **НЕ** происходит.

Команда **ECS** (escape -- побег) - обеспечивает передачу команды МП 88/86 внешним процессорам, например, арифметическому сопроцессору 8087. Сам же МП 88 ничего не делает, только читает какие-то данные и отбрасывает.

Команды **WAIT** и **ESC** используются для работы с сопроцессором 8087.

Префикс **LOCK** - это командный префикс, подобно **REP**-префиксу, он предназначен для работы в мультипроцессорных системах, когда несколько процессоров работают с одним и тем же участком памяти. Префикс **LOCK** принуждает захватить линии управления и тем самым получить исключительное право доступа к памяти на время обработки команды с префиксом.

```
MOV AL,1
LOCK XCHG AL,FLAG_BYTE
CMP AL,1
```

В общей области FLAG_BYTE установлено 1, если в ней работает другой МП, и 0, если никакой МП не работает. Когда там есть 1, то МП будет ожидать, пока область не освободится.

Пустая команда **NOP** - нечего не выполняется 2 такта, реализуется как **XCHG AX, AX**.

Эту команду можно использовать:

- в программах реального времени, когда нужен выдержать точную длительность фрагментов.
- при налаживании программы, когда отдельные фрагменты могут изменяться.

Например:

```
JZ ONE
....
ONE: MOV AX, SUM[BX]
```

Лучше:

```
JZ ONE
.....
ONE: NOP
MOV AX, SUM[BX]
```