

Основные команды обработки строк для IBM PC

В компьютерном программировании, строка традиционно является последовательность из символов в виде литеральной константы или какой-либо переменной.

Строка обычно рассматривается как тип данных и часто реализуется как структура данных массива из байтов (или слов), в которой хранятся последовательность элементов, обычно символов, с использованием некоторой кодировки символов .

Строка также может обозначать более общие массивы или другие последовательности (или список) типы данных и структуры.

ПРЕДСТАВЛЕНИЕ СТРОК И СИМВОЛОВ

Буфер фиксированной длины.

S	T	R	I	N	G		O	N	E								
---	---	---	---	---	---	--	---	---	---	--	--	--	--	--	--	--	--

Дескриптор длины.

*	S	T	R	I	N	G		T	W	O	?	?	?	?	?	?	?	?	?
---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	---	---

↑

0Ah

Строка с дескриптором

На языке Ассемблера она выглядит следующим образом:

```
Pstring db 0Ah, "STRING TWO",10 dup(?)
```

Строки с нулевым окончанием.

S	T	R	I	N	G		T	H	R	E	E	*	?	?	?	?	?	?	?
---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	---	---	---

↑

(0 байт)

На языке Ассемблера она выглядит следующим образом:

```
Cstring db "STRING THREE", 0, 6 dup (?)
```

Команды строковых примитивов

Строка символов в Ассемблере - это последовательность байт.

Цепочка символов — последовательность элементов размером больше байта — слово или двойное слово.

Команды строковых примитивов могут называться **цепочечными командами**.

Цепочечные команды позволяют проводить действия над блоками памяти, представляющими собой последовательности элементов следующего размера:

- 8 бит, байт;
- 16 бит, то есть слово;
- 32 бита, то есть двойное слово.

Команды строковых примитивов

Команда	Описание	Операнды
MOVS MOVSB MOVSW	Перемещает строки данных. Копирование строки Копирование строки байтов. Копирование строки слов.	MOVS [DI] , [SI]
CMPS CMPSB CMPSW	Сравнение строк. Сравнение строк байтов. Сравнение строк слов.	CMPS [SI] , [DI]
SCAS SCASB SCASW	Сканирование строки. Сравнивает регистры AL, AX или EAX с содержимым памяти, изменяя регистр флагов.	SCAS [DI] , AL SCAS [DI] , AX SCAS [DI] , EAX
STOS STOSB STOSW	Запись в строку. Сохраняет строку данных: сохраняет содержимое регистров AL, AX или EAX в памяти	STOS [DI] , AL STOS [DI] , AX STOS [DI] , EAX
LODS LODSB LODSW	Чтение строки. Загружает аккумулятор из строки. Загружает байт, слово или двойное слово в AL, AX или EAX из памяти	LODS [DI] , AL LODS [DI] , AX LODS [DI] , EAX

Регистры процессора x86

Регистры данных

AH	AL	AX Аккумулятор
BH	BL	BX Базовый регистр
CH	CL	CX Счетчик
DH	DL	DX Регистр данных

Сегментные регистры

CS	Регистр сегмента команд
DS	Регистр сегмента данных
ES	Регистр дополнительного сегмента данных
SS	Регистр сегмента стека

Регистры-указатели

SI	Индекс источника
DI	Индекс приемника
BP	Указатель базы
SP	Указатель стека

Прочие регистры

IP	Указатель команд
FLAGS	Регистр флагов

ESI(32)/SI(16) (source index register) – индекс источника

Как и регистр **BX**, регистр **SI** может использоваться, как указатель на ячейку памяти.

Например:

```
mov  ax,0
mov  ds,ax
mov  si,20    ;Здесь 8-битовое значение,
               ; содержащееся по адресу 20,
mov  al,[si]  ; записывается в регистр AL.
```

Особенно полезно использовать регистр **SI** для ссылки на память в строковых инструкциях процессора 8086.

SI всегда, как указатель на исходную ячейку памяти (источник).

Например:

```
mov ax,0
```

```
mov ds,ax
```

```
mov si,20 ; содержимое по адресу памяти,
```

```
mov al,[si] ; на который указывает SI,  
; сохраняется в регистре AX,
```

```
lodsb ; к SI также добавляется 1.
```

Это может оказаться очень эффективным при организации доступа к последовательным ячейкам памяти (**к строке текста**).

EDI(32)/DI(16) (destination index register) – индекс приёмника (получателя)

DI всегда служит указателем на целевую ячейку памяти (приемник).

При использовании его в строковых инструкциях он имеет также особые свойства.

Например:

```
mov  ax,0
mov  ds,ax
mov  di,1024 ; 8-битовое значение,
              ; расположенное по адресу 1024,
add  bl,[di] ; записывается в регистр BL.
lodsb
```

Содержимое индексных регистров SI и DI во время работы строковых команд изменяется автоматически в соответствии с правилом:

$$SI \leftarrow SI \pm \text{delta}, DI \leftarrow DI \pm \text{delta}.$$

Здесь

delta=1 для байтовых строк

delta=2 для 16-разрядных машинных слов.

Индексные регистры SI и DI хранят индексы (смещения) относительно некоторой базы (т.е. начала массива) при выборке операндов из памяти.

Строка - Приемник

Обязательно находиться в дополнительном сегменте памяти

Адресация → <ES:DI>
ES – заменить нельзя

Строка - Источник

Находиться в сегменте данных

Адресация → <DS:SI>
DS можно заменить

Операция пересылки

$(DS : SI) \rightarrow (ES : DI)$

Регистры ES и DS могут быть инициализированы одновременно.

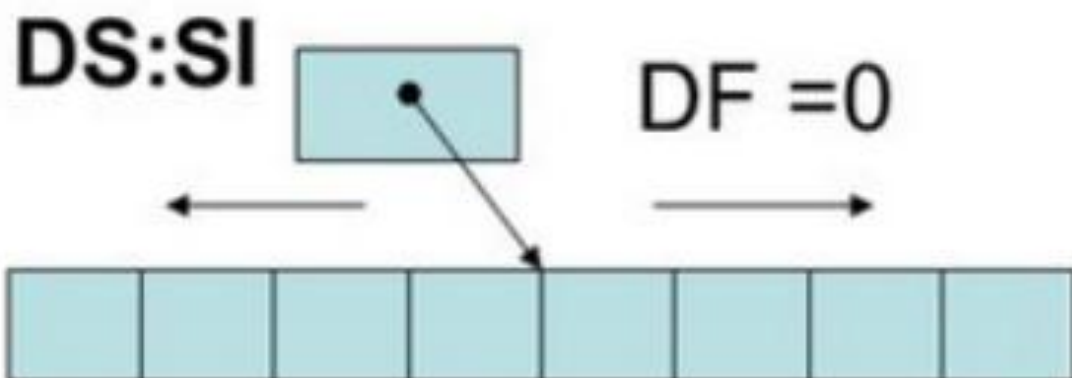
Пример

Mov ax, @data ; установить адрес сегмента данных

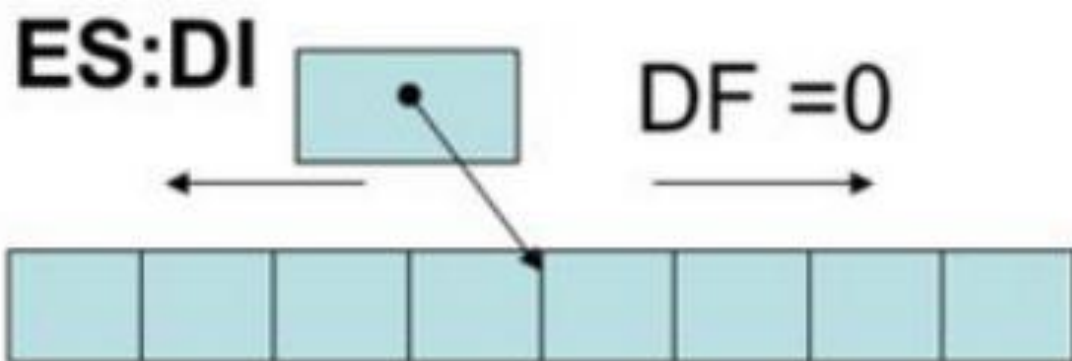
Mov ds, ax ; инициализировать DS

Mov es, ax ; инициализировать ES

Источник



Приемник



Для определения направления смещения команды строковых примитивов используют **флаг направления**.

Значение флага	Действия на SI и DI	Направление
$\langle DF \rangle = 0$	SI и DI – увеличиваются (инкрементируется)	Снизу вверх – строка обрабатывается слева на право – в сторону больших адресов
$\langle DF \rangle = 1$	SI и DI – уменьшаются (декрементируется)	Сверху вниз - строка обрабатывается справа налево – в сторону меньших адресов

Флаг направления изменяется командами CLD и STD:

CLD	;	сбросить флаг направления, направление - вверх
STD	;	установить флаг направления, направление - вниз

Данные команды не содержат операндов, т.к. имеют четкое назначение – изменение значения флага

Каждая команда строковых примитивов имеет три допустимых формата.

Общий формат может использоваться с байтами, словами или двойными словами.

Команды, заканчивающиеся буквой **B**, используют только 8-разрядные операнды.

Команды, заканчивающиеся буквой **W**, используют только 16-разрядные операнды.

Команды, заканчивающиеся буквой **D**, используют только 32-разрядные операнды.



Общие	Определенный размер	Описание	Шаг инкремента для SI и DI
MOVS	MOVSB	Перемещает (копирует) байт	1
	MOVSW	Перемещает (копирует) слово	2
	MOVSD*	Перемещает (копирует) двойное слово*	4



* - Только для процессоров Intel386, Intel486, Pentium.

Хоты команды обработки строк, как правило, включаются в программу без явного указания операндов, каждая команда использует два операнда.

MOVS *Получатель, отправитель* ; Копировать отправитель в получатель

CMPS *Получатель, отправитель* ; Сравнить отправитель и получатель

SCAS *Получатель* ; Сканировать строку

STOS *Получатель* ; Сохранить аккумулятор в получатель

LODS *Получатель* ; Загрузить аккумулятор из отправитель

До этого были инициализированы ES и DS, теперь в регистрах SI и DI должно быть установлено смещение операндов.

Mov SI, Offset source ; SI – указывает на отправителя

Mov DI, Offset dest ; DI – указывает на получателя

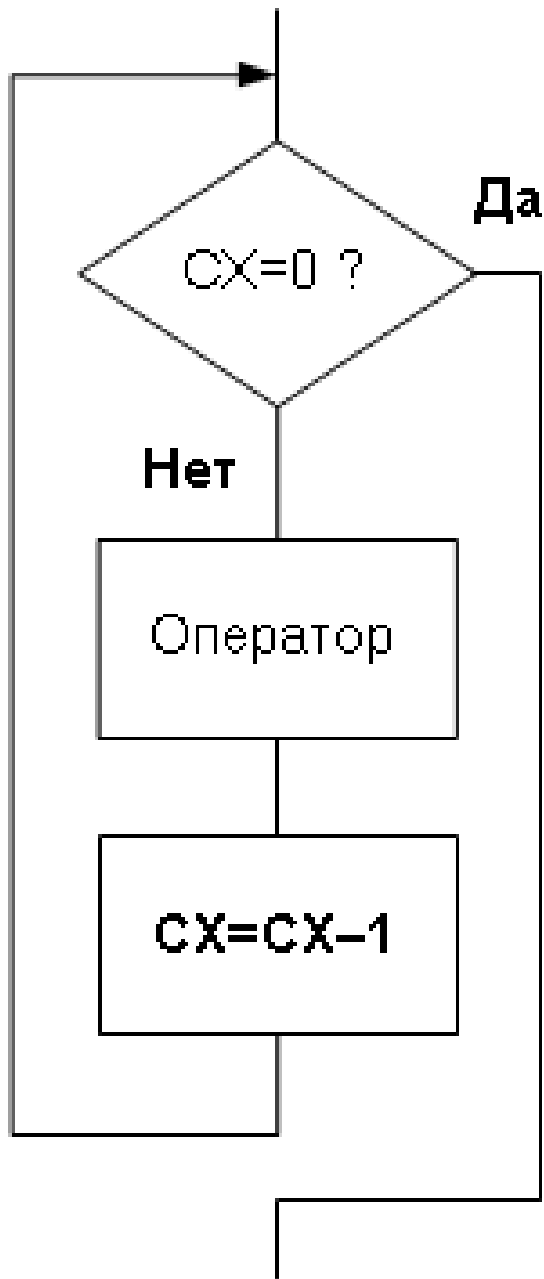
Movsb ; копирует отправитель в получатель

При однократном выполнении команды обрабатывают только один элемент, а для обработки строки команды должны повторяться одним из **префиксов повторения**.

REP ; Повторять, пока CX > 0

REPZ, REPE ; Повторять, пока флаг нуля установлен и CX <> 0

REPZ, REPNE ; Повторять, пока флаг нуля сброшен и CX <> 0



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

Алгоритм:

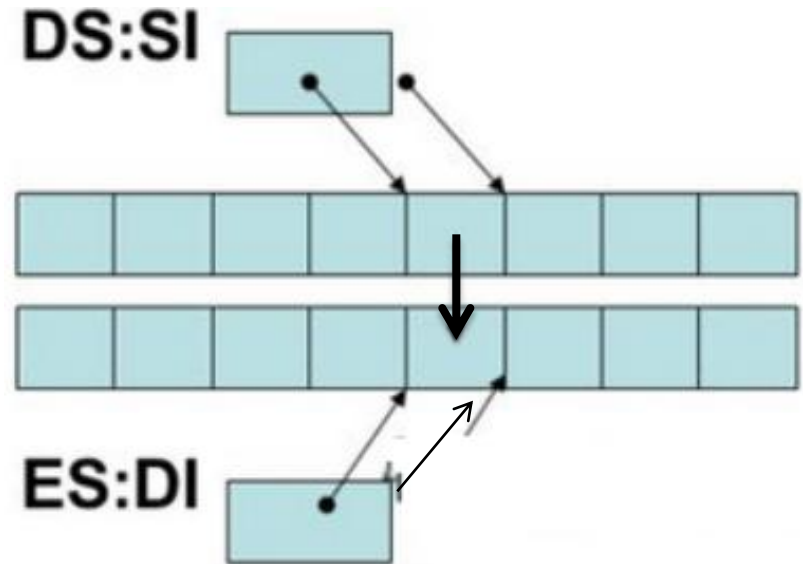
1. анализ содержимого **CX**;
2. если **CX<>0**, то выполнить строковую команду, следующую за данным префиксом и перейти к шагу 4;
3. если **CX=0**, то передать управление команде, следующей за данной строковой командой (выйти из цикла по **REP**);
4. уменьшить значение **CX=CX-1** и вернуться к шагу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

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

- перемещение элементов;
- сравнение элементов;
- сканирование элементов;
- загрузка элементов;
- хранение элементов;
- получение элементов цепочки из порта ввода-вывода;
- вывод элементов цепочки в порт ввода-вывода

Команда MOVSB

Команда перемещения
(копирования)
переносит данные из
места отправления,
адрес которого указан в
регистрах (DS : SI), в
место получения,
определяемого в
регистрах (ES : DI).



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**.

! 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.

Пример: копирование всех значений из массива source в массив target

Array_Size = 1000

Source dw array_size dup(?)

Target dw array_size dup(?)

mov AX, @data	; устанавливаем адрес сегмента данных
mov DS, AX	; инициализируем DS
mov ES, AX	; инициализируем ES
mov SI, Offset Source	; SI – указывает на отправителя
mov DI, Offset Target	; DI – указывает на получателя
mov CX, Array_Size	; счетчик повторений (число слов для копирования 1000)
Cld	; направление вверх
Rep movsw	; копирование из DS: SI в ES:DI

Выполняется пересылка 20 байт из STRING1 в STRING2 с использованием команды LEA.

Предположим, что оба регистра DS и ES инициализированы адресом сегмента данных:

```
STRING1 DB 20 DUP('*')  
STRING2 DB 20 DUP(' ')
```

...

CLD	;Сброс флага DF
MOV CX,20	;Счетчик на 20 байт
LEA DI,STRING2	;Адрес области "куда"
LEA SI,STRING1	;Адрес области "откуда"
REP MOVSB	;Переслать данные

Пример: Выборки данных из массива

Txt db 'A', 4, 'B', 4, 'A', 4, 'P', 4, 'И', 4, 'Я', '!', 4

Txt_len=\$ - Txt

; в программном сегменте

mov DI, 0B800h ; сегментный адрес видеобуфера

mov ES, AX ; инициализируем ES

; выведем на экран текст

mov DI, 1996 ; смещение к середине экрана

lea SI, txt ; DS:SI → txt

cld ; движение по строке вперед

mov CX, txt_Len/2 ; сколько слов переслать

Rep movsw ; пересылка в центр экрана красной
; (атрибут 4) надписи 'АВАРИЯ!'

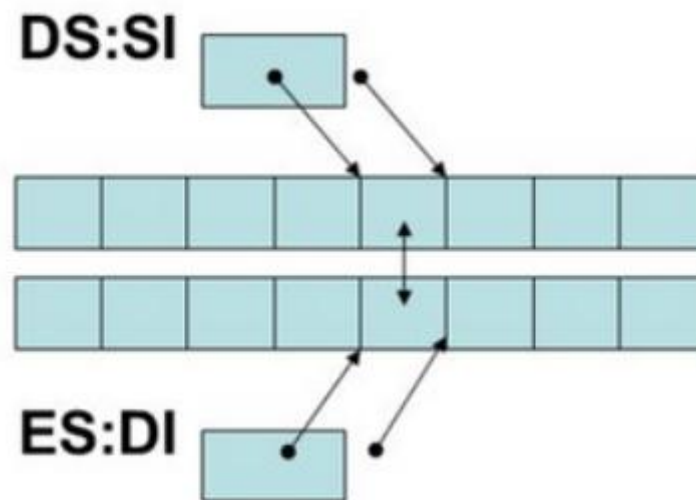
Команда CMPS (Compare String)

Команда CMPS сравнивает операнд-отправитель, адрес которого находится в регистрах DS:SI, с операндом получателем, на который указывают регистры ES:DI.

Осуществляя вычитание второго операнда из первого.

При выполнении данной команды необходимо использовать оба операнда

Команда CMPS превращается транслятором: на CMPSB или на CMPSW



Операцию сравнения можно условно изобразить следующим образом:

$(DS : SI) - (ES : DI) \rightarrow \text{флаги процессора}$

Порядок операндов


Следует быть особенно внимательным при использовании команды CMPS вместе с условными переходами.

Рассмотрим порядок выполнения операций:


Mov AX, 10


Cmp AX, 5 ; используется (AX – 5)

В команде CMP используется вычитание оператора-отправителя из оператора-получателя.


Cmps ES: dest, source ; используется (source – dest)

В команде CMPS, наоборот, используется вычитание оператора-получателя из оператора-отправителя.

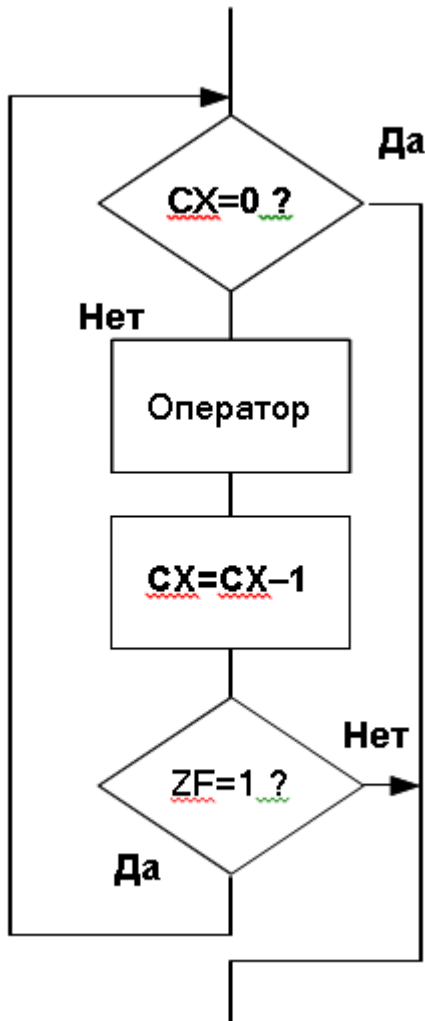
Чаще с данной командой для повторения употребляется префикс **REPE** (до первого отличия) или **REPNE** (до первого совпадения).

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

Алгоритм REPE и REPZ:

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**

для **REPNE** и **REPNZ** **ZF = 0**

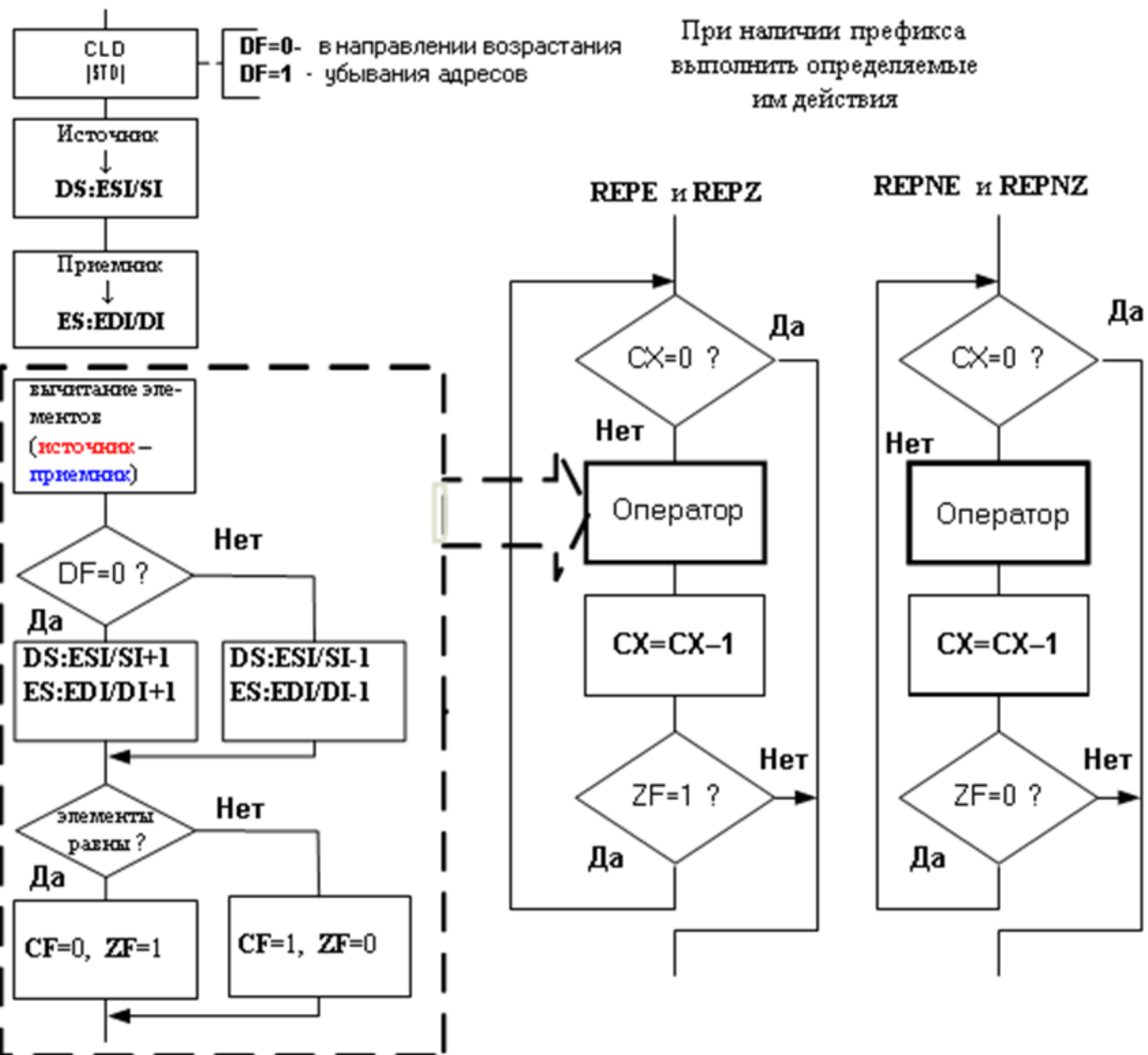


Пример

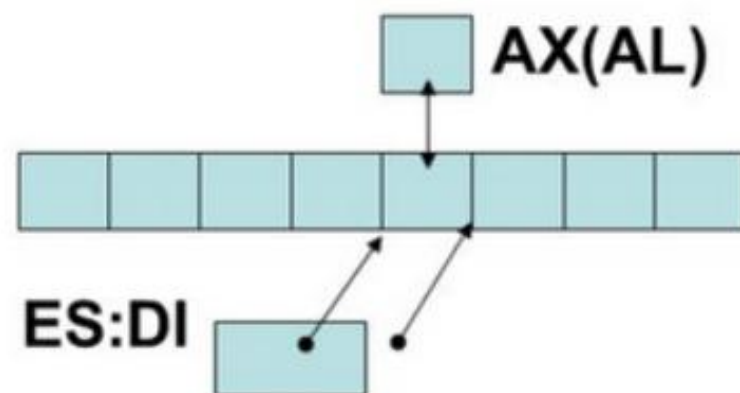
```
.data
; В полях данных сегмента данных, адресуемого через DS:
Str1    db  'FILE.001'          ; 1-я строка
; В полях данных сегмента данных, адресуемого через ES:
Str2    db  'FILE.002'          ; 2-я строка
.code
    cld                        ; сравнение вперед
    mov SI, offset Str1        ; DS:SI → Str1
    mov DI, offset Str2        ; ES:DI → Str2
    mov CX, 8                  ; длина сравниваемых строк
    repe cmpsb                 ; поиск различий в строке
    je equal                   ; переход, если строки совпадают
notequ:                          ; продолжение, если строки не совпадают
.
.
.
equal :                          ; продолжение, если строки совпадают
```

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.



Команда сканирования SCAS (Scan String)



Операцию сравнения можно условно изобразить следующим образом:

AX или AL – (ES : DI) → флаги процессора

Эти команды особенно удобны при работе с отдельными символами в длинной строке.

Могут быть использованы следующие префиксы повторения:

REP, REPE, REPZ ; строки в памяти и регистре совпадают

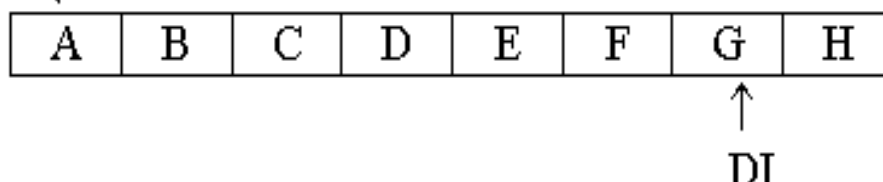
REPNE, REPNZ ; строки в памяти и регистре не совпадают

Сканирование для совпадающих символов.

В следующем примере строка STR сканируется до тех пор, пока символ F не будет найден.

```
.data
str db 'ABCDEFGH', 0
.code
    mov DI, seg alpha
    mov DS, DI
    mov DI, offset str      ; ES:DI указывает на строку
    mov AL, 'F'            ; поиск символа 'F'
    mov CX, 8              ; установить счетчик
    cld                    ; направление – вверх
    repne scasb            ; повторять пока не равно
    jnz exit               ; выход, если символ найден
    dec DI                 ; найден: декрементировать DI
```

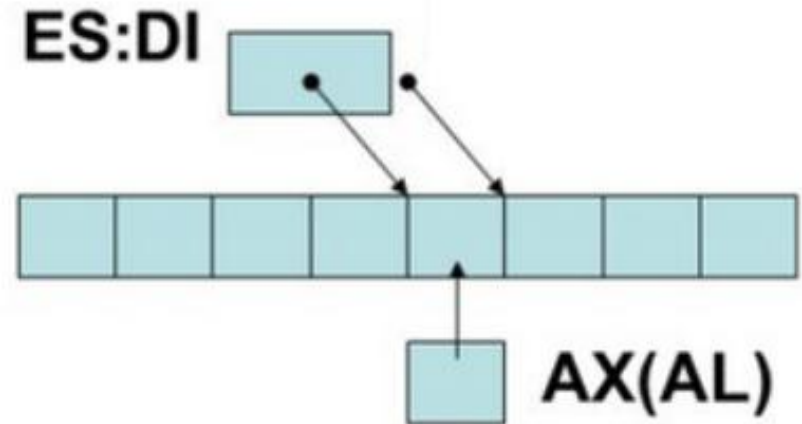
Сразу после того, как команда SCASB будет выполнена, DI укажет на символ, следующий за искомым символом.



Команды сохранения STOS (Store in String)

Команда STOS сохраняет содержимое регистров AL/AX/EAX в памяти по адресу ES:DI.

Регистры ES:DI содержат адрес операнда-получателя.



Пример 1.

Так же можно использовать STOS для инициализации памяти единственным значением. В следующем примере инициализируется каждый байт строки string1 значением 0FFh

; В полях данных сегмента данных, адресуемых через ES:

.data

string1 db 100 dup(?) ; Место под массив байт

.code

mov DI, Seg string1

mov ES, DI

mov AL, 0FFh ; значение для заполнения

mov DI, offset string1 ; ES:DI – адрес получателя

mov CX, 100 ; счетчик символов

cld ; направление – вверх

rep stosb ; заполнить содержимым AL

Команды сохранения STOS (Store in String)

Пример 2.

Допишем имеющуюся строку новыми значениями.

; В полях данных сегмента данных, адресуемых через ES:

.data

Immed db 'ID: '

.code

cld	;	движение по строке
mov DI, offset id+3	;	DI → за знаком ':'
mov AL, '3'	;	код ASCII цифры 3
stosb	;	отправим в строку
mov AL, '9'	;	код ASCII цифры 9
stosb	;	отправим в строку

; Теперь в строке Immed записано 'ID: 39'

Команды сохранения STOS (Store in String)

Пример 3.

Заполним только часть строки необходимыми значениями.

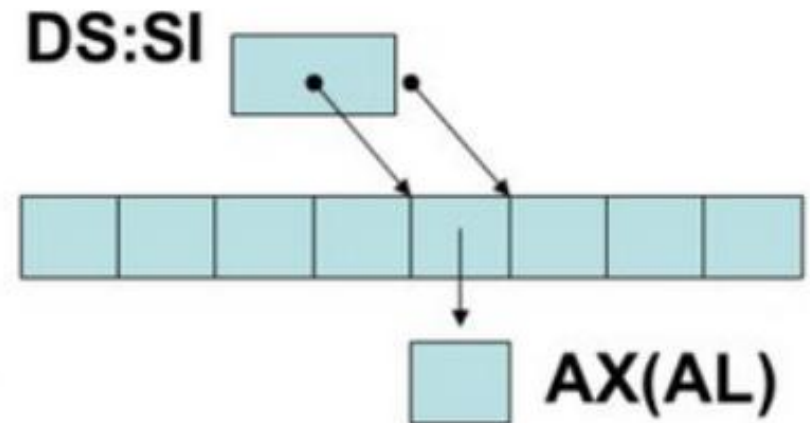
; В полях данных сегмента данных, адресуемых через ES:

```
.data
string1 db 100 dup(' ')      ;      Пустая строка символов
.code
    mov AL, '>'               ;      код ASCII цифры '>'
    mov CX, 5                 ;      заполнить 5 байтов
    cld                       ;      движение по строке вперед
    lea     DI, string1       ;      ES:DI → string1
    rep stos string1          ;      Первые 5 байтов строки string1
                                ;      заполняются кодом ASCII знаком '>'
```

Команды загрузки LODS (Load String)

загружают байт или первое слово в регистры AL/AX/EAX из памяти по адресу, указанному в DI:SI

В регистрах DS:SI содержится адрес операнда-отправителя.



Вместо отдельной команды LODSB, можно использовать следующие две команды:

`Mov AL, [si]` ; переместить байт по адресу DS:SI в AL

`Inc si` ; указать на следующий байт

Команда загрузки LODS (Load String)

Команды загрузки LODS загружают байт или первое слово в регистры AL/AX/EAX из памяти по адресу, указанному в DI:SI

В следующих командах производится сканирование буфера buffer, при этом очищается старший бит каждого байта и полученное значение сохраняется в output.

.data

buffer db 0C8h, 0FBh, 0F5h, 0CAh, 41h, 42h, 43h, 64h, 87h, 8Ch

output db 10 dup(?)

.code

cld	;	направление вверх
mov SI, offset buffer	;	буфер-отправитель
mov DI, offset output	;	буфер-получатель
mov CX, 10	;	длина буфера
L1: lodsb	;	копирование DS:[SI] в AL
and AL, 7Fh	;	очистить старший бит
stosb	;	сохранить AL в ES:[DI]
loop L1		

При выполнении фрагмента получаются следующие значения в выходной строке (повторяется каждый исходный байт со сброшенным старшим битом):

48 7B 75 4A 41 42 43 64 07 0C

INS Ввод из порта

Команды предназначены для ввода данных из порта непосредственно в память. Адрес порта указывается, в регистре DX, при этом задание адреса порта непосредственным значением не допускается. Данные пересылаются по адресу, находящемуся в паре регистров ES:EDI.

Замена сегмента не допускается.

После передачи данных регистр EDI получает положительное (если флаг DF=0) или отрицательное (если флаг DF=1) приращение. Величина приращения составляет 1, 2 или 4, в зависимости от размера передаваемых данных.

Формат: **ins строка, DX**

(что не избавляет от необходимости инициализировать регистры ES:EDI адресом строки).

Если устройство, адресуемое через порт, может передавать последовательность данных, то команды ins можно предварить префиксом повторения rep. В этом случае из порта принимается CX элементов данных заданного размера. Команды ins не воздействуют на флаги процессора.

Пример

;В сегменте данных, адресуемых через DS

mem dw 0

;В программном сегменте

push DS

pop ES ;ES=DS

mov DI,offset mem ;ES:DI -> mem

mov DX,303h ;Адрес порта

insw ;Ввод из порта 16-битового данного

OUTSW Вывод слова в порт

Команды предназначены для вывода данных в порт непосредственно из памяти.

Вариант команды outs имеет формат **outs DX, строка**

Пример 1

; В полях данных

mem dw 0FFh

;В программном сегменте

mov SI, offset mem

;ES:DI → mem

mov DX,303h

;Адрес порта

outsb

;Вывод в порт 8-битового данного