

2018

Глава 2 *Язык ассемблера IA-32*

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

2.1 Основные элементы языка ассемблера

Предложения ассемблера бывают четырех типов:

- **команды**, представляющие собой символические аналоги машинных команд. В процессе трансляции они преобразуются в машинные команды процессора;
- **директивы**, являющиеся указанием транслятору ассемблера на выполнение определенных действий. У директив нет аналогов в машинном представлении;
- **макрокоманды** — оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями из специальной библиотеки;
- **строки комментариев**, содержащие любые символы, в том числе и буквы русского алфавита, начинаются символом «;»

Виды лексем

При записи предложений языка используются:

- **служебные слова и символы** – мнемоники машинных команд, имена регистров процессора, имена директив и их атрибуты, знаки операций, встроенные идентификаторы и т.д.;
- **идентификаторы** – имена полей данных, метки команд, имена сегментов, имена процедур и т.п. – длина не должна превышать 247 байт, строчные и прописные буквы не различаются, первый символ – буква, символ подчеркивания «_», коммерческое «@» или знак «\$», например: k234, _delay;
- **литералы** – числа или строки в специальных ограничителях «'» или «"», например: 25, 'Пример'

Типы литералов

■ **целые константы:**

[<знак>] <целое> [<основание системы счисления>]

например:

- -43236, 236**d** – целые десятичные,
- 23**h**, 0AD**h** – целые шестнадцатеричные (если шестнадцатеричная константа начинается с буквы, то перед ней указывается 0),
- 0111010**b** – целое двоичное;

■ **вещественные константы:**

[<знак>] <целое> . [**E|e** [<знак>] <целое>],

например: -2., 34E-28;

- **символы** в кодировке ASCII или ANSI, например: 'A' или "A";
- **строковые константы**: 'ABCD' или "ABCD".

2.2 Структура программы на языке ассемблера

Программа на ассемблере для Win32 состоит из нескольких сегментов следующих типов:

- **сегмент кода**, содержащий команды ассемблера;
- **сегменты данных**:
 - **сегмент констант**, содержащий директивы объявления данных, изменение которых не предполагается;
 - **сегмент инициализированных данных**, содержащий директивы объявления данных, для которых заданы начальные значения – память под эти данные распределяется во время ассемблирования программы;
 - **сегмент неинициализированных данных**, содержащий директивы объявления данных – память под эти данные отводится во время загрузки программы на выполнение;
- **сегмент стека**, определяемый для ассемблера Win32 по умолчанию.

Упрощенные директивы сегментации

- **.MODEL** <Модель>[Модиф.][,<Язык>][,<Модификатор языка>]

где

- <Модель> – определяет набор и типы сегментов:
 - 32-х разрядная адр.: **FLAT** ;
- <Модификатор> – определяет тип адресации: **use16, use32, dos**
- <Язык>и <Модификатор языка>– определяют особенности передачи параметров при использовании разноязыковых модулей:

C, PASCAL, BASIC, FORTRAN, SYSCALL, STDCALL;

- **.CODE** [<Имя сегмента>] – начало или продолжение сегмента кода;
- **.STACK** [<Размер>] – начало или продолжение сегмента стека;
- **.DATA** – начало или продолжение сегмента инициализированных данных;
- **.DATA?** – начало или продолжение сегмента неинициализированных данных;
- **.CONST** – начало или продолжение сегмента неизменяемых данных

Пример 2.1 Консольное приложение

.586 ; разрешает использование набора команд i80586

.MODEL flat, **stdcall** ; определяет модель памяти

; и тип связи процедур

OPTION CASEMAP:NONE

; устанавливает чувствительность к регистру

Include kernel32.inc ; подключает файлы со стандартной

Include masm32.inc ; частью исходного текста

IncludeLib kernel32.lib ; подключает библиотеки

IncludeLib masm32.lib ; на этапе компоновки

.DATA ; сегмент инициализированных данных

Msg1 DB "Please Type Something: ", 0AH, 0DH, 0

Msg2 DB "Press Enter to Exit", 0AH, 0DH, 0

.DATA? ; сегмент неинициализированных данных

inbuf DB 100 **DUP** (?)

Пример 2.1 Консольное приложение (2)

.STACK 4096 ; сегмент стека – 4096 байт

.CODE ; сегмент кода

Start:

XOR EAX,EAX ; очистка регистра

Invoke StdOut,ADDR Msg1 ; вызов процедуры вывода

Invoke StdIn,ADDR inbuf,LengthOf inbuf

; вызов процедуры ввода

Invoke StdOut,ADDR inbuf; вызов процедуры вывода

Invoke StdOut,ADDR Msg2 ; вызов процедуры вывода

Invoke StdIn,ADDR inbuf,LengthOf inbuf

; вызов процедуры ввода

Invoke ExitProcess,0 вызов процедуры завершения

END Start

2.3 Директивы объявления данных

[<Имя>] <Директива> [<Число> DUP(<Инициализатор> [, Инициализатор] ...>)]

Директива	Описание типа данных
BYTE	8-разрядное целое без знака
SBYTE	8-разрядное целое со знаком
WORD	16-разрядное целое без знака или ближний указатель RM
SWORD	16-разрядное целое со знаком
DWORD	32-разрядное целое без знака или ближний указатель RM
SDWORD	32-разрядное целое со знаком
FWORD	48-разрядное целое или дальний указатель RM
QWORD	64-разрядное целое
TBYTE	80-разрядное целое
REAL4	32-разрядное короткое вещественное
REAL8	64-разрядное длинное вещественное
REAL10	80-разрядное расширенное вещественное

Примечание: Также могут использоваться директивы **DB**, **DW**, **DD**, **DQ**, **DT**, но в этом случае знаковые и беззнаковые, целые и вещественные типы транслятор не различает.

Примеры директив объявления данных

Инициализатор – константа соответствующего типа или символ ?.

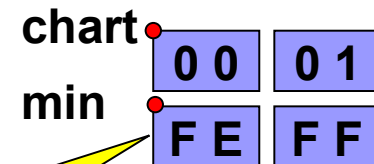
Примеры:

value1	BYTE	255	; <i>целое без знака</i>	value1	FF
value2	BYTE	'A'	; <i>символ</i>	value2	41
value3	SBYTE	-1	; <i>целое со знаком</i>	value3	FF
value4	BYTE	?	; <i>неинициализированное</i>	value4	?
value5	BYTE	10h	; <i>шестнадцатеричное</i>	value5	10
	BYTE	100101B	; <i>двоичное</i>		25
	DB	-128	; <i>целое со знаком</i>		80
beta	BYTE	23, 23h, 0ch		beta	17 23 0C
sdk	BYTE	"Hello", 0		sdk	48 65 6C 6C 6A 00

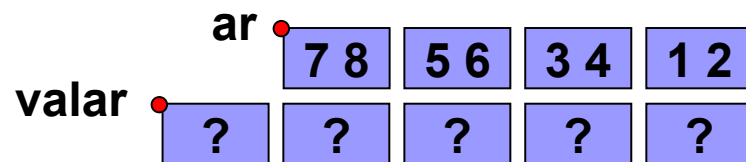
Примеры директив объявления данных (2)

Примеры:

chart WORD 256 ; *целое без знака*
min SWORD -32767 ; *целое со знаком*



ar DWORD 12345678h
valar BYTE 5 DUP (?)
rt REAL4 -2.1
de REAL10 4.6E+4096
vbn REAL4 20 DUP (0.0)
DT 4.8E-56



Символическая адресация данных

Если для некоторого поля данных, определяемого директивой, задано имя (имя поля данных), то в программе на ассемблере можно использовать это имя для указания поля, например:

A BYTE 25

. . .

mov AL, **A** ; поместить в регистр данное из **A**

В процессе трансляции ассемблер сопоставит имени смещение относительно начала сегмента данных и заменит все использования этого имени в качестве адреса данных на полученное смещение.

Начало сегмента
данных



mov AL, DS:[36]

2.4 Основные команды ассемблера

Формат команды ассемблера:

[<Метка>:] <Код операции> [<Список операндов>][;<Комментарий>]

Примеры:

- 1) `m1: mov AX, BX` ; пересылка числа
- 2) `cbw` ; преобразование байта в слово
- 3) ; суммы по месяцам

Метка – идентификатор, отмечающий адрес команды в памяти. В процессе трансляции ассемблер сопоставит метке смещение относительно начала сегмента кодов и заменит все использования метки в качестве адреса перехода на это смещение.

Адрес сегмента
кода



met: `mov EAX, EDX`

...

`jmp met`

met = 25

`jmp CS:[25]`

Операнды команд ассемблера

Операнды команд ассемблера могут:

- быть заданы неявно самой командой;
- задаваться непосредственно в коде команды:

`mov EAX, 25`

Непосредственный операнд

- находиться в регистрах процессора – в команде указывается имя регистра:

`mov EAX, EBX`

Операнд в регистре

Операнд в регистре

- храниться в оперативной памяти – указывается адрес операнда:

`mov EAX, 45[EBX+ECX*4]`

Адрес операнда
в памяти

Непосредственно
заданное смещение

База

Индекс

Масштаб

Размер операндов команд ассемблера

Длина операнда может определяться:

- **кодом команды** – если команда работает с единственным типом операндов, например,

`movsb ; работает только с байтами`

- **регистром**, используемым для хранения данных, например:

`mov EAX, A ; операнд – 4 байта`

- посредством использования **специальных описателей**:

- ☐ **BYTE PTR** – для операндов размером 1 байт,
- ☐ **WORD PTR** – для операндов размером 2 байта,
- ☐ **DWORD PTR** – для операндов размером 4 байта и т.д.

Описатели используют, если размер операнда

- не определяется первым или вторым способами:

`mov WORD PTR 0[EBX], 10`

- отличен от указанного при резервировании памяти:

`A WORD 2 dup (?)`

`. . .`

`mov AL, byte ptr A+3`

Адресация операндов в памяти

Адрес операнда (исполнительный) считается по формуле:

$$EA = (\text{База}) + (\text{Индекс}) * \text{Масштаб} + \text{Непосредственное смещение}$$

	База	Индекс	Масштаб	Смещение
	EAX			
CS:	EBX	EAX		
SS:	ECX	EBX	1	отсутств. ,
<u>DS</u> :	EDX	+ ECX	* 2	+ 8, 16 или
ES:	EBP	EDX	4	32 бита
FS:	ESP	EBP	8	
GS:	ESI	ESI		
	EDI	EDI		

Примеры:

inc	word ptr[500]	<i>; непосредственный адрес</i>
mov	ES:[ECX], EDX	<i>; задана только база</i>
mov	EAX, TABLE[ESI*4]	<i>; заданы индекс и масштаб</i>

Условные обозначения к описанию команд

r8 – один из 8-ми разрядных регистров: AL, AH, BL, BH, CL, CH, DL, DH;

r16 – один из 16-ти разрядных регистров: AX, BX, CX, DX, SI, DI, SP, BP;

r32 – один из 32-х разрядных регистров:

EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP;

reg – произвольный регистр общего назначения;

sreg – один из 16-разрядных сегментных регистров: CS, DS, ES, SS, FS, GS;

imm8 – непосредственно заданное 8-ми разрядное значение;

imm16 – непосредственно заданное 16-ти разрядное значение;

imm32 – непосредственно заданное 32-х разрядное значение;

imm – непосредственно заданное значение;

r/m8 – 8-ми разрядный операнд в регистре или в памяти;

r/m16 – 16-ти разрядный операнд в регистре или в памяти;

r/m32 – 32-ти разрядный операнд в регистре или в памяти;

mem – 8-ми, 16-ти или 32-х разрядный операнд в памяти;

rel8, rel16, rel32 – 8-ми, 16-ти или 32-х разрядная метка.

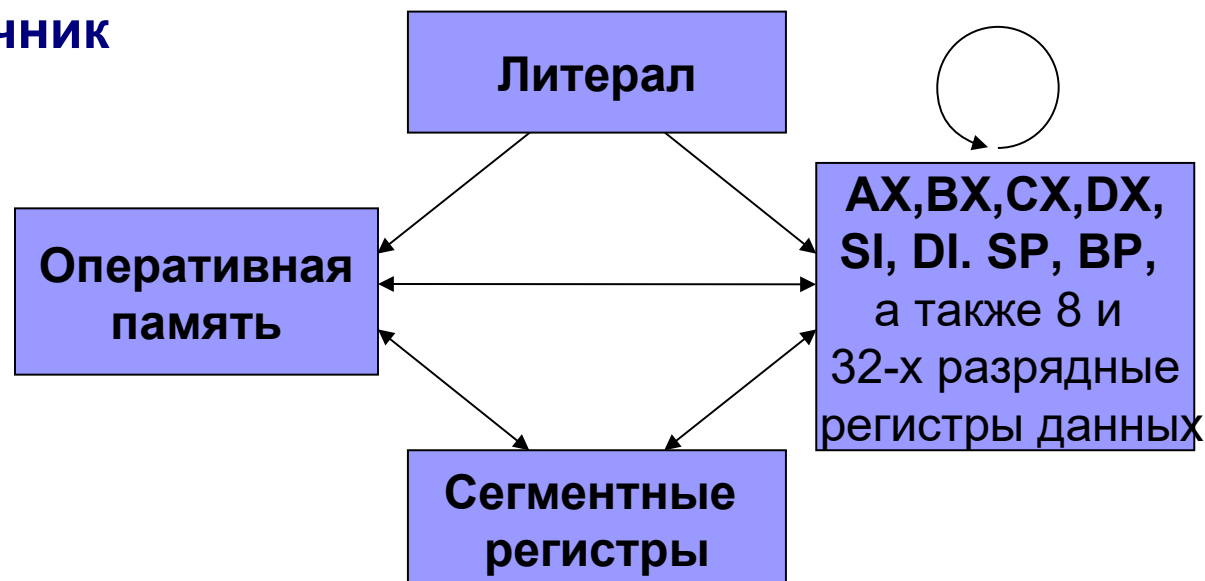
2.4.1 Команды пересылки / преобразования данных

1. Команда пересылки данных

MOV Приемник, Источник

Допустимые варианты:

```
mov reg, reg
mov mem, reg
mov reg, mem
mov mem, imm
mov reg, imm
mov r/m16, sreg
mov sreg, r/m16
```



Примеры:

- а) `mov AX, BX`
- б) `mov SI, 1000`
- в) `mov 0[EDI], AL`
- г) `mov AX, code`
`mov DS, AX`

Дополнительные ограничения:

- приемник и источник должны иметь один и тот же размер;
- в качестве приемника нельзя указывать CS, EIP и IP.

Команды пересылки / преобразования данных (3)

2. Команда обмена данных

XCHG Операнд1, Операнд 2

Допустимые варианты:

xchg reg, reg

xchg mem, reg

xchg reg, mem

Примеры:

а) **xchg EBX, ECX**

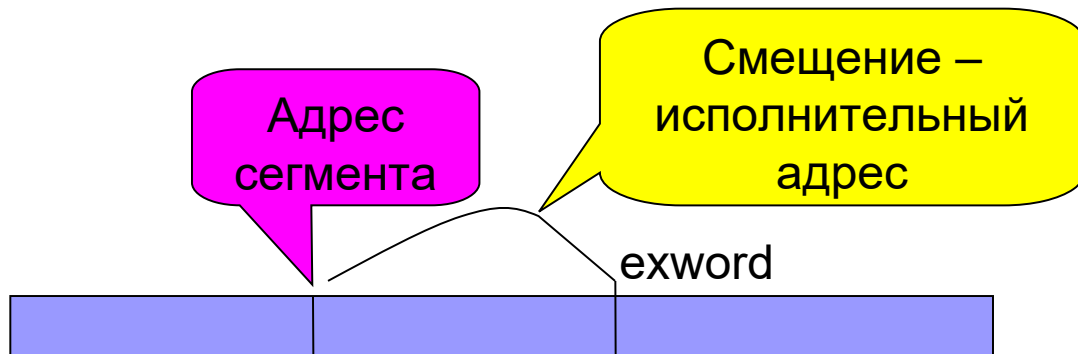
б) **xchg BX, 0[DI]**

3. Команда загрузки исполнительного адреса

LEA r32, mem

Примеры:

а) **lea EBX, exword**



б) **lea EDI, 6[EBX,ESI*2] ; $EA = (EBX) + (ESI)*2 + 6$**

Команды пересылки / преобразования данных (4)

4-5. Команды записи слова или двойного слова в стек и извлечения из стека

PUSH imm16 / imm32 / r16 / r32 / m16 / m32

POP r16 / r32 / m16 / m32

Если в стек помещается 16 разрядное значение, то значение
ESP := ESP-2, если помещается 32 разрядное значение, то
ESP := ESP-4.

Если из стека извлекается 16 разрядное значение, то значение
ESP := ESP+2, если помещается 32 разрядное значение, то
ESP := ESP+4.

Примеры:

push SI

pop word ptr [EBX]

Команды пересылки / преобразования данных (5)

6-7. Команды сложения

ADD Операнд1, Операнд2

ADC Операнд1, Операнд2

Допустимые варианты:

add reg, reg

add mem, reg

add reg, mem

Ограничение:

**операнды должны быть
одинаковыми по размеру.**

Складывает операнды и результат помещает по адресу первого операнда. В отличие от ADD команда ADC добавляет к результату значение бита флага переноса CF.

8-9. Команды вычитания

SUB Операнд1, Операнд2

SBB Операнд1, Операнд 2

Вычитает из первого операнда второй и результат помещает по адресу первого операнда. В отличие от SUB команда SBB вычитает из результата значение бита флага переноса CF. Ограничение то же.

Пример 2.2 Сложение 32 разрядных чисел

.DATA

```
A      SDWORD   -10
B      SDWORD    23
```

.DATA?

```
D      SDWORD   ?
```

.CODE

...

```
mov     EAX, A
```

```
add     EAX, B
```

```
mov     D, EAX
```

...

Тип (длина) данных при описании
и использовании совпадает,
следовательно дополнительный
описатель не нужен

Пример 2.3 Сложение 64-х разрядных чисел

.DATA

A QWORD -10

B QWORD 23

.DATA?

D QWORD ?

.CODE

...

mov EAX,dword ptr A

add EAX,dword ptr B

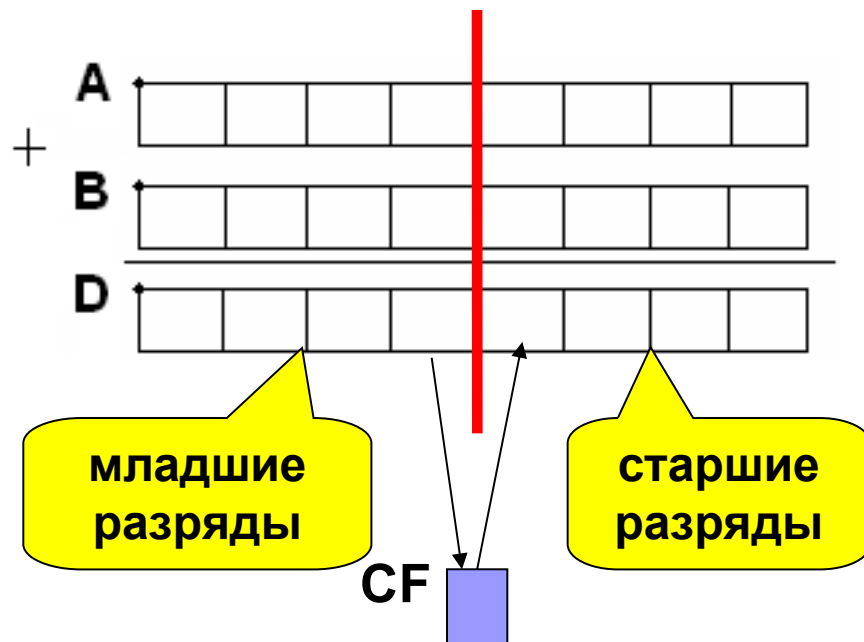
mov dword ptr D,EAX

mov EAX,dword ptr A+4

adc EAX,dword ptr B+4

mov dword ptr D+4,EAX

...



перенос в старшие разряды

Команды пересылки / преобразования данных (6)

10. Команда сравнения

CMP <Операнд 1> , <Операнд 2>

Выполняется как вычитание без записи результата.

Примеры:

Устанавливает флаги CF, SF, ZF и др.

а) `cmp AX,5`

б) `cmp byte ptr 0[BX], 'A'`

11-12. Команды добавления/вычитания единицы

INC reg/mem

DEC reg/mem

Примеры:

`inc AX`

`dec byte ptr 8[EBX,EDI]`

13. Команда изменения знака

NEG reg/mem

Команды пересылки / преобразования данных (6)

14-15. Команды умножения

MUL <Операнд2>

IMUL <Операнд2>

Команда MUL осуществляет
беззнаковое умножение, а
IMUL – знаковое.

Допустимые варианты:

mul/imul r|m8 ; AL * <Операнд2> ⇒ AX

mul/imul r|m16 ; AX * <Операнд2> ⇒ DX:AX

mul/imul r|m32 ; EAX * <Операнд2> ⇒ EDX:EAX

В качестве второго операнда нельзя указать непосредственное значение!!!

Регистры первого операнда в команде не указываются.

Местонахождение и длина результата операции зависит от размера второго операнда (байт, слово или двойное слово).

Пример:

mov AX,4

imul word ptr A ; DX:AX:=AX*A

Команды пересылки / преобразования данных (7)

16-19. Команды «развертывания» чисел

CBW ; байт в слово **AL -> AX**

CWD ; слово в двойное слово **AX -> DX:AX**

CDQ ; двойное слово в учетверенное **EAX -> EDX:EAX**

CWDE ; слово в двойное слово **AX -> EAX**

Команды не имеют операндов. Операнд и его длина определяются кодом команды и не могут быть изменены.

При выполнении команды происходит расширение записи числа до размера результата посредством размножения знакового разряда.

Команды используются при необходимости деления чисел одинаковой размерности для обеспечения удвоенной длины делимого (см. далее).

Команды пересылки / преобразования данных (8)

20-21. Команды деления

DIV <Операнд2>

IDIV <Операнд2>

Допустимые варианты:

div/idiv r|m8 ; AX:<Операнд2> \Rightarrow AL-результат, AH - остаток

div/idiv r|m16 ; (DX:AX):<Операнд2> \Rightarrow AX – рез. , DX - остаток

div/idiv r|m32 ; (EDX:EAX):<Операнд2> \Rightarrow EAX - рез. , EDX – ост.

В качестве второго операнда нельзя указать непосредственное значение!!!

Регистры первого операнда в команде не указываются.

Местонахождение и длина результата операции зависит от размера второго операнда.

Пример:

mov AX,40

cwd

idiv word ptr A; AX:=(DX:AX):A

Пример 2.4 Вычисление выражения

$$X = \frac{(A+D)(B-1)}{(D+8)}$$

.DATA

A WORD 25
B WORD -6
D WORD 11

.DATA?

X WORD ?

.CODE

```
...  
mov     CX,D  
add     CX,8;  CX:=D+8  
mov     BX,B  
dec     BX   ;  BX:=B-1  
mov     AX,A  
add     AX,D;  AX:=A+D  
imul    BX   ;  DX:AX:=(A+D)*(B-1)  
idiv    CX   ;  AX:=(DX:AX):CX  
mov     X,AX  
...
```

2.4.2 Команды передачи управления

1. Команда безусловного перехода

JMP	{	short	rel8
		<u>near ptr</u>	rel32 r32 m32
		far ptr	sreg:r32 m48

Команда выполняет безусловную передачу управления по указанному адресу:

- **rel8** – короткий переход – на -128..127 байт в пределах сегмента,
- **rel32, r32, m32** – ближний переход – в пределах сегмента (по умолч.),
- **m48** – дальний переход – в другой сегмент.

Примеры:

- а) `jmp short Label1 ; адрес задан меткой rel8`
- в) `jmp EBX ; адрес находится в регистре EBX`
- г) `jmp [EBX] ; адрес находится в памяти по адресу в EBX`
- б) `cycle: ...`
`jmp cycle ; адрес задан меткой rel32 или rel8`

Команды передачи управления (2)

2. Команды условного перехода

<Команда> rel8

Все команды имеют формат short, т.е. переход на -128..127 байт.

Мнемоники условного перехода:

- JZ – переход по "ноль";
- JE – переход по "равно";
- JNZ – переход по "не ноль";
- JNE – переход по "не равно";
- JL – переход по "меньше";
- JNG, JLE – переход по "меньше или равно";
- JG – переход по "больше";
- JNL, JGE – переход по "больше или равно";
- JA – переход по "выше" (беззнаковое "больше");
- JNA, JBE – переход по "не выше" (беззнаковое "не больше");
- JB – переход по "ниже" (беззнаковое "меньше");
- JNB, JAE – переход по "не ниже" (беззнаковое "не меньше").

Условный переход более чем на -128..127 байт

jz zero



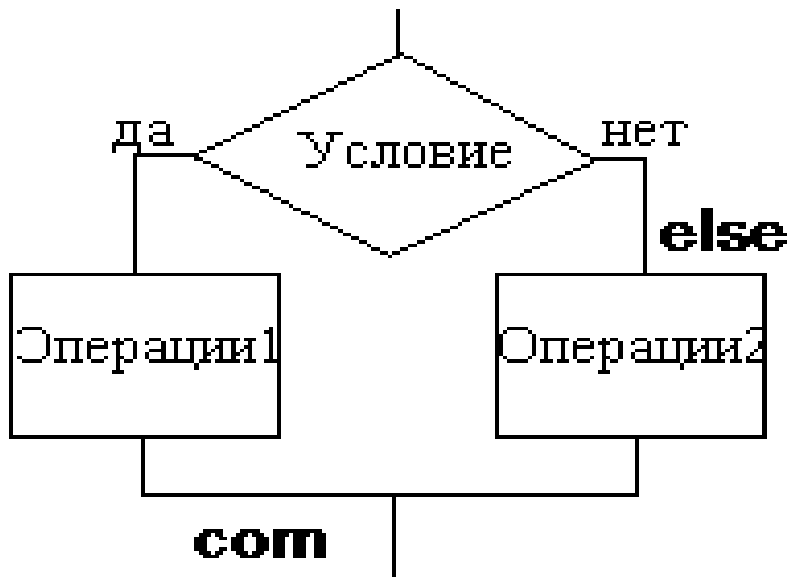
jnz continue

jmp zero

continue: ...

Если метка смещена
относительно текущего
адреса более, чем на
-128..127 байт, то переход
программируется
специальным образом

Программирование ветвлений



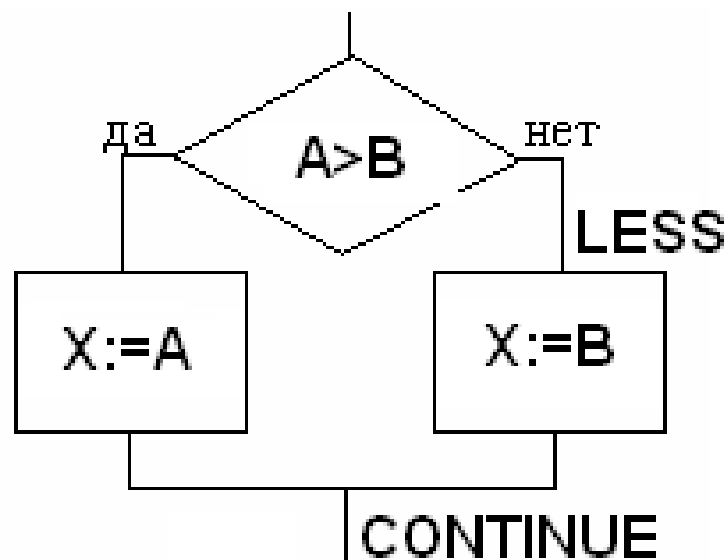
```
cmp    ...  
j<условие> ELSE  
<Операции 1>  
jmp    COM  
ELSE:  <Операции 2>  
COM:   <Продолжение>
```


Пример 2.5. Определение максимального из двух чисел

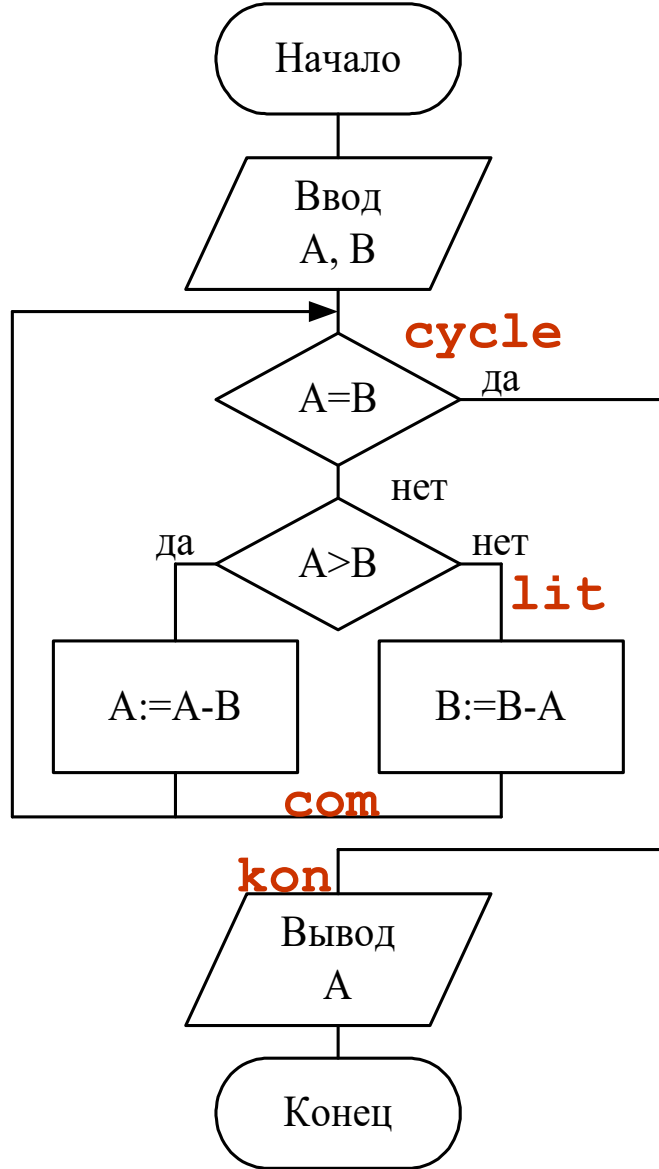
```
.DATA
A      SDWORD 334
B      SDWORD 745
X      SDWORD ?
```

```
.CODE
```

```
Start:  ...
        mov     EAX, A
        cmp     EAX, B      ; сравнение A и B
        jle     LESS        ; если первое меньше или равно
        mov     X, EAX
        jmp     short CONTINUE ; безусловный переход
LESS:    mov     EAX, B
        mov     X, EAX
CONTINUE: ...
```



Пример 2.6 Определение НОД



```

.DATA
A      WORD    24
B      WORD    18

```

```

.DATA?
D      WORD    ?

```

```

.CODE

Start:

```

```

        mov     AX,A
        mov     BX,B

cycle:   cmp     AX,BX
        je      kon
        jl      lit
        sub     AX,BX
        jmp     short com

lit:     sub     BX,AX
com:     jmp     cycle
kon:     mov     D,AX

```

...

Команды передачи управления (3)

3. Команды организации циклической обработки

1) Команда организации цикла

LOOP rel8

Выполнение команды:

- ECX:=ECX-1,
- если ECX=0, то происходит переход на следующую команду, иначе – короткий (-128..127 байт) переход на метку.

Пример:

```
        mov     ECX, loop_count
begin_loop:  <Тело цикла>
        ...
        loop    begin_loop
```

Примечание – Если в качестве счетчика используется CX, то перед командой следует вставить префикс размера адреса (67h):

```
        BYTE    67h
        loop    begin_loop
```

Команды передачи управления (4)

2) Команда перехода по обнуленному счетчику

JCXZ rel8

Если при входе в цикл значения счетчика равно 0, то произойдет «заикливание». Чтобы предотвратить заикливание значение регистра ECX надо проверить. Команда jcxz проверяет значение счетчика и, если оно равно нулю, то осуществляет переход на указанную метку.

Пример:

```
        mov    ECX, loop_count
        jcxz   end_of_loop
begin_loop: < Тело цикла >
        ...
        loop   begin_loop
end_of_loop:    ...
```

Команды передачи управления (5)

3) Команды организации цикла с условием

LOOPE rel8

LOOPNE rel8

Помимо регистра ECX команды проверяют значение флага ZF:

LOOPE осуществляет переход на метку, если $ZF=1$ & $ECX \neq 0$,

LOOPNE – если $ZF=0$ & $ECX \neq 0$,

иначе обе команды передают управление следующей команде.

Пример:

```
mov    ECX, loop_count
jcxz   end_of_loop
```

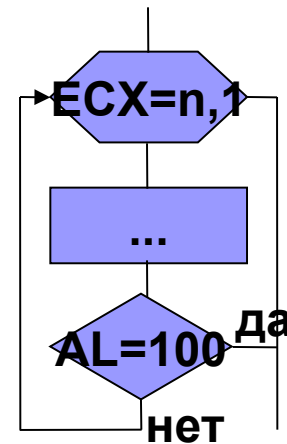
begin_loop:

< Тело цикла >

```
cmp     al, 100
```

```
loopne  begin_loop
```

end_of_loop: ...



Пример 2.7 Циклическая обработка

Определить сумму натуральных чисел 1..n.

.DATA

n DWORD 18

.DATA?

S SWORD ?

.CODE

Start:

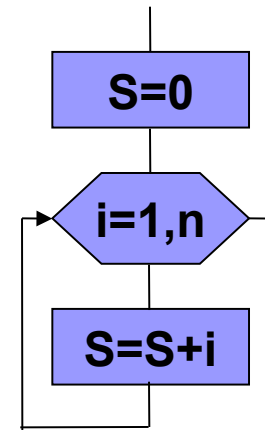
mov ECX,n

mov AX,0

cycle: add AX,CX

loop cycle

mov S,AX



mov ECX,n ; *счетчик*

mov AX,0 ; *сумма=0*

mov BX,1 ; *индекс*

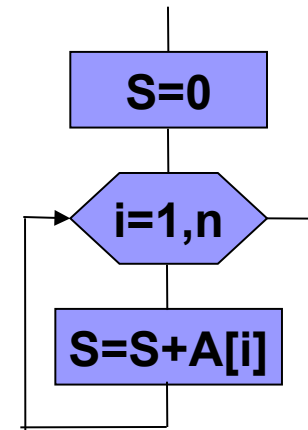
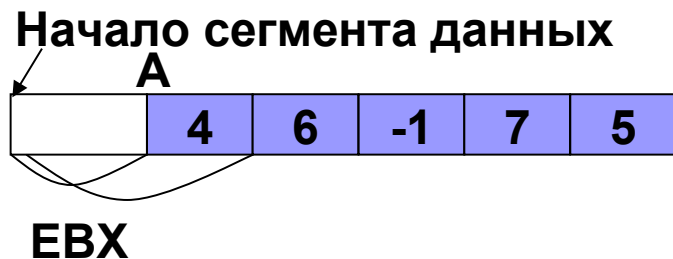
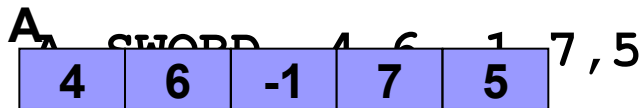
cycle: add AX,BX

inc BX ; *индекс++*

loop cycle

mov S,AX

Пример 2.8 Сумма элементов массива



Вариант 1

```
mov     AX, 0
lea     EBX, A ; смещение
mov     ECX, 5
cycle:  add     AX, 0[EBX]
        add     EBX, 2
        loop    cycle
        mov     S, AX
```

Вариант 2

```
mov     AX, 0
mov     EBX, 0 ; индекс
mov     ECX, 5
cycle:  add     AX, A[EBX*2]
        inc     EBX
        loop    cycle
        mov     S, AX
```

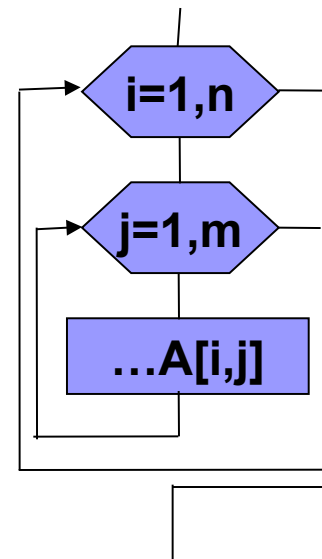
Построчная обработка матрицы

A	2	3	-1	-1	8
	6	-8	5	4	7
	8	6	-3	1	6

A SBYTE 2,3,1,-1,8
 SBYTE 6,-8,5,4,7
 SBYTE 8,6,3,1,6



```
mov     EBX, 0
mov     ECX, 3
cycle1: push  ECX
        mov     ECX, 5
cycle2: ...  A[EBX]
        inc     EBX
        loop    cycle2
        pop     ECX
        loop    cycle1
```

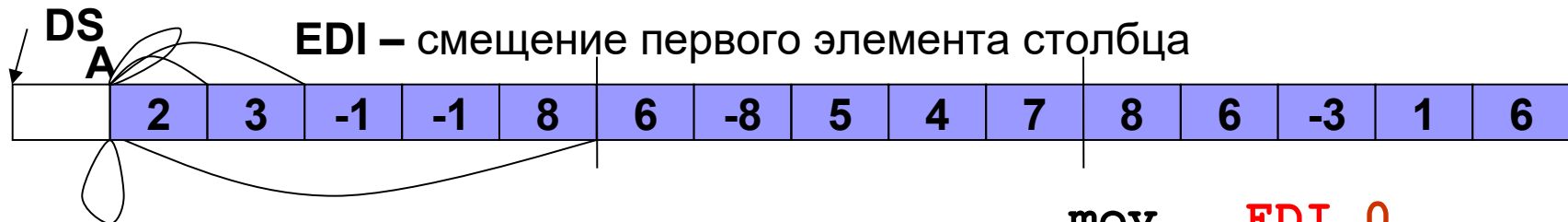


Обработка матрицы по столбцам

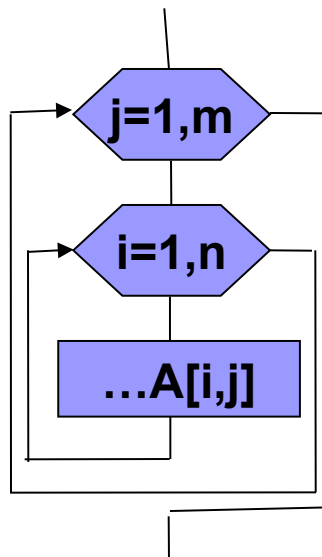
A

2	3	-1	-1	8
6	-8	5	4	7
8	6	-3	1	6

A SBYTE 2,3,1,-1,8
 SBYTE 6,-8,5,4,7
 SBYTE 8,6,3,1,6



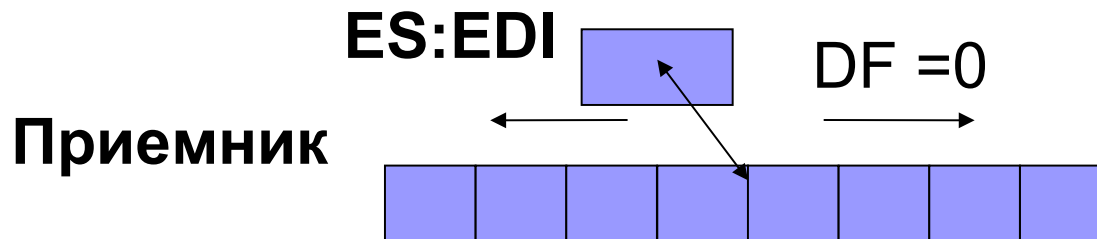
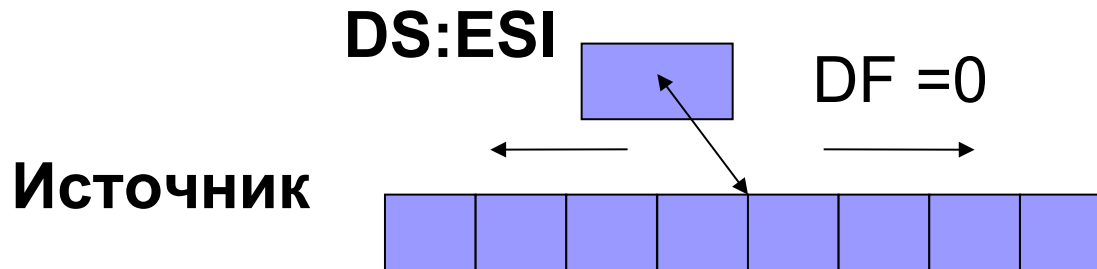
EBX – смещение элемента в столбце



```

mov     EDI,0
mov     ECX,5
cycle1: push ECX
mov     ECX,3
mov     EBX,0
cycle2: ... A[EBX,EDI]
add     EBX,5
loop    cycle2
pop     ECX
inc     EDI
loop    cycle1
  
```

2.5 Команды обработки цепочек



Элемент: байт, слово или двойное слово

Установка/сброс флага направления:

STD ; установить флаг *DF*

CLD ; сбросить флаг *DF*

Команды обработки строк (2)

1. Команда загрузки строки **LODS**

LODSB ; загрузка байта

LODSW ; загрузка слова

LODSD ; загрузка дв. слова

2. Команда записи строки **STOS**

STOSB ; запись байта

STOSW ; запись слова

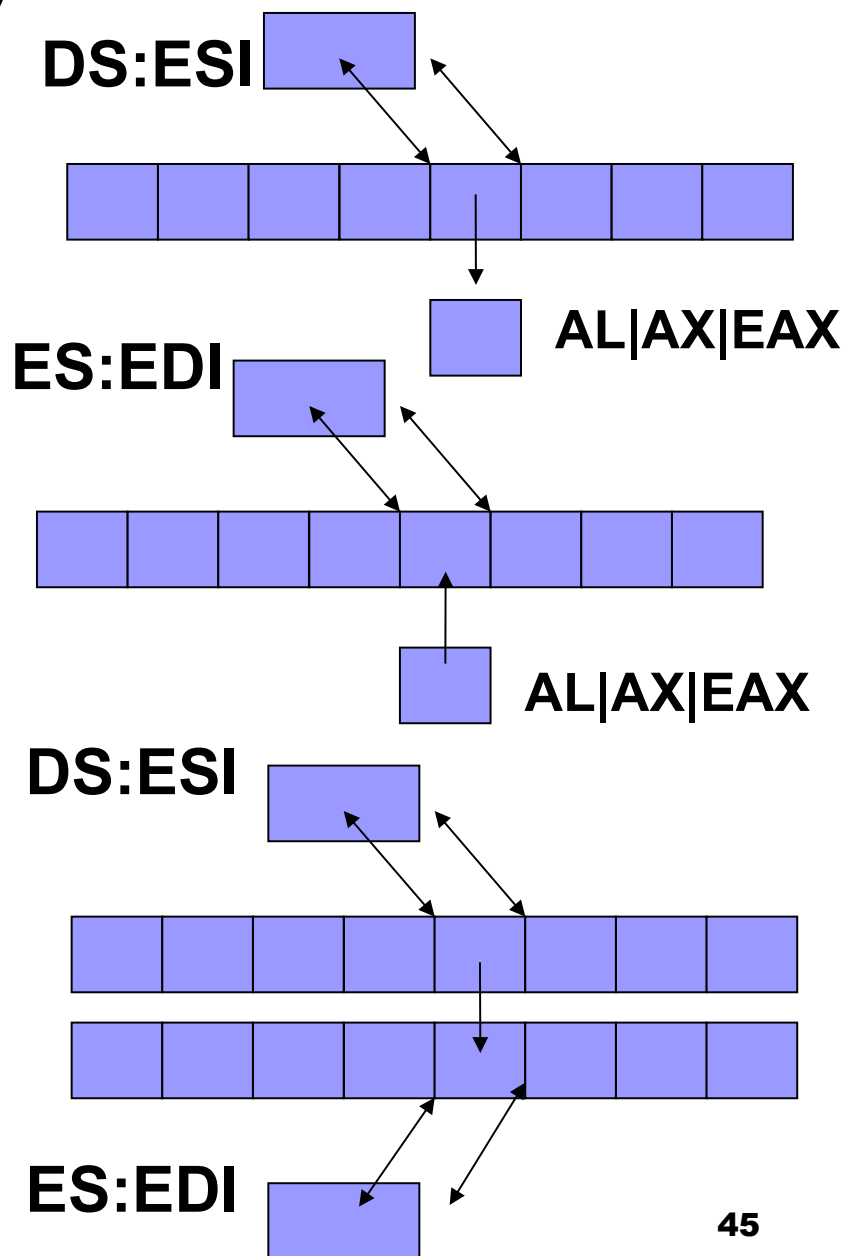
STOSD ; запись дв. слова

3. Команда пересылки **MOVS**.

MOVSB ; пересылка байта

MOVSW ; пересылки слова

MOVSD ; пересылки дв. слова



Команды обработки строк (3)

4. Префиксная команда повторения

REP {LODS | STOS | MOVS}

Пример:

.DATA

A BYTE "ABCDEFGFSRTQ"

.DATA?

B BYTE 10 DUP (?)

.CODE

Start:

cld ; сброс флага направления

mov ECX,10

lea ESI,A

lea EDI,B

rep movsb ; копирование строки из 10 символов

...

Команды обработки строк (4)

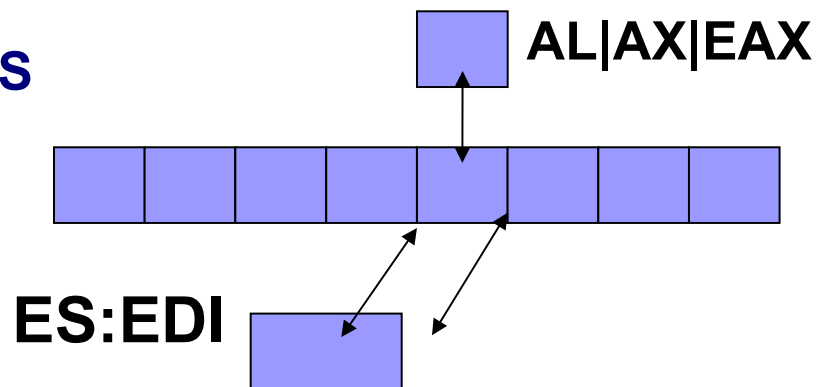
5. Команда сканирования строки **SCAS**

SCASB ;поиск байта

SCASW ;поиск слова

SCASD ;поиск дв. слова

AL|AX|EAX - (ES:EDI) -> флаги



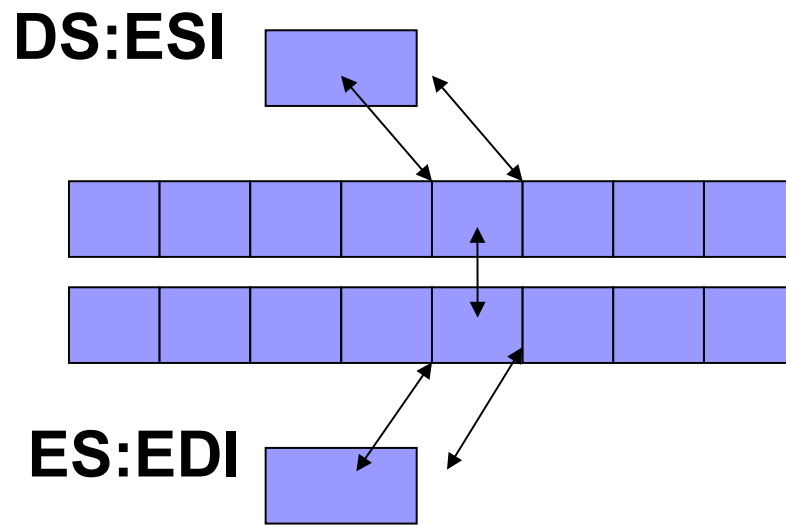
6. Команда сравнения строк **CMPS**

CMPSB ;сравнение байт

CMPSW ;сравнение слов

CMPSD ;сравнение дв. слов

(DS:ESI)-(ES:EDI) -> флаги



Команды обработки строк (5)

7. Префиксные команды "повторять, пока равно" и "повторять, пока не равно"

```
REPE    {SCAS | CMPS}  
REPNE   {SCAS | CMPS}
```

Пример:

```
.DATA
```

```
A      BYTE  "ABCDEFGSRTQ"
```

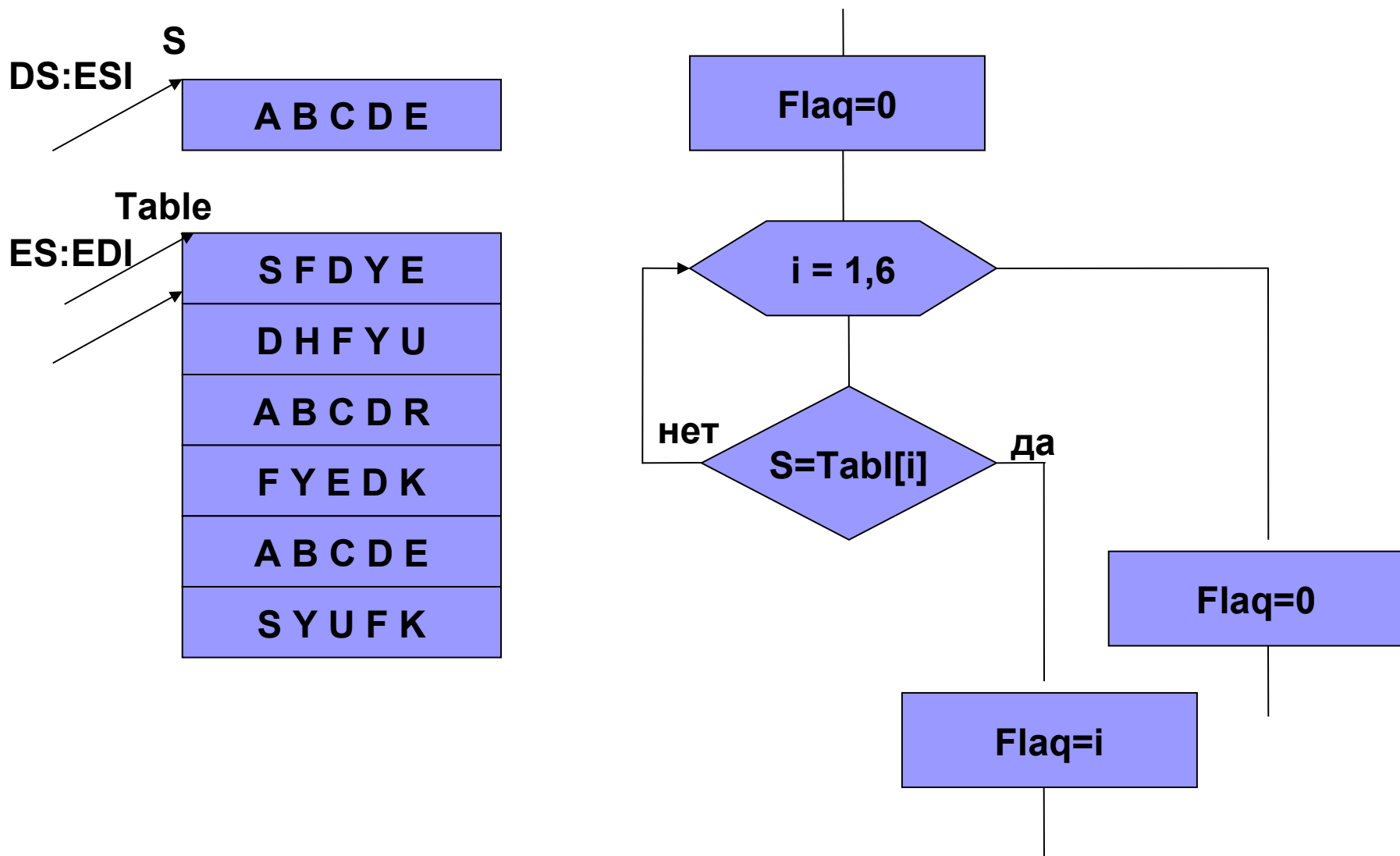
```
B      BYTE  "ABCDEFGSRTQ"
```

```
.CODE
```

Start:

```
cld          ; сброс флага направления  
mov    ECX,10  
lea    ESI,A  
lea    EDI,B  
repe   cmpsb ; сравнение строк из 10 символов
```

Пример 2.9 Поиск строки в таблице



Поиск строки в таблице (2)

.DATA

Flag	BYTE	0
S	BYTE	'ABCDE'
Table	BYTE	'ARTYG'
	BYTE	'FGJJU'
	BYTE	'FGHJK'
	BYTE	'ABCDY'
	BYTE	'ABCDE'
	BYTE	'FTYRG'

.CODE

Start:

```
lea    ESI,S
lea    EDI,Table
mov     ECX,6
mov     BL,1
cld
```


Поиск строки в таблице (3)

```
→ Cycle:  push    ESI
          push    EDI
          push    ECX
          mov     ECX, 5
          repe    cmpsb
          pop     ECX
          pop     EDI
          pop     ESI
          je      Found
          add     EDI, 5
          inc     BL
          loop    Cycle
          jmp     Not_Found
Found:    mov     Flag, BL
Not_Found: . . .
```

2.6 Команды манипулирования битами

1. Логические команды

NOT <Операнд> ; *логическое НЕ*

AND <Операнд 1>, <Операнд 2> ; *логическое И*

OR <Операнд 1>, <Операнд 2> ; *логическое ИЛИ*

XOR <Операнд 1>, <Операнд 2> ; *исключающее ИЛИ*

TEST <Операнд 1>, <Операнд 2> ; *И без записи результата*

Операции
выполняются
поразрядно

Пример. Выделить из числа в AL первый бит:

```
and    al, 10000000B
```

```
10110001
10000000
-----
10000000
```

Команды манипулирования битами (2)

2. Команды сдвига

<Код операции> <Операнд>, {CL | 1}

Мнемоника команд сдвига:

SAL – сдвиг влево арифметический;

SHL – сдвиг влево логический;

SAR – сдвиг вправо арифметический;

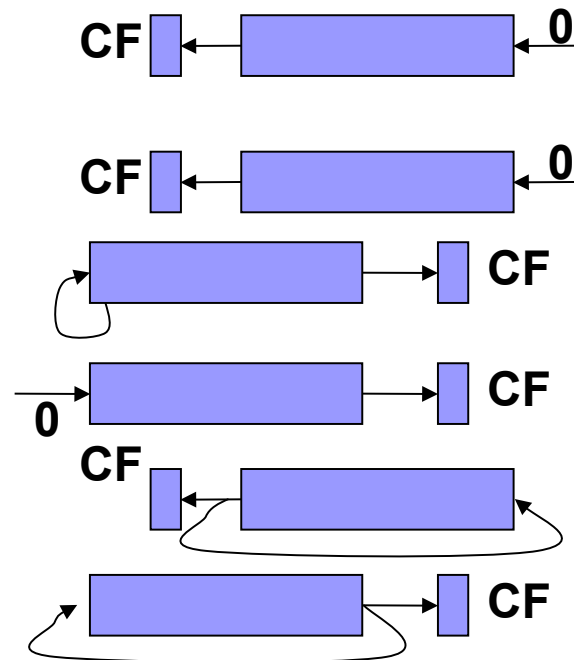
SHR – сдвиг вправо логический;

ROL – сдвиг влево циклический;

ROR – сдвиг вправо циклический;

RCL – сдвиг циклический влево с флагом переноса;

RCR – сдвиг циклический вправо с флагом переноса



Команды манипулирования битами (3)

Пример. Умножить число в AX на 10:

```
mov     BX,  AX
shl     AX,  1
shl     AX,  1
add     AX,  BX
shl     AX,  1
```

