

## Введение в язык Ассемблер

### План лекции:

регистр флагов;  
команды условного перехода;  
команды сравнения;  
пример программы сравнения двух строк.

### 1. Регистр флагов **EFLAGS**:

- ✓ флаги регистра **EFLAGS** устанавливают инструкции процессора или специальные команды;
- ✓ непосредственно регистр не доступен программисту;
- ✓ программист может проверять состояние флагов.

#### **Флаг установлен:**

значение соответствующего ему бита регистра **EFLAGS** равно 1.

#### **Флаг сброшен:**

значение соответствующего ему бита регистра **EFLAGS** равно 0.

Регистр флагов **EFLAGS** – это 32-разрядный регистр.

Старшие 16 разрядов используются при работе в защищённом режиме, и мы их рассматривать не будем.

К младшим 16 разрядам этого регистра можно обращаться как к отдельному регистру с именем **FLAGS**.

Флаги, находящиеся в младших 16 разрядах регистра **EFLAGS**:

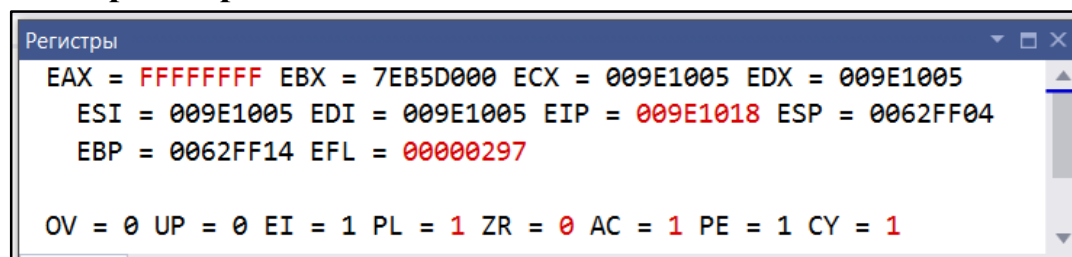
15								7 0							
0										0		0		1	
	<b>NT</b>	<b>IOPL</b>		<b>OF</b>	<b>DF</b>	<b>IF</b>	<b>TF</b>	<b>SF</b>	<b>ZF</b>		<b>AF</b>		<b>PF</b>		<b>CF</b>
x	x	x		s	c	x	x	s	s		s		s		s

s – состояние; x – системный; c – управляющий.

Все неиспользуемые биты помечены серым цветом и равны нулю за исключением 1-го бита, который всегда равен единице.

Бит	Обозначение Intel	Название	Описание	Обозначение в окне Registers VS
0	CF	Carry Flag	<b>Флаг переноса.</b> Устанавливается в 1, когда арифметическая операция генерирует перенос или выход за разрядную сетку результата. Сбрасывается в 0 в противном случае.	CY
1	1		Зарезервирован	
2	PF	Parity Flag	<b>Флаг чётности.</b> Устанавливается в 1, если результат последней операции имеет четное число единиц.	PE
3	0		Зарезервирован	
4	AF	Auxiliary Carry Flag	<b>Вспомогательный флаг переноса.</b> Устанавливается в 1, если арифметическая операция генерирует перенос из 3 бита в 4. Сбрасывается в 0 в противном случае. Этот флаг используется в двоично-десятичной арифметике.	AC
5	0		Зарезервирован	
6	ZF	Zero Flag	<b>Флаг нуля.</b> Устанавливается в 1, если результат нулевой. Сбрасывается в 0 в противном случае.	ZR
7	SF	Sign Flag	<b>Флаг знака.</b> Устанавливается равным старшему биту результата, который определяет знак в знаковых целочисленных операциях (0 – положительное число, 1 – отрицательное число).	PL
8	TF	Trap Flag	<b>Флаг трассировки</b> (пошаговое выполнение).	
9	IF	Interrupt Enable Flag	<b>Флаг разрешения прерываний.</b> При значении 1 микропроцессор реагирует на внешние аппаратные прерывания.	EI
10	DF	Direction Flag	<b>Флаг направления.</b>	UP
11	OF	Overflow Flag	<b>Флаг переполнения.</b> Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде. Этот флаг показывает наличие переполнения в знаковой целочисленной арифметике.	OV
12	IOPL	I/O Privilege Level	<b>Уровень приоритета ввода-вывода.</b>	
13				
14	NT	Nested Task	<b>Флаг вложенности задач.</b>	
15	0		Зарезервирован	

### Окно регистров отладчика VS:



## 2. Команды условного перехода:

Синтаксис команды условного перехода:

J<условие>

<метка\_перехода>

Команда условного перехода передает управление по указанной метке, если установлен соответствующий флаг состояния процессора.

Если флаг сброшен, то выполняется следующая за ней команда.

### 2.1 Команды перехода в зависимости от значения флагов состояния процессора:

**Флаг нуля:**

ZF (Zero Flag) Устанавливается в 1, если результат нулевой. (ZR)

Сбрасывается в 0 в противном случае.

Команда	Описание	Состояние флага
JZ	Переход по метке, если флаг нуля установлен	ZF=1
JNZ	Переход по метке, если флаг нуля сброшен	ZF=0

В противном случае выполняется команда, следующая за этой.

Регистры

EAX = FFFFFFFF EBX = 7EB5D000 ECX = 009E1005 EDX = 009E1005  
ESI = 009E1005 EDI = 009E1005 EIP = 009E1018 ESP = 0062FF04  
EBP = 0062FF14 EFL = 00000297

OV = 0 UP = 0 EI = 1 PL = 1 ZR = 0 AC = 1 PE = 1 CY = 1

120 %

```
6 ExitProc ; точка входа main
7 .START mov eax, 24 ; 24 десятичное -> eax
8 sub eax, 25 ; eax - 25 -> eax
9 .CONST
10 .DATA
11 .CODE
12
13 main PROC ; точка входа main
14 mov eax, 24 ; 24 десятичное -> eax
15 sub eax, 25 ; eax - 25 -> eax
16 jz zf1 ; if zf = 1 goto zf1
17 jnz zf0 ; if zf = 0 goto zf0
18 zf0:
19 mov ebx, 0 ; 0 -> ebx
20 jmp fin ; goto fin
21 zf1:
22 mov ebx, 1 ; 1 -> ebx
23 fin:
24 push 0 ; код возврата процесса Windows(параметр ExitProcess)
25 call ExitProcess ; так завершается любой процесс Windows
26 main ENDP ; конец процедуры
27
28 end main ; конец модуля main
```

Контрольные значения 1

Имя	Значение
ebx	0x00000000

```

6 ExitProc
7 .STACK EAX = 00000000 EBX = 7F55F000 ECX = 00951005 EDX = 00951005
8       ESI = 00951005 EDI = 00951005 EIP = 00951023 ESP = 002FFDB8
9 .CONST EBP = 002FFDC8 EFL = 00000246
10 .DATA
11 .CODE OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
12
13 main PROC      ; точка входа main
14     mov eax, 24 ; 24 десятичное -> eax
15     sub eax, 24 ; eax - 24 -> eax
16     jz  zf1     ; if zf = 1 goto zf1
17     jnz zf0     ; if zf = 0 goto zf0
18 zf0:
19     mov ebx, 0  ; 0 -> ebx
20     jmp fin     ; goto fin
21 zf1:
22     mov ebx, 1  ; 1 -> ebx
23 fin:
24     push 0      ; код возврата процесса Windows(параметр ExitProc
25     call ExitProcess ; так завершается любой процесс Windows
26 main ENDP      ; конец процедуры
27
28 end main       ; конец модуля main

```

Регистры

EAX = 00000000 EBX = 7F55F000 ECX = 00951005 EDX = 00951005  
ESI = 00951005 EDI = 00951005 EIP = 00951023 ESP = 002FFDB8  
EBP = 002FFDC8 EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

Контрольные значения 1

Имя	Значение
ebx	0x00000001

```

4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 ExitProc
7 .STACK EAX = 15C50000 EBX = 7F1F5000 ECX = 00111005 EDX = 00111005
8       ESI = 00111005 EDI = 00111005 EIP = 00111023 ESP = 00D7FB70
9 .CONST EBP = 00D7FB80 EFL = 00000246
10 .DATA
11 .CODE OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
12
13 main PROC      ; точка входа main
14     mov ax, 24  ; 24 десятичное -> ax
15     and ax, 111111100111b ; ax = 0000
16     jz  zf1     ; if zf = 1 goto zf1
17     jnz zf0     ; if zf = 0 goto zf0
18 zf0:
19     mov ebx, 0  ; 0 -> ebx
20     jmp fin     ; goto fin
21 zf1:
22     mov ebx, 1  ; 1 -> ebx
23 fin:
24     push 0      ; код возврата процесса Windows(параметр ExitProc
25     call ExitProcess ; так завершается любой процесс Windows
26 main ENDP      ; конец процедуры
27 end main       ; конец модуля main

```

Регистры

EAX = 15C50000 EBX = 7F1F5000 ECX = 00111005 EDX = 00111005  
ESI = 00111005 EDI = 00111005 EIP = 00111023 ESP = 00D7FB70  
EBP = 00D7FB80 EFL = 00000246

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0

Контрольные значения 1

Имя	Значение
ebx	0x00000001
ax	0x0000

## 2.2 Команды перехода в зависимости от значения флагов состояния процессора:

### Флаг знака:

SF (Sign Flag) Устанавливается равным старшему биту результата, который определяет знак в знаковых целочисленных операциях (0 – положительное число, 1 – отрицательное число).

Команда	Описание	Состояние
JS	переход, если флаг знака установлен	SF=1
JNS	переход, если флаг знака сброшен	SF =0

The screenshot shows an assembly editor with the following code:

```
4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 ExitProc
7 .START
8     EAX = 96DC8018 EBX = 00000001 ECX = 00021005 EDX = 00021005
9     ESI = 00021005 EDI = 00021005 EIP = 00021028 ESP = 004EFF04
10    EBP = 004EFF14 EFL = 00000286
11    OV = 0 UP = 0 EI = 1 PL = 1 ZR = 0 AC = 0 PE = 1 CY = 0
12
13 main PROC ; точка входа main
14     mov ax, 24 ; 24 десятичное -> ax
15     or ax, 1000000000000000b ; SF (PL) = 1
16     js zs1 ; if sf = 1 goto sf1
17     jns zs0 ; if sf = 0 goto sf0
18 zs0:
19     mov ebx, 0 ; 0 -> ebx
20     jmp fin ; goto fin
21 zs1:
22     mov ebx, 1 ; 1 -> ebx
23 fin:
24     push 0 ; код возврата процесса Windows(параметр ExitProc)
25     call ExitProcess ; так завершается любой процесс Windows
26 main ENDP ; конец процедуры
27 end main ; конец модуля main
```

Two windows are overlaid on the code:

- Регистры**: A window showing the state of registers. The 'PL' flag is highlighted with a red box and set to 1.
- Контрольные значения 1**: A window showing the values of variables. The 'ebx' variable is highlighted with a red box and set to 0x00000001.

```

.code                                ; сегмент кода
main PROC                            ; начало процедуры
mov ax, 24
sub ax, 24                           ; zs = 0
jz  zs1                               ; if zs = 1 goto zs1
jns zs0                              ; if zs = 0 goto zs0
zs0:
mov ebx, 0
jmp fin
zs1:
mov ebx, 1
fin:
push 0                               ; код возврата процесса
call ExitProcess                     ; так должен заканчива
main ENDP                            ; конец процедуры

end main                             ; конец модуля, main -

```

ebx	0x00000000
ax	0x0000

## 2.3 Команды перехода в зависимости от значения флагов состояния процессора:

### Флаг чётности:

PF (Parity Flag) Устанавливается в 1, если результат последней операции (PE) имеет четное число единиц.

Команда	Описание	Состояние
JP	переход, если флаг четности установлен	PF=1
JNP	переход, если флаг четности сброшен	PF =0

The screenshot shows an assembly editor with the following code:

```
4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 ExitProc
7 .STACK 1000h
8
9 .CONST
10 .DATA
11 .CODE
12
13 main PROC
14     mov ax, 0h ; 24 десятичное -> ax
15     add ax, 3h ; PF (PE) = 1
16     jp pf1 ; if pf = 1 goto pf1
17     jnp pf0 ; if pf = 0 goto pf0
18 pf0:
19     mov ebx, 0 ; 0 -> ebx
20     jmp fin ; goto fin
21 pf1:
22     mov ebx, 1 ; 1 -> ebx
23 fin:
24     push 0 ; код возврата процесса Windows(параметр ExitProc)
25     call ExitProcess ; так завершается любой процесс Windows
26 main ENDP ; конец процедуры
27 end main ; конец модуля main
```

Two windows are overlaid on the code:

- Регистры**: Shows register values. EAX = 327E0003, EBX = 7EEFE000, ECX = 00051005, EDX = 00051005, ESI = 00051005, EDI = 00051005, EIP = 00051023, ESP = 0083FD60, EBP = 0083FD70, EFL = 00000206. The PE flag is highlighted in red with the value 1.
- Контрольные значения 1**: A table showing the values of registers ebx and ax.

Имя	Значение
ebx	0x00000001
ax	0x0003



The screenshot shows a Windows assembly program being debugged. The assembly code is as follows:

```

main PROC
    mov ax, 1h          ; 24 десятичное -> ax
    add ax, 3h          ; PF (PE) = 0
    jp pf1              ; if pf = 1
    jnp pf0             ; if pf = 0
pf0:
    mov ebx, 0          ; 0 -> ebx
    jmp fin
pf1:
    mov ebx, 1          ; 1 -> ebx
fin:
    push 0              ; код возврата процесса Windows(параметр ExitProc
    call ExitProcess    ; так завершается любой процесс Windows
main ENDP

```

The CPU registers window is open, showing the following values:

Register	Value
EAX	2D0D0004
EBX	7E73F000
ECX	000B1005
EDX	000B1005
ESI	000B1005
EDI	000B1005
EIP	000B1018
ESP	005AFEC4
EBP	005AFED4
EFL	00000202
OV	0
UP	0
EI	1
PL	0
ZR	0
AC	0
PE	0
CY	0

The 'PF' (Parity Flag) is highlighted in red in the registers window, indicating its value is 0. The assembly code also includes a comment: ; PF (PE) = 0.

## 2.4 Команды перехода в зависимости от значения флагов состояния процессора:

### Флаг переполнения:

OF (Overflow Flag) Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде. Этот флаг показывает наличие переполнения в знаковой целочисленной арифметике.

Команда	Описание	Состояние
JO	переход, если возникло переполнение	OF=1
JNO	переход, если переполнения нет	OF =0

The screenshot shows an assembly editor with the following code:

```
4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 ExitProc
7 .START
8     EAX = CCD19C80 EBX = 7F2A6000 ECX = 00801005 EDX = 00801005
9     ESI = 00801005 EDI = 00801005 EIP = 00801014 ESP = 010EFD80
10    EBP = 010EFD90 EFL = 00000A92
11    .CODE
12
13    main PROC ; точка входа main
14        mov al, 7fh ;
15        add al, 1h ; OF (OV) = 1
16        jo of1 ; if of = 1 goto of1 ≤ 1 мс прошло
17        jno of0 ; if of = 0 goto of0
18    of0:
19        mov ebx, 0 ; 0 -> ebx
20        jmp fin ; goto fin
21    of1:
22        mov ebx, 1 ; 1 -> ebx
23    fin:
24        push 0 ; код возврата процесса Windows(параметр ExitProc)
25        call ExitProcess ; так завершается любой процесс Windows
26    main ENDP ; конец процедуры
27    end main ; конец модуля main
```

Two windows are overlaid on the code:

- Регистры**: Shows the state of registers. The **OV** flag is highlighted with a red box and set to **1**. Other flags shown are UP = 0, EI = 1, PL = 1, ZR = 0, AC = 1, PE = 0, CY = 0.
- Контрольные значения 1**: A table showing the values of control registers. The **ebx** register is highlighted with a red box and set to **0x00000001**. The **ax** register is set to **0x9c80**.

```

4  .MODEL FLAT, STDCALL          ; модель памяти, соглашение о вызовах
5  includelib kernel32.lib      ; компоновщик: компоновать с kernel32
6  ExitProcess                  ;
7  .STARTUP                     EAX = 673AEC7F EBX = 7F0D8000 ECX = 008E1005 EDX = 008E1005
8                               ESI = 008E1005 EDI = 008E1005 EIP = 008E1018 ESP = 007DF8E4
9                               EBP = 007DF8F4 EFL = 00000202
10 .DATA
11 .CODE
12                               OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 0 PE = 0 CY = 0
13 main PROC                    ; точка входа main
14     mov al, 7eh               ;
15     add al, 1h                ; OF (OV) = 0
16     jo of1                    ; if of = 1 goto of1
17     jno of0                   ; if of = 0 goto of0
18 of0:
19     mov ebx, 0                ; 0 -> ebx ≤ 1 мс прошло
20     jmp fin                   ; goto fin
21 of1:
22     mov ebx, 1                ; 1 -> ebx
23 fin:
24     push 0                    ; код возврата процесса Windows(параметр ExitProcess)
25     call ExitProcess          ; так завершается любой процесс Windows
26 main ENDP                    ; конец процедуры
27 end main                     ; конец модуля main

```

Регистры

EAX = 673AEC7F EBX = 7F0D8000 ECX = 008E1005 EDX = 008E1005  
 ESI = 008E1005 EDI = 008E1005 EIP = 008E1018 ESP = 007DF8E4  
 EBP = 007DF8F4 EFL = 00000202

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 0 PE = 0 CY = 0

120 %

Контрольные значения 1

Поиск (Ctrl+E)

Имя	Значение
ebx	0x00000000
ax	0x0c7f

### 3. Команды сравнения

#### Команда TEST

Выполняет операцию поразрядного логического И между соответствующими парами битов двух операндов.

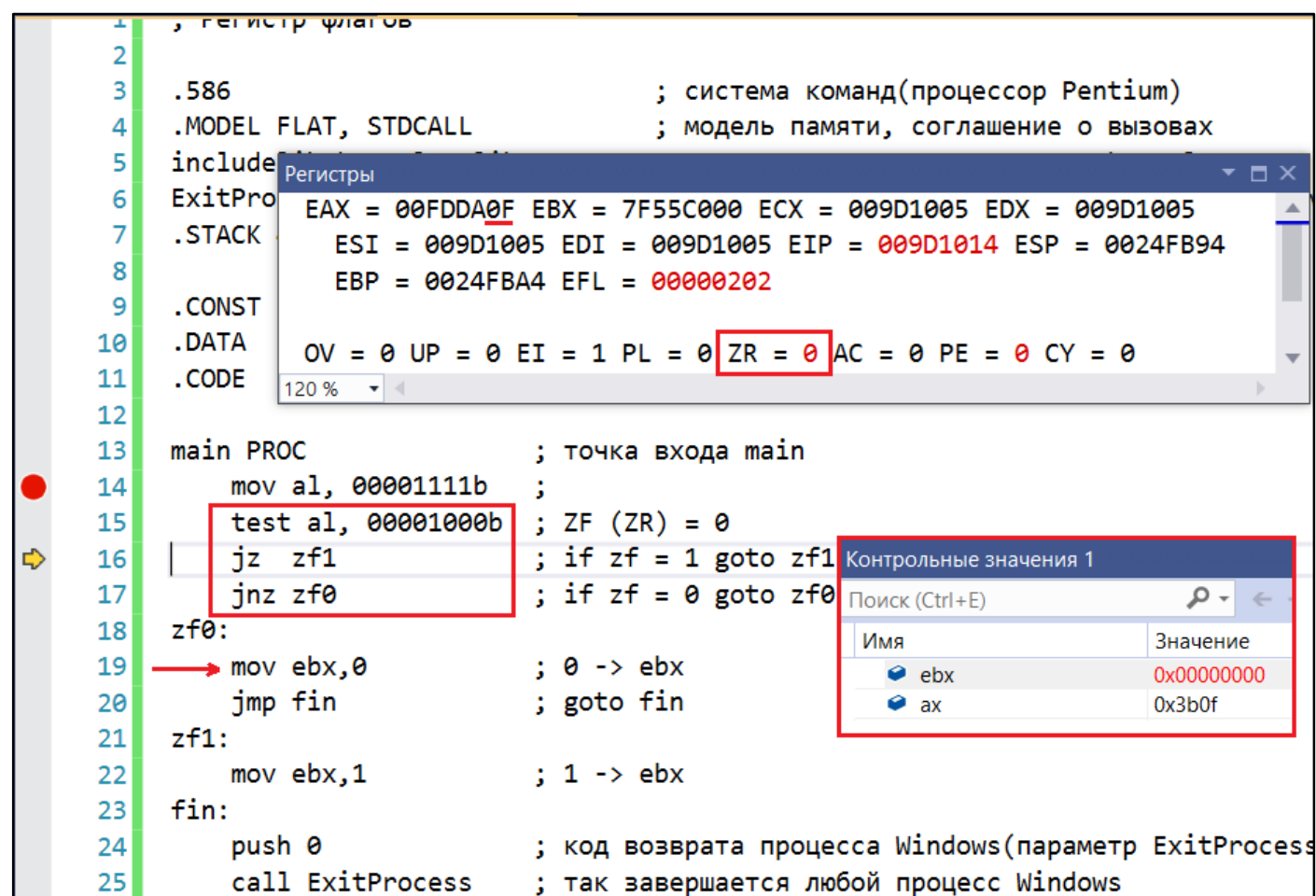
В зависимости от полученного результата устанавливает флаги состояния процессора.

Значение операнда-получателя *не изменяется*.

#### Флаг нуля:

ZF (Zero Flag) Устанавливается в 1, если результат нулевой.

(ZR) Сбрасывается в 0, если результат равен 1.



The screenshot displays an assembly editor with the following code:

```
1 ; регистр флагов
2
3 .586 ; система команд(процессор Pentium)
4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 include
6 ExitPro
7 .STACK
8
9 .CONST
10 .DATA
11 .CODE
12
13 main PROC ; точка входа main
14 mov al, 00001111b ;
15 test al, 00001000b ; ZF (ZR) = 0
16 jz zf1 ; if zf = 1 goto zf1
17 jnz zf0 ; if zf = 0 goto zf0
18 zf0:
19 mov ebx, 0 ; 0 -> ebx
20 jmp fin ; goto fin
21 zf1:
22 mov ebx, 1 ; 1 -> ebx
23 fin:
24 push 0 ; код возврата процесса Windows(параметр ExitProcess)
25 call ExitProcess ; так завершается любой процесс Windows
```

Two windows are overlaid on the code:

- Регистры (Registers):** Shows the state of various registers. The Zero Flag (ZF) is highlighted with a red box and shows a value of 0.
- Контрольные значения 1 (Control Values 1):** A table showing the values of the EBX and AX registers.

Имя	Значение
ebx	0x00000000
ax	0x3b0f

```

include110 kernel32.lib ; компоновщику. компоновать
ExitProcess PROTO :DWORD ; прототип функции
.stack 4096 ; сегмент стека объемом 4096
.const ; сегмент констант
.data ; сегмент данных

```

```

.code ; сегмент кода
main PROC ; начало процедуры
mov al, 00001111b
test al, 01100000b ; and zf = 1
jz f1 ; if zf = 1 goto f1
jnz f0 ; if zf = 0 goto f0

```

f0:

```

mov ebx, 0
jmp fin

```

f1:

```

mov ebx, 1

```

fin:

```

push 0 ; код возврата процесса (параметр)
call ExitProcess ; так должен заканчиваться процесс
main ENDP ; конец процедуры

end main ; конец модуля, main - точка входа

```

Имя	Значение
ebx	0x00000001
al	0x0f '\xf'

## Команда CMP

Команда вычитает исходный операнд из операнда-получателя и устанавливает следующие флаги:

*флаг переноса (CF);*

*флаг нуля (ZF);*

*флаг знака (SF);*

*флаг переполнения (OF);*

*флаг четности (PF);*

*флаг служебного переноса (AF).*

Значение операнда-получателя не изменяется.

### Флаг нуля:

ZF (Zero Flag) Устанавливается в 1, если результат нулевой.

(ZR) Сбрасывается в 0, если результат равен 1.

The screenshot shows an assembly editor with the following code:

```
1 ; регистр флагов
2
3 .586 ; система команд(процессор Pentium)
4 .MODEL FLAT, STDCALL ; модель памяти, соглашение о вызовах
5 include
6 ExitPro
7 .STACK
8
9 .CONST
10 .DATA
11 .CODE
12
13 main PROC ; точка входа main
14 mov eax, 25h ;
15 mov ebx, 25h
16 cmp eax, ebx ; ZF (ZR) = 1
17 jz zf1 ; if zf = 1 goto zf1
18 jnz zf0 ; if zf = 0 goto zf0
19 zf0:
20 mov ebx, 0 ; 0 -> ebx
21 jmp fin ; goto fin
22 zf1:
23 mov ebx, 1 ; 1 -> ebx
24 fin:
25 push 0 ; код возврата процесса Windows(параметр ExitProces
```

Two windows are overlaid on the code:

- Регистры**: A window showing the state of registers and flags. The ZR flag is highlighted with a red box and set to 1.
- Контрольные значения 1**: A window showing the values of control registers. The ebx register is highlighted with a red box and set to 0x00000001.

data ; сегмент данных

```
.code ; сегмент кода
main PROC ; начало процедуры
mov eax, 25h
mov ebx, 26h
cmp eax, ebx ; and zf = 1
jz f1 ; if zf = 1 goto f1
jnz f0 ; if zf = 0 goto f0
```

```
f0:
mov ebx, 0
jmp fin
```

```
f1:
mov ebx, 1
```

```
fin:
```

Имя	Значение
ebx	0x00000000
al	0x25 '%'

```
push 0 ; код возврата процесса (параметр ExitProcess )
call ExitProcess ; так должен заканчиваться любой процесс Windows
main ENDP ; конец процедуры

end main ; конец модуля, main - точка входа
```

## 4. Команды переходов при беззнаковом CMP-сравнении чисел

### 4.1 Команды перехода в зависимости от равенства операндов или равенства нулю регистра ECX (CX)

#### Флаг нуля:

ZF (Zero Flag) Устанавливается в 1, если результат нулевой.

(ZR) Сбрасывается в 0, если результат равен 1.

Команда	Описание	Состояние
JE	переход, если равны	ZF=1
JNE	переход, если не равны	ZF=0

The screenshot displays an assembly editor with the following code snippet:

```
4 .MODEL flat, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 Exit
7 .START EAX = 00000025 EBX = 00000026 ECX = 00E01005 EDX = 00E01005
8 ESI = 00E01005 EDI = 00E01005 EIP = 00E0101E ESP = 00BBF874
9 .COMMON
10 .DATA
11 .CODE OV = 0 UP = 0 EI = 1 PL = 1 ZR = 0 AC = 1 PE = 1 CY = 1
12
13 main PROC ; точка входа main
14 mov eax, 25h ;
15 mov ebx, 26h ;
16 cmp eax, ebx ; ZF (ZR) = 0
17 je zf1 ; if zf = 1 goto zf1
18 jne zf0 ; if zf = 0 goto zf0
19 zf0:
20 mov ebx, 0 ; 0 -> ebx
21 jmp fin ; goto fin
22 zf1:
23 mov ebx, 1 ; 1 -> ebx
24 fin:
25 push 0 ; код возврата процесса Windows (параметр ExitPro
26 call ExitProcess ; так завершается любой процесс Windows
27 main ENDP ; конец процедуры
28 end main ; конец модуля main
```

Two windows are overlaid on the code:

- Регистры**: A window showing the state of registers. The ZR flag is highlighted with a red box and set to 0.
- Контрольные значения 1**: A window showing the values of variables. The 'ebx' variable is highlighted with a red box and set to 0x00000000. The 'ax' variable is set to 0x0025.



## 4.2 Команды перехода в зависимости от равенства беззнаковых операндов

Команда	Описание	Состояние
JA	переход, если выше, т.е. левый операнд > правого операнда	ZF=0
JB	переход, если ниже, т.е. левый операнд < правого операнда	ZF=0

### Флаг нуля:

ZF (Zero Flag) Устанавливается в 1, если результат нулевой.  
Сбрасывается в 0, если результат равен 1.

The screenshot displays an assembly editor window with the following code:

```

4  .MODEL TEST, STDCALL
5  includelib kernel32.lib
6  Exit
7  .START EAX = FFFFFFF38 EBX = 00000000 ECX = 00FC1005 EDX = 00FC1005
8        ESI = 00FC1005 EDI = 00FC1005 EIP = 00FC1021 ESP = 0097F9C8
9  .COMMON EBP = 0097F9D8 EFL = 00000286
10 .DATA
11 .CODE
12
13 main PROC
14     mov eax, -200
15     cmp eax, 100
16     ja fa
17     jb fb
18 fa:
19     mov ebx, 0
20     jmp fin
21 fb:
22     mov ebx, 1
23 fin:
24     push 0
25     call ExitProcess
26 main ENDP
27 end main

```

The 'Registers' window shows the following values:

Регистры	Значение
EAX	FFFFFFF38
EBX	00000000
ECX	00FC1005
EDX	00FC1005
ESI	00FC1005
EDI	00FC1005
EIP	00FC1021
ESP	0097F9C8
EBP	0097F9D8
EFL	00000286

The 'Control Values' window shows the following values:

Имя	Значение
ebx	0x00000000
ax	0xff38

### 4.3 Команды перехода в зависимости от равенства беззнаковых операндов

Команда	Описание	Состояние
JAE	переход, если выше или равно, т.е. левый операнд $\geq$ правого операнда	ZR=0
JBE	переход, если ниже или равно, т.е. левый операнд $\leq$ правого операнда	ZR=0

#### Флаг нуля:

ZF (Zero Flag) Устанавливается в 1, если результат нулевой.  
Сбрасывается в 0, если результат равен 1.

The screenshot displays a debugger window with assembly code and two floating windows showing register and control values.

**Assembly Code:**

```

5  includelib kernel32.lib
6  Exit
7  .STA  EAX = FFFFFFF38 EBX = FFFFFFFCE0 ECX = 008C1005 EDX = 008C1005
8      ESI = 008C1005 EDI = 008C1005 EIP = 008C101C ESP = 00EAFE0C
9  .COM  EBP = 00EAFE1C EFL = 00000202
10 .DAT
11 .COD  OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 0 PE = 0 CY = 0
12
13 main PROC
14     mov eax, -200
15     mov ebx, -800
16     cmp eax, ebx
17     jae fa
18     jbe fb
19 fa:
20     mov ebx, 0
21     jmp fin
22 fb:
23     mov ebx, 1
24 fin:
25     push 0
26     call ExitProcess
27 main ENDP
28 end main

```

**Регистры (Registers):**

EAX	FFFFFFFF38
EBX	FFFFFFCE0
ECX	008C1005
EDX	008C1005
ESI	008C1005
EDI	008C1005
EIP	008C101C
ESP	00EAFE0C
EBP	00EAFE1C
EFL	00000202

**Контрольные значения 1 (Control Values 1):**

Имя	Значение
ebx	0x00000000
ax	0xff38

## 5. Команды переходов при CMP-сравнении чисел со знаком

### 5.1 Команды перехода после выполнения команд сравнения операндов со знаком

Команда	Описание
JG	переход, если больше, т.е. левый операнд > правого операнда
JL	переход, если меньше, т.е. левый операнд < правого операнда

The screenshot displays a debugger window with assembly code on the left and two floating windows on the right. The assembly code is for a `main` procedure. Line 16, `cmp eax, ebx`, is highlighted with a red box. Below it, lines 17 and 18, `jg fa` and `jnl fb`, are also highlighted with a red box. A red arrow points to line 20, `mov ebx, 0`. The `Регистры` (Registers) window shows the current state of registers: `EAX = FFFFFFF38`, `EBX = FFFFFFFCE0`, `ECX = 00C31005`, `EDX = 00C31005`, `ESI = 00C31005`, `EDI = 00C31005`, `EIP = 00C3101C`, `ESP = 0036F8DC`, `EBP = 0036F8EC`, `EFL = 00000202`. The `ZR` flag is highlighted with a red box and shows a value of 0. The `Контрольные значения 1` (Control Values 1) window shows the values of `ebx` as `0x00000000` and `ax` as `0xff38`.

```
4 .MODEL TEST, STDCALL ; модель памяти, соглашение о вызовах
5 includelib kernel32.lib ; компоновщик: компоновать с kernel32
6 Exit
7 .START EAX = FFFFFFF38 EBX = FFFFFFFCE0 ECX = 00C31005 EDX = 00C31005
8 ESI = 00C31005 EDI = 00C31005 EIP = 00C3101C ESP = 0036F8DC
9 .COMMON EBP = 0036F8EC EFL = 00000202
10 .DATA
11 .CODE OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 0 PE = 0 CY = 0
12
13 main PROC ; точка входа main
14 mov eax, -200 ;
15 mov ebx, -800 ;
16 cmp eax, ebx ; ZF (ZR) = 0
17 jg fa ; if eax > ebx goto fa
18 jnl fb ; if eax < ebx goto fa
19 fa:
20 mov ebx, 0 ; 0 -> ebx
21 jmp fin ; goto fin
22 fb:
23 mov ebx, 1 ; 1 -> ebx
24 fin:
25 push 0 ; код возврата процесса Windows(параметр ExitProc
26 call ExitProcess ; так завершается любой процесс Windows
27 main ENDP ; конец процедуры
28 end main ; конец модуля main
```

## 5.2 Команды перехода после выполнения команд сравнения операндов со знаком

Команда	Описание
JGE	переход, если больше или равно, т.е. левый операнд $\geq$ правого операнда
JLE	переход, если меньше или равно, т.е. левый операнд $\leq$ правого операнда

```

ddx dd 800
.code                               ; сегмент кода
main PROC                           ; начало процедуры
mov eax, -200
cmp eax, ddx                         ;
jge fa                               ; if eax >= ddx goto fa
jle fb                               ; if eax <= ddx goto fb
fa:
mov ebx, 1
jmp fin
fb:
mov ebx, 0
fin:
push 0                               ; код возврата процесса (п
call ExitProcess                     ; так должен заканчиваться
main ENDP                           ; конец процедуры

end main                             ; конец модуля, main - то

```

Имя	Значение
ebx	0x00000000
al	0x38 '8'

## 6. Пример программы сравнения двух строк

```
.stack 4096                ; сегмент стека объемом 4096
.const                     ; сегмент констант
.data                     ; сегмент данных
hw byte "Hello, World!!!"
pm byte "Привет, Мир!!!"

.code                     ; сегмент кода
main PROC                 ; начало процедуры

    mov ecx, sizeof hw
    cmp ecx, sizeof pm
    je mje                ; if sizeof hw == sizeof pm
    ja mhw                ; if sizeof hw > sizeof pm

mpm:                      ; hw < pm
    mov ebx, -1
    jmp fin

mje:                      ; sizeof hw == sizeof pm
    mov esi, 0
loopmje:
    mov al, hw[esi]
    cmp al, pm[esi]
    ja mhw
    jb mpm
    add esi, 1
    loop loopmje
    mov ebx, 0             ; hw = pm
    jmp fin

mhw:                      ; hw > pm
    mov ebx, 1

fin:
    push 0                ; код возврата процесса (параметр)
    call ExitProcess       ; так должен заканчиваться любой процесс
    main ENDP              ; конец процедуры

end main                  ; конец модуля, main - точка входа
```

## 7. Команды проверки и установки отдельных битов

Команды BT, BTR, BTC и BTS используются для работы с отдельными битами. Команды используют для организации семафоров.

Синтаксис команды тестирования бита:

BT	строка_битов, n
----	-----------------

### Флаг переноса:

CF (Carry Flag) Устанавливается в 1, когда арифметическая операция генерирует перенос или выход за разрядную сетку результата. Сбрасывается в 0 в противном случае.

Команда	Описание	Состояние
JC	переход, если перенос	CF=1
JNC	переход, если нет переноса	CF=0

The screenshot shows an assembly editor with the following code:

```
5  includelib kernel32.lib ; компоновщику: компоновать с kernel32
6  Exit
7  .START
8      EAX = D91FA42B EBX = 7F57A000 ECX = 00A61005 EDX = 00A61005
9      ESI = 00A61005 EDI = 00A61005 EIP = 00A6101B ESP = 0068FE80
10     EBP = 0068FE90 EFL = 00000246
11     b1 OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 0
12     b2
13     .CODE ; сегмент кода
14
15     main PROC ; точка входа main
16         bt b2, 7 ; SF (CY) = b2[7]
17         jc yes ; SF == 1
18         mov ebx, 0 ; 0 -> ebx
19         jmp fin ; goto fin
20     yes:
21         mov ebx, 1 ; 1 -> ebx
22     fin:
23         push 0 ; код возврата процесса Windows(параметр ExitPro
24         call ExitProcess ; так завершается любой процесс Windows
25     main ENDP ; конец процедуры
26     end main ; конец модуля main
```

Two windows are overlaid on the code:

- Регистры**: A window showing register values. The 'CY' flag is highlighted with a red box and shows a value of 0.
- Контрольные значения 1**: A window showing the values of variables 'ebx' and 'b2'. 'ebx' has a value of 0x00000000, and 'b2' has a value of 0x0004. Both are highlighted with red boxes.

## Команда тестирование бита:

```

5  includelib kernel32.lib
6  Exit
7  .STA EAX = 942FF691 EBX = 7F39D000 ECX = 003F1005 EDX = 003F1005
8      ESI = 003F1005 EDI = 003F1005 EIP = 003F1019 ESP = 002DF988
9      EBP = 002DF998 EFL = 00000247
10 .DAT
11 b1  OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 1
12 b2  120 %
13 .CODE ; сегмент кода
14
15 main PROC ; точка входа main
16     bt b2, 2 ; SF (CY) = b2[7]
17     jc yes ; SF == 1
18     mov ebx, 0 ; 0 -> ebx
19     jmp fin ; goto fin
20 yes:
21     mov ebx, 1 ; 1 -> ebx
22 fin:
23     push 0 ; код возврата процесса Windows(параметр ExitPro
24     call ExitProcess ; так завершается любой процесс Windows
25 main ENDP ; конец процедуры
26 end main ; конец модуля main

```

Регистры

EAX = 942FF691 EBX = 7F39D000 ECX = 003F1005 EDX = 003F1005  
ESI = 003F1005 EDI = 003F1005 EIP = 003F1019 ESP = 002DF988  
EBP = 002DF998 EFL = 00000247

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 1

Контрольные значения 1

Имя	Значение
ebx	0x00000001
b2	0x0004

## Команда тестирование бита с инверсией:

```

5  inc
6  Exit
7  .STA EAX = 346C63CE EBX = 7E80D000 ECX = 01361005 EDX = 01361005
8      ESI = 01361005 EDI = 01361005 EIP = 01361019 ESP = 00DBF90C
9      EBP = 00DBF91C EFL = 00000247
10 .CO OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 1
11 .DA
12 b1  120 %
13 b2  dw 000000000000000100b
14 .CODE ; сегмент кода
15
16 main PROC ; точка входа main
17     btc b2, 2 ; SF (CY) = b2[2]; b2[2] 0 !b2[2]
18     jc yes ; SF == 1
19     mov ebx, 0 ; 0 -> ebx
20     jmp fin ; goto fin
21 yes:
22     mov ebx, 1 ; 1 -> ebx
23 fin:
24     push 0 ; код возврата процесса Windows(параметр ExitPro
25     call ExitProcess ; так завершается любой процесс Windows
26 main ENDP ; конец процедуры
27 end main ; конец модуля main

```

Регистры

EAX = 346C63CE EBX = 7E80D000 ECX = 01361005 EDX = 01361005  
ESI = 01361005 EDI = 01361005 EIP = 01361019 ESP = 00DBF90C  
EBP = 00DBF91C EFL = 00000247

OV = 0 UP = 0 EI = 1 PL = 0 ZR = 1 AC = 0 PE = 1 CY = 1



Контрольные значения 1

Имя	Значение
ebx	0x00000001
b2	0x0000

### Команда тестирование бита с инверсией:

```
.data                                ; сегмент данных
;      876543210
b1 dw 0000000000000000b
b2 dw 0000000000000100b
.code                                ; сегмент кода
main PROC                           ; начало процедуры
    btc b1, 2                        ; cf = bt1[2] bt1[2] = !bt1[2]
    jc yes                           ; cf == 1
    mov ebx, 0
    jmp fin
yes:
    mov ebx, 1
fin:
    push 0                           ; код возврата процесса (параметр ExitProcess )
    call ExitProcess                 ; так должен заканчиваться любой процесс Windows
    main ENDP                        ; конец процедуры

end main                             ; конец модуля, main - точка входа
```

Имя	Значение
 _ebx	0x00000000
 b1	0x0004



### Команда тестирование бита с установкой:

```
.stack 4096          ; сегмент стека объемом 4096
.const              ; сегмент констант
.data              ; сегмент данных
;                876543210
b1 dw 0000000000000000b
b2 dw 00000000000000100b
.code              ; сегмент кода
main PROC         ; начало процедуры
    bts b1, 2      ; cf = bt1[2] bt1[2] = 1
    jc yes        ; cf == 1
    mov ebx, 0
    jmp fin
yes:
    mov ebx, 1
fin:
    push 0         ; код возврата процесса (парам
    call ExitProcess ; так должен заканчиваться лю
    main ENDP      ; конец процедуры
end main          ; конец модуля, main - точка
```

ebx	0x00000000
b1	0x0004

### Команда тестирование бита со сбросом:

```
.stack 4096          ; сегмент стека объемом 4096
.const              ; сегмент констант
.data              ; сегмент данных
;                876543210
b1 dw 0000000000000000b
b2 dw 00000000000000100b
.code              ; сегмент кода
main PROC         ; начало процедуры
    btr b2, 2      ; cf = bt2[2] bt2[2] = 0
    jc yes        ; cf == 1
    mov ebx, 0
    jmp fin
yes:
    mov ebx, 1
fin:
    push 0         ; код возврата процесса (параметр Exit
    call ExitProcess ; так должен заканчиваться любой проц
    main ENDP      ; конец процедуры
end main          ; конец модуля, main - точка входа
```

ebx	0x00000001
b2	0x0000