Jump instructions

- do not change flags

Unconditional jumps

Direct jump

```
jmp label
```

```
      jmp Continue
      xor eax,eax
Continue:  xor ecx,ecx
```

displacement = the difference between the target label and EIP (may also be negative)

Machine code:

```
0040340A EB 02   jmp Continue
0040340C 33 C0   xor eax,eax
0040340E 33 C9   Continue: xor ecx,ecx
```

The processor executes the jump by adding the displacement to the current value of EIP (EIP = 0040340C + 2 = 0040340E) => EIP will point to the instruction at which the program execution shall continue.

Indirect jump

jmp register/memory

A 32-bit operand contains the offset of the instruction, at which the program execution shall continue.

- Example: Write the string by letters.

```
.data
String DB “Hello!",0Dh,0Ah,0
Adr DD ?
.code
main PROC
        mov Adr,offset Stop
        mov ecx,offset WriteLetter
        mov edx,offset String
        mov edi,0
WriteLetter: mov al,[edx+edi]
        cmp al,0
        jne Continue; conditional jump cannot be indirect
        jmp Adr; jump to Stop
Continue: call WriteChar
        inc edi
        jmp ecx; return to WriteLetter
Stop:
exit
main ENDP
```

Conditional jumps

They allow to branch program execution according to the flags ZF, CF, OF, SF and PF.

```
jcc label
```

cc … condition code

Conditional jumps must be direct.

After comparison of unsigned numbers:

Instruction	Meaning – jump if	Condition
jb jnae jc	below not (above or equal) carry	CF = 1
jae jnb jnc	above or equal not below not carry	CF = 0
jbe jna	below or equal not above	CF = 1 or ZF = 1
ja jnbe	above not (below or equal)	CF = 0 and ZF = 0

```
mov al,1
cmp al,4
```

```
   0000 0001 (= 1)
 -       100 (= 4)
(1)1111 1101
```

CF = 1

After comparison of signed numbers:

Instruction	Meaning – jump if	Condition
`jl` `jnge`	less not (greater or equal)	SF ≠ OF
`jge` `jnl`	greater or equal not less	SF = OF
`jle` `jng`	less or equal not greater	ZF = 1 or SF ≠ OF
`jg` `jnle`	greater not (less or equal)	ZF = 0 and SF = OF

```
mov al,-1
cmp al,4
```

```
  1111 1111 (= -1)
-       100 (= 4)
  ---------
  1111 1011
```

OF = 0
SF = 1

Instruction	Meaning – jump if	Condition
`je` `jz`	equal zero	ZF = 1
`jne` `jnz`	not equal not zero	ZF = 0
`jp` `jpe`	parity parity even	PF = 1
`jnp` `jpo`	not parity parity odd	PF = 0
`js`	sign	SF = 1
`jns`	not sign	SF = 0
`jo`	overlfow	OF = 1
`jno`	not overflow	OF = 0
`jcxz`	CX is 0	CX = 0
`jecxz`	ECX is 0	ECX = 0

Loop instructions

- do not change flags

loop label

- Decrements the ECX register and compares it with 0 leaving the flags unchanged. If new ECX > 0, jumps to the label. Otherwise the program execution continues with the next instruction.
- Label is at the first instruction of the loop.

- Read a natural number n $\in \langle 2; 20 \rangle$. Calculate the second, third, etc. to the nth number of the Fibonacci sequence.

  F(0) = 0

  F(1) = 1

  F(2) = 1

  F(3) = 2

  …

  F(n) = F(n-1) + F(n-2)

```
F(0) = 0; F(1) = 1;
for (i = 2; i <= n; i++)
  F(i) = F(i-1) + F(i-2);
```

```
.data
Fibonacci DW 0, 1, 19 dup(?)
.code
main PROC
      call ReadInt; eax = n
      mov ecx,eax
      dec ecx; loop is executed (n-1)-times
      mov edi,0; i-2
      mov esi,1; i-1
Next:
      mov ax,Fibonacci[2*edi]
      add ax,Fibonacci[2*esi]
      inc edi
      inc esi
      mov Fibonacci[2*esi],ax
      loop Next
exit
main ENDP
```

```
F(0) = 0; F(1) = 1; i = 1;
while (i < n) {
  i++; F(i) = F(i-1) + F(i-2);
}
```

```
.code
main PROC
      call ReadInt; eax = n
      mov ecx,eax
      mov edi,0
      mov esi,1; esi = i
Next:
      cmp esi,ecx
      jnb Stop
      mov ax,Fibonacci[2*edi]
      add ax,Fibonacci[2*esi]
      inc edi
      inc esi
      mov Fibonacci[2*esi],ax
      jmp Next
Stop: exit
main ENDP
```

```
F(0) = 0; F(1) = 1; i = 1;
do {
  i++; F(i) = F(i-1) + F(i-2);
} while (i < n);
```

```
.code
main PROC
      call ReadInt; eax = n
      mov ecx,eax
      mov edi,0
      mov esi,1; esi = i
Next:
      mov ax,Fibonacci[2*edi]
      add ax,Fibonacci[2*esi]
      inc edi
      inc esi
      mov Fibonacci[2*esi],ax
      cmp esi,ecx
      jb Next
exit
main ENDP
```

- Calculate the Hamming distance of word variables Number1 and Number2 (the number of positions at which the corresponding bits are different).

```
loope label
```

```
loopz label
```

- Decrements the ECX register and compares it with 0 leaving the flags unchanged. If new ECX > 0 and ZF = 1, jumps to the label.

```
loopne label
```

```
loopnz label
```

- Decrements the ECX register and compares it with 0 leaving the flags unchanged. If new ECX > 0 and ZF = 0, jumps to the label.

- Read characters typed on the keyboard and store them to variable `String` until Enter is pressed or `MaxNumber` characters have been typed.

```
.data
MaxNumber EQU 80
String DB MaxNumber dup(?)
.code
main PROC
       mov ecx,MaxNumber
       jecxz Stop
       mov edx,offset String
       mov edi,0
Next:  call ReadChar
       call WriteChar
       mov [edx+edi],al; store the letter to String
       inc edi
       cmp al,0Dh; Enter was typed?
       loopne Next; repeat if not
Stop:
exit
main ENDP
```