

Низькорівневе програмування та програмування мікроконтролерів

З використанням мови програмування **Assembler**
Флаги, перевірка значень та переходи

класрум іm3ozqq4

Питання

Як реалізувати конструкцію
яка є в усіх мовах
програмування
без виключення

```
if (x > 5) {  
    ...  
}  
else {  
    ...  
}
```

Нам потрібна конструкція
яка “щось” порівнює
і базуючись на “цьому”
ми будемо робити перехід

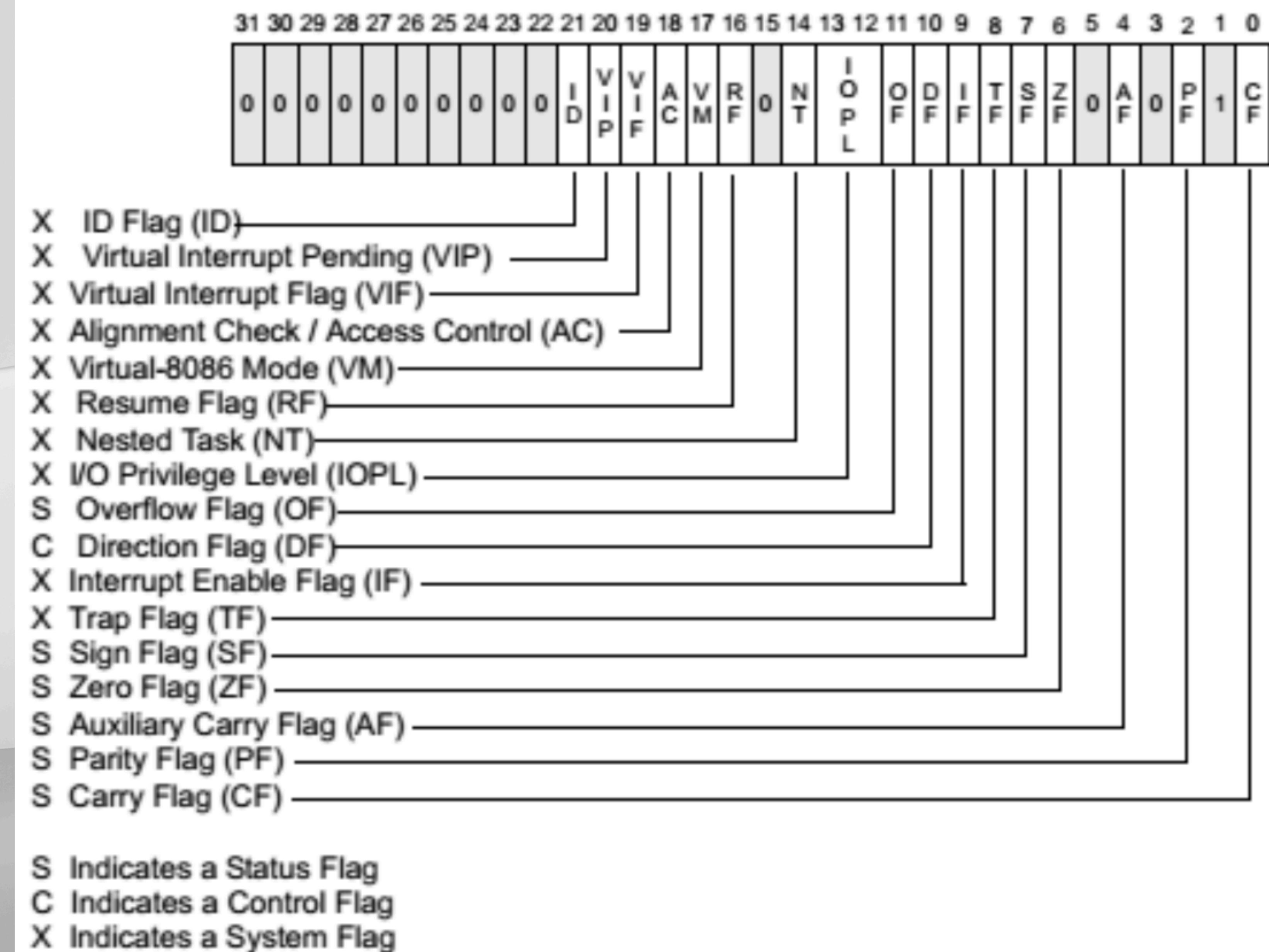
Однією з фундаментальних
операцій процесора,
є порівняння

Цією конструкцією є флаги

Як це працює?

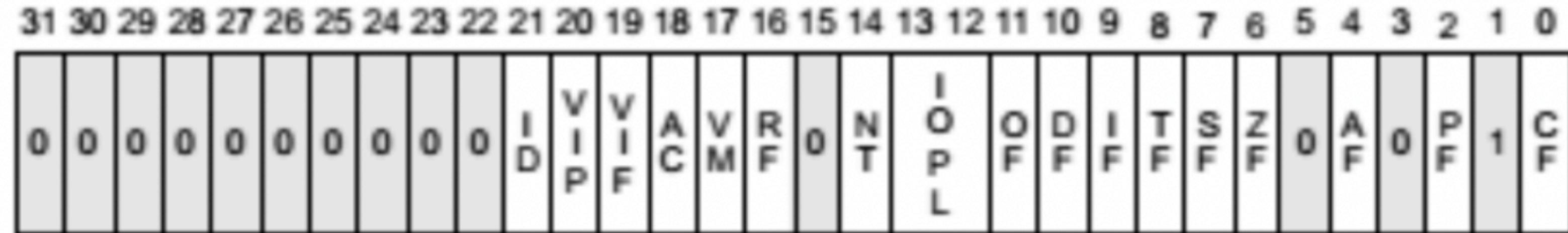
Флаги

Це набір бітів,
які
встановлюються
процесором
після виконання
деяких операцій



Флаги

Ми будемо використовувати НЕ ВСІ флаги



- X ID Flag (ID)
- X Virtual Interrupt Pending (VIP)
- X Virtual Interrupt Flag (VIF)
- X Alignment Check / Access Control (AC)
- X Virtual-8086 Mode (VM)
- X Resume Flag (RF)
- X Nested Task (NT)
- X I/O Privilege Level (IOPL)
- S Overflow Flag (OF)
- C Direction Flag (DF)
- X Interrupt Enable Flag (IF)
- X Trap Flag (TF)
- S Sign Flag (SF)
- S Zero Flag (ZF)
- S Auxiliary Carry Flag (AF)
- S Parity Flag (PF)
- S Carry Flag (CF)

Флаги, які використовуються для управління потоком виконання

CF – флаг преносу

PF – флаг парності

ZF – флаг нуля

SF – флаг знаку

DF – флаг направлення

OF – флаг переповнення

Хто встановлює флаги?

Процесор, після кожної
операції, яка щось модифікує

Arithmetic Instructions

Instruction	Description	Flags Affected
ADD dest, src	Add	CF, OF, SF, ZF, AF, PF
ADC dest, src	Add with carry	CF, OF, SF, ZF, AF, PF
SUB dest, src	Subtract	CF, OF, SF, ZF, AF, PF
SBB dest, src	Subtract with borrow	CF, OF, SF, ZF, AF, PF
INC dest	Increment	OF, SF, ZF, AF, PF (CF not changed)
DEC dest	Decrement	OF, SF, ZF, AF, PF (CF not changed)
NEG dest	Two's complement negate	CF, OF, SF, ZF, AF, PF

Multiplication & Division

Instruction	Description	Flags Affected
MUL src	Unsigned multiply	OF, CF = set if high part $\neq 0$; others undefined
IMUL src	Signed multiply	OF, CF = set if result doesn't fit; others undefined
DIV src	Unsigned divide	Flags undefined
IDIV src	Signed divide	Flags undefined

Bitwise Logic

Instruction	Description	Flags Affected
AND dest, src	Bitwise AND	SF, ZF, PF; CF=0, OF=0
OR dest, src	Bitwise OR	SF, ZF, PF; CF=0, OF=0
XOR dest, src	Bitwise XOR	SF, ZF, PF; CF=0, OF=0
NOT dest	Bitwise NOT	No flags affected

Shift & Rotate

Instruction	Description	Flags Affected
SHL/SAL dest, count	Shift left	CF=last out bit, OF=depends, SF, ZF, PF
SHR dest, count	Shift right logical	CF=last out bit, OF=MSB before shift, SF, ZF, PF
SAR dest, count	Shift right arithmetic	CF=last out bit, OF=0, SF, ZF, PF
ROL dest, count	Rotate left	CF=last out bit, OF=CF⊕MSB
ROR dest, count	Rotate right	CF=last out bit, OF=CF⊕MSB
RCL dest, count	Rotate left through CF	CF, OF updated; others undefined
RCR dest, count	Rotate right through CF	CF, OF updated; others undefined

Чи можна вручну встановлювати флаги?

Так, але не всі

Carry Flag Control

Instruction

Effect

CLC

Clear CF (Carry Flag = 0)

STC

Set CF (Carry Flag = 1)

CMC

Complement CF ($CF = \neg CF$)

Флаги можно зберігти В регистр АН

lahf



→ AX

Флаги можно загрузити з регістру AH

sahf

AX →



Окей, ми навчилися робити
з флагами що завгодно.
Як це використати?

Приклад

```
mov bl, 251
mov cl, 3
add bl, cl      ;# bl = 254
;# CF = 0
;# ZF = 0
;# SF = 0
;# PF = 1
```

Приклад

```
mov bl, 251
mov cl, 6
add bl, cl      ;# bl = 1
;# CF = 1
;# ZF = 0
;# SF = 0
;# PF = 0
```

Тобто оці флаги ми можемо використовувати як конструкцію в середині if

```
if (x > 5)
```

Залишилося розібратися що робити з

```
if (x > 5) {  
    ...  
} else {  
    ...  
}
```

Хотілося б
мати
функціонал

```
mov bl, 251  
mov cl, 3  
add bl, cl
```

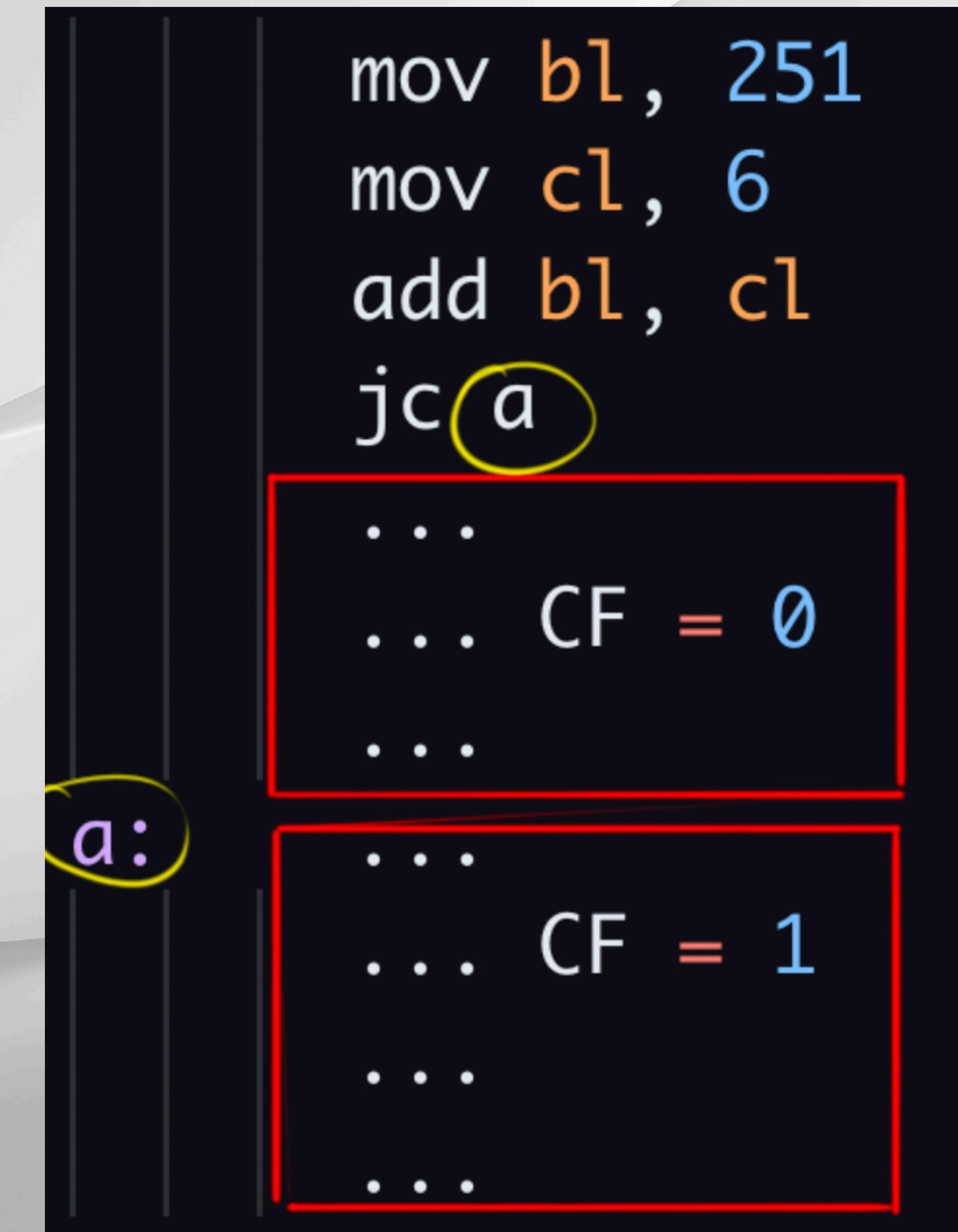
... CF = 0

... CF = 1

Синтаксис

Семантично

`jc` = Jump If CF=1



Але в машиному коді нема міток.

Мітки існують тільки в момент компіляції

Зберігається
відстань
в байтах
куди потрібно
зробити перехід

	mov bl, 251	00401000 b3fb	mov bl, 0xfb
	mov cl, 6	00401002 b106	mov cl, 6
	add bl, cl	00401004 00cb	add bl, cl
	jc a	00401006 7210	jb +16
	mov dx, 1	00401008 66 ba0100	mov dx, 1
	mov dx, 1	0040100c 66 ba0100	mov dx, 1
	mov dx, 1	00401010 66 ba0100	mov dx, 1
	mov dx, 1	00401014 66 ba0100	mov dx, 1
a:	mov cx, 1	00401018 66 b90100	mov cx, 1
	mov cx, 1	0040101c 66 b90100	mov cx, 1
	mov cx, 1	00401020 66 b90100	mov cx, 1
	mov cx, 1	00401024 66 b90100	mov cx, 1

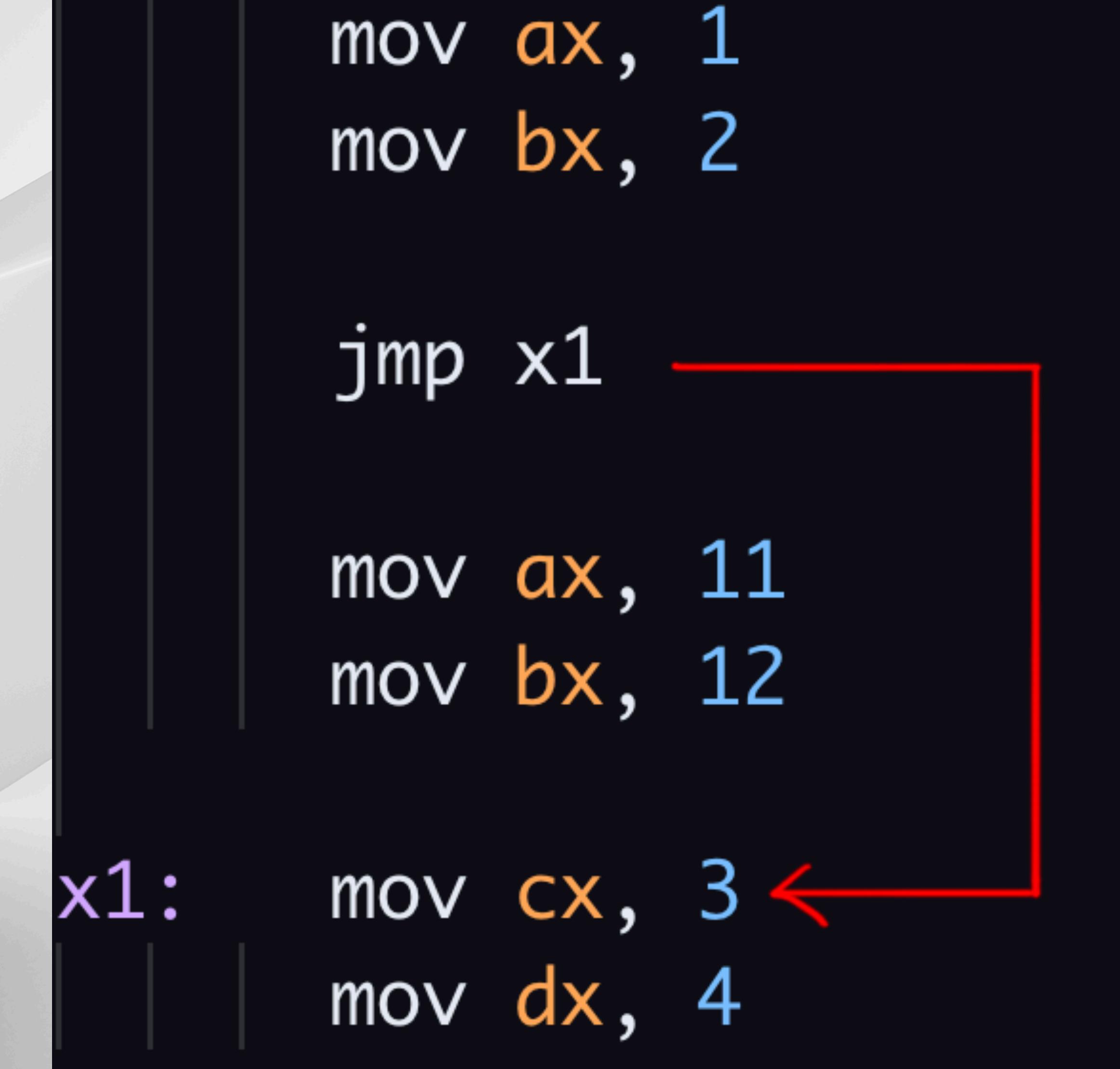
Але маємо проблему

	mov bl, 251	00401000 b3fb	mov bl, 0xfb
	mov cl, 6	00401002 b106	mov cl, 6
	add bl, cl	00401004 00cb	add bl, cl
	jc a	00401006 7210	jb +16
	mov dx, 1	00401008 66 ba0100	mov dx, 1
	mov dx, 1	0040100c 66 ba0100	mov dx, 1
	mov dx, 1	00401010 66 ba0100	mov dx, 1
	mov dx, 1	00401014 66 ba0100	mov dx, 1
a:	mov cx, 1	00401018 66 b90100	mov cx, 1
	mov cx, 1	0040101c 66 b90100	mov cx, 1
	mov cx, 1	00401020 66 b90100	mov cx, 1
	mov cx, 1	00401024 66 b90100	mov cx, 1

Рішення —

безумовний перехід

```
mov ax, 1
mov bx, 2
jmp x1
mov ax, 11
mov bx, 12
x1:  mov cx, 3
      mov dx, 4
```



Таким чином маємо код

```
        mov bl, 251  
        mov cl, 6  
        add bl, cl  
        jc a  
  
        mov dx, 1 ; CF = 0  
        mov dx, 1  
        mov dx, 1  
        jmp exit  
  
a:  
    mov cx, 1 ; CF = 1  
    mov cx, 1  
    mov cx, 1  
  
exit:
```

00401000 b3fb	mov bl, 0xfb
00401002 b106	mov cl, 6
00401004 00cb	add bl, cl
00401006 720e	jb +14
00401008 66 ba0100	mov dx, 1
0040100c 66 ba0100	mov dx, 1
00401010 66 ba0100	mov dx, 1
00401014 eb0c	jmp +12
00401016 66 b90100	mov cx, 1
0040101a 66 b90100	mov cx, 1
0040101e 66 b90100	mov cx, 1

Під кожен флаг маємо
відповідну команду переходу

Mnemonic	Jumps if...	Flag test
JO	overflow	OF = 1
JNO	no overflow	OF = 0
JS	sign (negative)	SF = 1
JNS	not sign (non-negative)	SF = 0
JZ	zero / equal	ZF = 1
JNZ	not zero / not equal	ZF = 0
JC	carry / below (unsigned)	CF = 1
JNC	not carry / not below (unsigned)	CF = 0
JP	parity even	PF = 1
JNP	parity odd	PF = 0

Але не завжди потрібно робити
операцію **add**, або якусь іншу,
щоб зробити перехід

Можливо просто...

...порівняти

Синтаксис

```
mov ax, 5  
mov bx, 7  
cmp ax, bx
```

Можливі варіанти

ax == bx
ax != bx
ax > bx
ax >= bx
ax < bx
ax <= bx

je
jne
ja
jae
jb
jbe

```
mov ax, 5
mov bx, 7
cmp ax, bx
```

Альтернативи

```
ax == bx  
ax != bx  
ax > bx  
ax >= bx  
ax < bx  
ax <= bx
```

```
je  
jne  
ja  
jae  
jb  
jbe
```

```
jnbe  
jnb  
jnae  
jna
```

```
mov ax, 5  
mov bx, 7  
cmp ax, bx
```

A = above

B = below

Але є нюанс...

Але є нюанс...

Як процесор зрозуміє, що
255, це 255 або -1

-3	11111101
-2	11111110
-1	11111111
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
.....
124	01111100
125	01111101
126	01111110
127	01111111
-128	10000000
-127	10000001
-126	10000010
-125	10000011

Але є нюанс...

Як процесор зрозуміє, що
255, це 255 або -1

НІЯК

-3	11111101
-2	11111110
-1	11111111
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
.....
124	01111100
125	01111101
126	01111110
127	01111111
-128	10000000
-127	10000001
-126	10000010
-125	10000011

Йому треба допомогти

Операція**Unsigned****Signed**

==

JE / JZ

!=

JNE / JNZ

>

JA (Above)

>=

JAE / JNB (Above or
Equal)

<

JB (Below)

<=

JBE (Below or Equal)

JNE / JNZ

JE / JZ

JB (Below)

JAE / JNB (Above or
Equal)

JA (Above)

JE / JZ

JNE / JNZ

JG (Greater)

JGE (Greater or Equal)

JL (Less)

JLE (Less or Equal)

JNE / JNZ

JE / JZ

JLE (Less or Equal)

JL (Less)

JGE (Greater or Equal)

JG (Greater)