Branching

References

- BH chapter 3
- Kl 6.3, 6.4 , 6.5

Branching

- CPU executes program sequentially
- However control can be transferred to other parts.
- Transfers can be conditional or unconditional
- Conditional Transfers:
  - Control is transferred to a new location if a certain condition is true
- Unconditional Transfers:
  - Control is transferred to a new location in all cases.
- The program will start running sequentially from where the control was transferred.

Branching in HLL

- Following is an example of conditional transfer in C++

1. If (x!=0)

```
{
     Z=Y/X
}
```

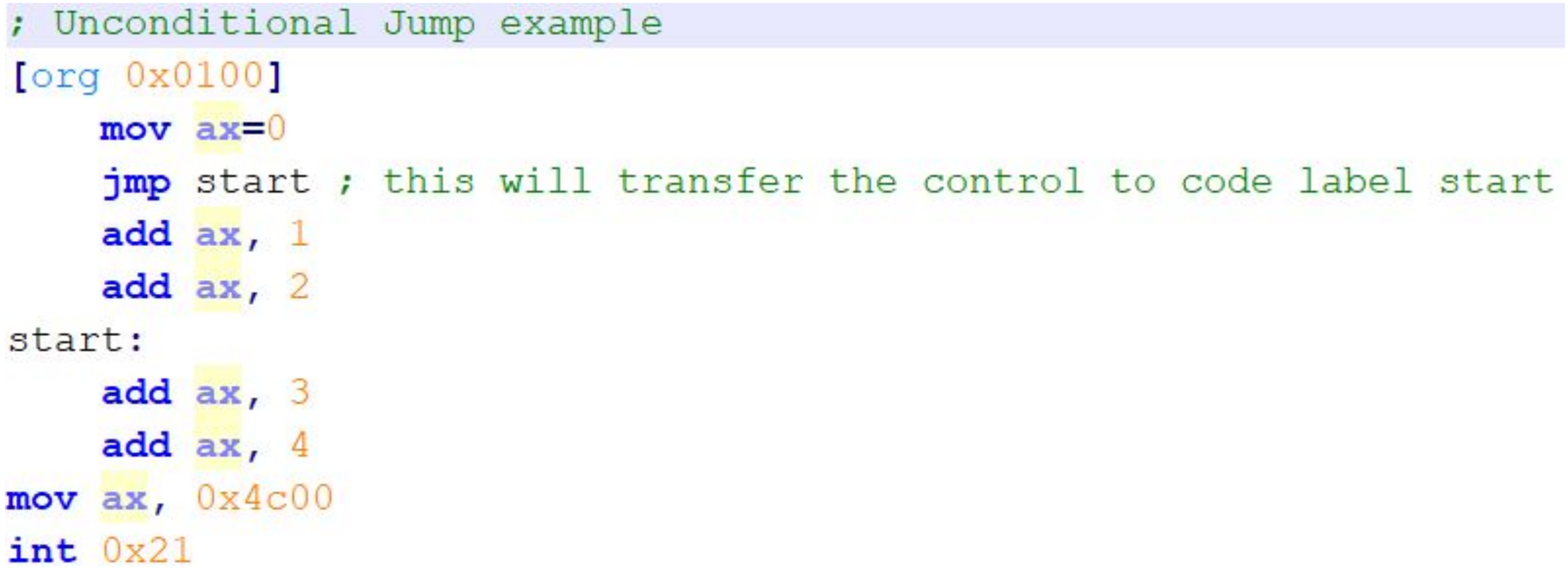
2. while (c<10)

```
{
 sum= a[c--]
}
```

3. goto in C++ can be used for unconditional jump

Unconditional Jump in Assembly

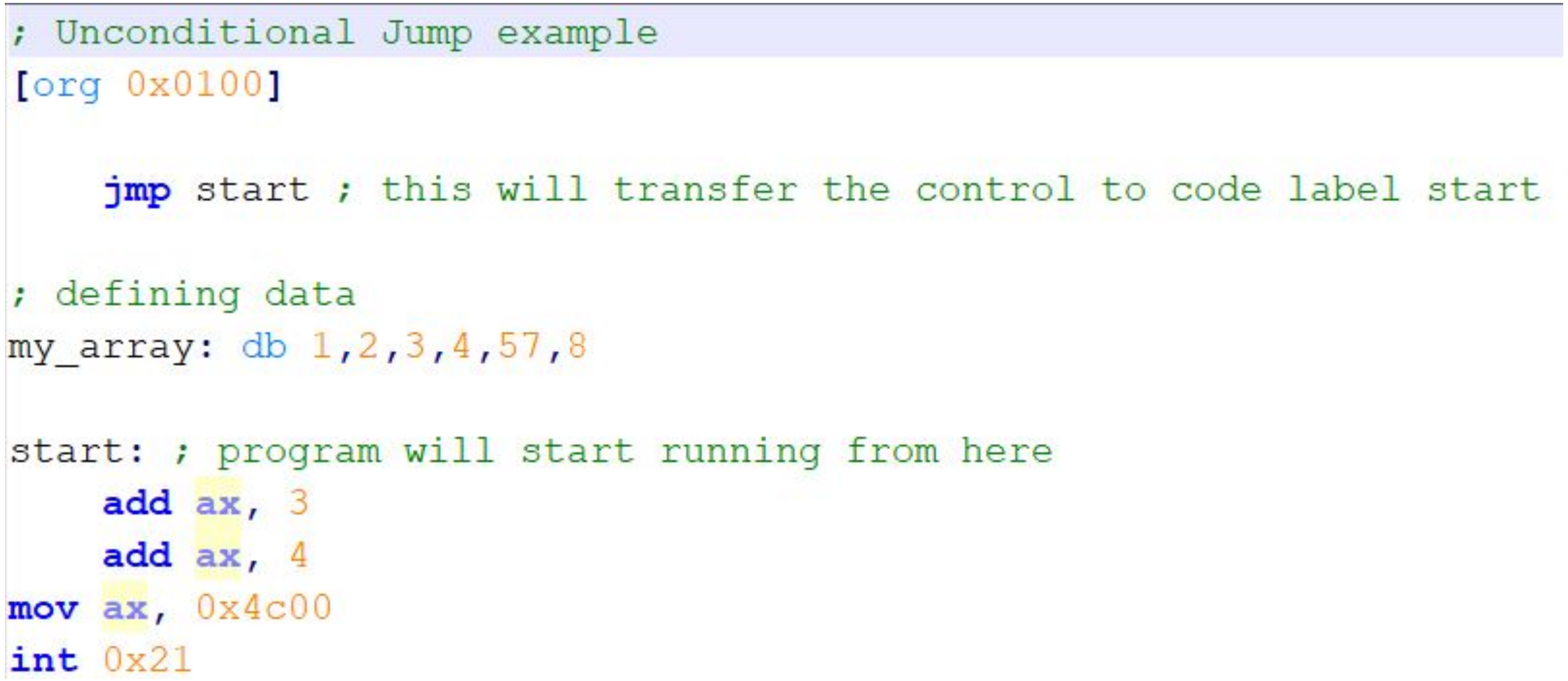
- JMP is used for unconditional jump to another part of code
- Format
  - `JMP <destination>`
  - Where destination can be any 16 bit address where your desired instruction is located.
  - Usually it’s a code label
- Example:



```
; Unconditional Jump example
[org 0x0100]
    mov ax=0
    jmp start ; this will transfer the control to code label start
    add ax, 1
    add ax, 2
start:
    add ax, 3
    add ax, 4
mov ax, 0x4c00
int 0x21
```

Unconditional Jump in Assembly

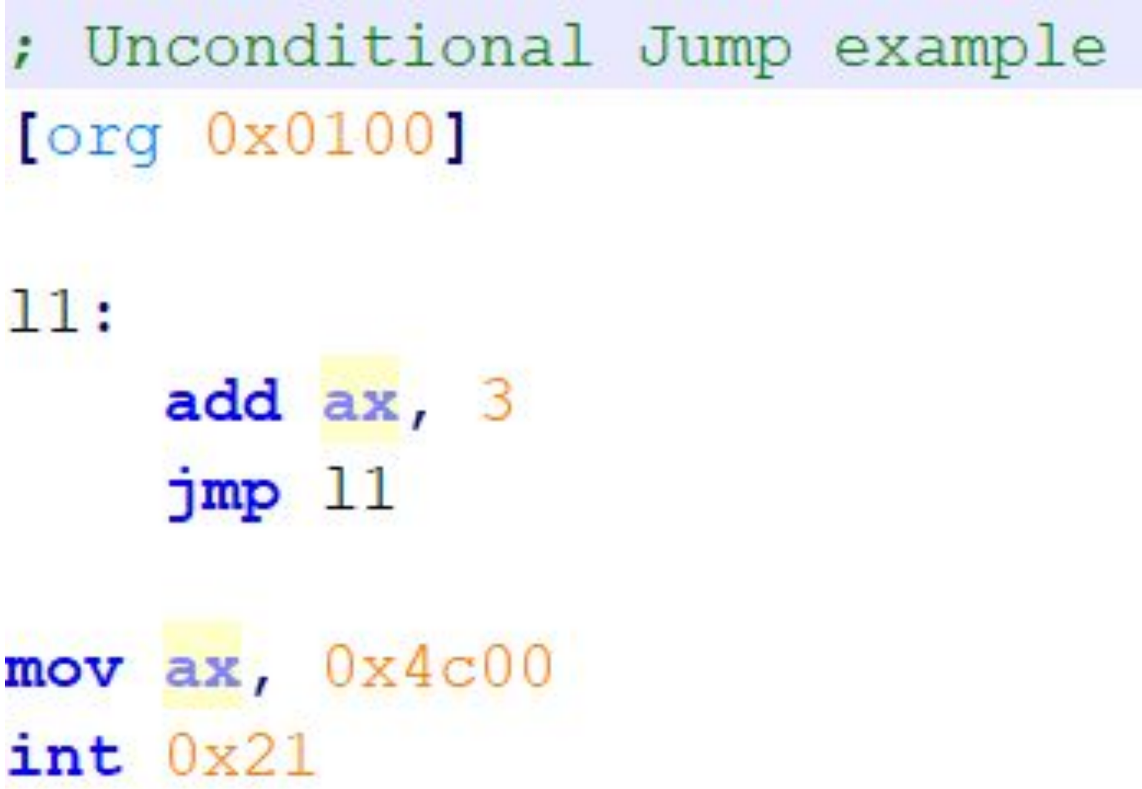
- Especially used when data is defined within code



```
; Unconditional Jump example
[org 0x0100]

    jmp start ; this will transfer the control to code label start

; defining data
my_array: db 1,2,3,4,57,8

start: ; program will start running from here
    add ax, 3
    add ax, 4
mov ax, 0x4c00
int 0x21
```

Question:

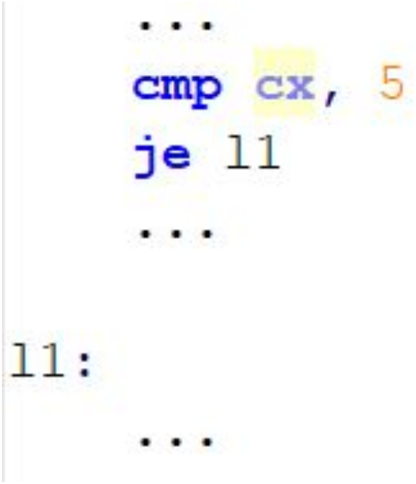
- What is the following code doing?



```
; Unconditional Jump example
[org 0x0100]

l1:
    add ax, 3
    jmp l1

mov ax, 0x4c00
int 0x21
```

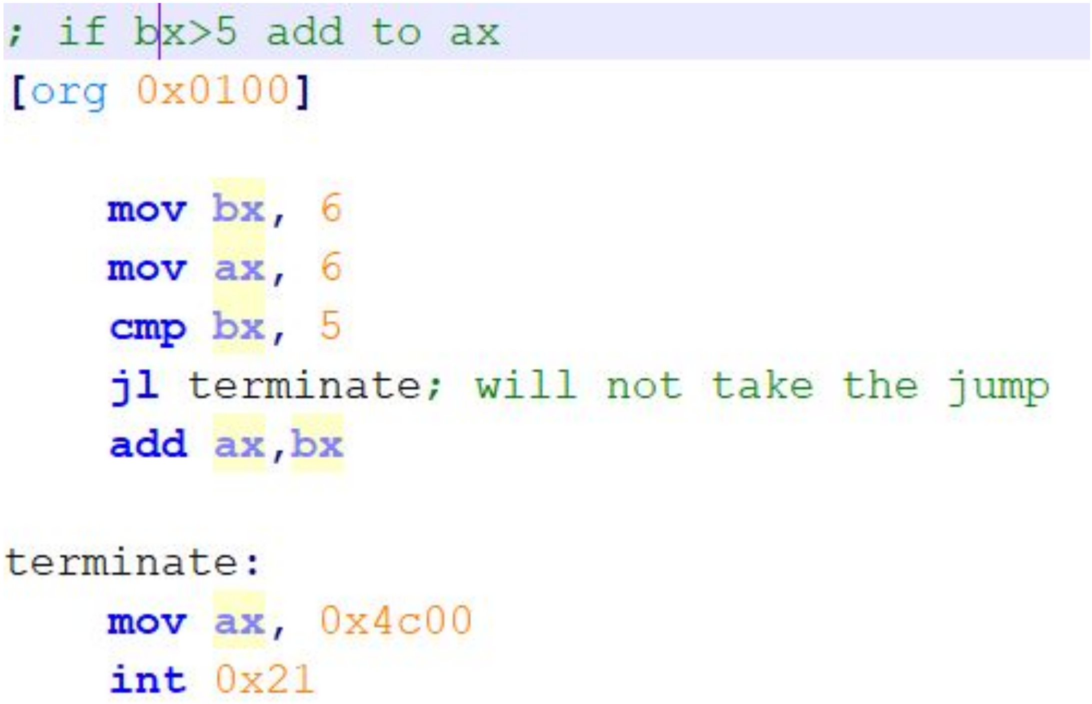
Conditional Jumps

- Conditional jump and LLL are implemented a combination of comparisons and jumps.
- It’s a two step process:
  - First, an operation such as CMP, AND, or SUB modifies the CPU status flags.
  - Second, a conditional jump instruction tests the flags and causes a branch to a new address.
- Example of code snippet

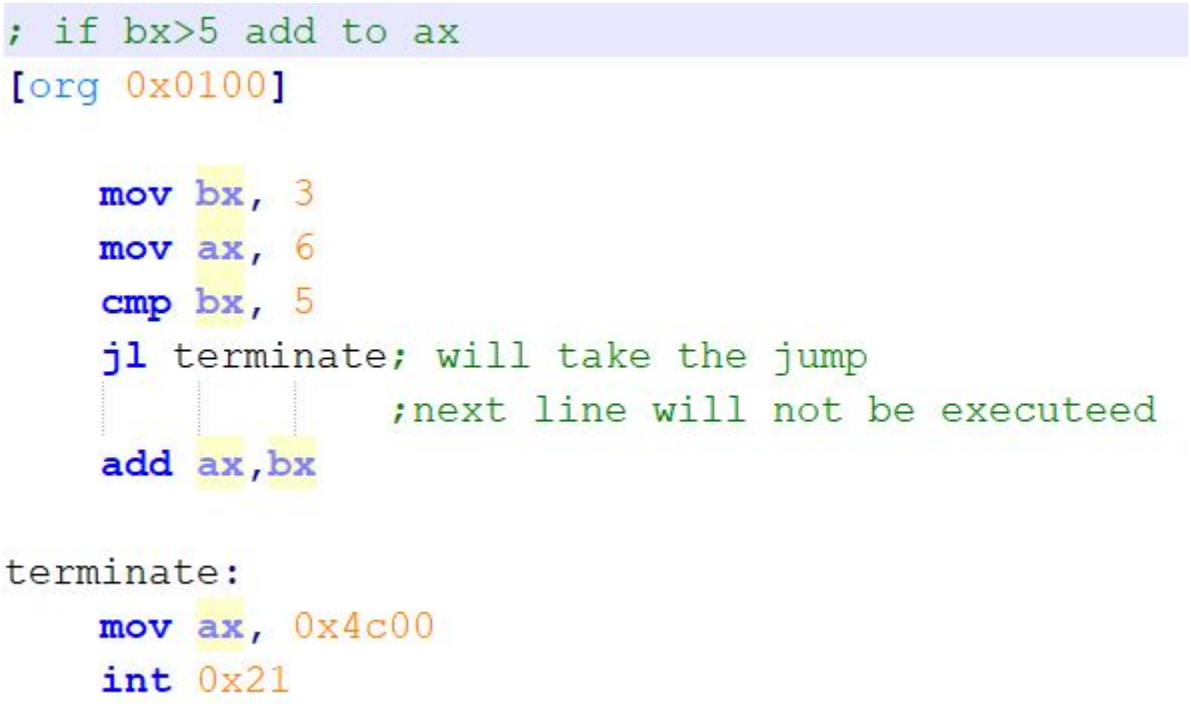


```
        ...
        cmp cx, 5
        je l1
        ...

l1:
        ...
```

Example



```
; if bx>5 add to ax
[org 0x0100]

    mov bx, 6
    mov ax, 6
    cmp bx, 5
    jl terminate; will not take the jump
    add ax,bx

terminate:
    mov ax, 0x4c00
    int 0x21
```



```
; if bx>5 add to ax
[org 0x0100]

    mov bx, 3
    mov ax, 6
    cmp bx, 5
    jl terminate; will take the jump
                ;next line will not be executeed
    add ax,bx

terminate:
    mov ax, 0x4c00
    int 0x21
```

Cmp instrution

- The CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand.
- Neither operand is modified
- The CMP instruction changes the Overflow, Sign, Zero, Carry, Auxiliary Carry, and Parity flags according to the value the destination operand would have had if actual subtraction had taken place.

Conditional jump instruction

- In last examples you saw je and jl instructions.
- These are example of conditional jump instructions
- Je stands for jump if equal, jl stands for jump if larger.
- There are many other conditional jumps.
- These instructions uses flags to check whether to take or not take the jump.
    - For example:
        - je will check ZF to see if the numbers were equal or not.
        - cmp would have set ZF to 1 is both operands were equal
- Format is
    - `<conditional jump instruction> <destination>`

Types of Conditional jumps

- Jumps based on specific flag values
- Jumps based on equality between operands or the value of CX
- Jumps based on comparisons of unsigned operands
- Jumps based on comparisons of signed operands

Types of Conditional jumps

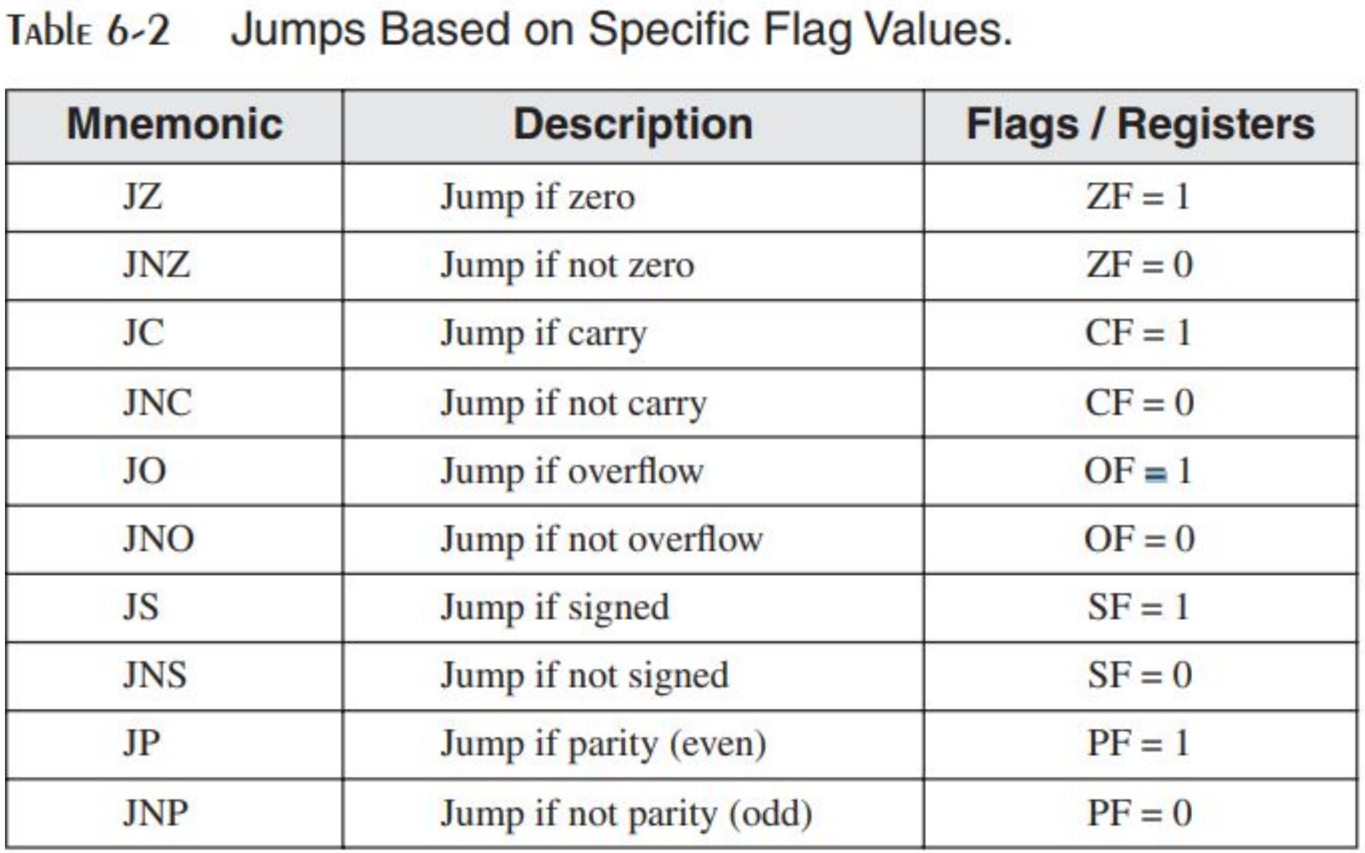
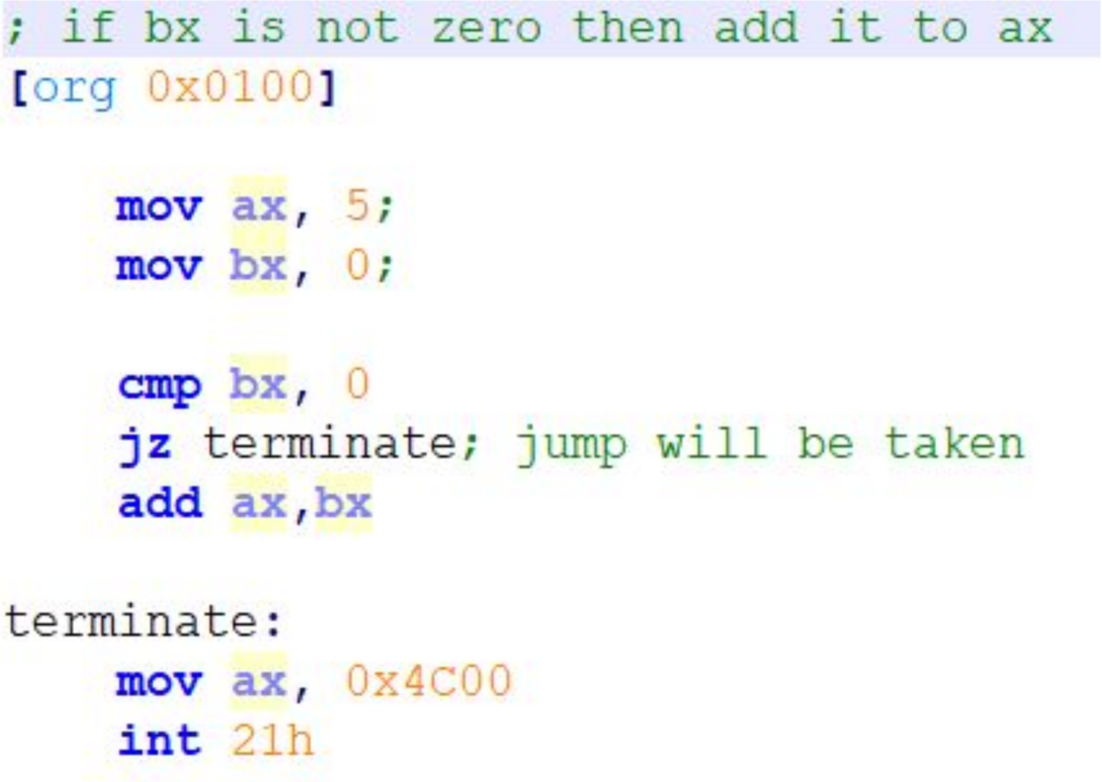


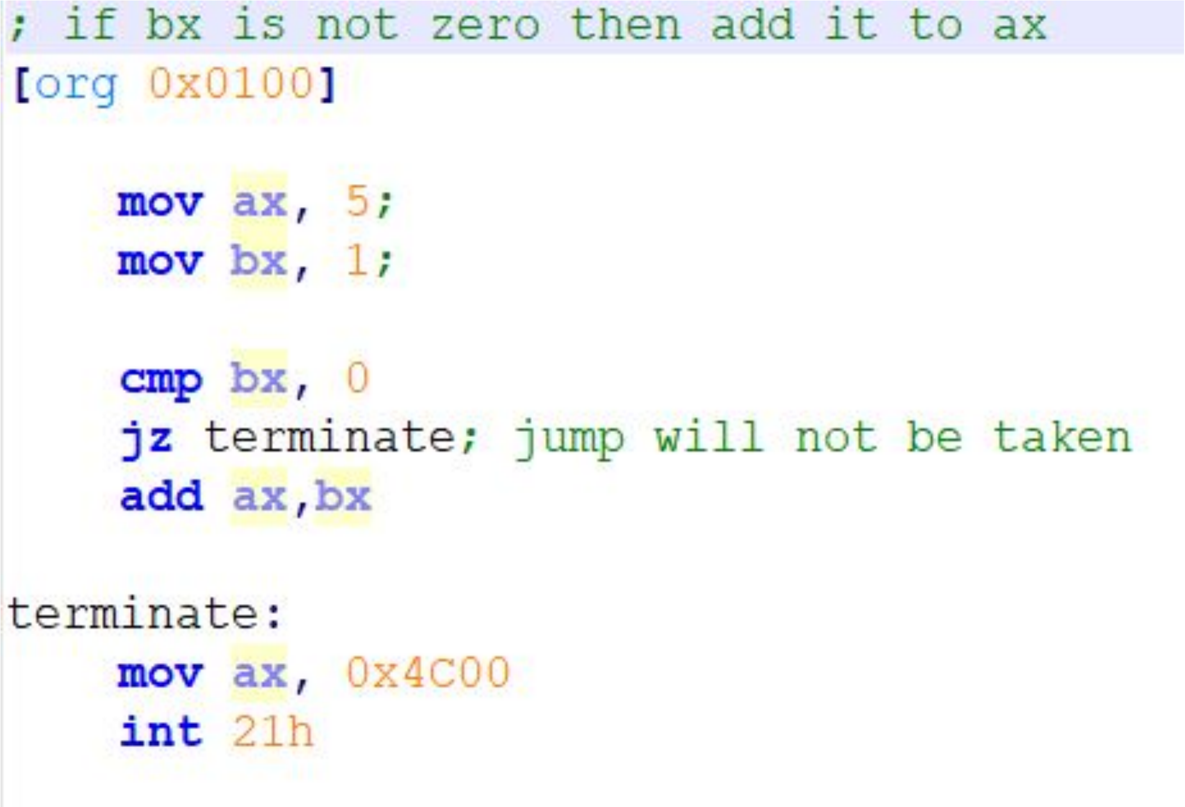
Table 6-2 Jumps Based on Specific Flag Values.

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Example

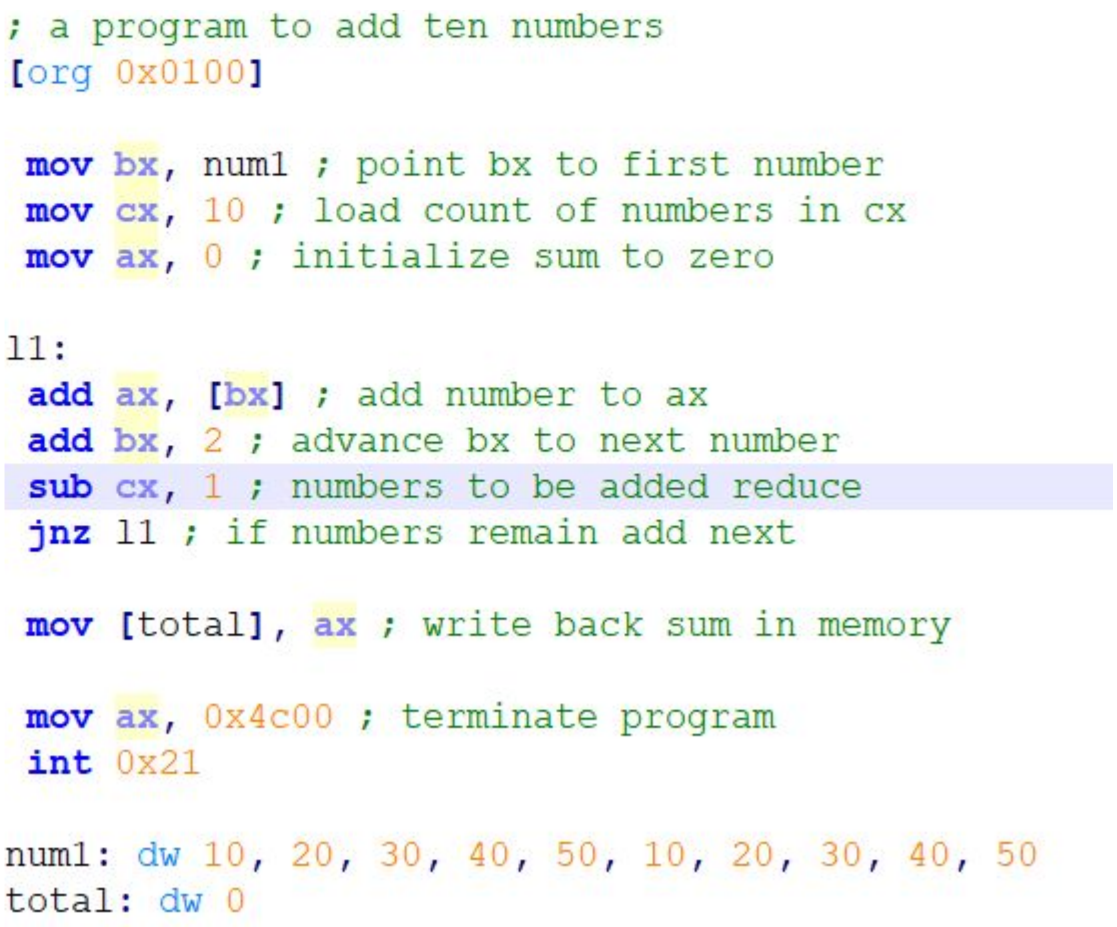


```
; if bx is not zero then add it to ax
[org 0x0100]

    mov ax, 5;
    mov bx, 0;

    cmp bx, 0
    jz terminate; jump will be taken
    add ax,bx

terminate:
    mov ax, 0x4C00
    int 21h
```



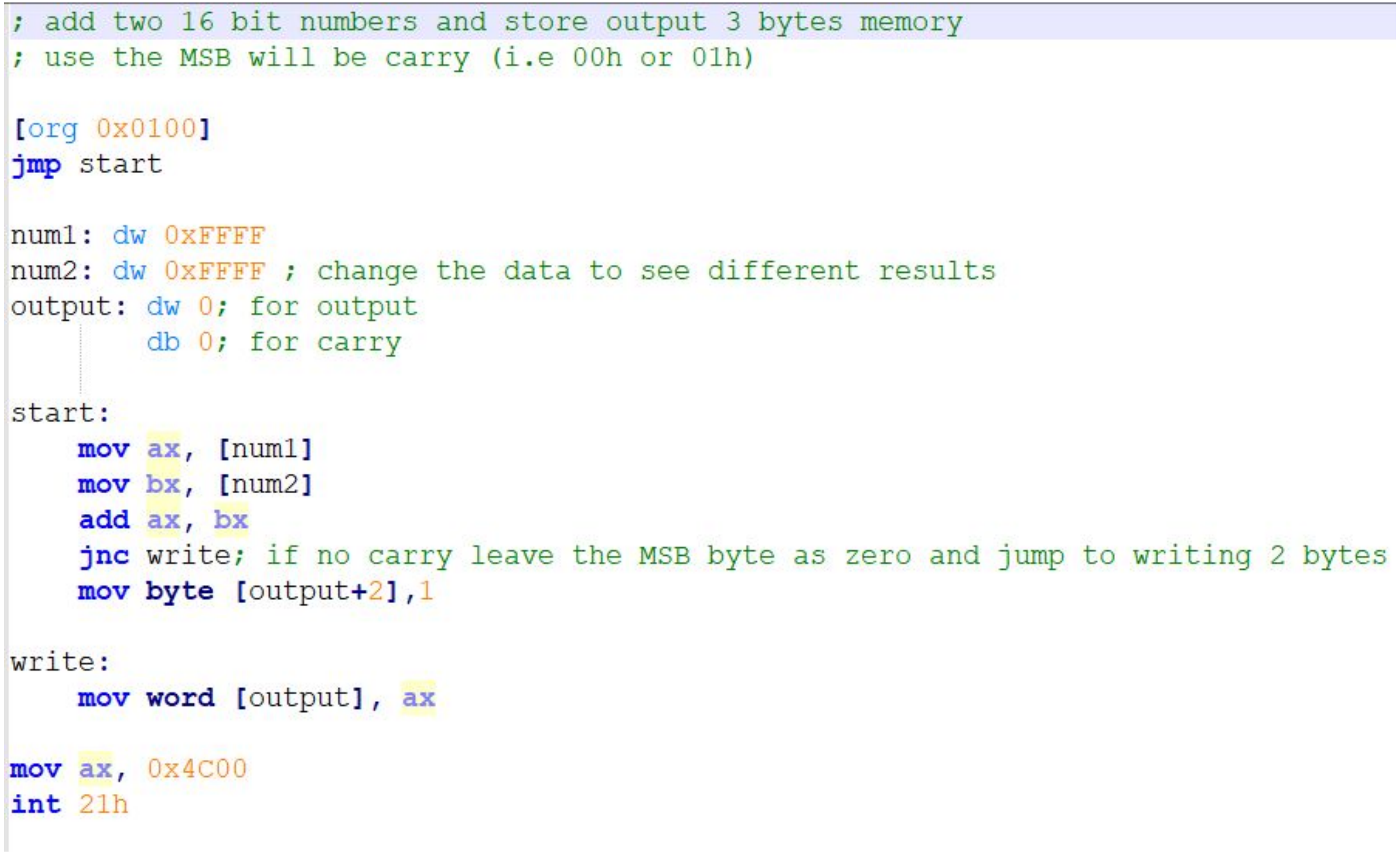
```
; if bx is not zero then add it to ax
[org 0x0100]

    mov ax, 5;
    mov bx, 1;

    cmp bx, 0
    jz terminate; jump will not be taken
    add ax,bx

terminate:
    mov ax, 0x4C00
    int 21h
```

Example



```
; a program to add ten numbers
[org 0x0100]

 mov bx, num1 ; point bx to first number
 mov cx, 10 ; load count of numbers in cx
 mov ax, 0 ; initialize sum to zero

l1:
 add ax, [bx] ; add number to ax
 add bx, 2 ; advance bx to next number
 sub cx, 1 ; numbers to be added reduce
 jnz l1 ; if numbers remain add next

 mov [total], ax ; write back sum in memory

 mov ax, 0x4c00 ; terminate program
 int 0x21

num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

Example



```
; add two 16 bit numbers and store output 3 bytes memory
; use the MSB will be carry (i.e 00h or 01h)

[org 0x0100]
jmp start

num1: dw 0xFFFF
num2: dw 0xFFFF ; change the data to see different results
output: dw 0; for output
        db 0; for carry

start:
    mov ax, [num1]
    mov bx, [num2]
    add ax, bx
    jnc write; if no carry leave the MSB byte as zero and jump to writing 2 bytes
    mov byte [output+2],1

write:
    mov word [output], ax

mov ax, 0x4C00
int 21h
```

Types of Conditional jumps

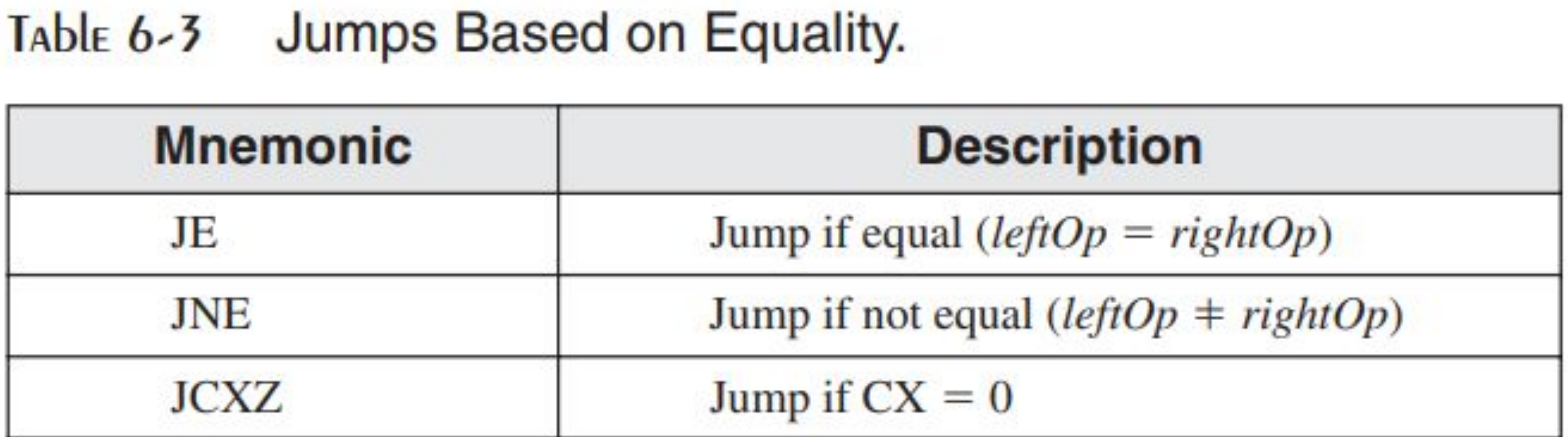
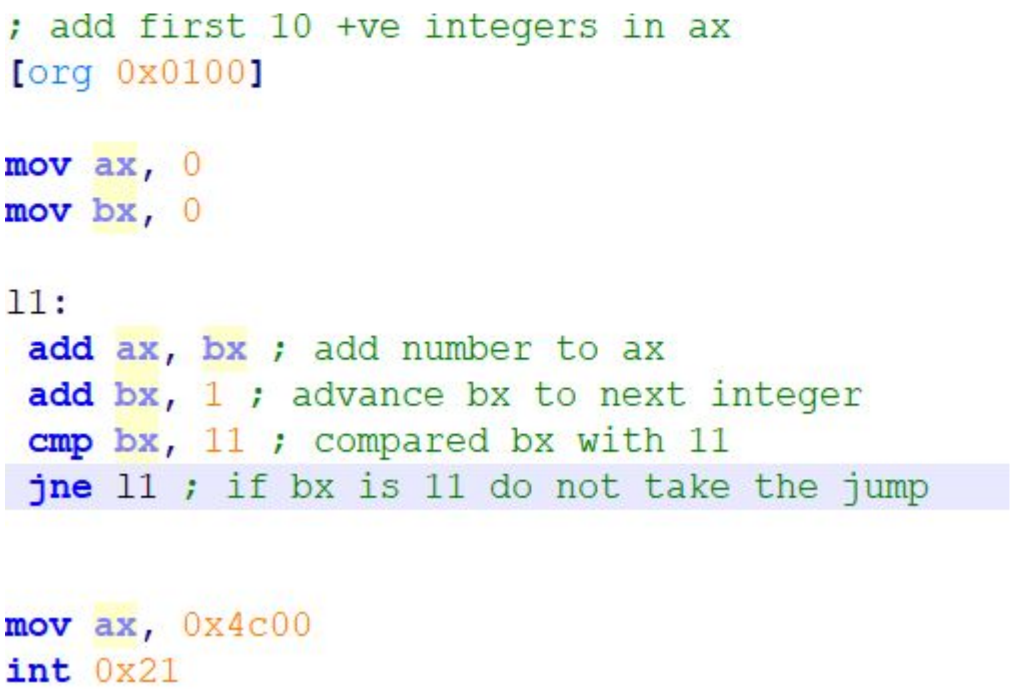


Table 6-3 Jumps Based on Equality.

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0

Example



```
; add first 10 +ve integers in ax
[org 0x0100]

mov ax, 0
mov bx, 0

l1:
 add ax, bx ; add number to ax
 add bx, 1 ; advance bx to next integer
 cmp bx, 11 ; compared bx with 11
 jne l1 ; if bx is 11 do not take the jump


mov ax, 0x4c00
int 0x21
```

Types of Conditional jumps

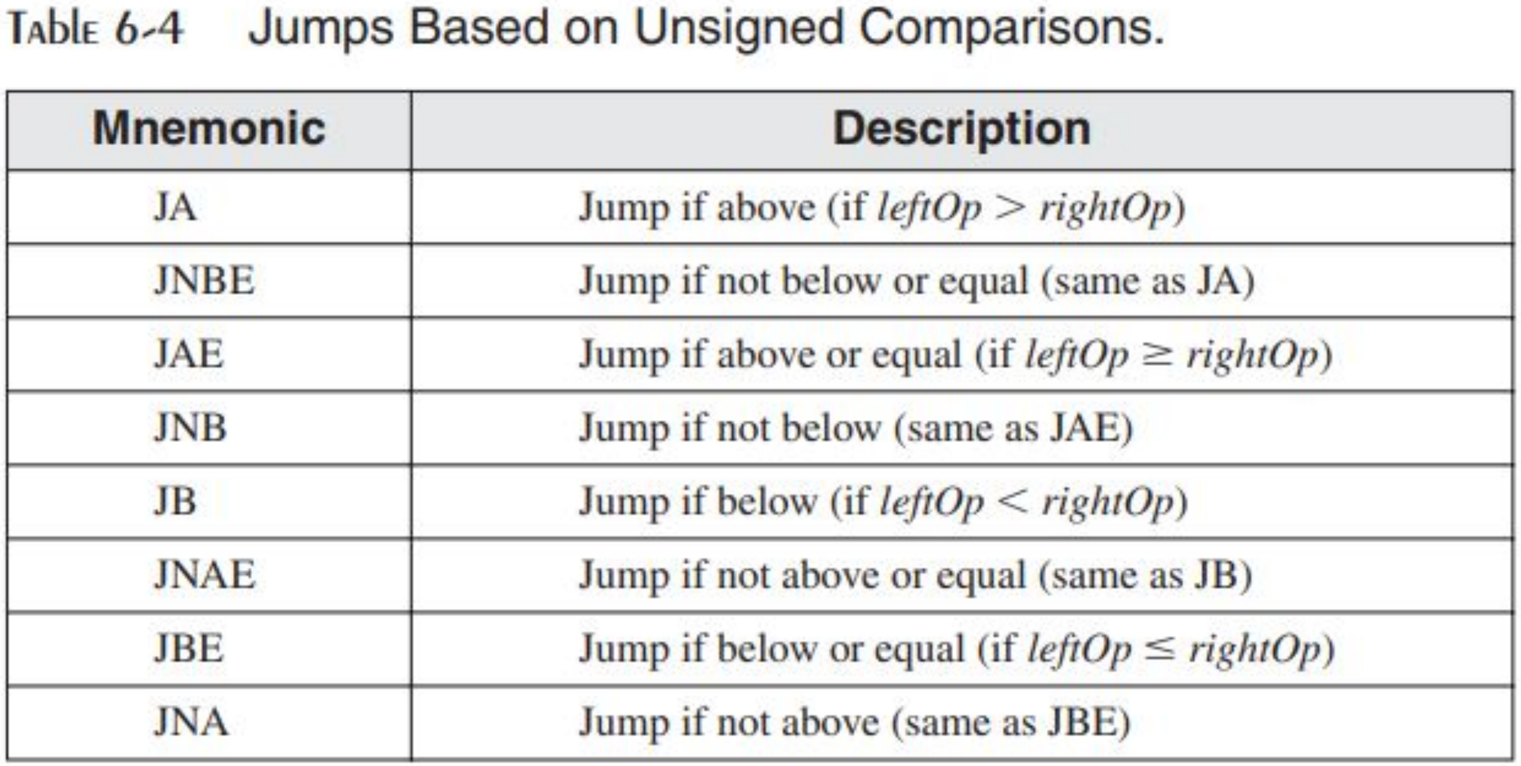


Table 6-4 Jumps Based on Unsigned Comparisons.

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* ≥ *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* ≤ *rightOp*)
JNA	Jump if not above (same as JBE)

Types of Conditional jumps

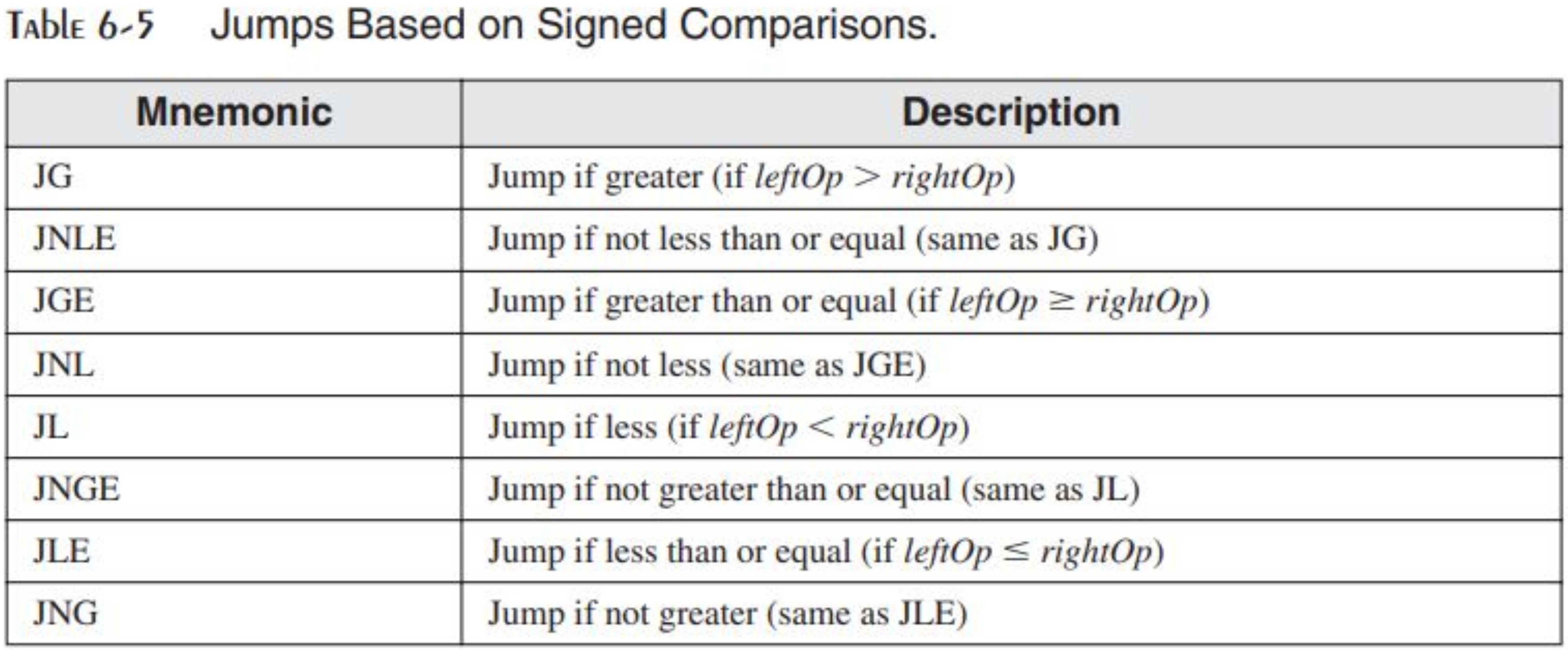


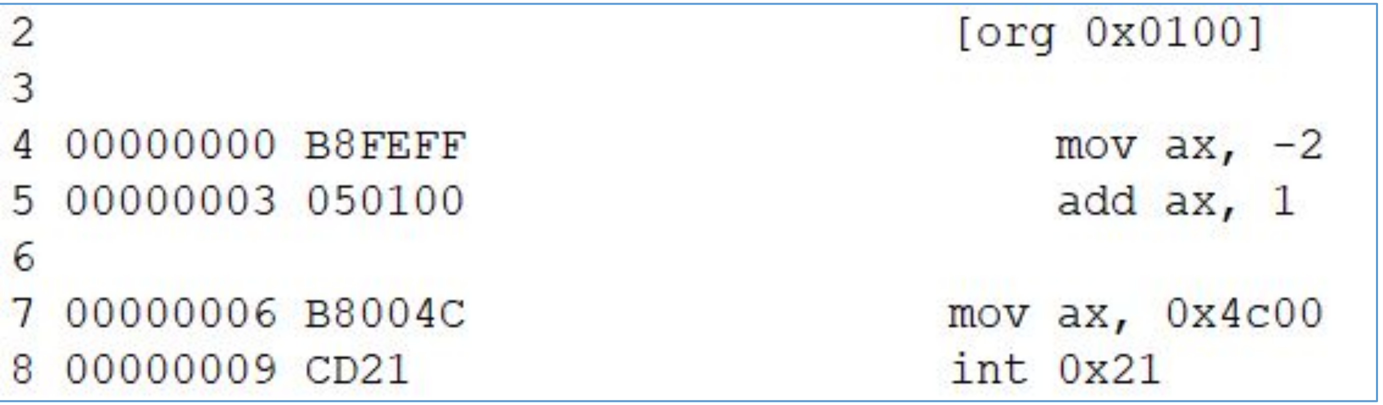
Table 6-5 Jumps Based on Signed Comparisons.

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

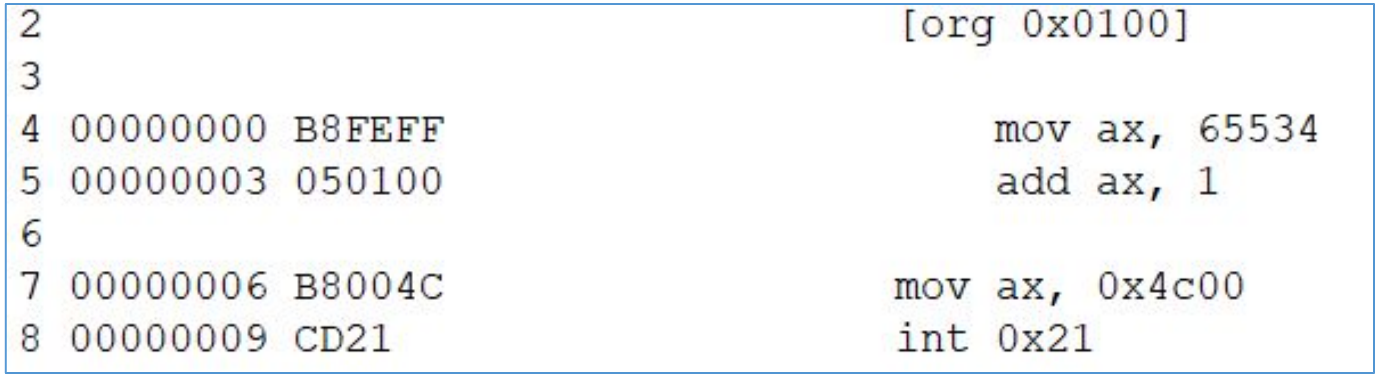
Difference Between Signed and Unsigned number

- The processor does not consider the difference between signed or unsigned number
- It only maintains flags for either case
- It depends on programmer how they interpret the flag and which jump instructions they use.

Example



```
                                   [org 0x0100]

00000000 B8FEFF                        mov ax, -2
00000003 050100                        add ax, 1

00000006 B8004C                    mov ax, 0x4c00
00000009 CD21                      int 0x21
```

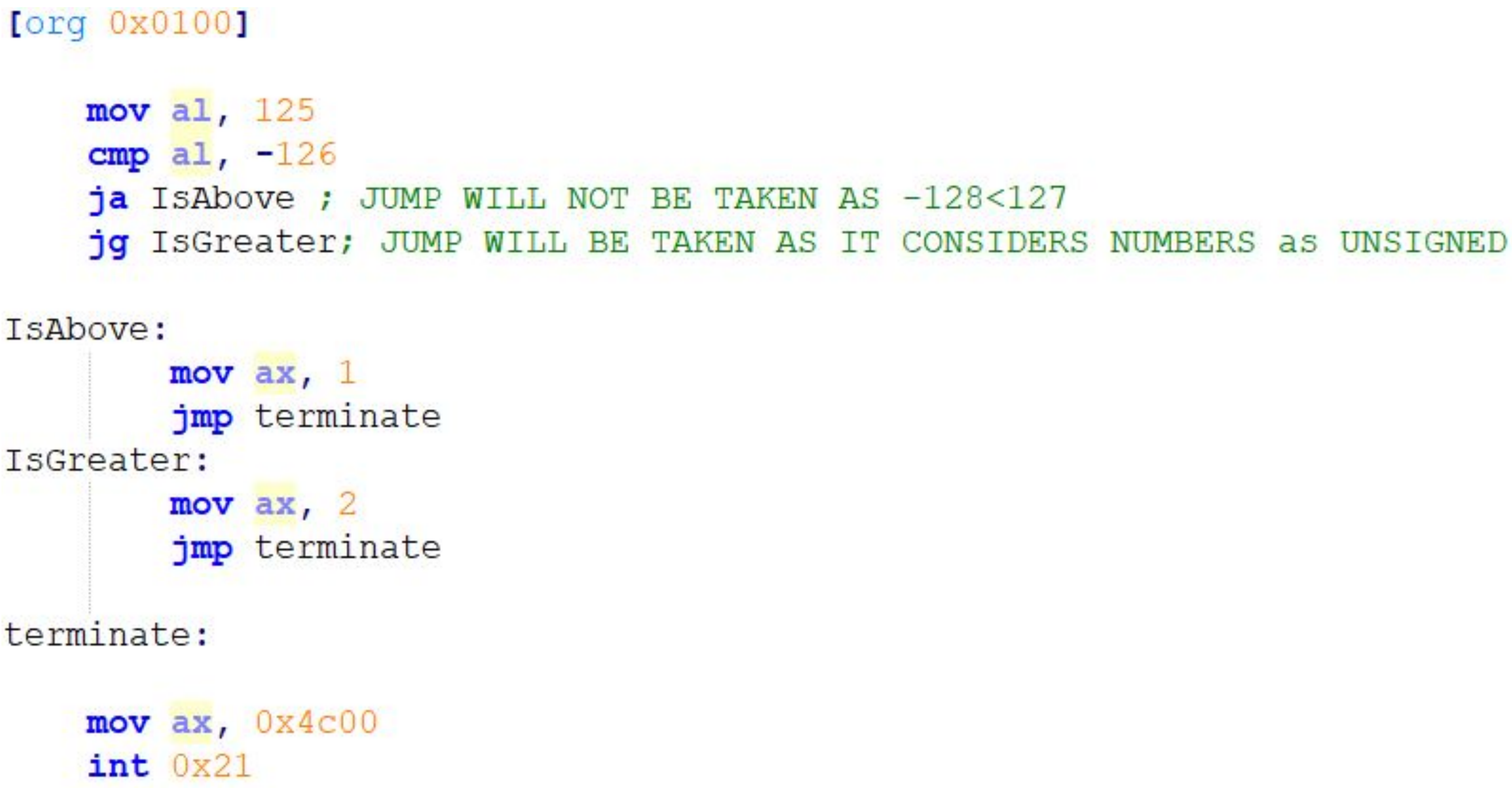


```
                                 [org 0x0100]

00000000 B8FEFF                      mov ax, 65534
00000003 050100                      add ax, 1

00000006 B8004C                  mov ax, 0x4c00
00000009 CD21                    int 0x21
```

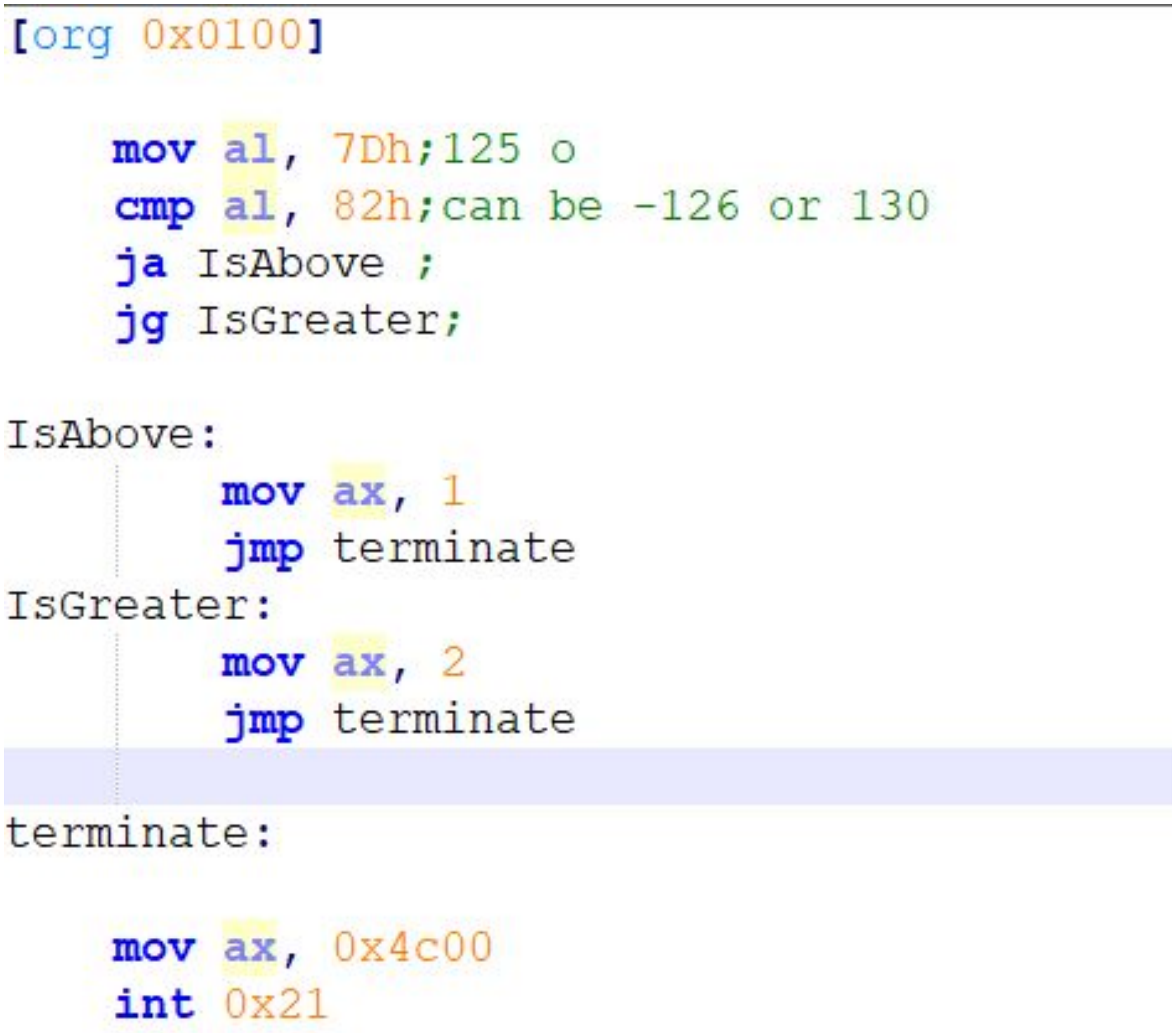
- Two different assembly codes
- Same machine code
- Same ax after code ends
    - Ax=FFFF
- Processor will on SF in both cases
    - SF=1
- It depends on the programmer to interpret AX as 65535 or as -1

Example

- Difference between jumps for signed and unsigned comparison.
- After cmp, CF=1, SF=1, OF=1



```
[org 0x0100]

    mov al, 125
    cmp al, -126
    ja IsAbove ; JUMP WILL NOT BE TAKEN AS -128<127
    jg IsGreater; JUMP WILL BE TAKEN AS IT CONSIDERS NUMBERS as UNSIGNED

IsAbove:
        mov ax, 1
        jmp terminate
IsGreater:
        mov ax, 2
        jmp terminate

terminate:

    mov ax, 0x4c00
    int 0x21
```



```
[org 0x0100]

    mov al, 7Dh;125 o
    cmp al, 82h;can be -126 or 130
    ja IsAbove ;
    jg IsGreater;

IsAbove:
        mov ax, 1
        jmp terminate
IsGreater:
        mov ax, 2
        jmp terminate

terminate:

    mov ax, 0x4c00
    int 0x21
```

Example

Dec	Hex	Binary
125	7D	0111 1101
-126	82	1000 0010
125-(-126)= 251	7D-82= FB	1111 1011

- CF is on because 7D-82 needs borrow
- SF is on because MSB of answer is 1
- OF flag is on because, 251 is out of range of -128 to -127
    - You can also determine OF by looking at the difference in MSB of 1st operand and result (i.e. 7D and FB, they are different)

FLAGs associate with jumps

- JB will check CF=1
- JL will check SF ≠ OF
- JA will check ZF = 0 AND CF = 0
- JG will check ZF = 0 AND SF = OF
- You can see all these association in table given in BH page 42