# 6. Intel Assembly II
# - Control Transfer Instructions

# Control Transfer Instructions

- Unconditional transfers
  - Jump
  - Call and return (covered later)
- Conditional transfers
- Software interrupts (covered later)

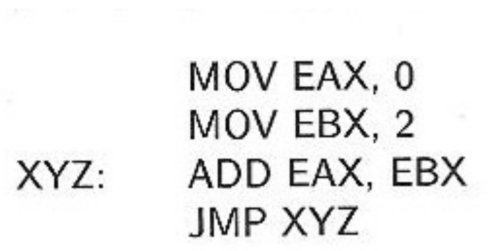
## Unconditional Transfer

- jmp
  - Transfers program control to a different point in the instruction stream without recording return info
  - The destination operand can be an immediate value, a general-purpose register, or a memory location  (absolute offset vs. relative offset)
  - Immediate operand can be specified by code label

- Code label
  - A label <u>in the code segment</u> that the program control is transferred to
  - The label is created by programmer; long and understandable labels are useful
  - Colon is often appended
  - No two lines in the code segment may have the same label

## JMP

- Program continues adding 2 to the EAX register forever; it is a infinite loop



```
        MOV EAX, 0
        MOV EBX, 2
XYZ:    ADD EAX, EBX
        JMP XYZ
```

## Variations of Jump

- Near jump: within a segment (default jump)

```
jmp [eax]         ;Jump to
jmp LABEL.BEL     ;Jump to address given by LABEL.
```

- Far jump: allows control to move to another code segment

```
jmp far LABEL     ;Jump to address given by LABEL.
```

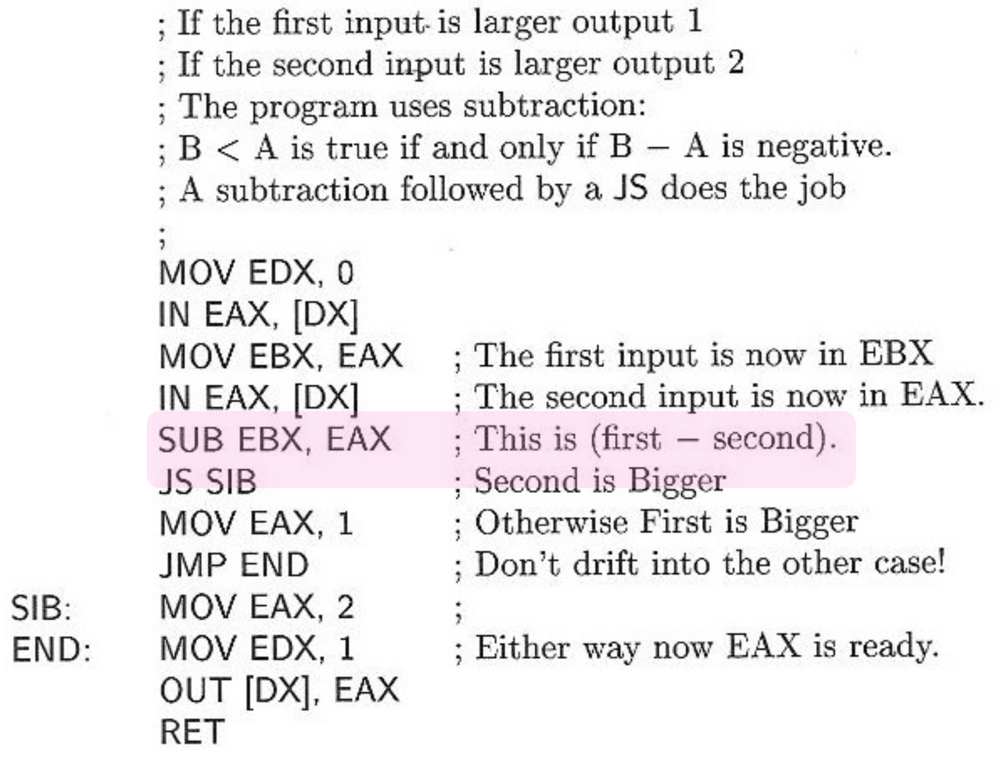
- Short jump: uses a single byte to store the displacement of the jump (+127/-128 bytes)

```
NEXT:   add ax, bx
        jmp short NEXT
```

## Conditional Transfer

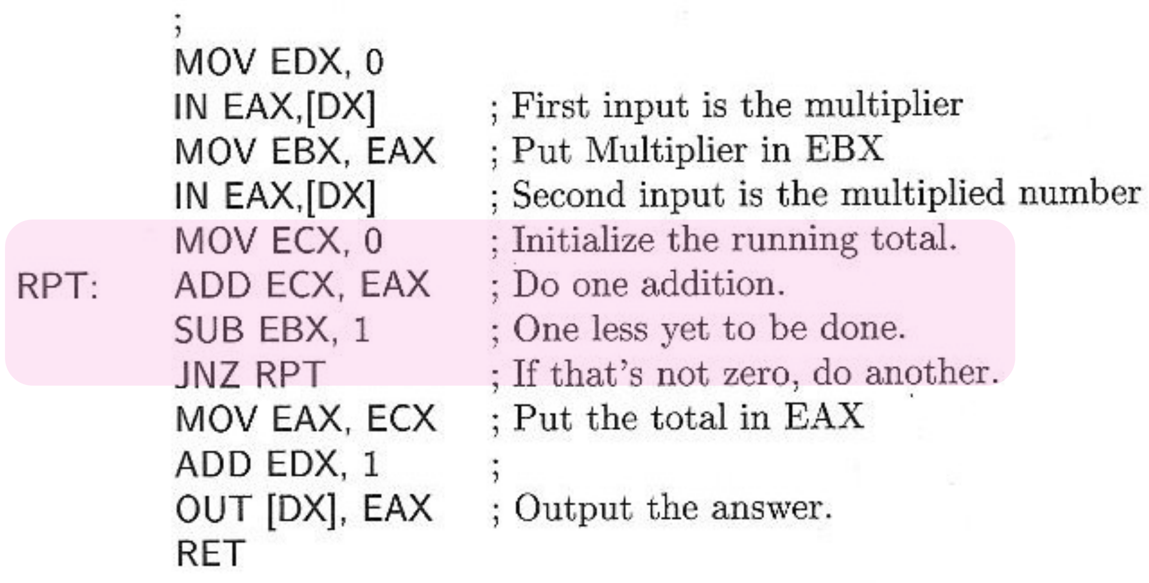
- Conditional jump is a jump carried out on the basis of a truth value
- Decisions are based on one-bit in *eflags*: ZF, SF, CF, OF, AF, and PF
  - JZ branches only if ZF is set
  - JNZ branches only if ZF is unset
  - JO branches only if OF is set
  - JNO branches only if OF is unset
  - JS branches only if SF is set
  - JNS branches only if SF is unset
  - JC branches only if CF is set
  - JNC branches only if CF is unset
  - JP branches only if PF is set
  - JNP branches only if PF is unset

# Conditional Jump Example - JS



```
         ; If the first input is larger output 1
         ; If the second input is larger output 2
         ; The program uses subtraction:
         ; B < A is true if and only if B − A is negative.
         ; A subtraction followed by a JS does the job
         ;
         MOV EDX, 0
         IN EAX, [DX]
         MOV EBX, EAX    ; The first input is now in EBX
         IN EAX, [DX]    ; The second input is now in EAX.
         SUB EBX, EAX    ; This is (first − second).
         JS SIB          ; Second is Bigger
         MOV EAX, 1      ; Otherwise First is Bigger
         JMP END         ; Don't drift into the other case!
SIB:     MOV EAX, 2      ;
END:     MOV EDX, 1      ; Either way now EAX is ready.
         OUT [DX], EAX
         RET
```

## Programs with loops



```
        ;
        MOV EDX, 0
        IN EAX,[DX]         ; First input is the multiplier
        MOV EBX, EAX        ; Put Multiplier in EBX
        IN EAX,[DX]         ; Second input is the multiplied number
        MOV ECX, 0          ; Initialize the running total.
RPT:    ADD ECX, EAX        ; Do one addition.
        SUB EBX, 1          ; One less yet to be done.
        JNZ RPT             ; If that's not zero, do another.
        MOV EAX, ECX        ; Put the total in EAX
        ADD EDX, 1          ;
        OUT [DX], EAX       ; Output the answer.
        RET
```

Program 4.5

## Comparison-based Jump

cmp vleft, vright

- For unsigned integer : ZF and CF
  - If vleft == vright, ZF == 1 and CF == 0
  - If vleft > vright, ZF == 0 and CF == 0
  - If vleft < vright, ZF == 0 and CF == 1

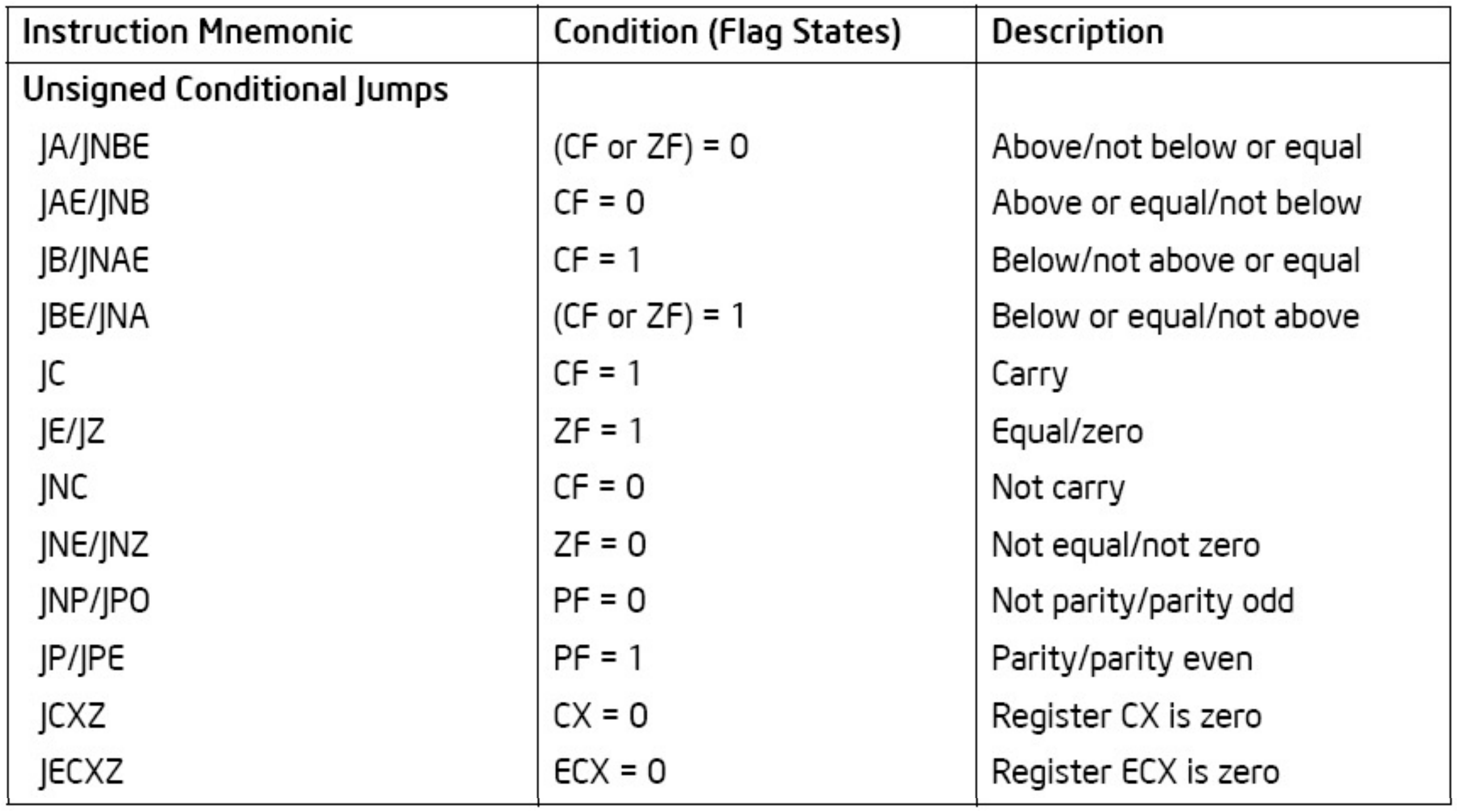
- For signed integer: ZF, OF, and SF
  - If vleft == vright, ZF == 1, OF == 0, and SF == 0
  - If vleft > vright, ZF == 0 and OF == SF
  - If vleft < vright, ZF == 0 and OF ≠ SF

# Comparison-based jump

cmp vleft, vright

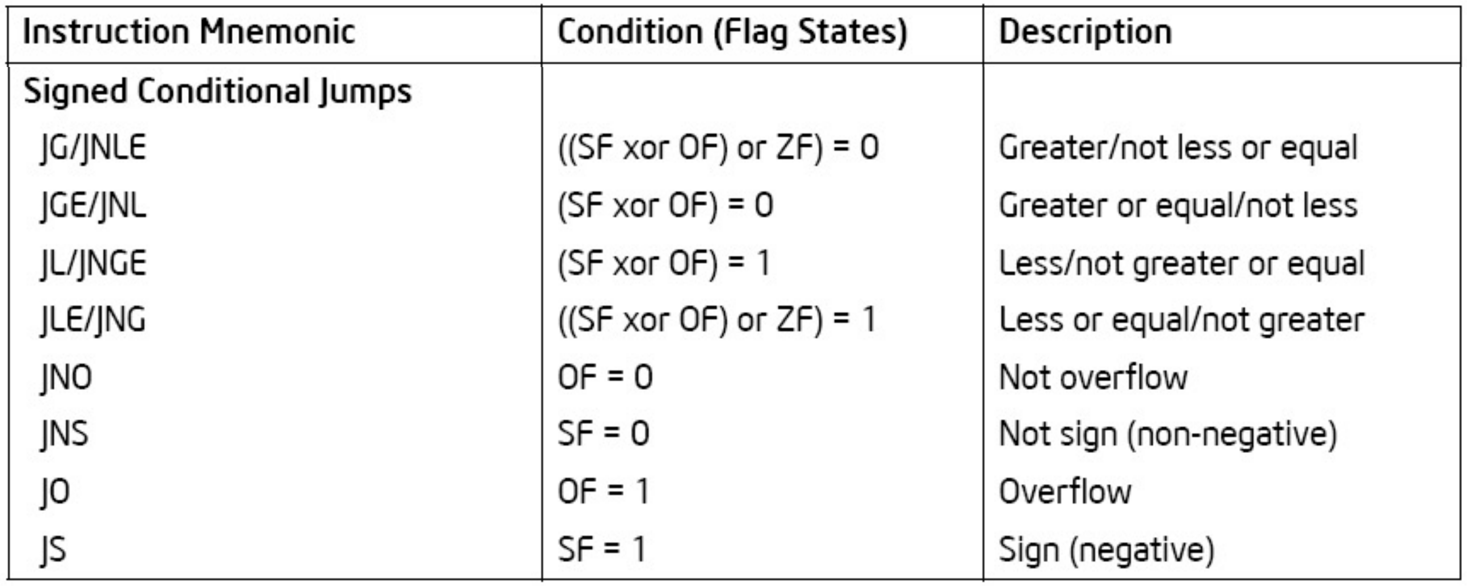
| Signed | Unsigned |
|---|---|
| JE branches if vleft = vright | JZ branches if vleft = vright |
| JNE branches if vleft ≠ vright | JNZ branches if vleft ≠ vright |
| JL, JNGE branches if vleft < vright | JB, JNAE branches if vleft < vright |
| JLE, JNG branches if vleft ≤ vright | JBE, JNA branches if vleft ≤ vright |
| JG, JNLE branches if vleft > vright | JA, JNBE branches if vleft > vright |
| JGE, JNL branches if vleft ≥ vright | JAE, JNB branches if vleft ≥ vright |

# Unsigned Conditional Jumps



| Instruction Mnemonic | Condition (Flag States) | Description |
|---|---|---|
| **Unsigned Conditional Jumps** | | |
| JA/JNBE | (CF or ZF) = 0 | Above/not below or equal |
| JAE/JNB | CF = 0 | Above or equal/not below |
| JB/JNAE | CF = 1 | Below/not above or equal |
| JBE/JNA | (CF or ZF) = 1 | Below or equal/not above |
| JC | CF = 1 | Carry |
| JE/JZ | ZF = 1 | Equal/zero |
| JNC | CF = 0 | Not carry |
| JNE/JNZ | ZF = 0 | Not equal/not zero |
| JNP/JPO | PF = 0 | Not parity/parity odd |
| JP/JPE | PF = 1 | Parity/parity even |
| JCXZ | CX = 0 | Register CX is zero |
| JECXZ | ECX = 0 | Register ECX is zero |

# Signed Conditional Jumps



| Instruction Mnemonic | Condition (Flag States) | Description |
|---|---|---|
| **Signed Conditional Jumps** | | |
| JG/JNLE | ((SF xor OF) or ZF) = 0 | Greater/not less or equal |
| JGE/JNL | (SF xor OF) = 0 | Greater or equal/not less |
| JL/JNGE | (SF xor OF) = 1 | Less/not greater or equal |
| JLE/JNG | ((SF xor OF) or ZF) = 1 | Less or equal/not greater |
| JNO | OF = 0 | Not overflow |
| JNS | SF = 0 | Not sign (non-negative) |
| JO | OF = 1 | Overflow |
| JS | SF = 1 | Sign (negative) |

## Translating Standard Control Structures - if

- if statements

```
if (condition)
   then_block;
```

```
   ; code to set FLAGS
   jxx endif  ; select xx so that branches
              ; if condition false
   ; code for then_block
endif:
```

## Translating Standard Control Structures – if-else

- if-else statements

```
if (condition)
   then_block;
else
   else_block
```

```
   ; code to set FLAGS
   jxx else_block  ; select xx so that
                   ;branches if condition false
   ; code for then_block
   jmp endif
else_block:
   ; code for else_block
endif:
```

## Translating Standard Control Structures - while

- while loops

```
while (condition) {
   body_of_loop;
}
```

```
while:
   ; code to set FLAGS based on condition
   jxx end_while  ; select xx so that
                   ;branches if condition false
   ; body_of_loop
   jmp while
end_while:
```

## Translating Standard Control Structures - do-while

- do -while loops

```
do {
   body_of_loop;
} while (condition);
```

```
do:
   ; body_of_loop
   ; code to set FLAGS based on condition
   jxx do    ; select xx so that
              ;branches if condition false
```

## Loop Instruction

- LOOP Instruction
  - Combination of decrement ecx and jnz conditional jump.
    - Decrement ecx
    - If ecx != 0, jump to label
    - else fall through.
  - LOOP, LOOPE (loop while equal), LOOPZ (loop while zero), LOOPNE (loop while not equal), and LOOPNZ (loop while not zero)

```
loop LABEL  ;Jump if ecx != 0
loope      ;Jump if (Z = 1 AND ecx != 0)
loopne     ;Jump if (Z = 0 AND ecx != 0)
```