# Micro-Processors

**Tutorial 4**
***Jumps, Macros and Procedures***

Noura Gamal

# Jumps

Divided into 2 types:

1. Unconditional Jumps
    a. **NEAR** – This procedure targets within the same code segment. (Intra-segment)
    b. **SHORT** - This procedure also targets within the same code segment, but the offset is 1 byte long. (Intra-segment)
    c. **FAR** - In this procedure, the target is outside the segment and the size of the pointer is double word. (Inter-segment)
2. Conditional Jumps

# Short Jump (Unconditional -IntraSegment Jump)

- **Short jump** is a 2-byte instruction that allows jumps or branches to memory locations within **+127** and **−128** bytes.
- 8 bit signed displacement.
- From the address **following** the jump
- Relocatable; relative jumps.

# Short Jump

Example

```
Offset   Machine Code Source Code
0100     B4 02       start:  mov ah, 2  ;loop start
0102     B2 41               mov  dl, 'A' ;
0104     CD 21               int     21h    ;disp A
0106     EB  F8              jmp     start  ;jmp back
0108     ….. (rest of program)
```

How does the compiler know it's a SHORT jump?

0100 –0108 = -8 = F8

**Short JMP**

| OPCODE (EBH) | DISP |
|---|---|

# Near Jump (Unconditional -IntraSegment Jump)

- A near jump passes control to an instruction in the current code segment located within ±32K bytes from the near jump instruction.
- Near jump is a 3-byte instruction with opcode  followed by a signed 16-bit displacement.
- Relocatable; relative jumps.

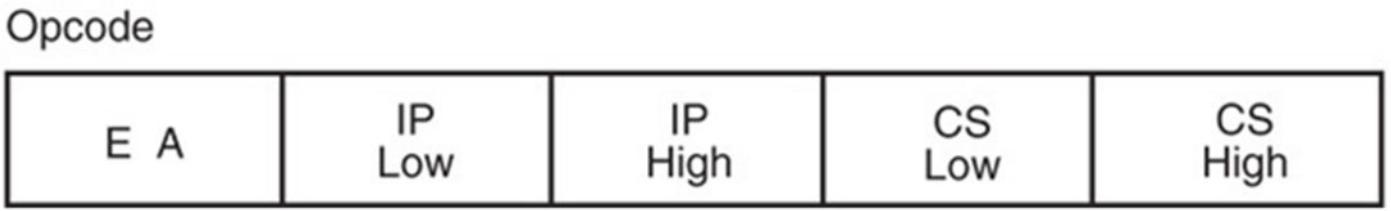
**Example: Encodings of short, near, and far jumps.**

```
0005  33 C0                          XOR AX, AX
0007  40                   Back:      INC AX
0008  EB 10                          JMP Forward
000A  B9 000A                        MOV CX, 10
000D  E9 000A                        JMP Near PTR Forward
0010  B9 0014                        MOV cx, 20
0013  EA ---- 001A R                 JMP Far PTR Forward
0018  8B C1                          MOV AX, CX
001A  03 C0                Forward:  ADD AX, AX
001C  EB E9                          JMP Back
```

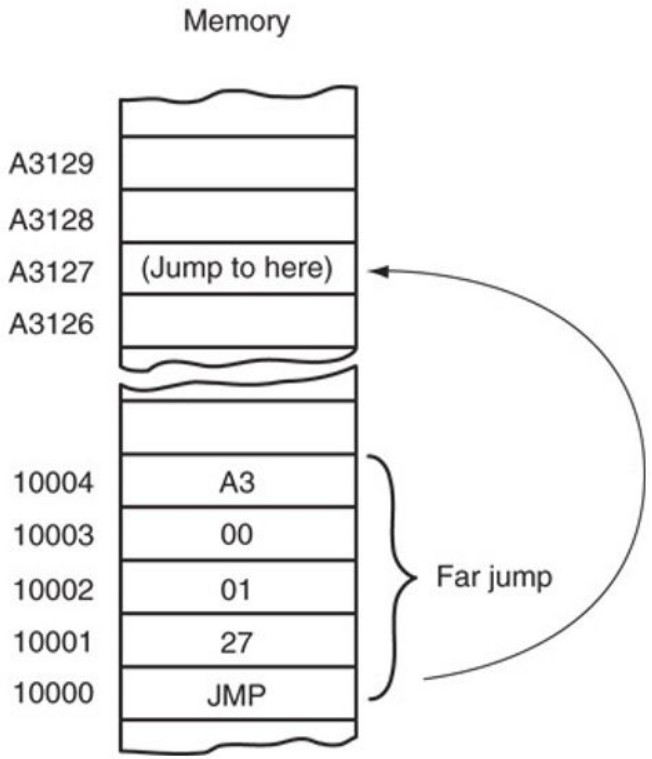
| | | | | | |
|---|---|---|---|---|---|
| **Short JMP** | **OPCODE (EBH)** | **DISP** | | | |
| **Near JMP** | **OPCODE (E9H)** | **IP Low** | **IP High** | | |
| **Intersegment JMP** | **OPCODE (EAH)** | **IP Low** | **IP High** | **CS Low** | **CS High** |

# Far Jump

- Obtains a new segment and offset address to accomplish the jump:
- bytes 2 and 3 of this 5-byte instruction contain the new offset address
- bytes 4 and 5 contain the new segment address



A far jump instruction replaces the contents of both CS and IP with 4 bytes following the opcode.



# Conditional Jump

- Always short jumps in 8086 - 80286.
- limits range to within +127 and –128 bytes from the location following the conditional jump
- Conditional jump instructions test flag bits:
  - sign (S), zero (Z), carry ©
  - parity (P), overflow (0)

| Mnemonic | Condition Tested | "Jump IF ..." |
|---|---|---|
| JA/JNBE | (CF = 0) and (ZF = 0) | above/not below nor zero |
| JAE/JNB | CF = 0 | above or equal/not below |
| JB/JNAE | CF = 1 | below/not above nor equal |
| JBE/JNA | (CF or ZF) = 1 | below or equal/not above |
| JC | CF = 1 | carry |
| JE/JZ | ZF = 1 | equal/zero |
| JG/JNLE | ((SF xor OF) or ZF) = 0 | greater/not less nor equal |
| JGE/JNL | (SF xor OF) = 0 | greater or equal/not less |
| JL/JNGE | (SF xor OR) = 1 | less/not greater nor equal |
| JLE/JNG | ((SF xor OF) or ZF) = 1 | less or equal/not greater |
| JNC | CF = 0 | not carry |
| JNE/JNZ | ZF = 0 | not equal/not zero |
| JNO | OF = 0 | not overflow |
| JNP/JPO | PF = 0 | not parity/parity odd |
| JNS | SF = 0 | not sign |
| JO | OF = 1 | overflow |
| JP/JPE | PF = 1 | parity/parity equal |
| JS | SF = 1 | sign |

# Loops

Open loop slide from lecture from 62 too 65.

# Procedures

- The syntax for procedure declaration:

```
name PROC

    ; here goes the code
    ; of the procedure ...

    RET
name ENDP
```

- CALL name: used to call a proc

- Example:

```
ORG    100h

CALL   m1

MOV    AX, 2

RET                ; return to operating
system.

m1     PROC

MOV    BX, 5

RET                ; return to caller.

m1     ENDP

END
```

# Procedures

- CALL pushes the address of the instruction following the CALL (**return address**) on the **stack**.
- RET instruction **removes an address** from the stack so the program returns to the instruction following the CALL.

# Macros

- The syntax for macros declaration:

  Macro definition:

  name MACRO [parameters,...]

  <instructions>

  ENDM

- name: used to call a macro

- Example:

```
MyMacro   MACRO  p1, p2, p3
          MOV AX, p1
          MOV BX, p2
          MOV CX, p3
ENDM
ORG 100h
MyMacro 1, 2, 3
MyMacro 4, 5, DX
RET
```

The above code is expanded into:

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```