

---

## **Week 5**

# **8088/8086 Microprocessor Programming II**

# Flag Control Instructions

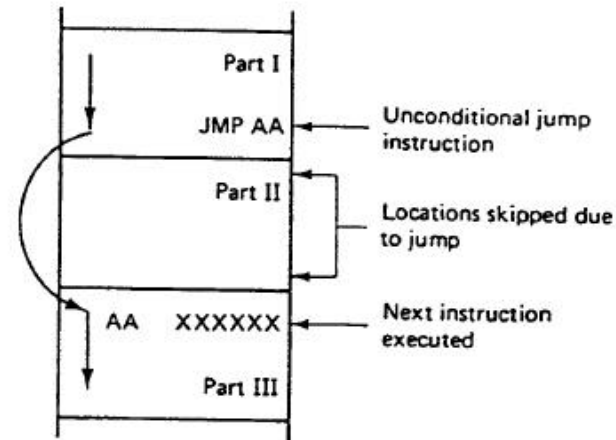
---

SF	ZF		AF		PF		CF
----	----	--	----	--	----	--	----

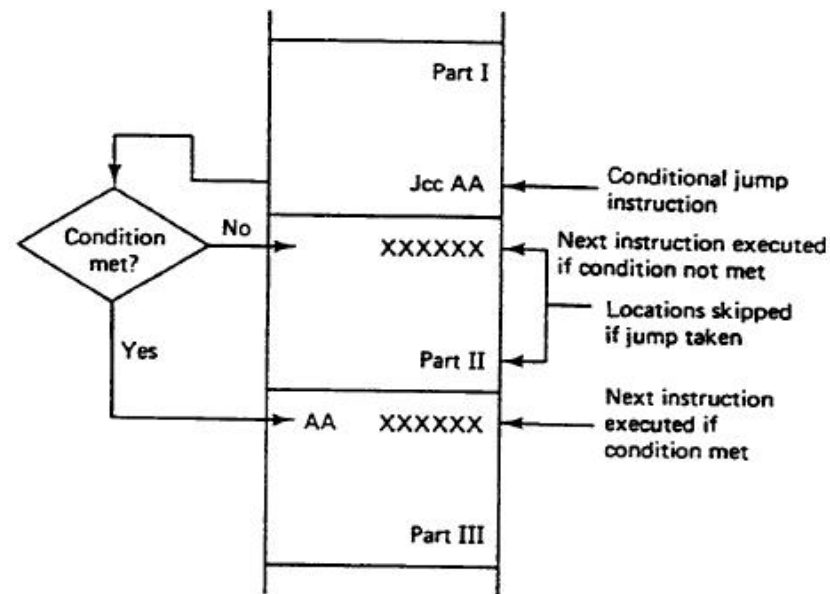
- LAHF Load AH from flags  $(AH) \leftarrow (\text{Flags})$
- SAHF Store AH into flags  $(\text{Flags}) \leftarrow (AH)$ 
  - Flags affected: SF, ZF, AF, PF, CF
- CLC Clear Carry Flag  $(CF) \leftarrow 0$
- STC Set Carry Flag  $(CF) \leftarrow 1$
- CLI Clear Interrupt Flag  $(IF) \leftarrow 0$
- STI Set interrupt flag  $(IF) \leftarrow 1$
- Example (try with debug)  
LAHF  
MOV [MEM1], AH  
MOV ah, [MEM2]  
SAHF  
; MEM1 = 150 (FF) MEM2 = 151 (01)

# Jump Instructions

- Unconditional vs conditional jump



(a)



(b)

Example. `JMP [BX]`  
Assume `BX = 1000h`  
`Ds:1000 = 200h`

# Compare

---

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) – (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

# Conditional Jump

- Above and below for comparison of unsigned numbers
- Less and greater for comparison of signed numbers

Mnemonic	Meaning	Format	Operation	Flags affected
Jcc	Conditional jump	Jcc Operand	If the specified condition cc is true the jump to the address specified by the operand is initiated; otherwise the next instruction is executed.	None

(a)

Mnemonic	Meaning	Condition
JA	above	CF = 0 and ZF = 0
JAЕ	above or equal	CF = 0
JB	below	CF = 1
JBE	below or equal	CF = 1 or ZF = 1
JC	carry	CF = 1
JCXZ	CX register is zero	(CF or ZF) = 0
JE	equal	ZF = 1
JG	greater	ZF = 0 and SF = OF
JGE	greater or equal	SF = OF
JL	less	(SF xor OF) = 1
JLE	less or equal	((SF xor OF) or ZF) = 1
JNA	not above	CF = 1 or ZF = 1
JNAE	not above nor equal	CF = 1
JNB	not below	CF = 0
JNBE	not below nor equal	CF = 0 and ZF = 0
JNC	not carry	CF = 0
JNE	not equal	ZF = 0
JNG	not greater	((SF xor OF) or ZF) = 1
JNGE	not greater nor equal	(SF xor OF) = 1
JNL	not less	SF = OF
JNLE	not less nor equal	ZF = 0 and SF = OF
JNO	not overflow	OF = 0
JNP	not parity	PF = 0
JNS	not sign	SF = 0
JNZ	not zero	ZF = 0
JO	overflow	OF = 1
JP	parity	PF = 1
JPE	parity even	PF = 1
JPO	parity odd	PF = 0
JS	sign	SF = 1
JZ	zero	ZF = 1

(b)

# Subroutines and Subroutine Handling Functions

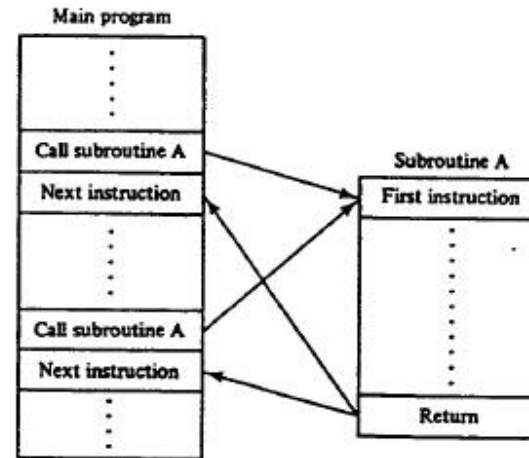
A subroutine is a special segment of a program that be called for execution from any point in the program

A return instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment

Examples. Call 1234h  
Call BX  
Call [BX]

Two calls

intradsegment  
intersegment



(a)

Mnemonic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

(b)

Operand
Near-proc
Far-proc
Memptr16
Regptr16
Memptr32

(c)

Figure 6-20 (a) Subroutine concept. (b) Subroutine call instruction. (c) Allowed operands.

# Example

---

- Write an 8086 program that adds two packed BCD numbers input from the keyboard and computes and displays the result on the system video monitor
- Data should be in the form 64+89= The answer 153 should appear in the next line.

Mov dx, buffer address

Mov ah,0a

Mov si,dx

Mov byte ptr [si], 8

Int 21

Mov ah,0eh

Mov al,0ah

Int 10 ; BIOS service 0e line feed position cursor

## Example Continued

---

```
sub byte ptr[si+2], 30h  
sub byte ptr[si+3], 30h  
sub byte ptr[si+5], 30h  
sub byte ptr[si+6], 30h
```

```
Mov cl,4  
Rol byte ptr [si+3],cl  
Rol byte ptr [si+6],cl  
Ror word ptr [si+2], cl  
Ror word ptr [si+2], cl
```

```
Mov al, [si+3]  
Add al, [si+6]  
Daa  
Mov bh,al  
Jnc display  
Mov al,1  
Call display  
Mov al,bh  
Call display  
Int 20
```

# Display Subroutine

---

```
mov    bl,al           ;Save original number
and    al,f0           ;Force bits 0-3 low
mov    cl,4            ;Four rotates
ror    al,cl           ;Rotate MSD into LSD
add    al,30           ;Convert to ASCII
mov    ah,0e           ;BIOS video service 0E
int     10             ;Display character
mov    al,bl           ;Recover original number
and    al,0f           ;Force bits 4-7 low
add    al,30           ;Convert to ASCII
int     10             ;Display character
ret                  ;Return to calling program
;
```

;Input buffer begins here

## Push S

$$\begin{aligned} (\text{SP}) &\leftarrow (\text{SP}) - 2 \\ ((\text{SP})) &\leftarrow (\text{S}) \end{aligned}$$

Pop D

$$\begin{aligned} (D) &\leftarrow ((SP)) \\ (SP) &\leftarrow (SP) + 2 \end{aligned}$$

# Loop and Loop Handling Instructions

---

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	$(CX) \leftarrow (CX) - 1$ Jump is initiated to location defined by short-label if $(CX) \neq 0$ ; otherwise, execute next sequential instruction
LOOPE/LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ Short-label	$(CX) \leftarrow (CX) - 1$ Jump to location defined by short-label if $(CX) \neq 0$ and $(ZF) = 1$ ; otherwise, execute next sequential instruction
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ Short-label	$(CX) \leftarrow (CX) - 1$ Jump to location defined by short-label if $(CX) \neq 0$ and $(ZF) = 0$ ; otherwise, execute next sequential instruction

Figure 6-28 Loop instructions.

# Loop

---

```

NEXT:  MOV CX,COUNT
        .
        .
        .
        .
        .
        .
        LOOP NEXT

```

Load count for the number of repeats

Body of routine that is repeated

Loop back to label NEXT if count not zero

(a)

```

MOV     AX,DATASEGADDR
MOV     DS,AX
MOV     SI,BLK1ADDR
MOV     DI,BLK2ADDR
MOV     CX,N
NXTPT:  MOV     AH,[SI]
        MOV     [DI],AH
        INC     SI
        INC     DI
        LOOP    NXTPT
        HLT

```

(b)

# String Instructions

---

8088 is equipped with special instructions to handle string operations

String: A series of data words (or bytes) that reside in consecutive memory locations

Examples: move, scan, compare

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/MOVSW	$((ES)0 + (DI)) \leftarrow ((DS)0 + (SI))$ $(SI) \leftarrow (SI) \pm 1 \text{ or } 2$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	None
CMPS	Compare string	CMPSB/CMPSW	Set flags as per $((DS)0 + (SI)) - ((ES)0 + (DI))$ $(SI) \leftarrow (SI) \pm 1 \text{ or } 2$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCASB/SCASW	Set flags as per $(AL \text{ or } AX) - ((ES)0 + (DI))$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODSB/LODSW	$(AL \text{ or } AX) \leftarrow ((DS)0 + (SI))$ $(SI) \leftarrow (SI) \pm 1 \text{ or } 2$	None
STOS	Store string	STOSB/STOSW	$((ES)0 + (DI)) \leftarrow (AL \text{ or } AX) \pm 1 \text{ or } 2$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	None

Figure 6-32 Basic string instructions.

# Example

---

```

                                MOV     AX,DATASEGADDR
                                MOV     DS,AX
                                MOV     ES,AX
                                MOV     SI,BLK1ADDR
                                MOV     DI,BLK2ADDR
                                MOV     CX,N
                                CLD
NXTPT:  MOVSB
                                LOOP    NXTPT
                                HLT
```

# Repeat String REP

---

Basic string operations must be repeated in order to process arrays of data; this is done by inserting a repeat prefix.

Prefix	Used with:	Meaning
REP	MOVS STOS	Repeat while not end of string $CX \neq 0$
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal $CX \neq 0$ and $ZF = 1$
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal $CX \neq 0$ and $ZF = 0$

**Figure 6–36** Prefixes for use with the basic string operations.

# Direction Flag

---

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	$(DF) \leftarrow 0$	DF
STD	Set DF	STD	$(DF) \leftarrow 1$	DF

DF = 0 autoincrement mode

DF = 1 autodecrement mode

Example

Mov ax, 0

Mov ds,ax

Mov es,ax

Mov al,05

Mov di,a000h

Mov cx, 0fh

CLD

Repstosb

Initializing a block of memory by  
repeating the stosb instruction

## Example. Find and replace

---

- Write a program that scans the name “Mr. Gones” and replaces the “G” with the letter “J”.

```
Data1 db "Mr. Gones",'$'  
      mov ax,ds  
      mov es,ax  
      cld  
      mov di, offset data1  
      mov cx,09  
      mov al,'G'  
      repne scasb  
      jne over  
      dec di  
      mov byte ptr[di], 'J'  
Over:  mov ah,09  
      mov dx,offset data1  
      int 21h
```

## Example. Display the ROM BIOS Date

---

- Write an 8086 program that searches the BIOS ROM for its creation date and displays that date on the monitor.
- If a date cannot be found display the message “date not found”
- Typically the BIOS ROM date is stored in the form xx/xx/xxxx beginning at system address F000:FFF5
- Each character is in ASCII form and the entire string is terminated with the null character (00)
- First we scan the null character, if the null character is found in the first eight characters, an invalid date is assumed
- If not, the entire string is copied into RAM
- Add a '\$' character to the end of the string and make it ready for DOS function 09, INT 21

**I. Locate Date String in ROM**

- A. Save Es
- B. Search for null byte
  - 1. Point ES:DI to F000:FFF5
  - 2. Direction flag = auto increment
  - 3. 12 bytes to scan
  - 4. Scan for null character
  - 5. Null Found?
    - i. No: Goto ILE
    - ii. Yes: Continue
- C. Compute length of string and save in CX
  - 1. Subtract FFF5 from address of null character
- D. Make sure string is the correct format.
  - 1. Found null at character 1-8?
    - i. Yes: Goto ILE
    - ii. No: Continue

**II. Display Date String**

- A. Add date string to message header
  - 1. Recover ES
  - 2. Save DS
  - 3. Point DS:SI to F000:FFF5
  - 4. Point ES:DI to end of message header
  - 5. Copy date string to end of header
- B. Recover DS and append end of message (\$) to the full string
- C. Display the complete message (header string plus date string)
  - 1. Point DS:DX to start of header
  - 2. DOS INT 21, function 9
- D. Goto III.A
- E. Display date not found message
  - 1. Recover DS
  - 2. Point DS:DX to date not found string
  - 3. DOS INT 21, function 9

**III. Return to DOS**

- A. INT 20H

**IV. Set up Messages**

- A. Header: "Your ROM BIOS is dated:"
- B. Date: "xx/xx/xxxx"
- C. Not found: "Date not found"

```

push    es

mov     ax,f000
mov     es,ax
mov     di,fff5
cld
mov     cx,0c
mov     al,0
repne   scasb
jnz     II.E
mov     cx,di
sub     cx,fff5
cmp     cx,9
jb      II.E

pop     es
push    ds
mov     ax,f000
mov     ds,ax
mov     si,fff5
mov     di,start_of_date_string
rep     movsb
pop     ds
mov     [di],'$'

mov     dx,start_of_header_string
mov     ah,9
int     21
jmp     III.A
II.E    pop     ds
        mov     dx,start_of_not_found_string
        mov     ah,9
        int     21

III.A   int     20

start_of_header:
        db      "Your ROM BIOS is dated"
start_of_date_string:
        db      "xx/xx/xxxx"

```

## Example. Binary (Hex) to ASCII (Decimal) Conversion

---

- $34Dh = 3 \times 256 + 4 \times 16 + 13 \times 1$   
 $= 845$
- $34Dh / A = 84$  remainder 5
- $84h / A = 8$  remainder 4
- $8 < A$  the process stops
- Taking the remainders in reverse order gives : 845 decimal
- Write a program to convert a word sized hex number in data item BINNUM
- The result will be five digits, each digit will be converted to ASCII and placed in ASCNUM, the lowest digit will be in high memory as the convention of ASCII storage in DOS

# Program

---

	BINNUM	DW 34dh
	ASCNUM	DB 5 DUP('0')
	MOV	BX, 10
	MOV SI	OFFSET ASCNUM
	ADD	SI,5
	DEC	SI
	MOV	AX,BINNUM
BACK:	SUB	DX, DX
	DIV	BX
	OR	DL,30h
	MOV	[SI], DL
	DEC	SI
	CMP	AX,0
	JA	BACK
	INT	20h