Chapter 6

Real-Mode 80386DX Microprocessor Programming 2

Part 1

Introduction

6.2 Flag control Instructions

6.3 Compare and Set Instructions

6.4 Jump Instructions

6.5 Subroutines and Subroutine-Handling Instructions

6.6 The Loop and Loop-Handling Instructions

6.7 Strings and Sting-Handling Instructions

Flag Control Instructions- Loading, Storing, and Modifying Flags

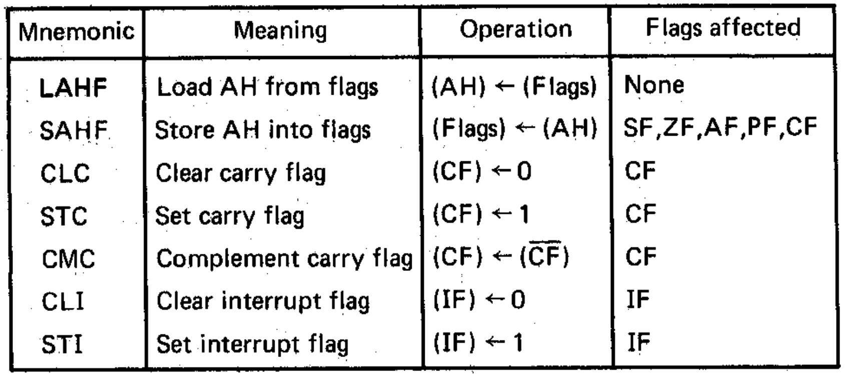
- LAHF/SAHF→ Load/save control flags
- CLC/STC/CMC → Modify carry
- CLI/STI → Modify interrupt flag

- Modifying the carry flag—CLC/STC/CMC
  - Used to initialize the carry flag
  - Clear carry flag

    CLC ; 0 → (CF)
  - Set carry flag

    STC ; 1 → (CF)
  - Complement carry flag

    CMC ; (CF*) → (CF)



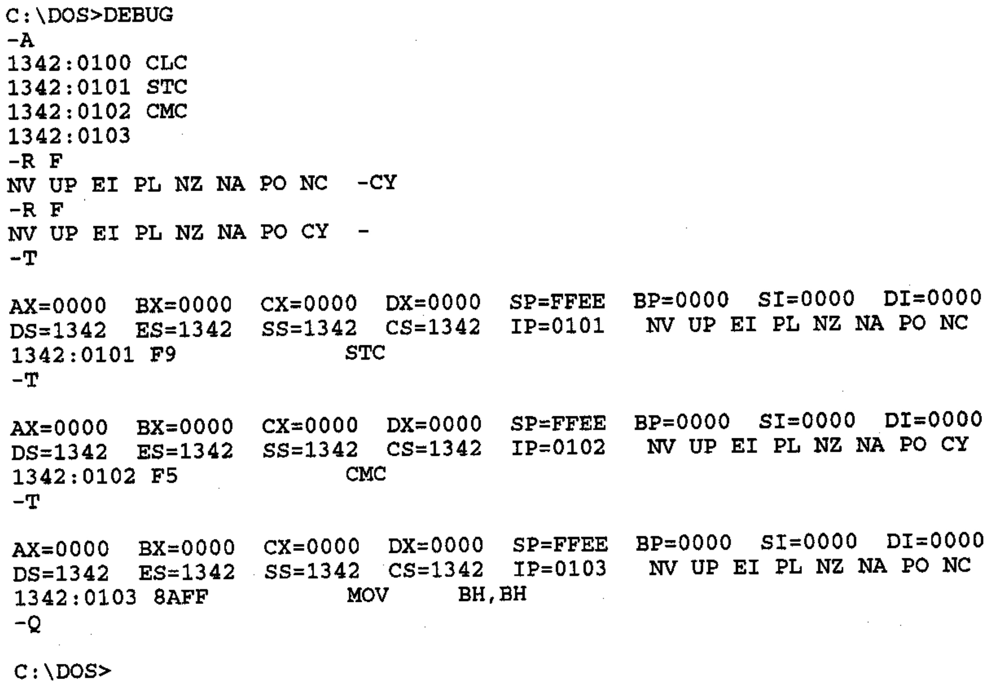
Mnemonic	Meaning	Operation	Flags affected
LAHF	Load AH from flags	(AH) ← (Flags)	None
SAHF	Store AH into flags	(Flags) ← (AH)	SF,ZF,AF,PF,CF
CLC	Clear carry flag	(CF) ← 0	CF
STC	Set carry flag	(CF) ← 1	CF
CMC	Complement carry flag	(CF) ← ($\overline{\text{CF}}$)	CF
CLI	Clear interrupt flag	(IF) ← 0	IF
STI	Set interrupt flag	(IF) ← 1	IF

- Modifying the interrupt flag—CLI/STI
  - Used to turn on/off external hardware interrupts
  - Clear interrupt flag

    CLI ; 0 → (IF) Disable interrupts
  - Set interrupt flag

    STI ; 1 → (IF) Enable interrupts

Flag Control Instructions- Example



```
C:\DOS>DEBUG
-A
1342:0100 CLC
1342:0101 STC
1342:0102 CMC
1342:0103
-R F
NV UP EI PL NZ NA PO NC  -CY
-R F
NV UP EI PL NZ NA PO CY  -
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0101   NV UP EI PL NZ NA PO NC
1342:0101 F9            STC
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0102   NV UP EI PL NZ NA PO CY
1342:0102 F5            CMC
-T

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0103   NV UP EI PL NZ NA PO NC
1342:0103 8AFF          MOV     BH,BH
-Q

C:\DOS>
```

- Debug flag notation
  - CF → CY = 1, NC = 0
- Example—Execution of carry flag modification instructions

  CY=1

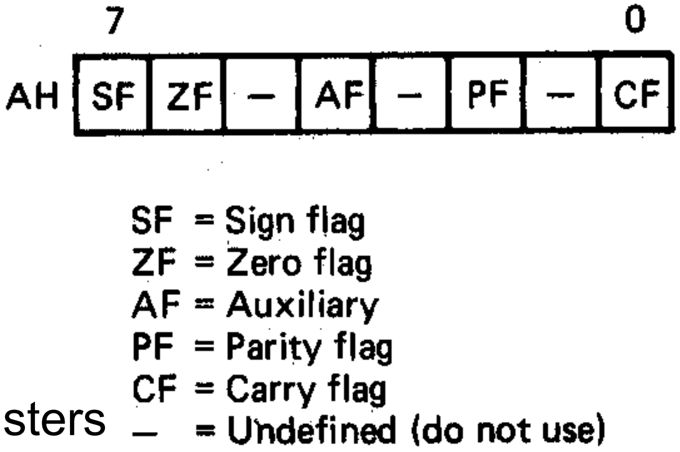
  CLC ;Clear carry flag

  STC ;Set carry flag

  CMC ;Complement carry flag

Loading and Saving the Flag Register

- All loads and stores of flags take place through the AH register
  - Format of the flags in the AH register
    - B0 = CF
    - B2 = PF
    - B4 = AF
    - B6 = ZF
    - B7 = SF



	7							0
AH	SF	ZF	–	AF	–	PF	–	CF

SF = Sign flag
ZF = Zero flag
AF = Auxiliary
PF = Parity flag
CF = Carry flag
– = Undefined (do not use)

- Load the AH register with the content of the flags registers

  LAHF

  (Flags) → (AH)

  Flags unchanged

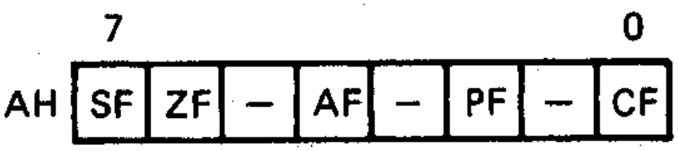
- Store the content of AH in the flags register—SAHF

  SAHF

  (AH) → (Flags)

  SF,ZF,AF,PF,CF → updated

Loading and Saving the Flag Register



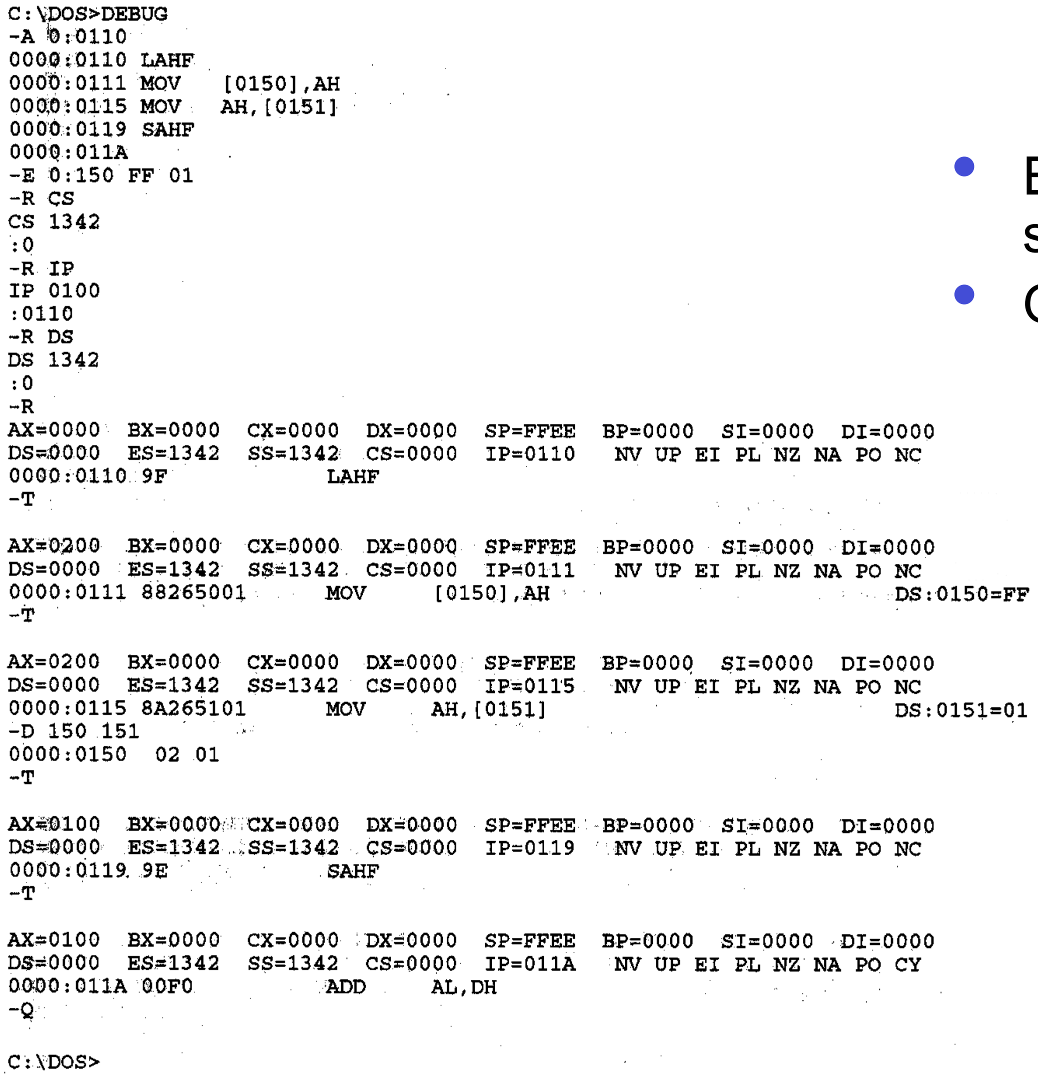
	7							0
AH	SF	ZF	–	AF	–	PF	–	CF

SF = Sign flag
ZF = Zero flag
AF = Auxiliary
PF = Parity flag
CF = Carry flag
– = Undefined (do not use)

- Application—saving a copy of the flags and initializing with new values

```
LAHF            ;Load of flags into AH
MOV [MEM1],AH   ;Save old flags at address MEM1
MOV AH,[MEM2]   ;Read new flags from MEM2 into AH
SAHF            ;Store new flags in flags register
```

Flag Control Instructions- Example



```
C:\DOS>DEBUG
-A 0:0110
0000:0110 LAHF
0000:0111 MOV    [0150],AH
0000:0115 MOV    AH,[0151]
0000:0119 SAHF
0000:011A
-E 0:150 FF 01
-R CS
CS 1342
:0
-R IP
IP 0100
:0110
-R DS
DS 1342
:0
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=0110   NV UP EI PL NZ NA PO NC
0000:0110 9F            LAHF
-T

AX=0200  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=0111   NV UP EI PL NZ NA PO NC
0000:0111 88265001      MOV     [0150],AH                          DS:0150=FF
-T

AX=0200  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=0115   NV UP EI PL NZ NA PO NC
0000:0115 8A265101      MOV     AH,[0151]                          DS:0151=01
-D 150 151
0000:0150  02 01
-T

AX=0100  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=0119   NV UP EI PL NZ NA PO NC
0000:0119 9E            SAHF
-T

AX=0100  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=011A   NV UP EI PL NZ NA PO CY
0000:011A 00F0          ADD     AL,DH
-Q

C:\DOS>
```

- Example—Execution of the flags save and initialization sequence
- Other flag notation:

  Flag = 1/0

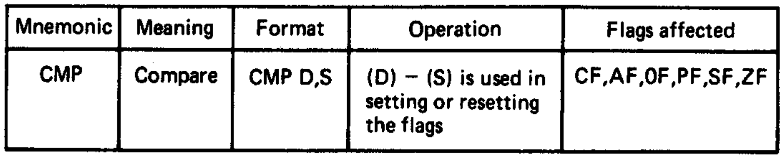
  SF = NG/PL

  ZF = ZR/NZ

  AF = AC/NA

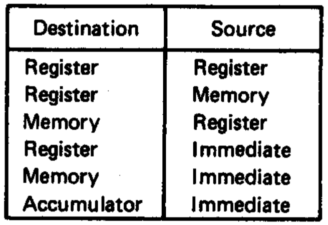
  PF = PE/PO

Compare Instructions



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)

- Used to compare two values of data and update the state of the flags to reflect their relationship
- General format: CMP D,S
- Operation: Compares the content of the source to the destination

  (D) - (S) → Flags updated to reflect relationship

  - Source and destination contents unchanged
  - Allowed operand variations:
    - Values in two registers
    - Values in a memory location and a register
    - Immediate source operand and a value in a register or memory
  - Allows SW to perform conditional control flow—typically testing of a flag by jump instruction
    - ZF = 1 → D = S = Equal
    - ZF = 0, CF = 1 → D < S = Unequal, less than
    - ZF = 0, CF = 0 → D > S = Unequal, greater than

Compare Instructions- Example

- Example—Initialization of internal registers with immediate data and compare. Example:

```
MOV AX,1234H      ;Initialize AX
MOV BX,ABCDH      ;Initialize BX
CMP AX,BX         ;Compare AX-BX
```

  - Data registers AX and BX initialized from immediate data

    IMM16 → (AX) = 1234H → + integer

    IMM16 → (BX) = ABCDH → - integer

  - Compare computation performed as:

    (AX) = 0001001000110100_2 (BX) = 1010101111001101_2

    (AX) – (BX) = 0001001000110100_2 - 1010101111001101_2

    ZF = 0 = NZ

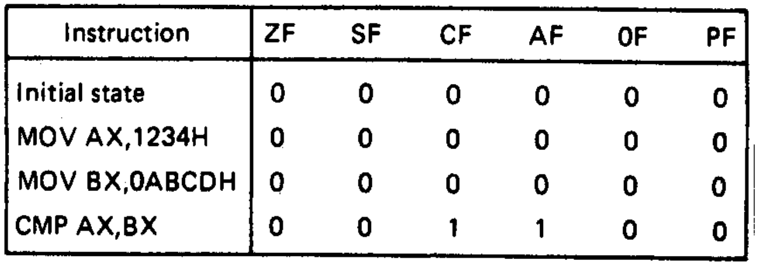
    SF = 0 = PL ;treats as signed numbers

    CF = 1 = CY

    AF = 1 = AC

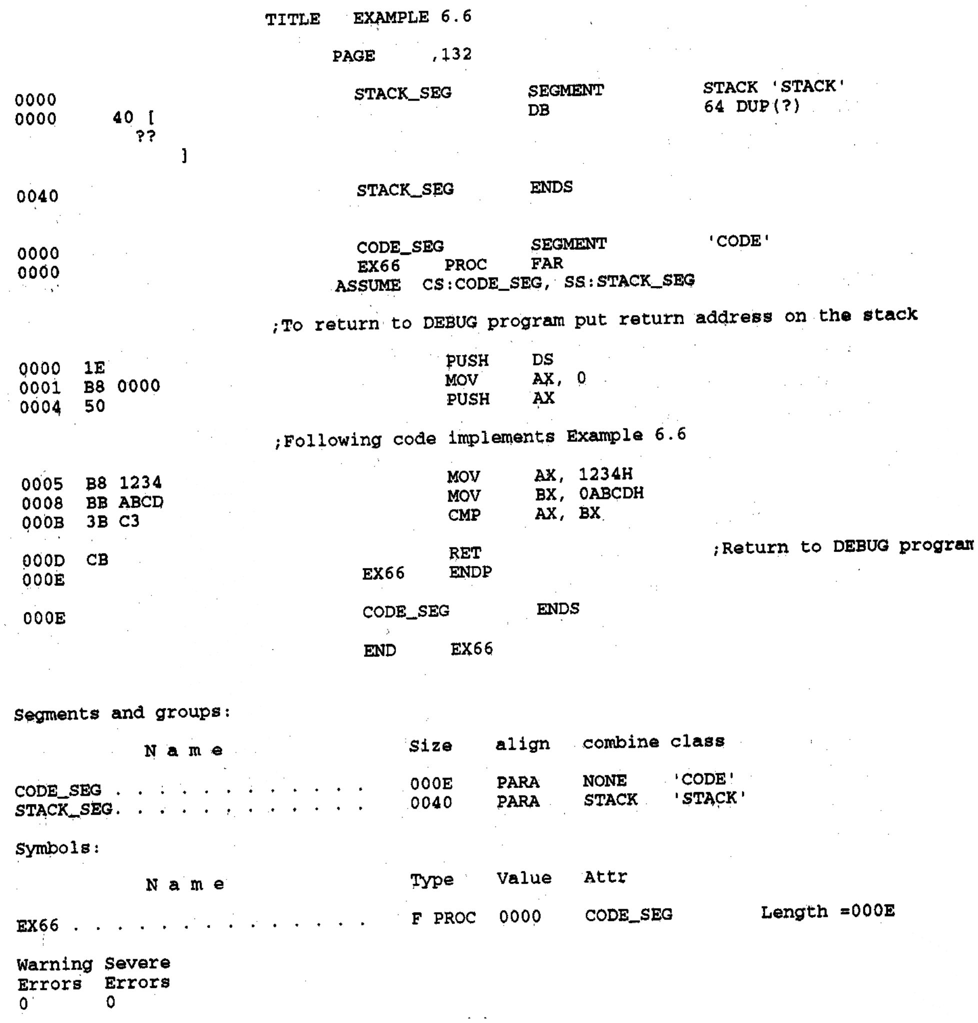
    OF = 0 = NV

    PF = 0 = PO

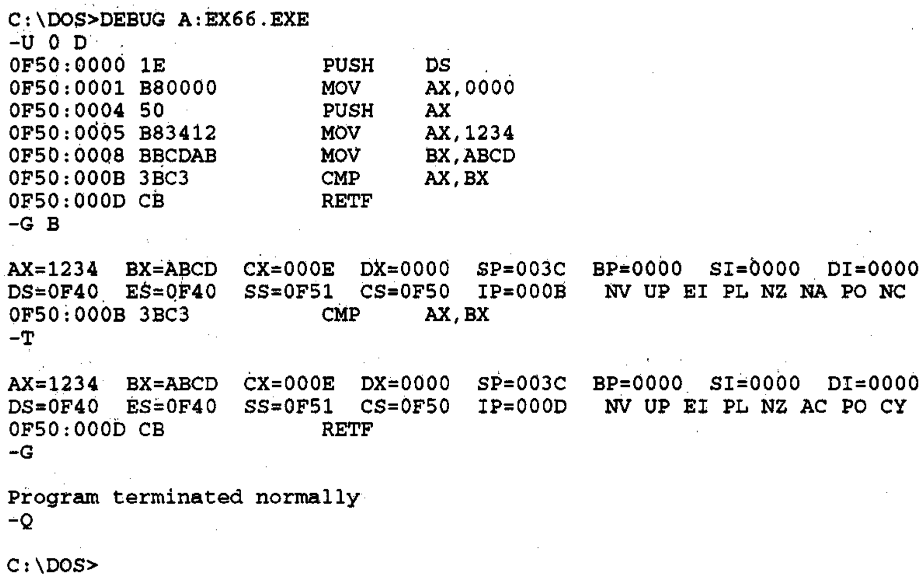


Instruction	ZF	SF	CF	AF	OF	PF
Initial state	0	0	0	0	0	0
MOV AX,1234H	0	0	0	0	0	0
MOV BX,0ABCDH	0	0	0	0	0	0
CMP AX,BX	0	0	1	1	0	0

Compare Instructions- Listing and Debug Execution

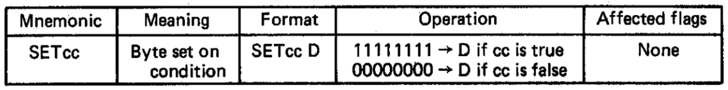


```
                          TITLE    EXAMPLE 6.6

                               PAGE      ,132

0000                              STACK_SEG        SEGMENT          STACK 'STACK'
0000      40 [                                     DB               64 DUP(?)
            ??
               ]

0040                              STACK_SEG        ENDS


0000                              CODE_SEG         SEGMENT          'CODE'
0000                              EX66      PROC   FAR
                              ASSUME   CS:CODE_SEG, SS:STACK_SEG

                          ;To return to DEBUG program put return address on the stack

0000   1E                                 PUSH     DS
0001   B8 0000                            MOV      AX, 0
0004   50                                 PUSH     AX

                          ;Following code implements Example 6.6

0005   B8 1234                            MOV      AX, 1234H
0008   BB ABCD                            MOV      BX, 0ABCDH
000B   3B C3                              CMP      AX, BX

000D   CB                                 RET                        ;Return to DEBUG program
000E                              EX66    ENDP

000E                              CODE_SEG         ENDS

                                  END     EX66


Segments and groups:

             N a m e                       Size    align    combine class

CODE_SEG . . . . . . . . . . . .           000E    PARA     NONE    'CODE'
STACK_SEG. . . . . . . . . . . .           0040    PARA     STACK   'STACK'

Symbols:

             N a m e                       Type    Value    Attr

EX66 . . . . . . . . . . . . . .           F PROC  0000     CODE_SEG       Length =000E

Warning Severe
Errors  Errors
0       0
```



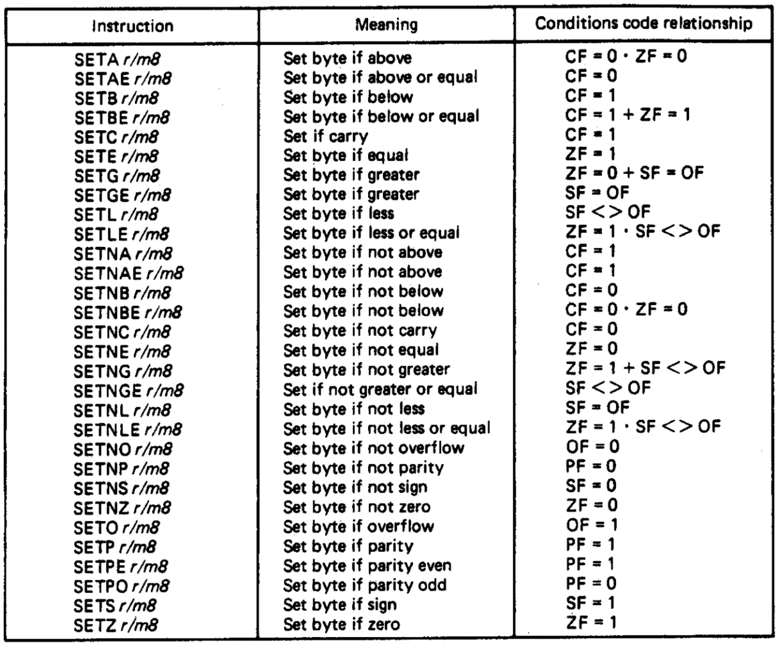
```
C:\DOS>DEBUG A:EX66.EXE
-U 0 D
0F50:0000 1E            PUSH    DS
0F50:0001 B80000        MOV     AX,0000
0F50:0004 50            PUSH    AX
0F50:0005 B83412        MOV     AX,1234
0F50:0008 BBCDAB        MOV     BX,ABCD
0F50:000B 3BC3          CMP     AX,BX
0F50:000D CB            RETF
-G B

AX=1234  BX=ABCD  CX=000E  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0F40  ES=0F40  SS=0F51  CS=0F50  IP=000B   NV UP EI PL NZ NA PO NC
0F50:000B 3BC3          CMP     AX,BX
-T

AX=1234  BX=ABCD  CX=000E  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0F40  ES=0F40  SS=0F51  CS=0F50  IP=000D   NV UP EI PL NZ AC PO CY
0F50:000D CB            RETF
-G

Program terminated normally
-Q

C:\DOS>
```

Byte Set on Condition Instruction



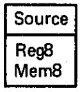
Mnemonic	Meaning	Format	Operation	Affected flags
SETcc	Byte set on condition	SETcc D	11111111 → D if cc is true 00000000 → D if cc is false	None

(a)



Instruction	Meaning	Conditions code relationship
SETA *r/m8*	Set byte if above	CF = 0 · ZF = 0
SETAE *r/m8*	Set byte if above or equal	CF = 0
SETB *r/m8*	Set byte if below	CF = 1
SETBE *r/m8*	Set byte if below or equal	CF = 1 + ZF = 1
SETC *r/m8*	Set if carry	CF = 1
SETE *r/m8*	Set byte if equal	ZF = 1
SETG *r/m8*	Set byte if greater	ZF = 0 + SF = OF
SETGE *r/m8*	Set byte if greater	SF = OF
SETL *r/m8*	Set byte if less	SF <> OF
SETLE *r/m8*	Set byte if less or equal	ZF = 1 · SF <> OF
SETNA *r/m8*	Set byte if not above	CF = 1
SETNAE *r/m8*	Set byte if not above	CF = 1
SETNB *r/m8*	Set byte if not below	CF = 0
SETNBE *r/m8*	Set byte if not below	CF = 0 · ZF = 0
SETNC *r/m8*	Set byte if not carry	CF = 0
SETNE *r/m8*	Set byte if not equal	ZF = 0
SETNG *r/m8*	Set byte if not greater	ZF = 1 + SF <> OF
SETNGE *r/m8*	Set if not greater or equal	SF <> OF
SETNL *r/m8*	Set byte if not less	SF = OF
SETNLE *r/m8*	Set byte if not less or equal	ZF = 1 · SF <> OF
SETNO *r/m8*	Set byte if not overflow	OF = 0
SETNP *r/m8*	Set byte if not parity	PF = 0
SETNS *r/m8*	Set byte if not sign	SF = 0
SETNZ *r/m8*	Set byte if not zero	ZF = 0
SETO *r/m8*	Set byte if overflow	OF = 1
SETP *r/m8*	Set byte if parity	PF = 1
SETPE *r/m8*	Set byte if parity even	PF = 1
SETPO *r/m8*	Set byte if parity odd	PF = 0
SETS *r/m8*	Set byte if sign	SF = 1
SETZ *r/m8*	Set byte if zero	ZF = 1

(b)



Source
Reg8 Mem8

(c)

Byte set on condition instruction

- Used to test results in the flags, such as those of a compare operation, for a specific conditional relationships and then produce a logical result of True or False reflecting the result
- General format:

  SETcc D

  cc = one of the supported conditional relationships

Byte Set on Condition Instruction

- Operation: Flags tested for conditions defined by “cc” and the destination in a register or memory updated as follows

  If cc test True:

  11111111_2 = FFH → D

  If cc test False:

  00000000_2 = 00H → D

- Examples of conditional tests:

  SETE = set byte if equal → ZF = 1

  SETC = set byte if carry → CF =1

  SETBE = set byte if below or equal → CF = 1 +(or) ZF = 1

- Example: SETA AL = set byte if above

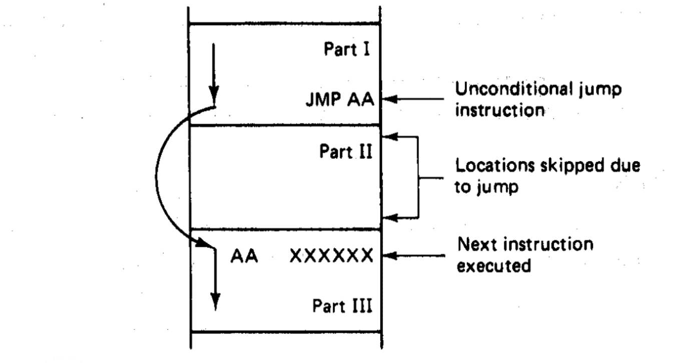
  if CF = 0 • (and) ZF = 0

  (AL) = FFH

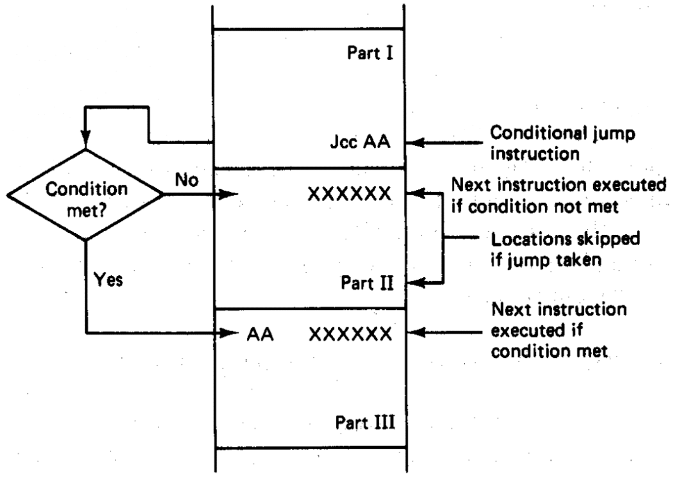
  Otherwise,

  (AL) =00H

Jump Instructions


Part I
JMP AA
Unconditional jump instruction
Part II
Locations skipped due to jump
AA XXXXXX
Next instruction executed
Part III

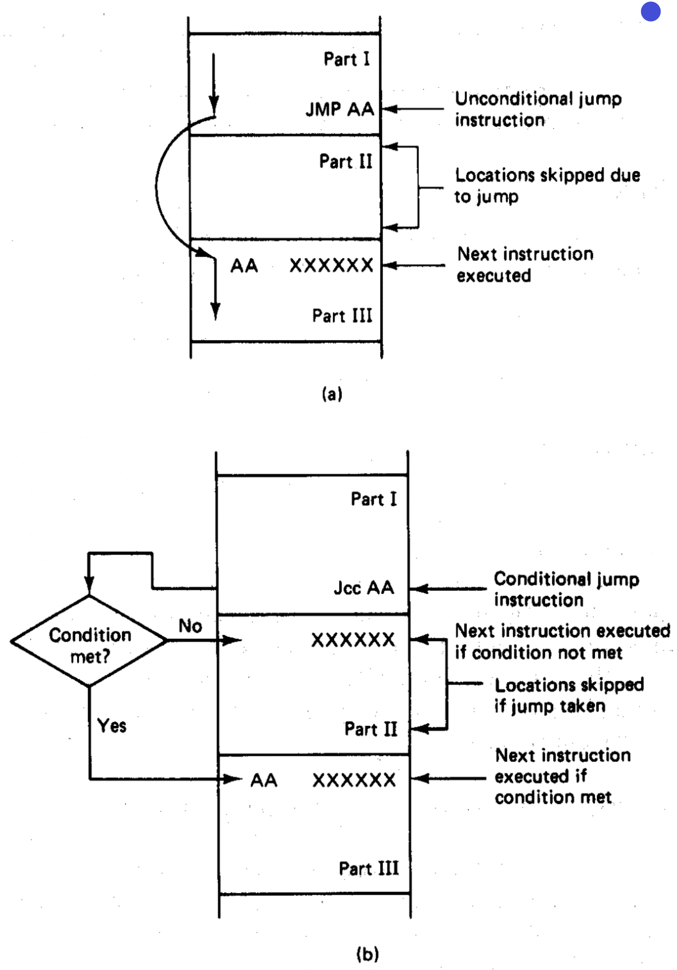
(a)


Part I
Jcc AA
Conditional jump instruction
Condition met?
No
XXXXXX
Next instruction executed if condition not met
Locations skipped if jump taken
Yes
Part II
AA XXXXXX
Next instruction executed if condition met
Part III

(b)

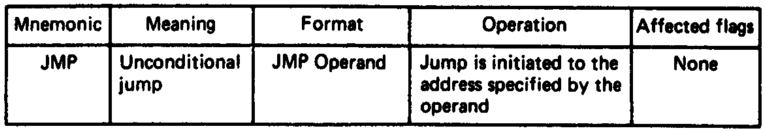
- Jump operation alters the execution path of the instructions in the program—flow control
  - Unconditional Jump
    - Always takes place
    - No status requirements are imposed
    - Example
      - JMP AA instructions in part I executed
      - Control passed to next instruction identified by AA in Part III
      - Instructions in Part II skipped

Jump Instructions


Part I
JMP AA
Unconditional jump instruction
Part II
Locations skipped due to jump
AA XXXXXX
Next instruction executed
Part III
(a)
Part I
Jcc AA
Conditional jump instruction
Condition met?
No
XXXXXX
Next instruction executed if condition not met
Locations skipped if jump taken
Yes
Part II
AA XXXXXX
Next instruction executed if condition met
Part III
(b)

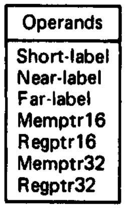
- Conditional jump
  - May or may not take place
  - Status conditions must be satisfied
  - Example
    - Jcc AA instruction in Part 1 executed
    - Conditional relationship specified by cc is evaluated
    - If conditions met, jump takes place and control is passed to next instruction identified by AA in Part III
    - Otherwise, execution continues sequentially with first instruction in Part II
  - Condition cc specifies a relationship of status flags such as CF, PF, ZF, etc.

Unconditional Jump Instruction



Mnemonic	Meaning	Format	Operation	Affected flags
JMP	Unconditional jump	JMP Operand	Jump is initiated to the address specified by the operand	None

(a)



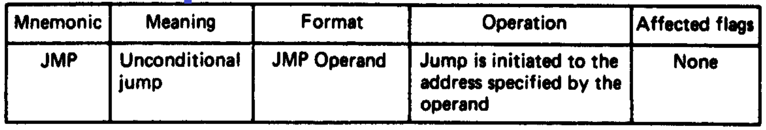
Operands
Short-label Near-label Far-label Memptr16 Regptr16 Memptr32 Regptr32

(b)

- Unconditional jump instruction
  - Implements the unconditional jump operation needed by:
    - Branch program control flow structures
    - Loop program control flow structures
  - General format:

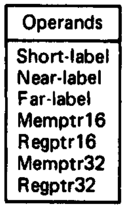
    JMP Operand

Types of Unconditional Jump Instruction



Mnemonic	Meaning	Format	Operation	Affected flags
JMP	Unconditional jump	JMP Operand	Jump is initiated to the address specified by the operand	None

(a)



Operands
Short-label Near-label Far-label Memptr16 Regptr16 Memptr32 Regptr32

(b)

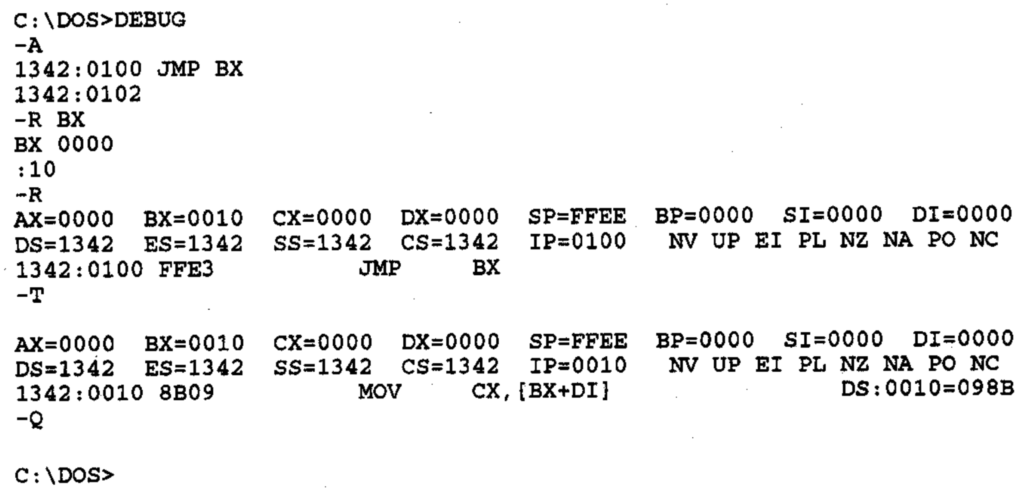
- Types of unconditional jumps
  - Intrasegment—branch to address is located in the current code segment
    - Only IP changes value
    - short-label
      - 8-bit signed displacement coded into the instruction
      - Immediate addressing
      - Range equal –126 to +129
      - New address computed as:

        (Current IP) + short-label → (IP)

        Jump to address = (Current CS):(New IP)
    - near-label
      - 16-bit signed IP offset coded in the instruction
      - Example

        JMP 1234H

regptr16 Unconditional Jump Example



```
C:\DOS>DEBUG
-A
1342:0100 JMP BX
1342:0102
-R BX
BX 0000
:10
-R
AX=0000  BX=0010  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 FFE3          JMP     BX
-T

AX=0000  BX=0010  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0010   NV UP EI PL NZ NA PO NC
1342:0010 8B09          MOV     CX,[BX+DI]                   DS:0010=098B
-Q

C:\DOS>
```

- regptr16
  - 16-bit value of IP specified as the content of a register
  - Register addressing
  - Operation:

    (BX) → (IP)

  Jump to address = (Current (CS):(New IP)
- Example

  1342:0100 JMP BX
  - Prior to execution

    (IP) = 0100H

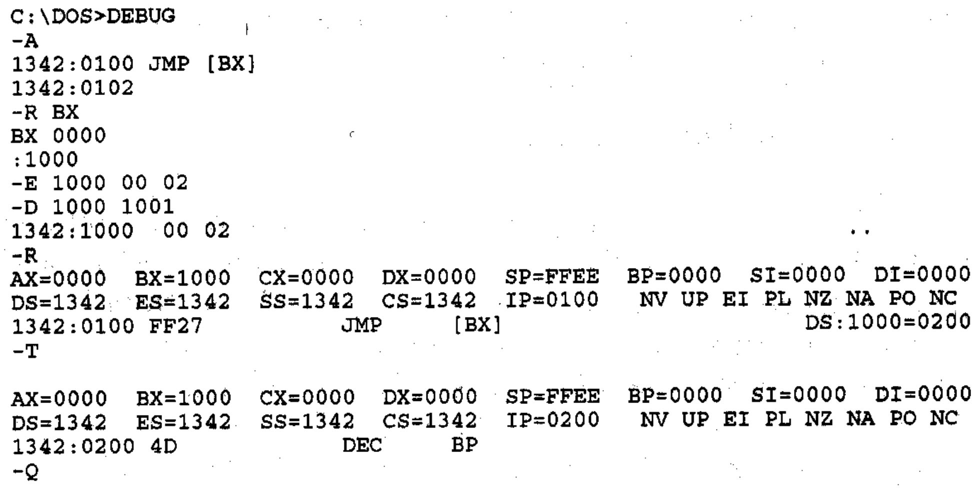
    (BX) =0010H
  - After execution

    (IP) =0010H

  Address of next instruction

  (CS:IP) = 1342:0010

memptr16 Unconditional Jump Example



```
C:\DOS>DEBUG
-A
1342:0100 JMP [BX]
1342:0102
-R BX
BX 0000
:1000
-E 1000 00 02
-D 1000 1001
1342:1000  00 02
-R
AX=0000  BX=1000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0100   NV UP EI PL NZ NA PO NC
1342:0100 FF27          JMP     [BX]                        DS:1000=0200
-T

AX=0000  BX=1000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1342  ES=1342  SS=1342  CS=1342  IP=0200   NV UP EI PL NZ NA PO NC
1342:0200 4D            DEC     BP
-Q
```

- memptr16
  - 16-bit value of IP specified as the content of a storage location in memory
  - Memory addressing
- Example

  1342:0100   JMP [BX]

  - Prior to execution

    (IP) = 0100H

    (DS) = 1342H

    (BX) = 1000H

  (DS:BX) = (1342H:1000H) = 0200H

  - After execution

    (IP) = 0200H

  - Next instruction

    (CS:IP) = 1342:0200H

Intersegment Unconditional Jump Operation (1)

- Intersegment—branch to address is located in another code segment
  - Both CS and IP change value
  - far-label
    - 32-bit immediate operand coded into the instruction
    - New address computed as:
      - 1st 16 bits → (IP)
      - 2nd 16 bits → (CS)

Jump to address = (New CS) : (New IP)

Intersegment Unconditional Jump Operation (2)

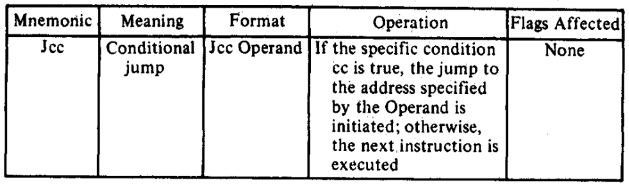
- Intersegment—branch to address is located in another code segment
  - regptr32
    - 32-bit value specified as the content of a register
    - Register addressing
  - memptr32
    - 32-bit value specified in memory
    - Memory addressing
    - Example: JMP DWORD PTR [SI]
    - Operation:

      (DS:SI) → new IP

      (DS:SI +2) → new CS

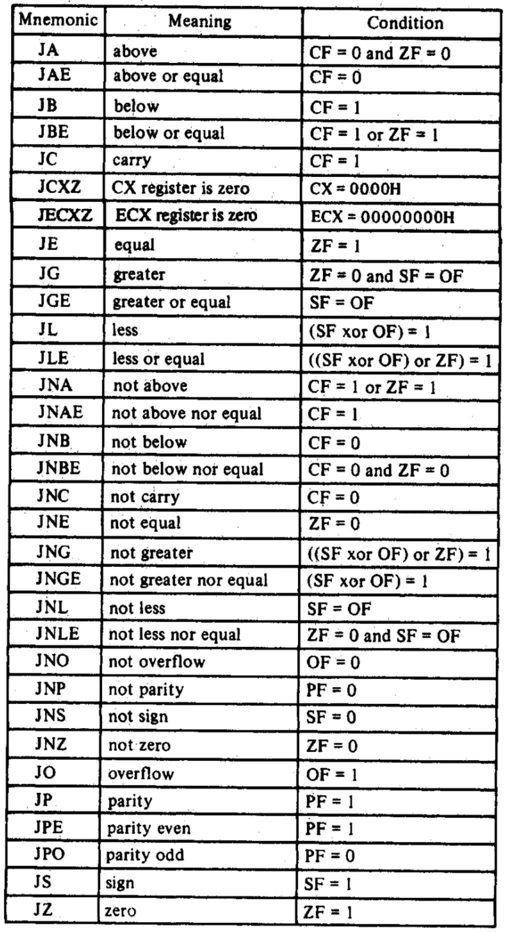
  Jump to address = (New CS) : (New IP)

Conditional Jump Instruction



Mnemonic	Meaning	Format	Operation	Flags Affected
Jcc	Conditional jump	Jcc Operand	If the specific 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
JAE	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	CX = 0000H
JECXZ	ECX register is zero	ECX = 00000000H
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

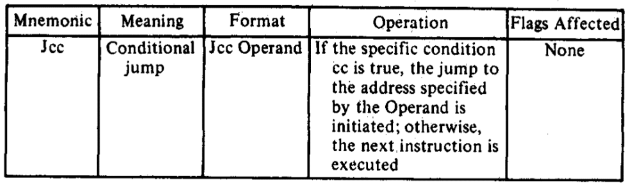
(b)

- Condition jump instruction
  - Implements the conditional jump operation
  - General format:

    Jcc Operand

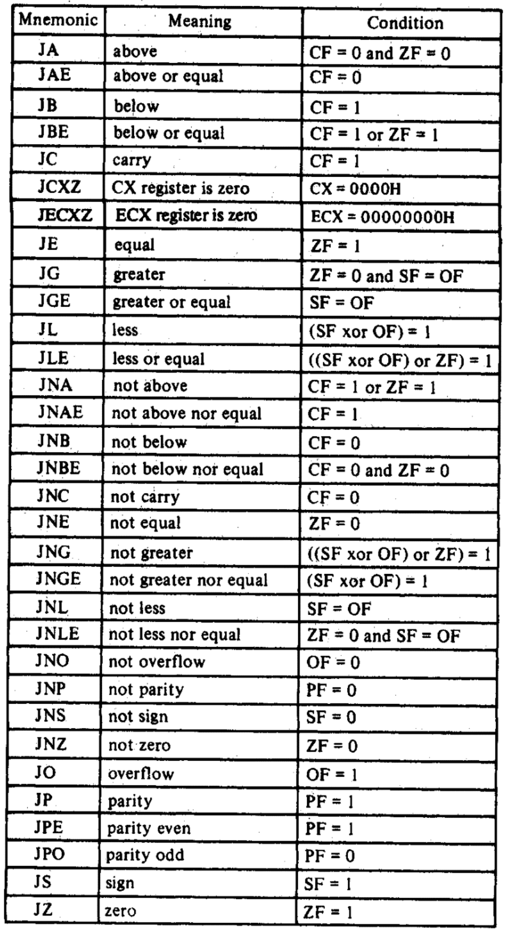
    - cc = one of the supported conditional relationships
    - Supports the same operand types as unconditional jump

Conditional Jump Instruction



Mnemonic	Meaning	Format	Operation	Flags Affected
Jcc	Conditional jump	Jcc Operand	If the specific 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
JAE	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	CX = 0000H
JECXZ	ECX register is zero	ECX = 00000000H
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)

- Condition jump instruction
  - Operation: Flags tested for conditions defined by cc and:
    - If cc test True:

      IP, or IP and CS are updated with new value(s)—Jump is taken
    - If cc test False:

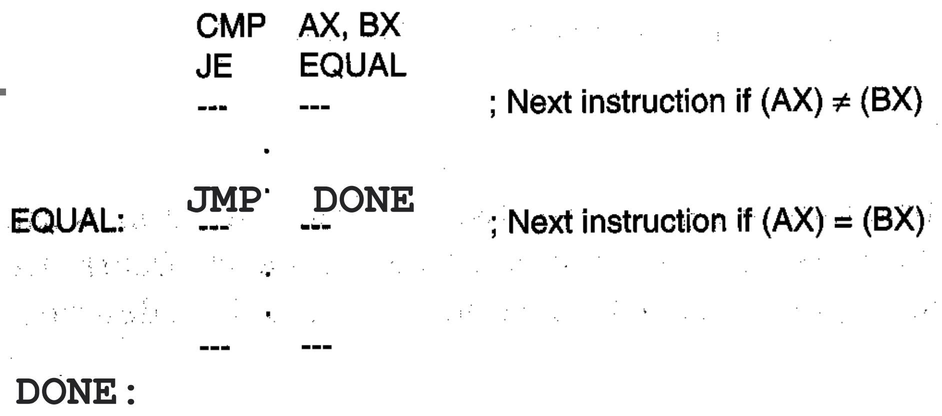
      IP, or IP and CS are unchanged—continues with sequential execution
  - Examples of conditional tests:

    JC = jump on carry → CF = 1

    JPE/JP = jump on parity even → PF =1

    JE/JZ = jump on equal → ZF = 1

Branch Program Structures

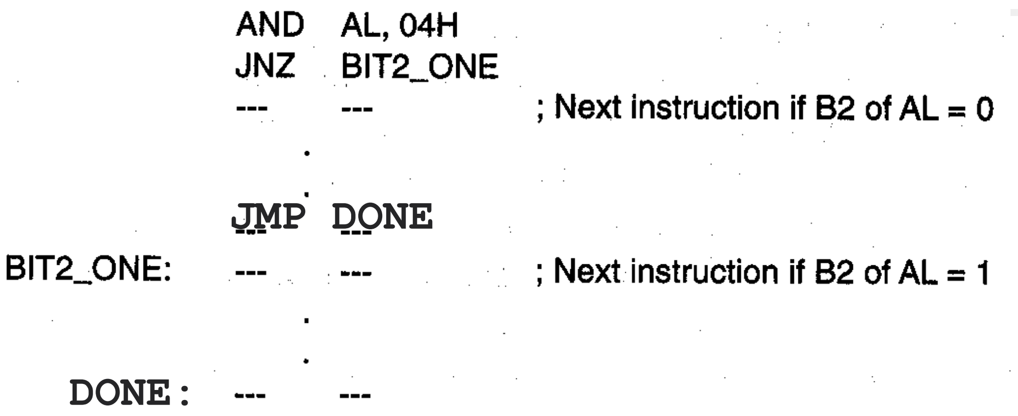


```
        CMP   AX, BX
        JE    EQUAL
        ---   ---          ; Next instruction if (AX) ≠ (BX)
         .
        JMP   DONE
EQUAL:  ---   ---          ; Next instruction if (AX) = (BX)
         .
         .
        ---   ---
DONE:
```

- Example—IF-THEN-ELSE using a flag condition
  - One of the most widely used flow control program structure
  - Implemented with CMP,JE, and JMP instructions
  - Operation
    - AX compared to BX to update flags
    - JE tests for:

      ZF = 1
    - If (AX) ≠ (BX); ZF = 0 → ELSE—next sequential instruction is executed
    - If (AX) = (BX); ZF =1 → THEN—instruction pointed to by EQUAL executes

Branch Program Structures



```
            AND   AL, 04H
            JNZ   BIT2_ONE
            ---   ---            ; Next instruction if B2 of AL = 0
                .
                .
            JMP   DONE
BIT2_ONE:   ---   ---            ; Next instruction if B2 of AL = 1
                .
                .
DONE:       ---   ---
```

- Example—IF-THEN-ELSE using a register bit test
  - Conditional test is made with JNZ instruction and taken if ZF =0
  - Generation of test condition

(AL) = xxxxxxx AND 00000100 = 00000x00

if bit 2 = 1 ZF =0

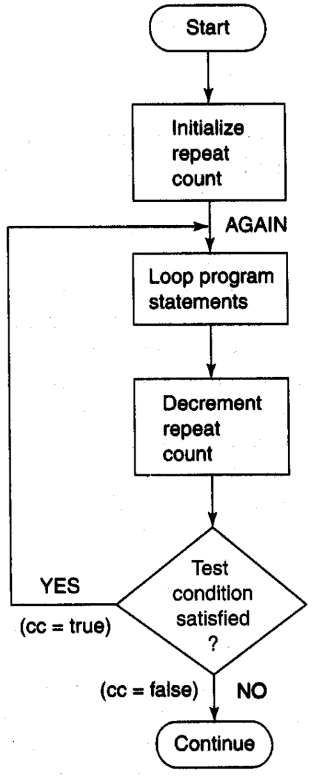
if bit 2 = 0 ZF =1

Therefore, jump to BIT2_ONE only takes place if bit 2 of AL equals 1

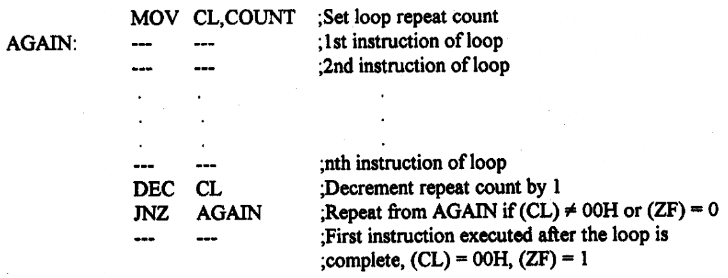
  - Same operation can be performed by shifting bit 2 to the CF and then testing with JC

    CF =1

Loop Program Structures



(a)



```
         MOV  CL,COUNT   ;Set loop repeat count
AGAIN:   ---  ---        ;1st instruction of loop
         ---  ---        ;2nd instruction of loop
          .    .              .
          .    .              .
          .    .              .
         ---  ---        ;nth instruction of loop
         DEC  CL         ;Decrement repeat count by 1
         JNZ  AGAIN      ;Repeat from AGAIN if (CL) ≠ 00H or (ZF) = 0
         ---  ---        ;First instruction executed after the loop is
                         ;complete, (CL) = 00H, (ZF) = 1
```

(b)

- Example—Loop program structure
  - Allows a part of a program to be conditionally repeated over an over
  - Employs post test—conditional test at end of sequence
  - Important parameters
    - Initial count → count register
    - Terminal count → zero or other value
  - Program flow of control:
    - Initialize count

      MOV CL,COUNT
    - Perform body of loop operation

      AGAIN: --- --- first of multiple instructions
    - Decrement count

      DEC CL
    - Conditional test for completion

      JNZ AGAIN

Loop Program Structures

- Example—Another loop program structure
  - Allows a part of a program to be conditionally repeated over an over
  - Employs pre-test—at entry of loop
  - Important parameters
    - Initial count → count register
    - Terminal count → zero or other value
  - Program flow of control:
    - Initialize count

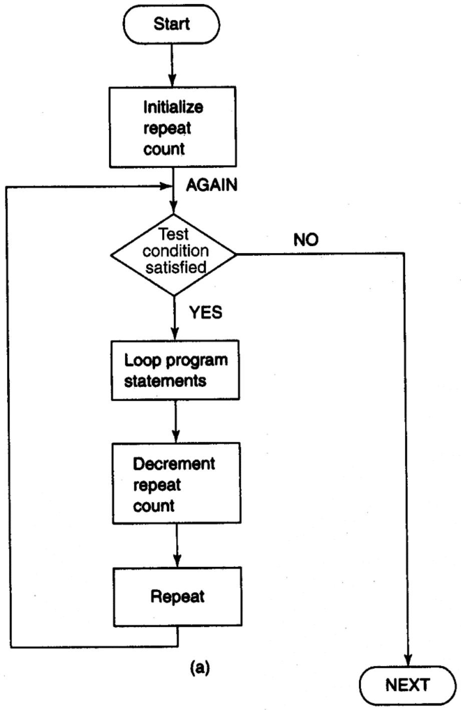
      MOV CL,COUNT
    - Pre-test

      AGAIN: JZ NEXT
    - Perform body of loop operation

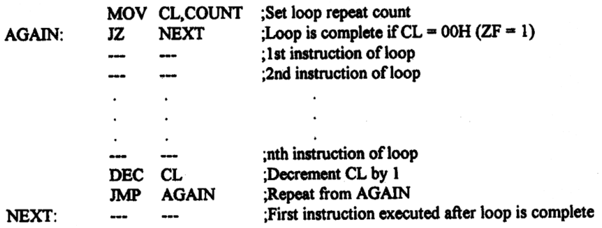
      --- --- first of multiple instructions
    - Decrement count

      DEC CL
    - Unconditional return to start of loop

      JMP AGAIN



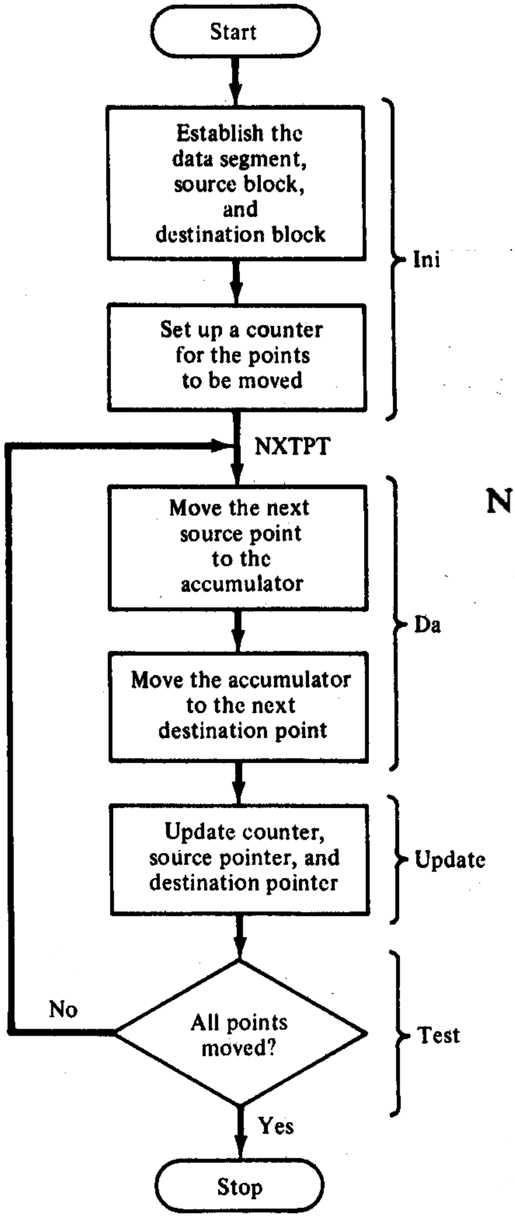
(a)



```
        MOV  CL,COUNT   ;Set loop repeat count
AGAIN:  JZ   NEXT       ;Loop is complete if CL = 00H (ZF = 1)
        ---  ---        ;1st instruction of loop
        ---  ---        ;2nd instruction of loop
         .    .               .
         .    .               .
         .    .               .
        ---  ---        ;nth instruction of loop
        DEC  CL         ;Decrement CL by 1
        JMP  AGAIN      ;Repeat from AGAIN
NEXT:   ---  ---        ;First instruction executed after loop is complete
```

(b)

Block Move Program


Start
Establish the data segment, source block, and destination block
Ini
Set up a counter for the points to be moved
NXTPT
Move the next source point to the accumulator
Da
Move the accumulator to the next destination point
Update counter, source pointer, and destination pointer
Update
All points moved?
No
Yes
Test
Stop

```
        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
        DEC     CX
        JNZ     NXTPT
        HLT
```