Chapter 6

8088/8086 Microprocessor Programming 2

Introduction

6.1 Flag-Control Instructions—✓

6.2 Compare Instruction—✓

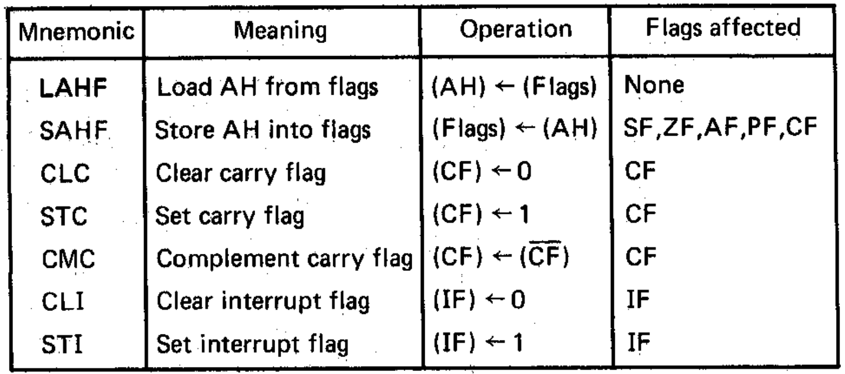
6.3 Jump Instructions—✓

6.4 Subroutines and Subroutine-Handling Instructions —✓

6.5 The Loop and Loop-Handling Instructions —✓

6.6 Strings and Sting-Handling Instructions

6.1 Flag Control Instructions- Loading, Storing, and Modifying Flags



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) ← $(\overline{CF})$	CF
CLI	Clear interrupt flag	(IF) ← 0	IF
STI	Set interrupt flag	(IF) ← 1	IF

- Variety of flag control instructions provide support for loading, saving, and modifying content of the flags register
  - LAHF/SAHF→ Load/store control flags
  - CLC/STC/CMC → Modify carry flag
  - 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

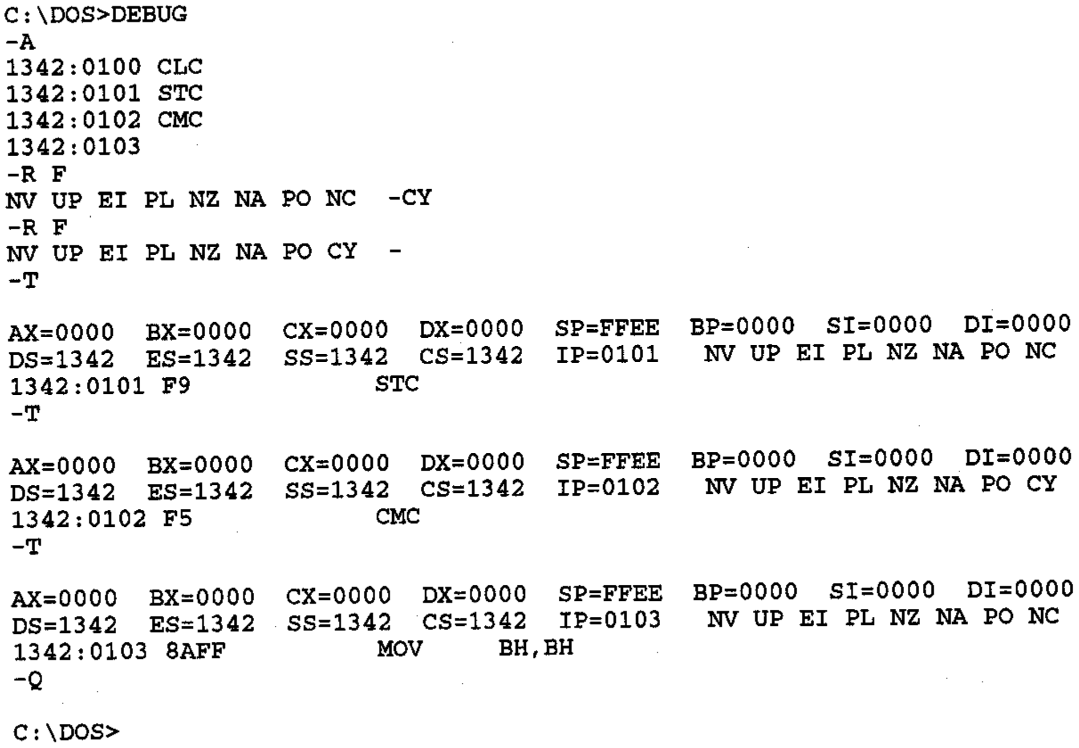
    (CF*) → (CF) * stands for overbar (NOT)
- Modifying the interrupt flag—CLI/STI
  - Used to turn on/off external hardware interrupts
  - Clear interrupt flag

    CLC

    0 → (CF) Disable interrupts
  - Set interrupt flag

    STC

6.1 Flag Control Instructions- Debug 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

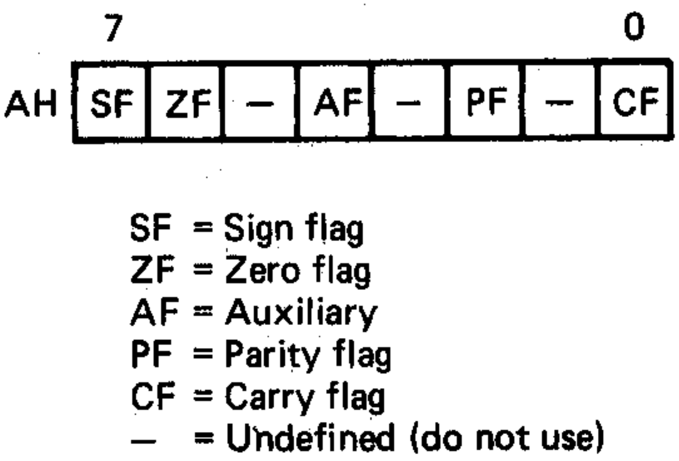
  CY=1 → initial sate

  CLC ;Clear carry flag

  STC ;Set carry flag

  CMC ;Complement carry flag

6.1 Flag Control Instructions- Loading and Storing the Flags 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)

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

  LAHF

  (Flags) → (AH)

  Flags unchanged
- Store the content of AH into the flags register

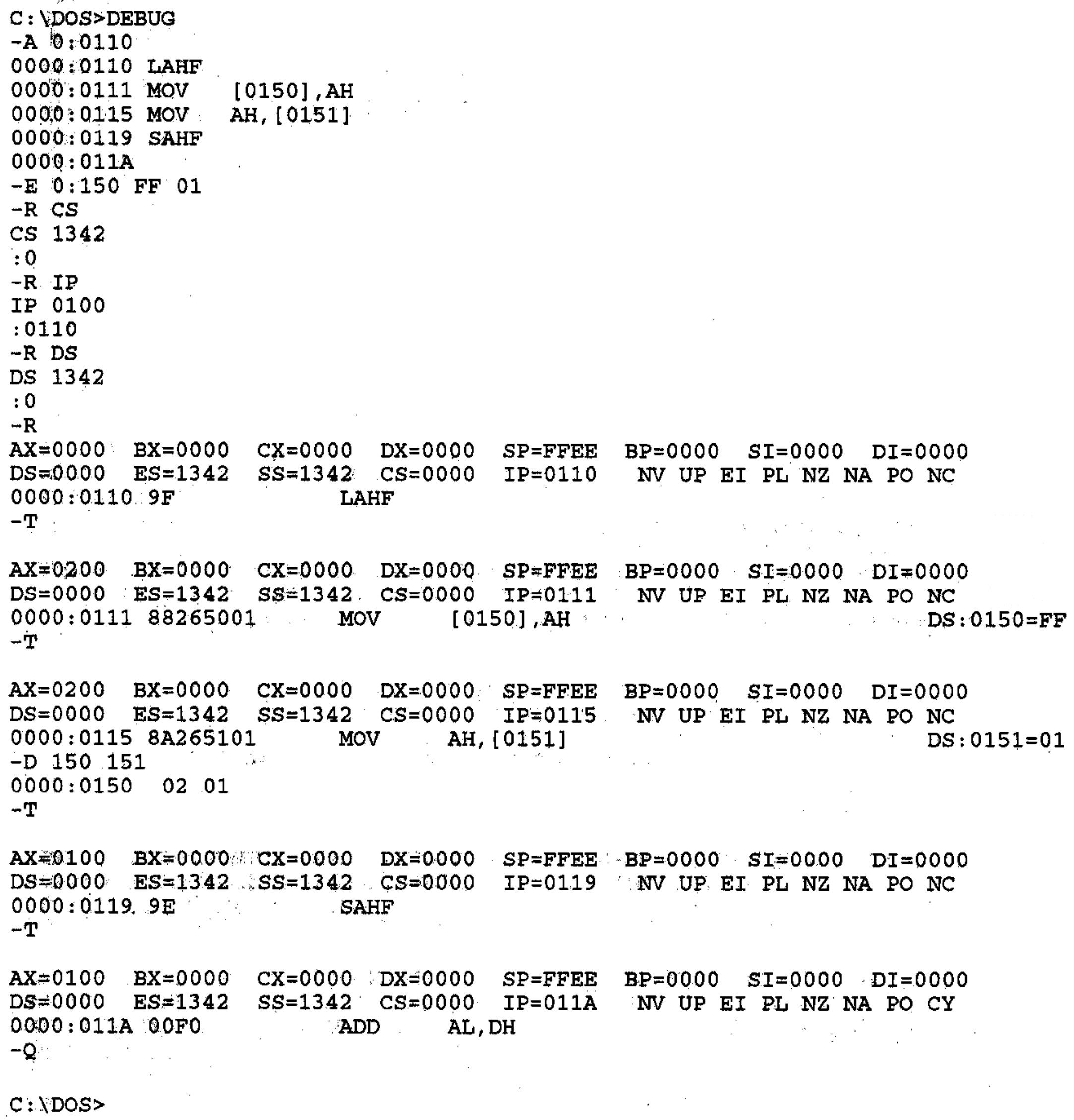
  SAHF

  (AH) → (Flags)

  SF,ZF,AF,PF,CF → updated
- Application—saving a copy of the flags in memory and initializing with new values from memory

```
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
```

6.1 Flag Control Instructions- Debug Example



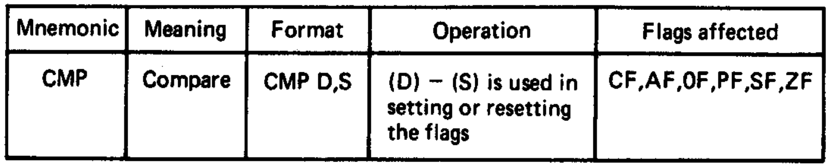
```
C:\DOS>DEBUG
-A 0:0110
0000:0110 LAHF
0000:0111 MOV   [0150],AH
0000:0115 MOV   AH,[0151]
0000:0119 SAHF
0000:011A
-E 0:150 FF 01
-R CS
CS 1342
:0
-R IP
IP 0100
:0110
-R DS
DS 1342
:0
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0000  ES=1342  SS=1342  CS=0000  IP=0110   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 flag save and initialization sequence

Other flag notation:

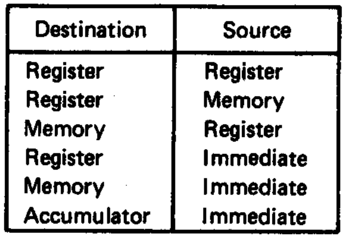
- Flag = 1/0
- SF = NG/PL
- ZF = ZR/NZ
- AF = AC/NA
- PF = PE/PO

6.2 Compare Instruction- Instruction Format and Operation



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

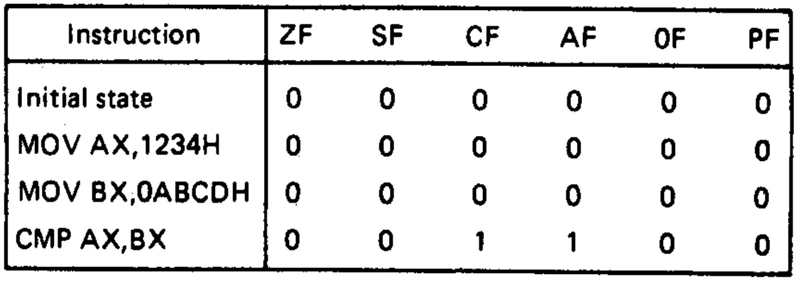
(b)

- Compare instruction
  - 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; updates flags based on result

    (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

6.2 Compare Instruction- Compare Example



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

- Example:

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

- Initialization of data registers AX and BX with immediate data:

  IMM16 → (AX) = 1234H

  IMM16 → (BX) = ABCDH

- Compare computation performed as:

  $(AX) = 0001001000110100_2$

  $(BX) = 1010101111001101_2$

$(AX) – (BX) = \underline{0}00100100011\underline{0}100_2 - \underline{1}01010111100\underline{1}101_2$

  ZF = 0 = NZ

  SF = 0 = PL → treats operands as signed numbers

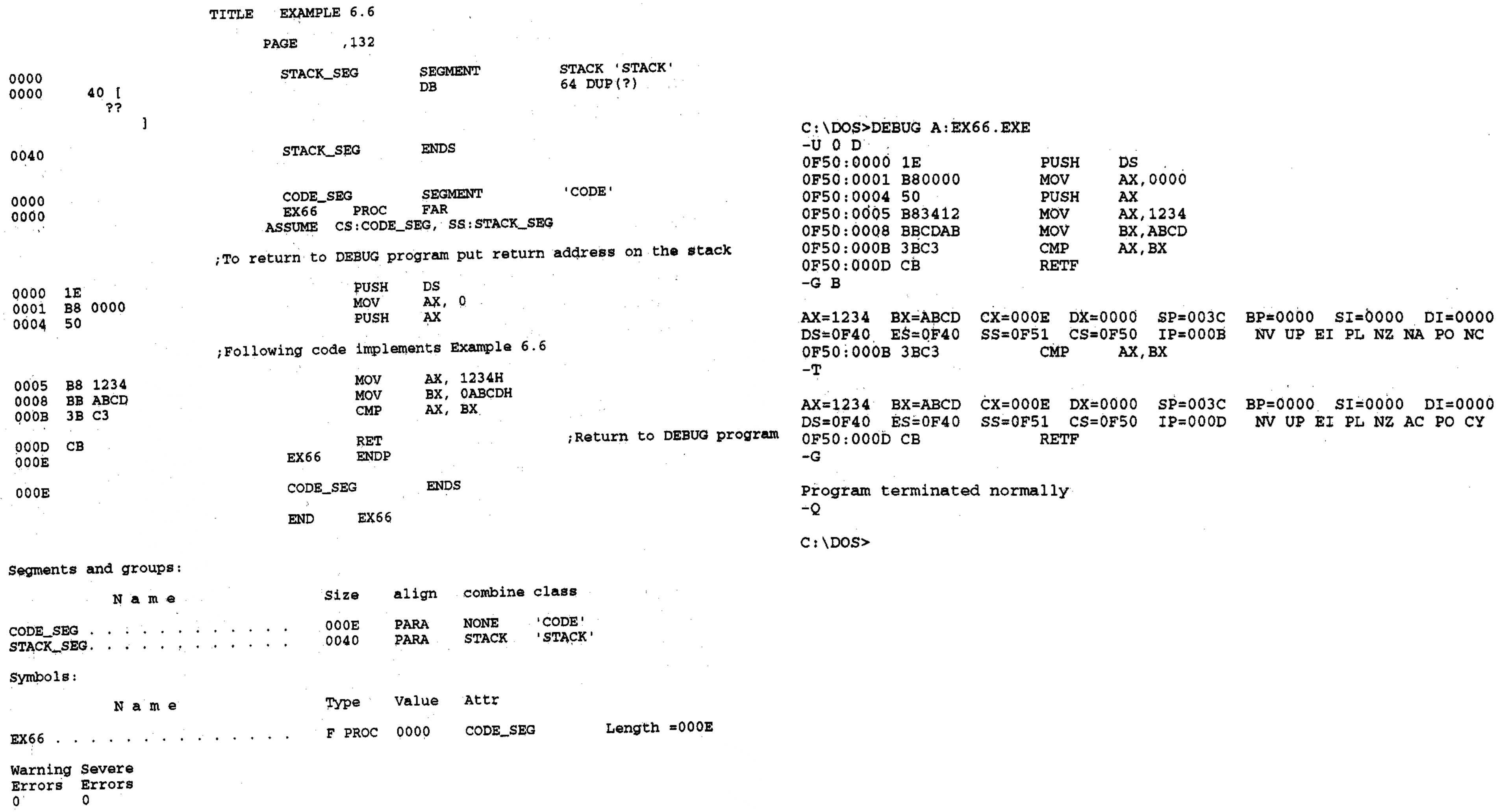
  CF = 1 = CY

  AF = 1 = AC

  OF = 0 = NV

  PF = 0 = PO

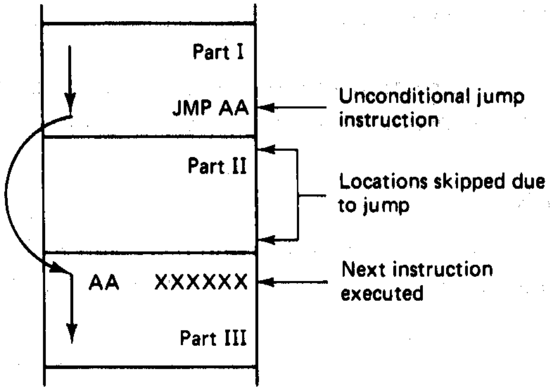
6.2 Compare Instruction- Listing and Debug Execution



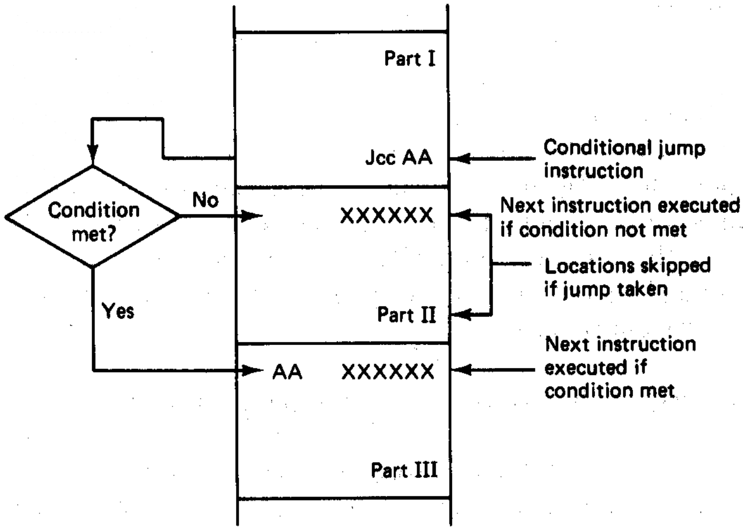
```
                    TITLE   EXAMPLE 6.6

                            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>
```

6.3 Jump Instructions- Unconditional and Conditional Jump Control Flow


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

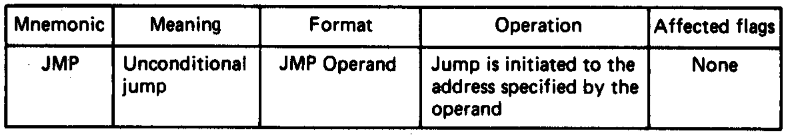
(a)


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)

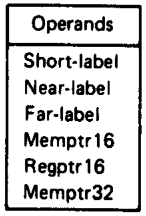
- Jump operation alters the execution path of the instructions in the program—flow control
  - Unconditional Jump
    - Always takes place
    - No status requirements are imposed
    - Example
      - JMP AA instructions in Part I executed
      - Control passed to next instruction identified by AA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

6.3 Jump Instructions- 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

(b)

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

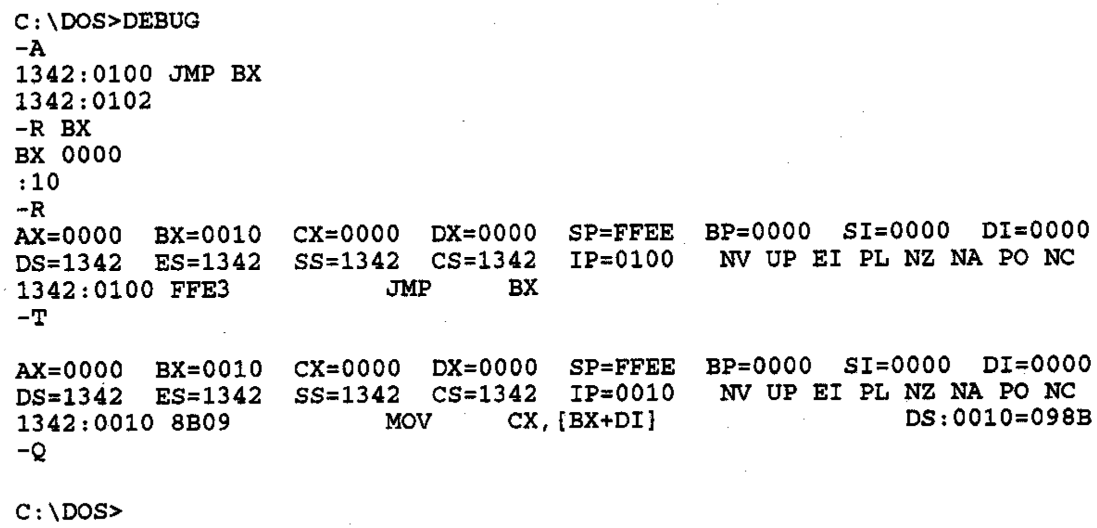
    JMP Operand
  - Types of unconditional jumps
    - Intrasegment—branch to address is located in the current code segment
      - Only IP changes value
      - short-label
        - 8-bit signed displacement coded into the instruction
        - Immediate addressing
        - Range equal –126 to +129
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

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

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

          JMP 1234H

6.3 Jump Instructions- 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(0)) + (New IP)

- Example

  1342:0100 JMP BX

  Prior to execution

  (IP) = 0100H

  (BX) =0010H

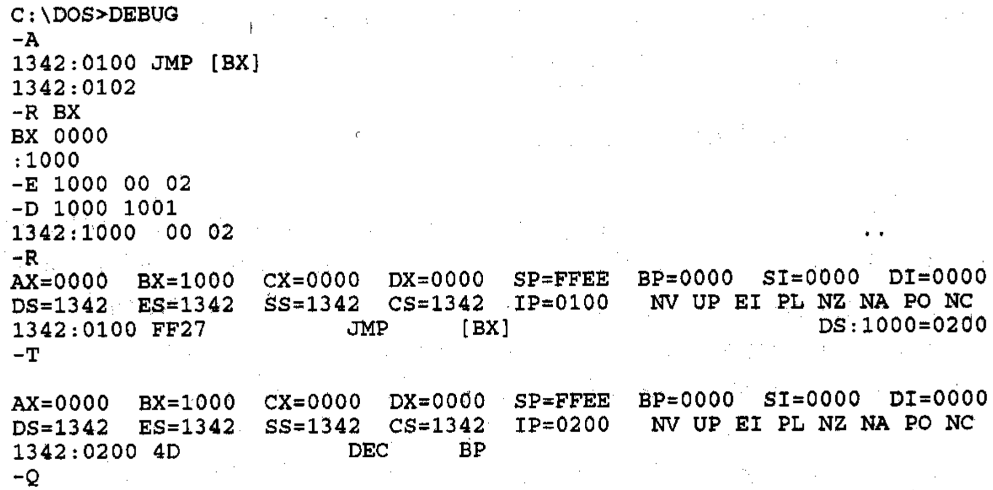
  After execution

  (IP) =0010H

  Address of next instruction

  (CS:IP) = 1342:0010

6.3 Jump Instructions- 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
  - Register indirect 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

6.3 Jump Instructions- Intersegment Unconditional Jump Operation

- Intersegment—branch to address is located in another code segment
  - Both CS and IP change values
  - 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)
  - memptr32
    - 32-bit value specified in memory
    - Memory indirect addressing
    - Example

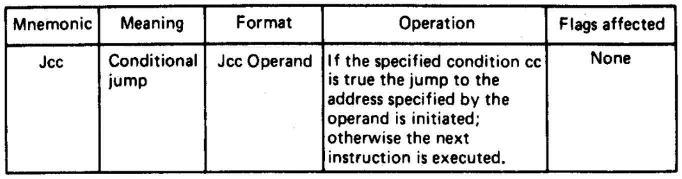
      JMP DWORD PTR [DI]
    - Operation:

      (DS:DI) → new IP

      (DS:DI +2) → new CS

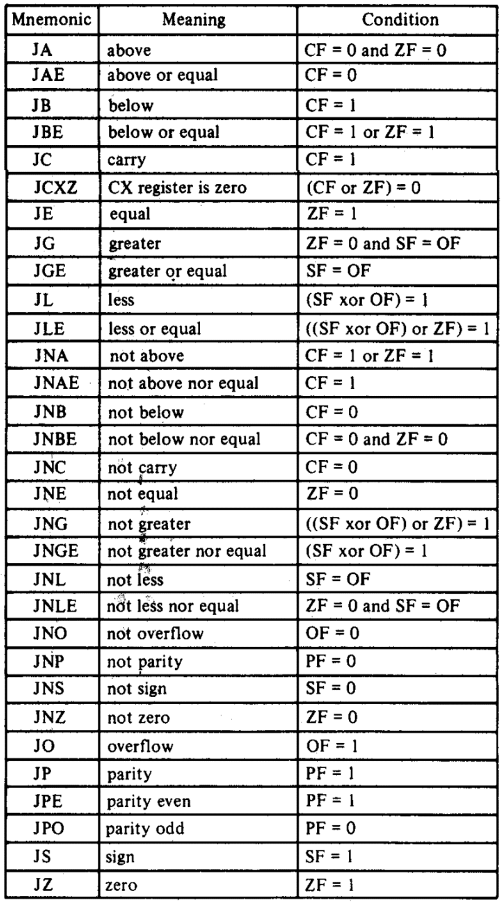
      Jump to address = (New CS):(New IP)

6.3 Jump Instructions- Conditional Jump Instruction



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
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	(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)

- 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
  - Operation: Flags tested for conditions defined by cc and:

    If cc test True:

    IP, or IP and CS are updated with new value

    - Jump is taken
    - Execution resumes at jump to target address

    If cc test False:

    IP, or IP and CS are unchanged

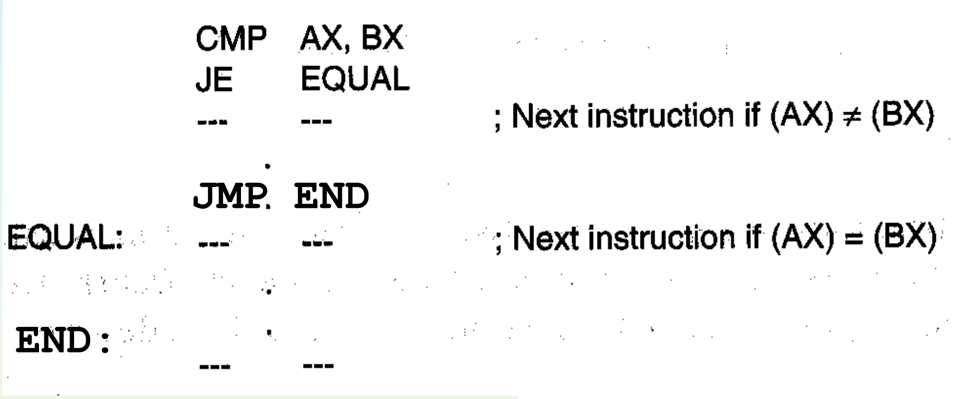
    - Jump is not taken
    - Execution continues with the next sequential instruction
  - Examples of conditional tests:

    JC = jump on carry → CF = 1

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

    JE/JZ = jump on equal → ZF = 1

6.3 Jump Instructions- Branch Program Structures

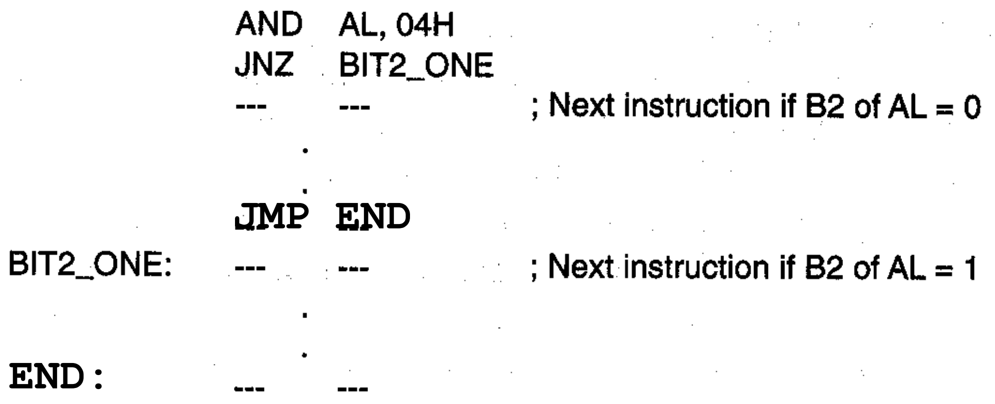


```
             CMP   AX, BX
             JE    EQUAL
             ---   ---              ; Next instruction if (AX) ≠ (BX)

             JMP   END
EQUAL:       ---   ---              ; Next instruction if (AX) = (BX)

END :
             ---   ---
```

- Example—IF-THEN-ELSE comparing values
  - 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 → THEN path—next sequential instruction is executed
    - If (AX) = (BX); ZF =1 → ELSE path—instruction pointed to by EQUAL executes
    - JMP instruction used in THEN path to bypass the ELSE path when

6.3 Jump Instructions- Branch Program Structures



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

- Example—IF-THEN-ELSE using a register bit test
- Conditional test is made with JNZ instruction and branch taken if

  ZF =0
- Generation of test condition

  (AL) = xxxxxxx AND 00000100

  = 00000x00

  if bit 2 = 1 ZF =0 (not zero)

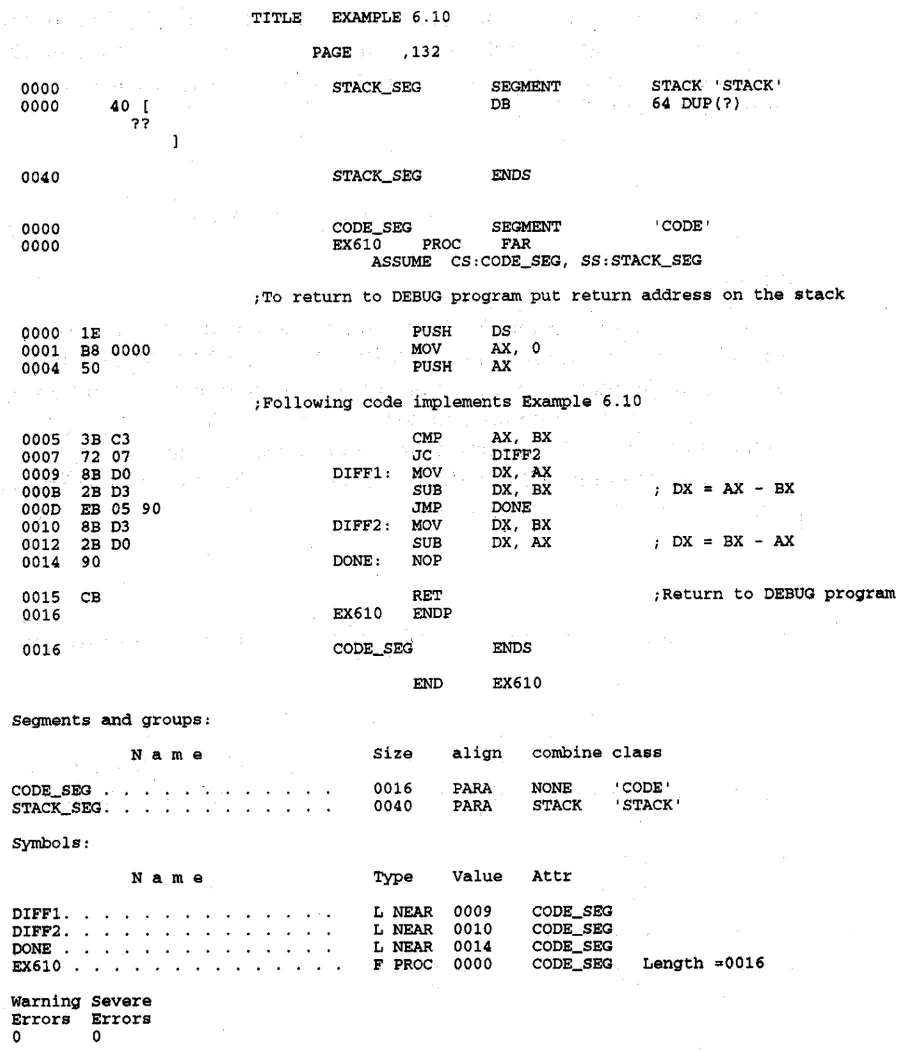
  if bit 2 = 0 ZF =1

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

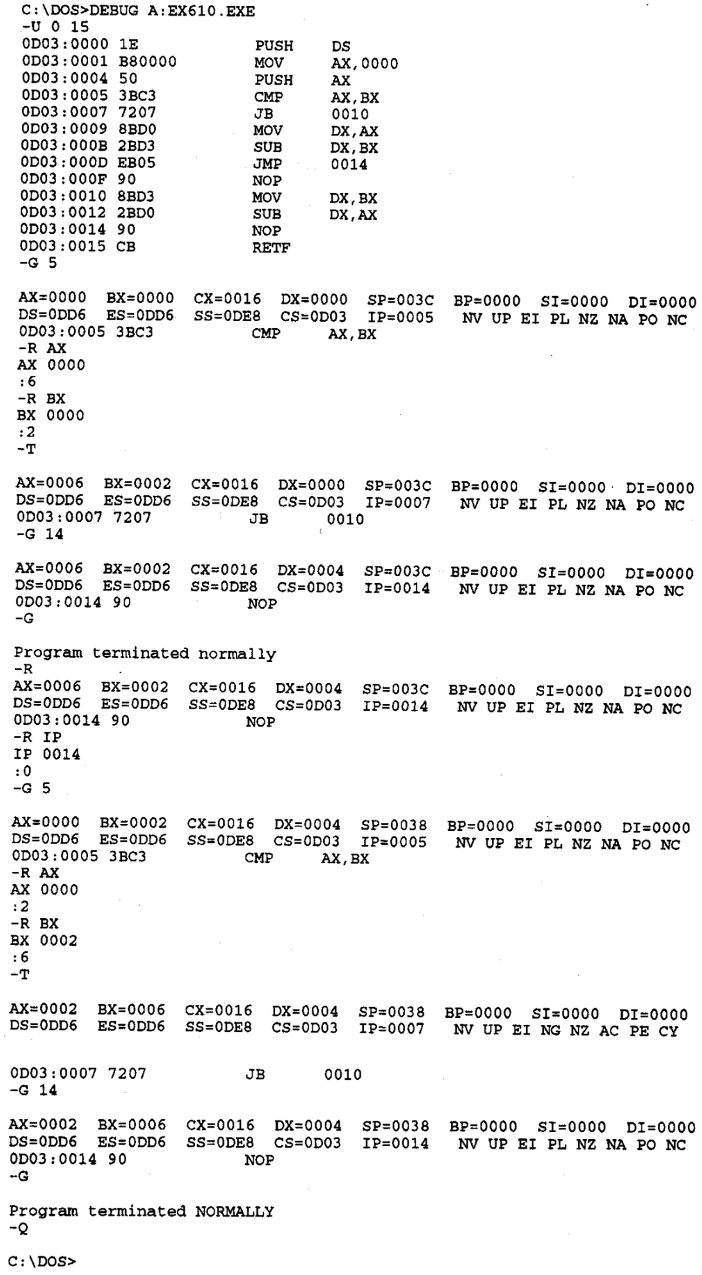
- Same operation can be performed by shifting bit 2 to the CF and then testing with JC

  CF =1

6.3 Jump Instructions- Program Applying Branching

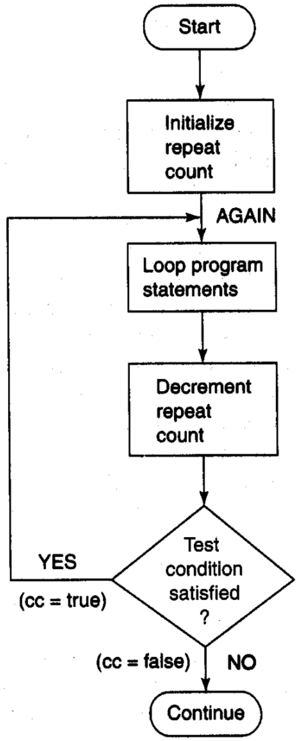


```
                       TITLE   EXAMPLE 6.10

                           PAGE     ,132

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

0040                       STACK_SEG          ENDS


0000                       CODE_SEG           SEGMENT          'CODE'
0000                       EX610     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 6.10

0005   3B C3                        CMP      AX, BX
0007   72 07                        JC       DIFF2
0009   8B D0               DIFF1:   MOV      DX, AX
000B   2B D3                        SUB      DX, BX          ; DX = AX - BX
000D   EB 05 90                     JMP      DONE
0010   8B D3               DIFF2:   MOV      DX, BX
0012   2B D0                        SUB      DX, AX          ; DX = BX - AX
0014   90                  DONE:    NOP

0015   CB                           RET                      ;Return to DEBUG program
0016                       EX610    ENDP

0016                       CODE_SEG          ENDS

                                    END      EX610

Segments and groups:

           N a m e                  Size    align    combine class

CODE_SEG . . . . . . . . . . . . .  0016    PARA     NONE     'CODE'
STACK_SEG. . . . . . . . . . . . .  0040    PARA     STACK    'STACK'

Symbols:

           N a m e                  Type    Value    Attr

DIFF1. . . . . . . . . . . . . . .  L NEAR  0009     CODE_SEG
DIFF2. . . . . . . . . . . . . . .  L NEAR  0010     CODE_SEG
DONE . . . . . . . . . . . . . . .  L NEAR  0014     CODE_SEG
EX610  . . . . . . . . . . . . . .  F PROC  0000     CODE_SEG    Length =0016

Warning Severe
Errors  Errors
0       0
```

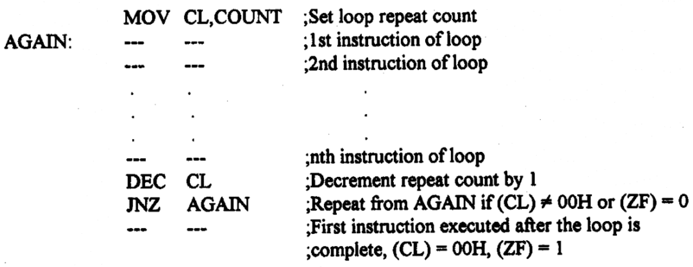


```
C:\DOS>DEBUG A:EX610.EXE
-U 0 15
0D03:0000 1E               PUSH     DS
0D03:0001 B80000           MOV      AX,0000
0D03:0004 50               PUSH     AX
0D03:0005 3BC3             CMP      AX,BX
0D03:0007 7207             JB       0010
0D03:0009 8BD0             MOV      DX,AX
0D03:000B 2BD3             SUB      DX,BX
0D03:000D EB05             JMP      0014
0D03:000F 90               NOP
0D03:0010 8BD3             MOV      DX,BX
0D03:0012 2BD0             SUB      DX,AX
0D03:0014 90               NOP
0D03:0015 CB               RETF
-G 5

AX=0000  BX=0000  CX=0016  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0005   NV UP EI PL NZ NA PO NC
0D03:0005 3BC3           CMP     AX,BX
-R AX
AX 0000
:6
-R BX
BX 0000
:2
-T

AX=0006  BX=0002  CX=0016  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0007   NV UP EI PL NZ NA PO NC
0D03:0007 7207           JB      0010
-G 14

AX=0006  BX=0002  CX=0016  DX=0004  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0014   NV UP EI PL NZ NA PO NC
0D03:0014 90             NOP
-G

Program terminated normally
-R
AX=0006  BX=0002  CX=0016  DX=0004  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0014   NV UP EI PL NZ NA PO NC
0D03:0014 90             NOP
-R IP
IP 0014
:0
-G 5

AX=0000  BX=0002  CX=0016  DX=0004  SP=0038  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0005   NV UP EI PL NZ NA PO NC
0D03:0005 3BC3           CMP     AX,BX
-R AX
AX 0000
:2
-R BX
BX 0002
:6
-T

AX=0002  BX=0006  CX=0016  DX=0004  SP=0038  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0007   NV UP EI NG NZ AC PE CY


0D03:0007 7207           JB      0010
-G 14

AX=0002  BX=0006  CX=0016  DX=0004  SP=0038  BP=0000  SI=0000  DI=0000
DS=0DD6  ES=0DD6  SS=0DE8  CS=0D03  IP=0014   NV UP EI PL NZ NA PO NC
0D03:0014 90             NOP
-G

Program terminated NORMALLY
-Q

C:\DOS>
```

6.3 Jump Instructions- 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—Repeat-Until program structure
  - Allows a part of a program to be conditionally repeated over an over
  - Employs post test—conditional test at end of sequence; always performs one iteration
  - Important parameters
    - Initial count → count register
    - Terminal count → zero or other value
  - Program flow of control:
    - Initialize count

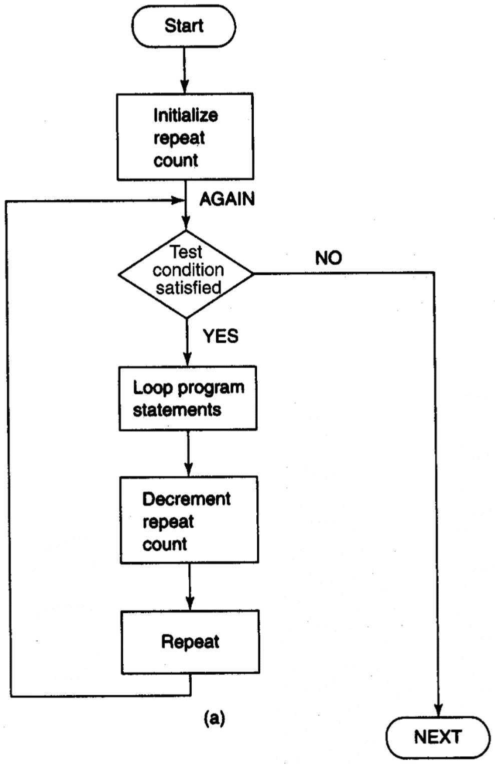
      MOV CL,COUNT
    - Perform body of loop operation

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

      DEC CL
    - Conditional test for completion

      JNZ AGAIN

6.3 Jump Instructions- Loop Program Structures



(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)

- Example—While-Do program structure
  - Allows a part of a program to be conditionally repeated over an over
  - Employs pre-test—at entry of loop; may perform no iterations
  - Important parameters
    - Initial count → count register
    - Terminal count → zero or other value
  - Program flow/control:
    - Initialize count

      MOV CL,COUNT
    - Pre-test

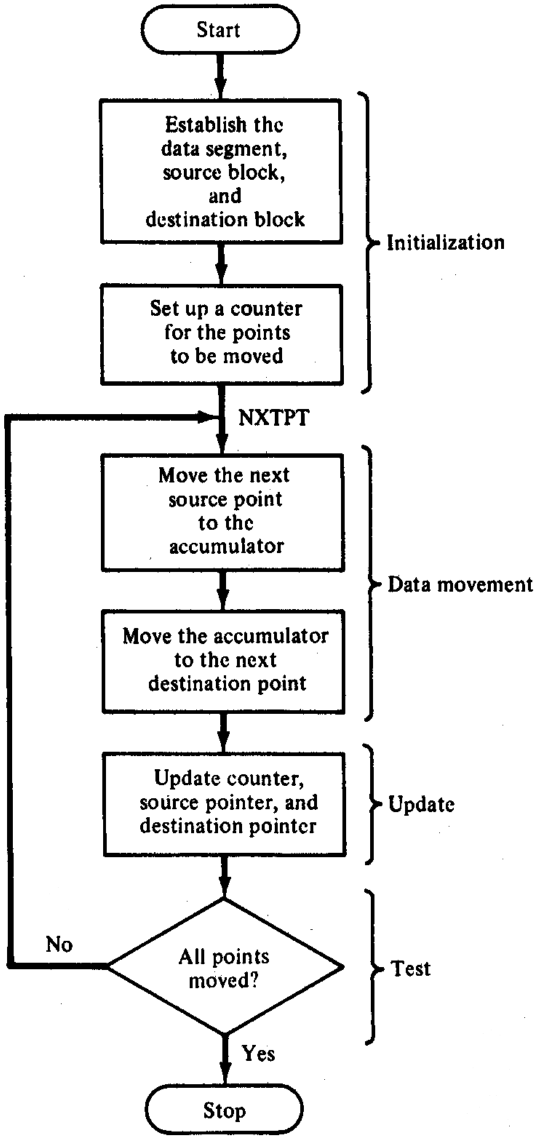
      AGAIN: JZ NEXT
    - Perform body of loop operation

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

      DEC CL
    - Unconditional return to start of loop

      JMP AGAIN

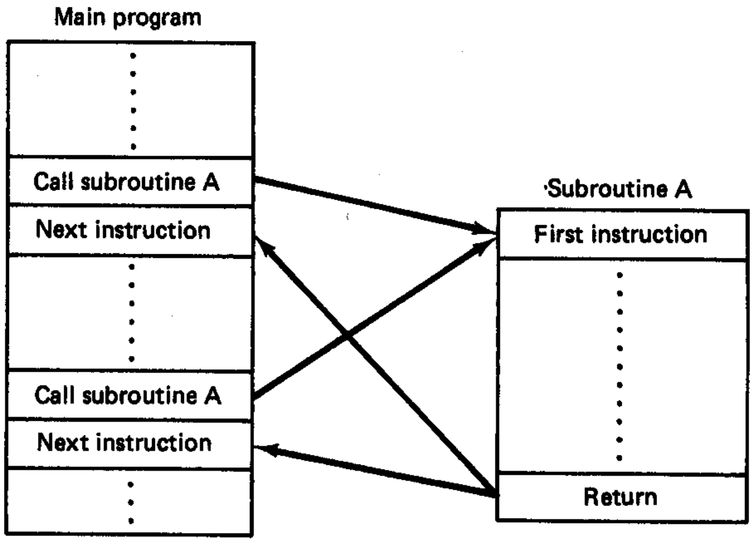
6.3 Jump Instructions- Block Move Program


Start
Establish the data segment, source block, and destination block
Set up a counter for the points to be moved
Initialization
NXTPT
Move the next source point to the accumulator
Move the accumulator to the next destination point
Data movement
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
```

6.4 Subroutines and Subroutine-Handling Instructions- Subroutine

- Subroutine—special segment of program that can be called for execution from any point in a program
  - Program structure that implements HLL “functions” and “procedures”
  - Written to perform an operation (function/procedure) that must be performed at various points in a program
  - Written as a subroutine and only included once in the program
  - Example:
    - Instruction in Main part of program calls “Subroutine A”
    - Program flow of control transferred to first instruction of Subroutine A
    - Instructions of Subroutine A execute sequentially
    - Return initiated by last instruction of Subroutine A
    - Same sequence repeated when the subroutine is called again later in the program
  - Instructions
    - Call instruction—initiates the subroutine from the main part of program
    - Return instruction—initiates return of control to the main program at completion of the subroutine
    - Push and pop instructions used to save register content and pass parameters



6.4 Subroutines and Subroutine-Handling Instructions- Call Instruction

- Call Instruction
  - Implements two types of calls
    - Intrasegment call
    - Intersegment call
  - Intrasegment call—starting address of subroutine is located in the current code segment
    - Only IP changes value
    - near-proc
      - 16-bit offset coded in the instruction
      - Example

        CALL 1234H
      - Operation:

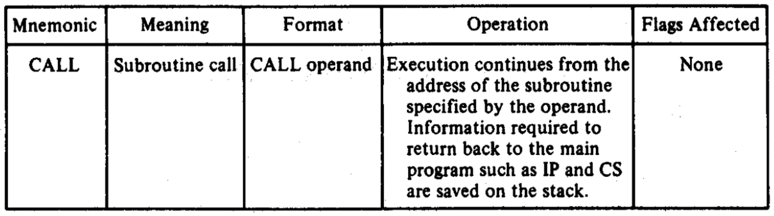
        1. IP of next instruction saved on top of stack

        2. SP is decremented by 2

        3. New value from call instruction is loaded into IP

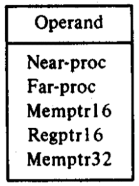
        4. Instruction fetch restarts with first instruction of subroutine

        Current CS:New IP



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)

6.4 Subroutines and Subroutine-Handling Instructions- Intrasegment Call Operation (Continued)

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

    CALL BX
  - Operation:
    - Same as near-proc except

      (BX) → New IP
- memptr16
  - 16-bit value of IP specified as the content of a storage location in memory
  - Memory addressing modes—register addressing
  - Example

    CALL [BX]
    - Same as near-proc except

      (DS:BX) → New IP

6.4 Subroutines and Subroutine-Handling Instructions- Intersegment Call Operation

- Intersegment—start address of the subroutine points to another code segment
  - Both CS and IP change values
  - far-proc
    - 32-bit immediate operand coded into the instruction
    - New address computed as:
      - 1st 16 bits → New IP
      - 2nd 16 bits → New CS

    Subroutine starts at = New CS:New IP
  - memptr32
    - 32-bit value specified in memory
    - Memory addressing modes—register indirect addressing
    - Example

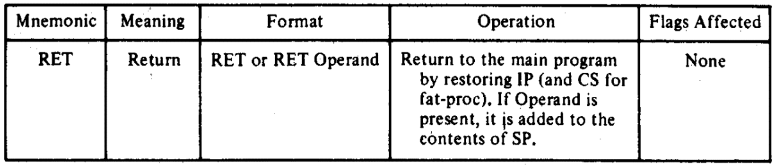
      CALL DWORD PTR [DI]
    - Operation:

      (DS:DI) → New IP

      (DS:DI +2) → New CS

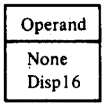
      Starting address of subroutine = New CS:New IP

6.4 Subroutines and Subroutine-Handling Instructions- Return Instruction



Mnemonic	Meaning	Format	Operation	Flags Affected
RET	Return	RET or RET Operand	Return to the main program by restoring IP (and CS for fat-proc). If Operand is present, it is added to the contents of SP.	None

(a)



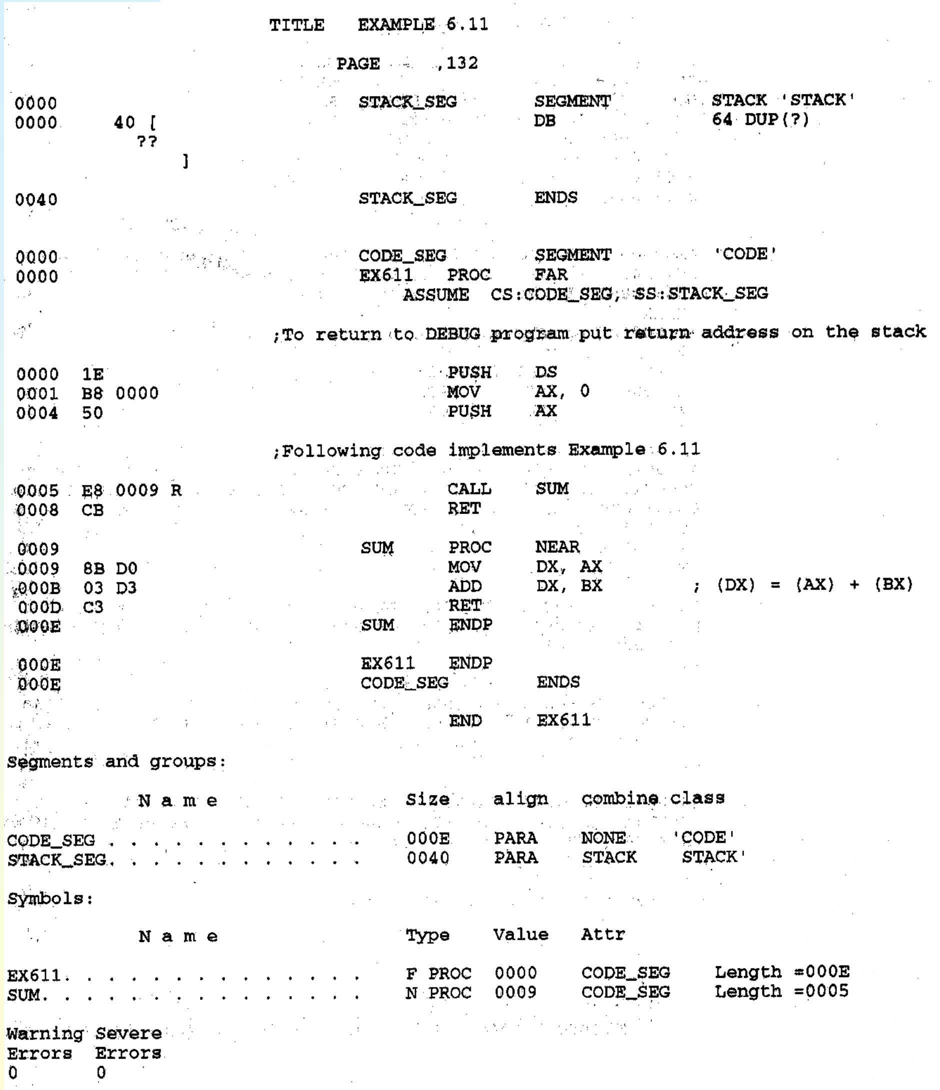
Operand
None
Disp16

(b)

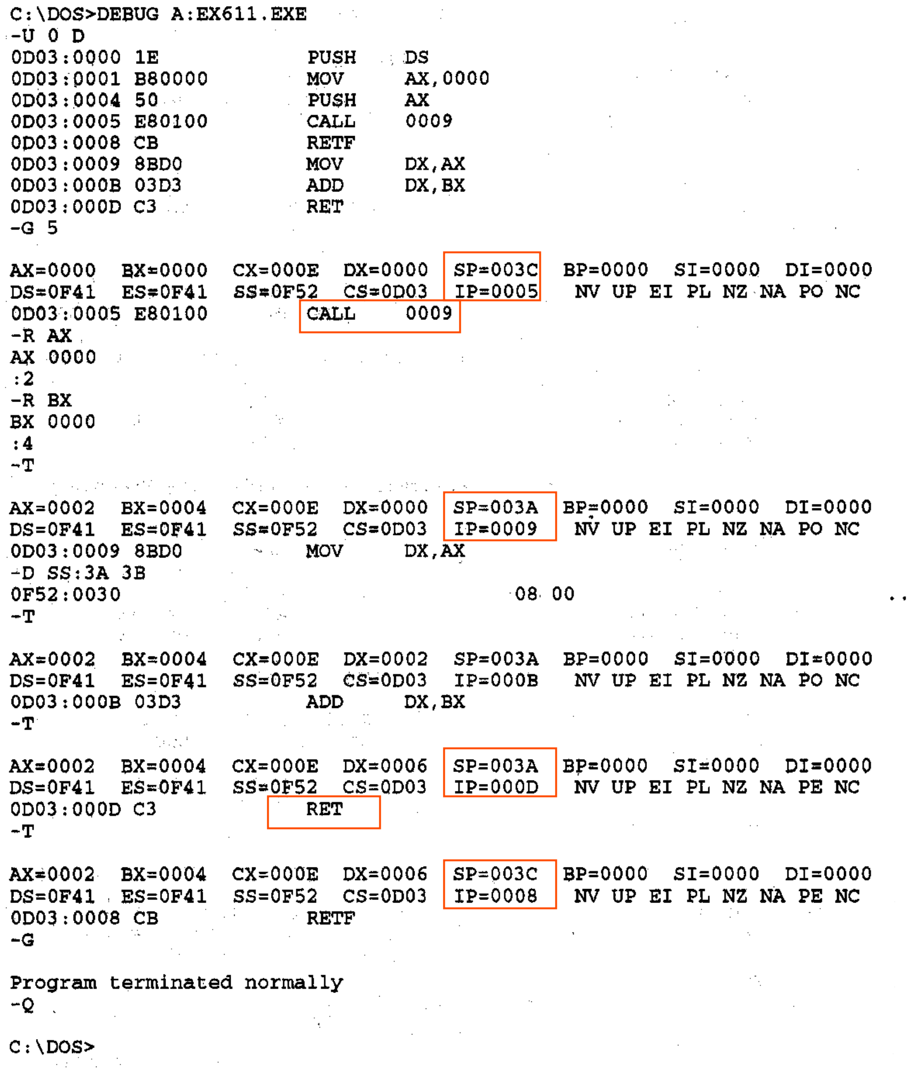
- Return instruction
  - Every subroutine must end with a return instruction
  - Initiates return of execution to the instruction in the main program following that which called the subroutine
  - Example:

    RET
    - Causes the value of IP (intrasegment return) or both IP and CS (intersegment return) to be popped from the stack and put back into the IP and CS registers
    - Increments SP by 2/4

6.4 Subroutines and Subroutine-Handling Instructions- Example of a Subroutine Call

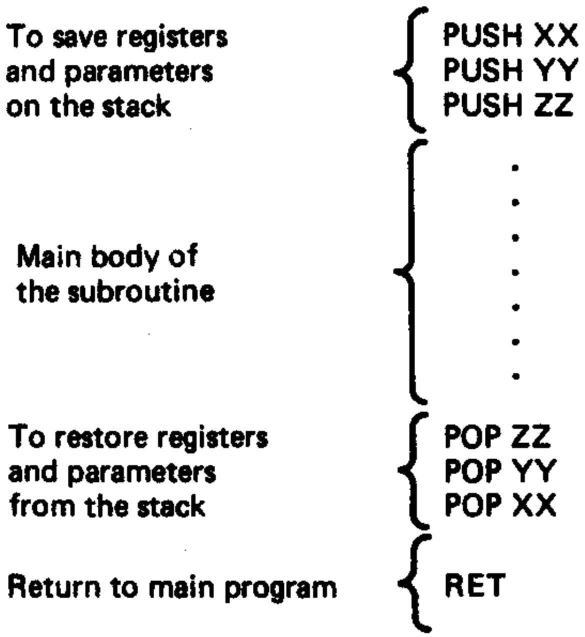


```
                         TITLE   EXAMPLE 6.11

                             PAGE      ,132

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

0040                         STACK_SEG        ENDS


0000                         CODE_SEG         SEGMENT        'CODE'
0000                         EX611    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 6.11

0005  E8 0009 R                         CALL    SUM
0008  CB                                RET

0009                         SUM        PROC    NEAR
0009  8B D0                             MOV     DX, AX
000B  03 D3                             ADD     DX, BX          ; (DX) = (AX) + (BX)
000D  C3                                RET
000E                         SUM        ENDP

000E                         EX611      ENDP
000E                         CODE_SEG           ENDS

                                        END     EX611

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

EX611. . . . . . . . . . . . . . . .   F PROC  0000    CODE_SEG    Length =000E
SUM. . . . . . . . . . . . . . . . .   N PROC  0009    CODE_SEG    Length =0005

Warning Severe
Errors  Errors
0       0
```



```
C:\DOS>DEBUG A:EX611.EXE
-U 0 D
0D03:0000 1E            PUSH    DS
0D03:0001 B80000        MOV     AX,0000
0D03:0004 50            PUSH    AX
0D03:0005 E80100        CALL    0009
0D03:0008 CB            RETF
0D03:0009 8BD0          MOV     DX,AX
0D03:000B 03D3          ADD     DX,BX
0D03:000D C3            RET
-G 5

AX=0000  BX=0000  CX=000E  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0F41  ES=0F41  SS=0F52  CS=0D03  IP=0005   NV UP EI PL NZ NA PO NC
0D03:0005 E80100        CALL    0009
-R AX
AX 0000
:2
-R BX
BX 0000
:4
-T

AX=0002  BX=0004  CX=000E  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=0F41  ES=0F41  SS=0F52  CS=0D03  IP=0009   NV UP EI PL NZ NA PO NC
0D03:0009 8BD0          MOV     DX,AX
-D SS:3A 3B
0F52:0030                                08 00                         ..
-T

AX=0002  BX=0004  CX=000E  DX=0002  SP=003A  BP=0000  SI=0000  DI=0000
DS=0F41  ES=0F41  SS=0F52  CS=0D03  IP=000B   NV UP EI PL NZ NA PO NC
0D03:000B 03D3          ADD     DX,BX
-T

AX=0002  BX=0004  CX=000E  DX=0006  SP=003A  BP=0000  SI=0000  DI=0000
DS=0F41  ES=0F41  SS=0F52  CS=0D03  IP=000D   NV UP EI PL NZ NA PE NC
0D03:000D C3            RET
-T

AX=0002  BX=0004  CX=000E  DX=0006  SP=003C  BP=0000  SI=0000  DI=0000
DS=0F41  ES=0F41  SS=0F52  CS=0D03  IP=0008   NV UP EI PL NZ NA PE NC
0D03:0008 CB            RETF
-G

Program terminated normally
-Q

C:\DOS>
```

6.4 Subroutines and Subroutine-Handling Instructions- Structure of a Subroutine



To save registers and parameters on the stack

```
PUSH XX
PUSH YY
PUSH ZZ
```

Main body of the subroutine

```
.
.
.
.
.
.
.
```

To restore registers and parameters from the stack

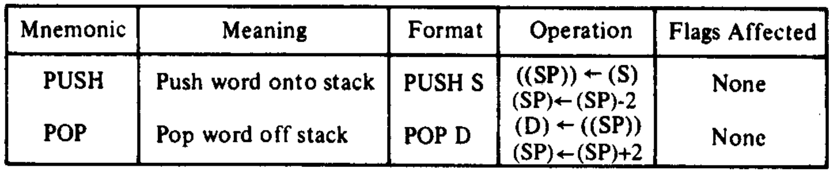
```
POP ZZ
POP YY
POP XX
```

Return to main program

```
RET
```

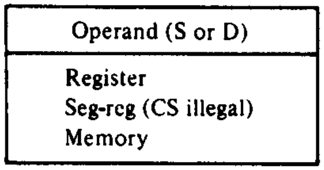
- Elements of a subroutine
  - Save of information to stack—PUSH
  - Main body of subroutine—Multiple instructions
  - Restore of information from stack—POP
  - Return to main program—RET
- Save of information
  - Must save content of registers/memory locations to be used or other program parameters (FLAGS)
  - PUSH, PUSHF
- Main body
  - Retrieve input parameters passed from main program via stack—stack pointer indirect address
  - Performs the algorithm/function/operation required of the subroutine
  - Prepare output parameters/results for return to main body via stack—stack pointer indirect addressing
- Restore information
  - Register/memory location contents saved on stack at entry of subroutine must be restored before return to main program—POP, POPF

6.4 Subroutines and Subroutine-Handling Instructions- Push and Pop Instruction



Mnemonic	Meaning	Format	Operation	Flags Affected
PUSH	Push word onto stack	PUSH S	((SP)) ← (S) (SP)← (SP)-2	None
POP	Pop word off stack	POP D	(D) ← ((SP)) (SP)←(SP)+2	None

(a)



Operand (S or D)
Register Seg-reg (CS illegal) Memory

(b)

- Push instruction
  - General format:

    PUSH S

    - Saves a value on the stack—content of:
      - Register/segment register
      - Memory
  - Example:

    PUSH AX

    (AH) → ((SP)-1)

    (AL) → ((SP)-2)

    (SP)-2 → (SP) = New top of stack
- Pop instruction
  - General format:

    POP D

    - Restores a value on the stack—content to: register, segment register, memory
  - Example:

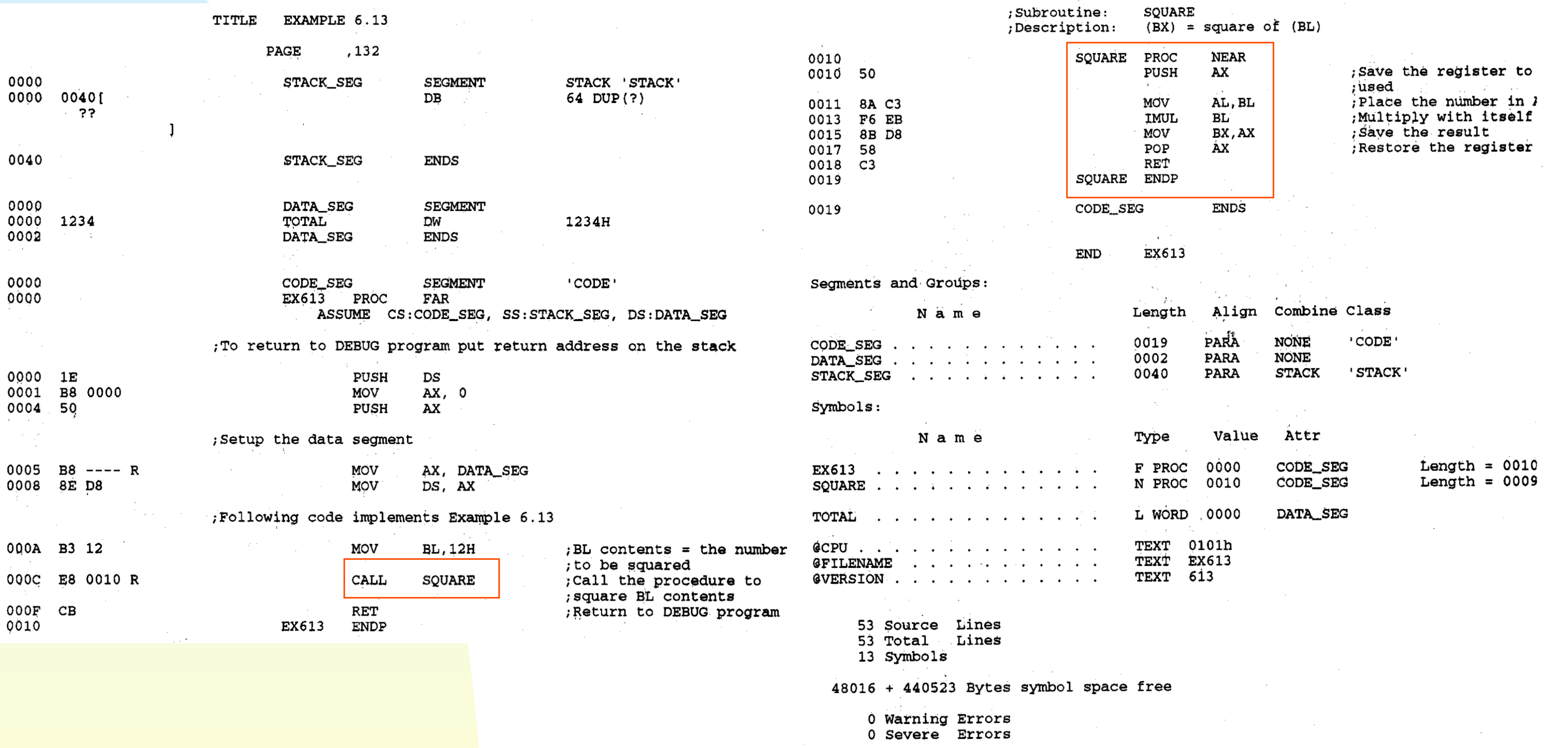
    POP AX

    ((SP)) → AL

    ((SP)+1) → AH

    ((SP)+2) → SP =Old top of stack

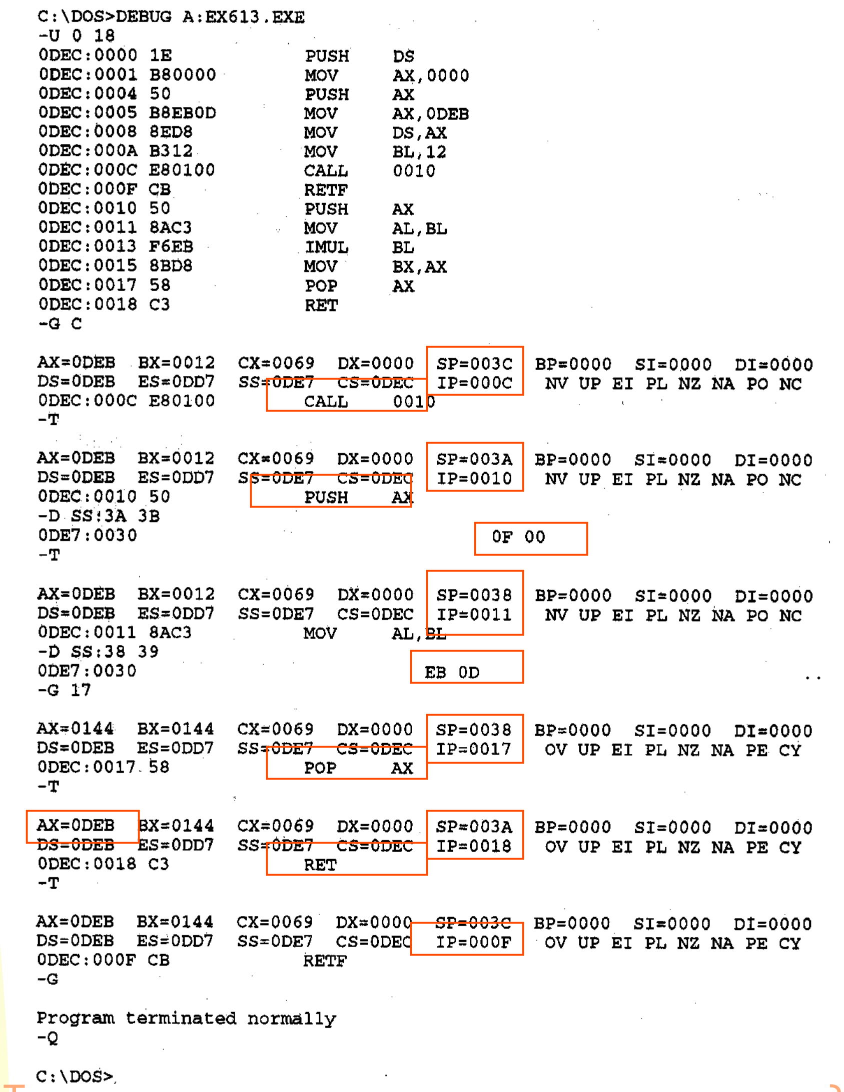
6.4 Subroutines and Subroutine-Handling Instructions- Subroutine Call Involving PUSH and POP



```
                        TITLE    EXAMPLE 6.13

                             PAGE      ,132

0000                         STACK_SEG        SEGMENT          STACK 'STACK'
0000   0040[                                  DB               64 DUP(?)
         ??
                   ]

0040                         STACK_SEG        ENDS

0000                         DATA_SEG         SEGMENT
0000   1234                  TOTAL            DW               1234H
0002                         DATA_SEG         ENDS

0000                         CODE_SEG         SEGMENT          'CODE'
0000                         EX613     PROC   FAR
                                  ASSUME   CS:CODE_SEG, SS:STACK_SEG, DS:DATA_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

                        ;Setup the data segment

0005   B8 ---- R                       MOV      AX, DATA_SEG
0008   8E D8                           MOV      DS, AX

                        ;Following code implements Example 6.13

000A   B3 12                           MOV      BL,12H          ;BL contents = the number
                                                                ;to be squared
000C   E8 0010 R                       CALL     SQUARE          ;Call the procedure to
                                                                ;square BL contents
000F   CB                              RET                      ;Return to DEBUG program
0010                         EX613     ENDP
```

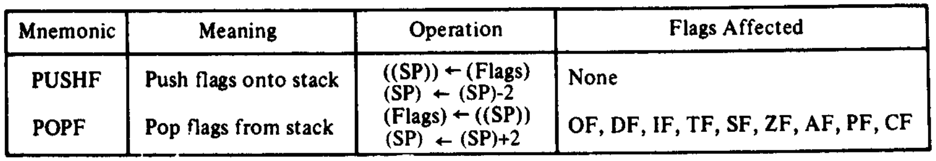
```
                                  ;Subroutine:     SQUARE
                                  ;Description:    (BX) = square of (BL)

0010                              SQUARE   PROC     NEAR
0010   50                                  PUSH     AX              ;Save the register to
                                                                    ;used
0011   8A C3                               MOV      AL,BL           ;Place the number in
0013   F6 EB                               IMUL     BL              ;Multiply with itself
0015   8B D8                               MOV      BX,AX           ;Save the result
0017   58                                  POP      AX              ;Restore the register
0018   C3                                  RET
0019                              SQUARE   ENDP

0019                              CODE_SEG          ENDS


                                  END      EX613

Segments and Groups:

                N a m e                   Length   Align   Combine Class

CODE_SEG . . . . . . . . . . . . .        0019     PARA    NONE    'CODE'
DATA_SEG . . . . . . . . . . . . .        0002     PARA    NONE
STACK_SEG  . . . . . . . . . . . .        0040     PARA    STACK   'STACK'

Symbols:

                N a m e                   Type     Value   Attr

EX613  . . . . . . . . . . . . . .        F PROC   0000    CODE_SEG        Length = 0010
SQUARE . . . . . . . . . . . . . .        N PROC   0010    CODE_SEG        Length = 0009

TOTAL  . . . . . . . . . . . . . .        L WORD   0000    DATA_SEG

@CPU . . . . . . . . . . . . . . .        TEXT  0101h
@FILENAME  . . . . . . . . . . . .        TEXT  EX613
@VERSION . . . . . . . . . . . . .        TEXT  613


     53 Source  Lines
     53 Total   Lines
     13 Symbols

  48016 + 440523 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

6.4 Subroutines and Subroutine-Handling Instructions- Subroutine Call involving PUSH and POP (continued)



```
C:\DOS>DEBUG A:EX613.EXE
-U 0 18
0DEC:0000 1E             PUSH    DS
0DEC:0001 B80000         MOV     AX,0000
0DEC:0004 50             PUSH    AX
0DEC:0005 B8EB0D         MOV     AX,0DEB
0DEC:0008 8ED8           MOV     DS,AX
0DEC:000A B312           MOV     BL,12
0DEC:000C E80100         CALL    0010
0DEC:000F CB             RETF
0DEC:0010 50             PUSH    AX
0DEC:0011 8AC3           MOV     AL,BL
0DEC:0013 F6EB           IMUL    BL
0DEC:0015 8BD8           MOV     BX,AX
0DEC:0017 58             POP     AX
0DEC:0018 C3             RET
-G C

AX=0DEB  BX=0012  CX=0069  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=000C   NV UP EI PL NZ NA PO NC
0DEC:000C E80100         CALL    0010
-T

AX=0DEB  BX=0012  CX=0069  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=0010   NV UP EI PL NZ NA PO NC
0DEC:0010 50             PUSH    AX
-D SS:3A 3B
0DE7:0030                                0F 00
-T

AX=0DEB  BX=0012  CX=0069  DX=0000  SP=0038  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=0011   NV UP EI PL NZ NA PO NC
0DEC:0011 8AC3           MOV     AL,BL
-D SS:38 39
0DE7:0030                          EB 0D
-G 17

AX=0144  BX=0144  CX=0069  DX=0000  SP=0038  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=0017   OV UP EI PL NZ NA PE CY
0DEC:0017 58             POP     AX
-T

AX=0DEB  BX=0144  CX=0069  DX=0000  SP=003A  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=0018   OV UP EI PL NZ NA PE CY
0DEC:0018 C3             RET
-T

AX=0DEB  BX=0144  CX=0069  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DEB  ES=0DD7  SS=0DE7  CS=0DEC  IP=000F   OV UP EI PL NZ NA PE CY
0DEC:000F CB             RETF
-G

Program terminated normally
-Q

C:\DOS>
```

6.4 Subroutines and Subroutine-Handling Instructions- Push Flags Instruction



Mnemonic	Meaning	Operation	Flags Affected
PUSHF	Push flags onto stack	((SP)) ← (Flags) (SP) ← (SP)-2	None
POPF	Pop flags from stack	(Flags) ← ((SP)) (SP) ← (SP)+2	OF, DF, IF, TF, SF, ZF, AF, PF, CF

- Push flags instruction
  - General formats:

    PUSHF
    - Saves flags onto the stack
    - Operation

    (FLAGS) → ((SP))

    (SP)-2 → (SP) = New top of stack
- Pop flags instruction
  - General formats:

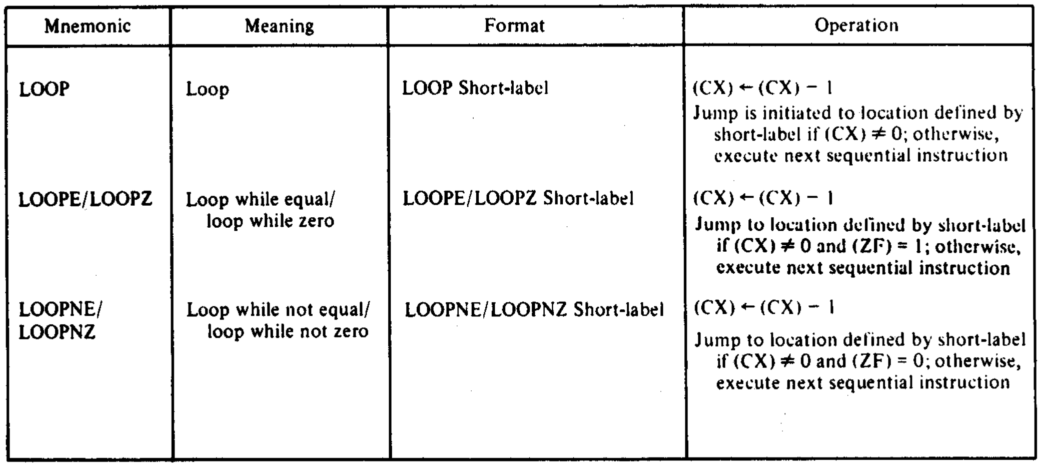
    POPF
    - Restores flags from the stack

    ((SP)) → FLAGS

    (SP)+2 → (SP) =Old top of stack

6.5 The Loop and Loop-Handling Instructions- Loop Instructions

- Loop—segment of program that is repeatedly executed
  - Can be implemented with compare, conditional jump, and decrement instructions
- Loop instructions
  - Special instructions that efficiently perform basic loop operations
  - Replace the multiple instructions with a single instruction
  - LOOP—loop while not zero
    - CX ≠ 0 — repeat while count not zero
  - LOOPE/LOOPZ– loop while equal
    - CX ≠ 0 — repeat while count not zero, and
    - ZF = 1—result of prior instruction was equal
  - LOOPNE/LOOPNZ—loop while not equal
    - CX ≠ 0 — repeat while count not zero, and
    - ZF = 0—result from prior instruction was not equal



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

6.5 The Loop and Loop-Handling Instructions- Loop Program Structure and Operation

- Structure of a loop
  - Initialization of the count in CX
  - Body—instruction sequence that is to be repeated; short label identifying beginning
  - Loop instruction– determines if loop is complete or if the body is to repeat
- Example

1.Initialize data segment, source and destination block pointers, and loop count

2. Body of program is executed—source element read, written to destination, and then both pointers incremented by 1

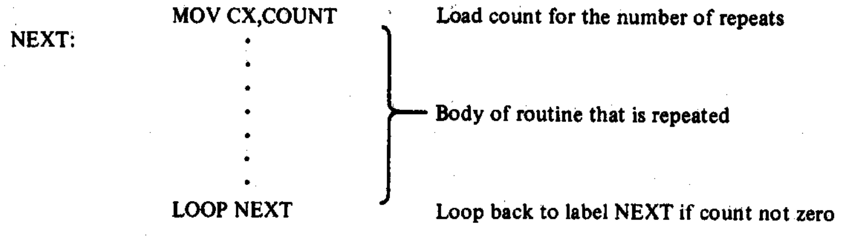
3. Loop test

a. Contents of CX decremented by 1

b. Contents of CX check for zero

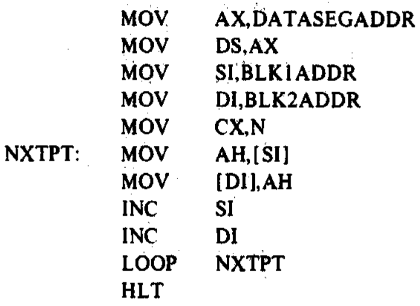
c. If CX = 0, loop is complete and next sequential instruction (HLT) is executed

d. If CX ≠ 0, loop of code is repeated by returning control to the instruction corresponding to the Short-Label (NXTPT:) operand



```
            MOV CX,COUNT          Load count for the number of repeats
NEXT:            .
                 .
                 .                Body of routine that is repeated
                 .
                 .
                 .
            LOOP NEXT             Loop back to label NEXT if count not zero
```

(a)

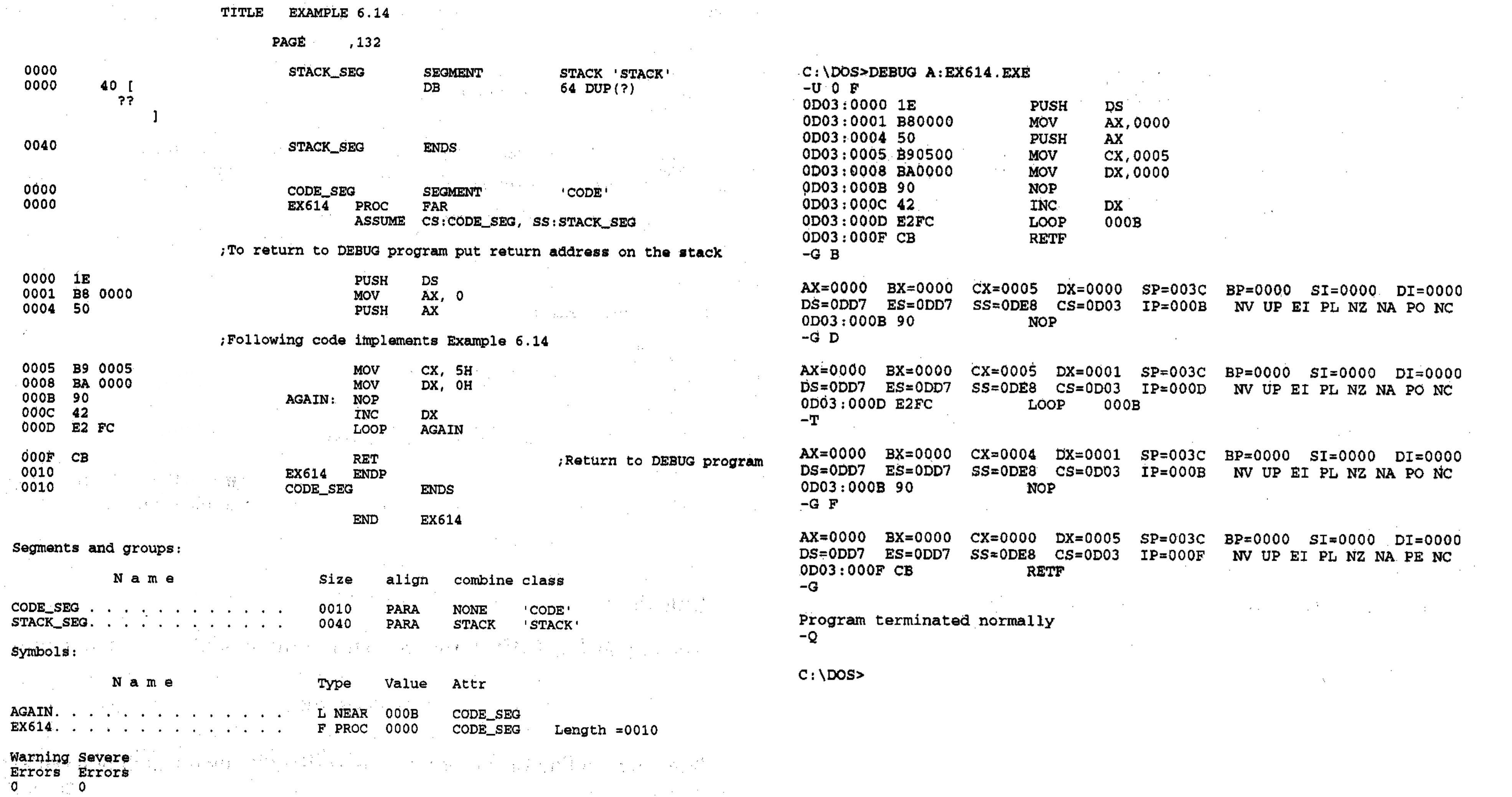


```
        MOV     AX,DATASEGADDR
        MOV     DS,AX
        MOV     SI,BLK1ADDR
        MOV     DI,BLK2ADDR
        MOV     CX,N
NXTPT:  MOV     AH,[SI]
        MOV     [DI],AH
        INC     SI
        INC     DI
        LOOP    NXTPT
        HLT
```

(b)

6.5 The Loop and Loop-Handling Instructions-

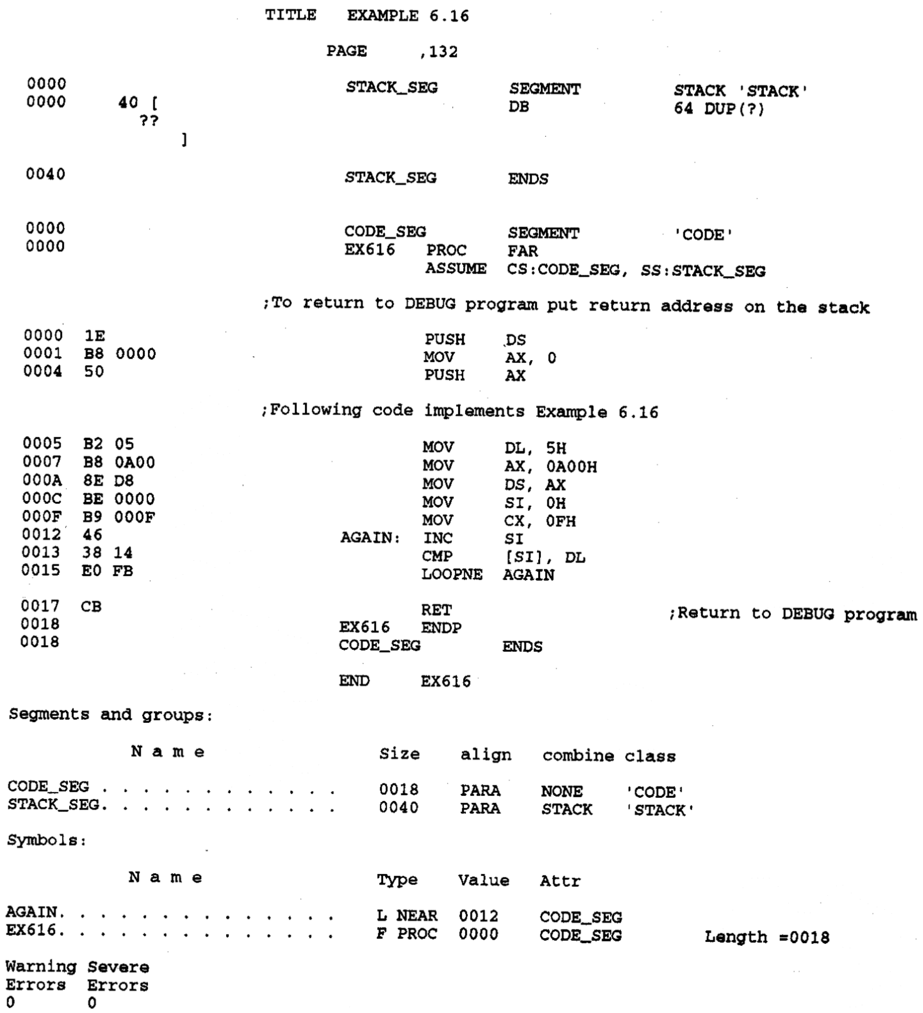
Loop Example—Loop Count Operation



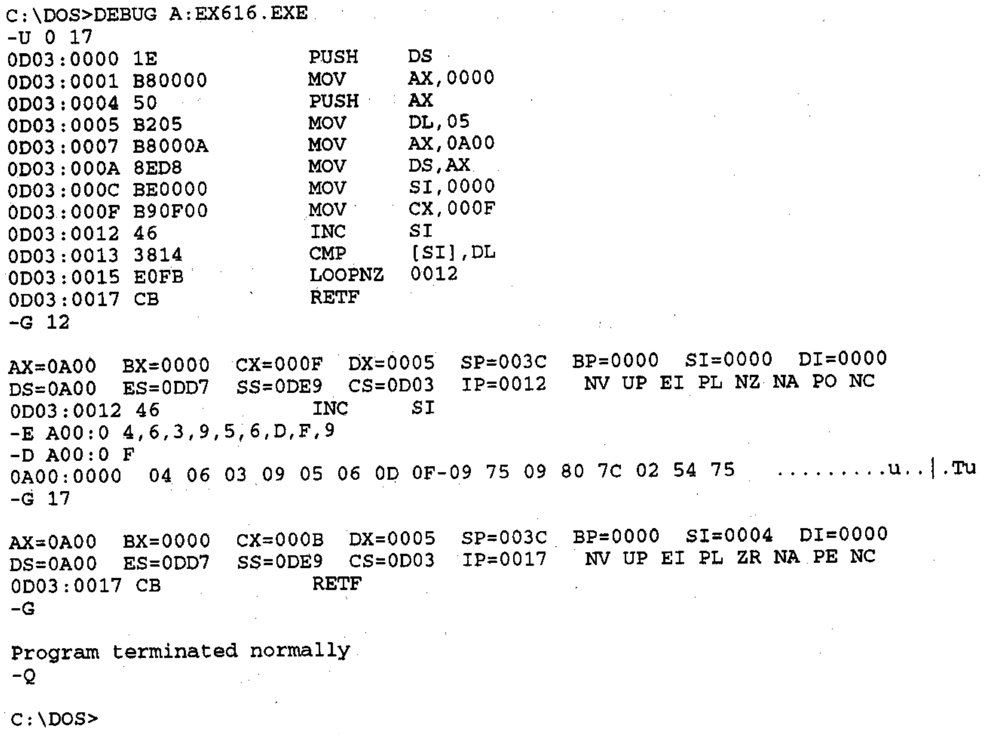
```
                  TITLE   EXAMPLE 6.14

                          PAGE     ,132

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

0040                      STACK_SEG        ENDS

0000                      CODE_SEG         SEGMENT          'CODE'
0000                      EX614     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 6.14

0005  B9 0005                       MOV    CX, 5H
0008  BA 0000                       MOV    DX, 0H
000B  90                  AGAIN:    NOP
000C  42                            INC    DX
000D  E2 FC                         LOOP   AGAIN

000F  CB                            RET                     ;Return to DEBUG program
0010                      EX614     ENDP
0010                      CODE_SEG         ENDS

                                    END    EX614

Segments and groups:

            N a m e                 Size    align   combine class

CODE_SEG . . . . . . . . . . . . .  0010    PARA    NONE    'CODE'
STACK_SEG. . . . . . . . . . . . .  0040    PARA    STACK   'STACK'

Symbols:

            N a m e                 Type    Value   Attr

AGAIN. . . . . . . . . . . . . . .  L NEAR  000B    CODE_SEG
EX614. . . . . . . . . . . . . . .  F PROC  0000    CODE_SEG     Length =0010

Warning Severe
Errors  Errors
0       0
```

```
C:\DOS>DEBUG A:EX614.EXE
-U 0 F
0D03:0000 1E            PUSH    DS
0D03:0001 B80000        MOV     AX,0000
0D03:0004 50            PUSH    AX
0D03:0005 B90500        MOV     CX,0005
0D03:0008 BA0000        MOV     DX,0000
0D03:000B 90            NOP
0D03:000C 42            INC     DX
0D03:000D E2FC          LOOP    000B
0D03:000F CB            RETF
-G B

AX=0000  BX=0000  CX=0005  DX=0000  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD7  ES=0DD7  SS=0DE8  CS=0D03  IP=000B   NV UP EI PL NZ NA PO NC
0D03:000B 90            NOP
-G D

AX=0000  BX=0000  CX=0005  DX=0001  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD7  ES=0DD7  SS=0DE8  CS=0D03  IP=000D   NV UP EI PL NZ NA PO NC
0D03:000D E2FC          LOOP    000B
-T

AX=0000  BX=0000  CX=0004  DX=0001  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD7  ES=0DD7  SS=0DE8  CS=0D03  IP=000B   NV UP EI PL NZ NA PO NC
0D03:000B 90            NOP
-G F

AX=0000  BX=0000  CX=0000  DX=0005  SP=003C  BP=0000  SI=0000  DI=0000
DS=0DD7  ES=0DD7  SS=0DE8  CS=0D03  IP=000F   NV UP EI PL NZ NA PE NC
0D03:000F CB            RETF
-G

Program terminated normally
-Q

C:\DOS>
```

6.5 The Loop and Loop-Handling Instructions- Loop Example—Block Search Operation



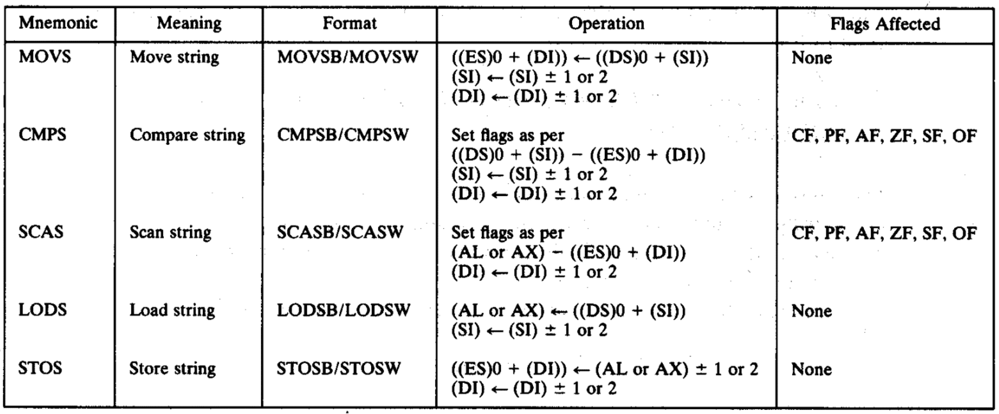
```
                         TITLE    EXAMPLE 6.16

                             PAGE      ,132

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

0040                          STACK_SEG        ENDS

0000                          CODE_SEG         SEGMENT          'CODE'
0000                          EX616    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 6.16

0005  B2 05                            MOV     DL, 5H
0007  B8 0A00                          MOV     AX, 0A00H
000A  8E D8                            MOV     DS, AX
000C  BE 0000                          MOV     SI, 0H
000F  B9 000F                          MOV     CX, 0FH
0012  46                      AGAIN:   INC     SI
0013  38 14                            CMP     [SI], DL
0015  E0 FB                            LOOPNE  AGAIN

0017  CB                               RET                      ;Return to DEBUG program
0018                          EX616    ENDP
0018                          CODE_SEG         ENDS

                              END      EX616

Segments and groups:

          N a m e                Size    align    combine class

CODE_SEG . . . . . . . . . . . . 0018    PARA     NONE    'CODE'
STACK_SEG. . . . . . . . . . . . 0040    PARA     STACK   'STACK'

Symbols:

          N a m e                Type    Value    Attr

AGAIN. . . . . . . . . . . . . . L NEAR  0012     CODE_SEG
EX616. . . . . . . . . . . . . . F PROC  0000     CODE_SEG         Length =0018

Warning Severe
Errors  Errors
0       0
```



```
C:\DOS>DEBUG A:EX616.EXE
-U 0 17
0D03:0000 1E               PUSH    DS
0D03:0001 B80000           MOV     AX,0000
0D03:0004 50               PUSH    AX
0D03:0005 B205             MOV     DL,05
0D03:0007 B8000A           MOV     AX,0A00
0D03:000A 8ED8             MOV     DS,AX
0D03:000C BE0000           MOV     SI,0000
0D03:000F B90F00           MOV     CX,000F
0D03:0012 46               INC     SI
0D03:0013 3814             CMP     [SI],DL
0D03:0015 E0FB             LOOPNZ  0012
0D03:0017 CB               RETF
-G 12

AX=0A00  BX=0000  CX=000F  DX=0005  SP=003C  BP=0000  SI=0000  DI=0000
DS=0A00  ES=0DD7  SS=0DE9  CS=0D03  IP=0012   NV UP EI PL NZ NA PO NC
0D03:0012 46               INC     SI
-E A00:0 4,6,3,9,5,6,D,F,9
-D A00:0 F
0A00:0000  04 06 03 09 05 06 0D 0F-09 75 09 80 7C 02 54 75   .........u..|.Tu
-G 17

AX=0A00  BX=0000  CX=000B  DX=0005  SP=003C  BP=0000  SI=0004  DI=0000
DS=0A00  ES=0DD7  SS=0DE9  CS=0D03  IP=0017   NV UP EI PL ZR NA PE NC
0D03:0017 CB               RETF
-G

Program terminated normally
-Q

C:\DOS>
```

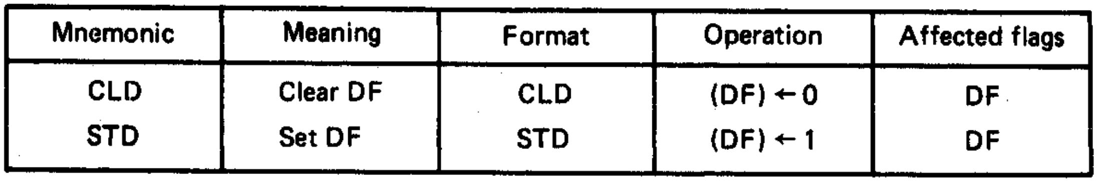
6.6 Strings and String-Handling Instructions-
String Instructions

- String—series of bytes or words of data that reside at consecutive memory addresses
- String instructions
  - Special instructions that efficiently perform basic string operations
  - Replaces multiple instructions with a single instruction
  - Examples
    - Move string
    - Compare string
    - Scan string
    - Load string
    - Store string
    - Repeated string
- Typical string operations
  - Move a string of data elements from one part of memory to another—block move
  - Scan through a string of data elements in memory looking for a specific value
  - Compare the elements of two strings of data elements in memory to determine if they are the same or different
  - Initialize a group of consecutive storage locations in memory



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

6.6 Strings and String-Handling Instructions- Autoindexing



Mnemonic	Meaning	Format	Operation	Affected flags
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

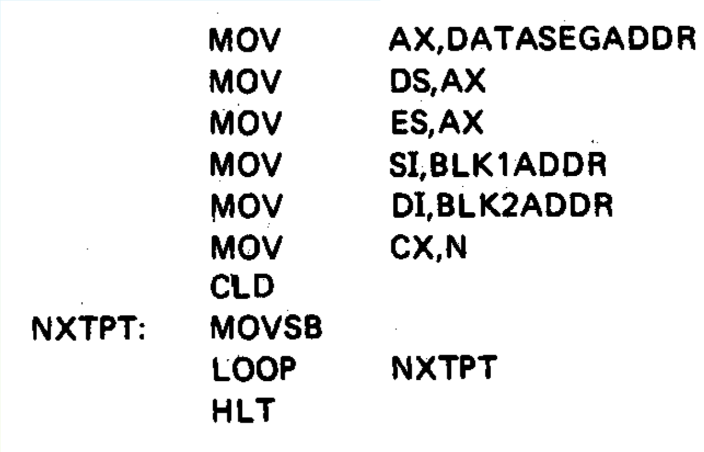
- Autoindexing—name given to the process of automatically incrementing or decrementing the source and destination addresses by the string instructions
  - Direction (DF) control flag of the status register determines mode of operation
    - DF= 0 → autoincrement
    - DF = 1 → autodecrement
  - Increment or decrement is by 1 or 2 depending on size data specified in the instruction
  - Direction flag instructions permit the DF bit to be cleared or set as part of a string routine
    - CLD—clear direction flag

      0 → (DF) = autoincrement
    - STD—set direction flag

      1 → (DF) = autodecrement

Moved from later

6.6 Strings and String-Handling Instructions- Move String Instruction



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

- Move string instruction
  - Used to move an element of data between a source and destination location in memory:
  - General format:

    MOVSB—move string byte

    MOVSW—move string word
  - Operation: Copies the content of the source to the destination; autoincrements/decrements both the source and destination addresses

    ((DS)0+(SI)) → ((ES)0+(DI))

    (SI) ±1 or 2 → (SI)

    (DI) ±1 or 2 → (DI)
- Direction flag determines increment/decrement

  DF = 0 → autoincrement

  DF = 1 → autodecrement
- Application example—block move

  1. Initialize DS & ES to same value
  2. Load SI and DI with block starting addresses
  3. Load CX with the count of elements in the string
  4. Set DF for autoincrement
  5. Loop on string move to copy N elements
- MOVSB and LOOP replaces multiple move and increment/decrement instructions

6.6 Strings and String-Handling Instructions- Compare/Scan String Instructions

- Compare string instruction
  - Used to compare the destination element of data in memory to the source element in memory and reflect the result of the comparison in the flags
  - General format:

    CMPSB,SW—compare string byte, word
  - Operation: Compares the content of the destination to the source; updates the flags; autoincrements/decrements both the source and destination addresses

    ((DS)0+(SI)) - ((ES)0+(DI))

    update status flags

    (SI) ± 1 or 2 → (SI)

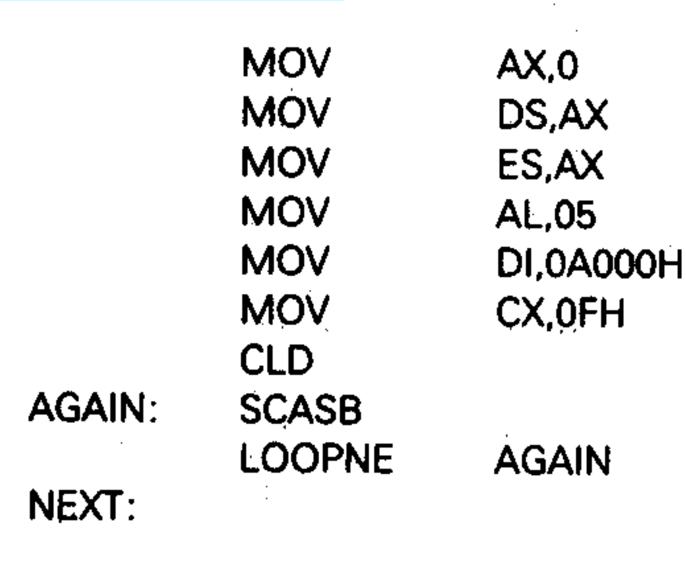
    (DI) ± 1 or 2 → (DI)
- Scan string instruction—SCAS
  - Same operation as CMPS except destination is compared to a value in the accumulator (A) register

    (AL,AX) - ((ES)0+(DI))
- Application example—block scan

  1. Initialize DS & ES to same value

  2. Load AL with search value; DI with block starting address; and CX with the count of elements in the string; clear DF

  3. Loop on scan string until the first element equal to 05H is found



```
        MOV     AX,0
        MOV     DS,AX
        MOV     ES,AX
        MOV     AL,05
        MOV     DI,0A000H
        MOV     CX,0FH
        CLD
AGAIN:  SCASB
        LOOPNE  AGAIN
NEXT:
```

6.6 Strings and String-Handling Instructions-
Load/Store String Instructions

- Load string instruction
  - Used to load a source element of data from memory into the accumulator register.
  - General format:

    LODSB,SW—load string byte, word
  - Operation: Loads the content of the source element in the accumulator; autoincrements/decrements the source addresses

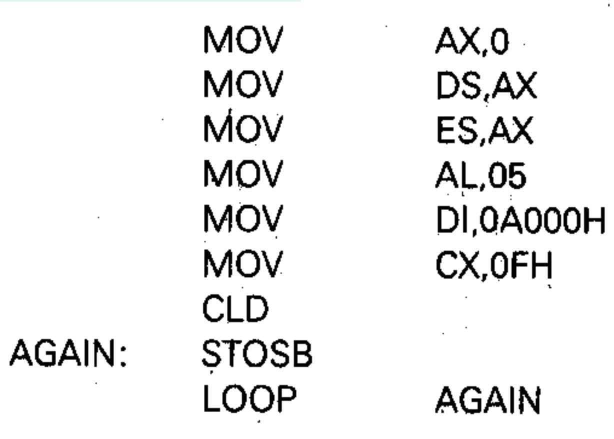
    ((DS)0+(SI)) → (AL or AX)

    update status flags

    (SI) ± 1 or 2 → (SI)
- Store string instruction—STOS
  - Same operation as LODS except value in accumulator is stored in destination is memory

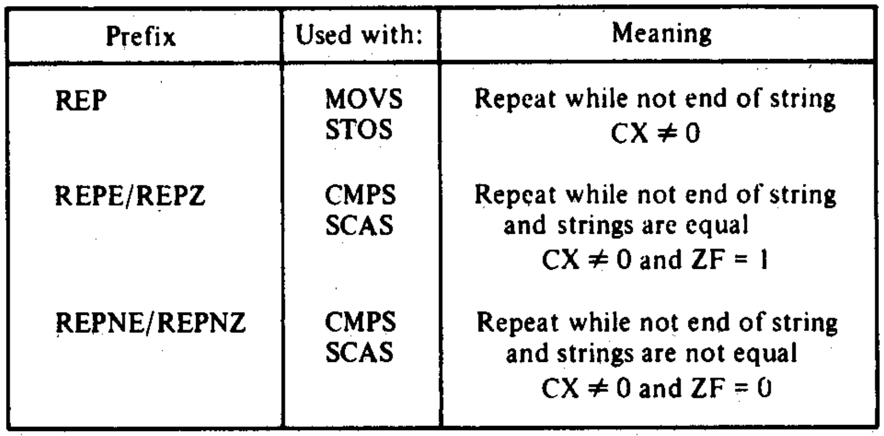
    (AL,AX) → ((ES)0+(DI))
- Application example—initializing a block of memory

1. Initialize DS & ES to same value
2. Load AL with initialization value; DI with block starting address, CX with the count of elements in the string; and clear DF
3. Loop on store string until all element of the string are initialized to 05H



```
        MOV     AX,0
        MOV     DS,AX
        MOV     ES,AX
        MOV     AL,05
        MOV     DI,0A000H
        MOV     CX,0FH
        CLD
AGAIN:  STOSB
        LOOP    AGAIN
```

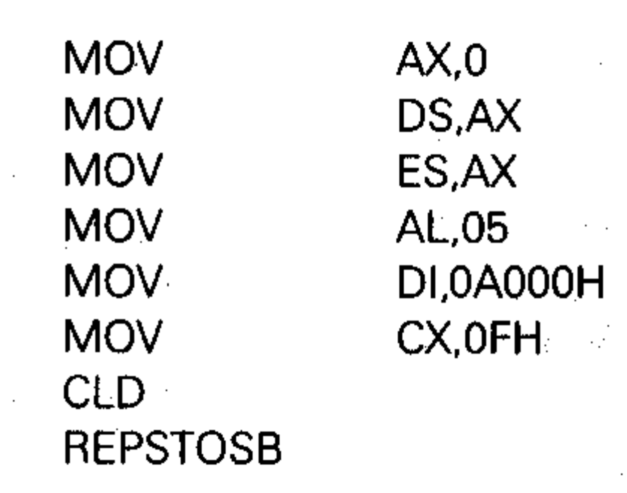
6.6 Strings and String-Handling Instructions-
Repeat String Instructions



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

- Repeat string—in most applications the basic string operations are repeated
  - Requires addition of loop or compare & conditional jump instructions
  - Repeat prefix provided to make coding of repeated sting more efficient
- Repeat prefixes
  - REP
    - CX ≠ 0 — repeat while not end of string
    - Used with: MOVS and STOS
  - REPE/REPZ
    - CX ≠ 0—repeat while not end of string,
      and
      ZF = 1—strings are equal
    - Used with: CMPS and SCAS
  - REPNE/REPNZ—Used with: CMPS and SCAS
    - CX ≠ 0—repeat while not end of string,
      and
      ZF = 0—strings are not equal
    - Used with: CMPS and SCAS

6.6 Strings and String-Handling Instructions- Repeat String Examples and Application



```
MOV         AX,0
MOV         DS,AX
MOV         ES,AX
MOV         AL,05
MOV         DI,0A000H
MOV         CX,0FH
CLD
REPSTOSB
```

- General format:

  REPXXXX

  Where: XXXX = one of string instructions
- Examples:

  REPMOVB

  REPESCAS

  REPNESCAS
- Application example—initializing a block of memory

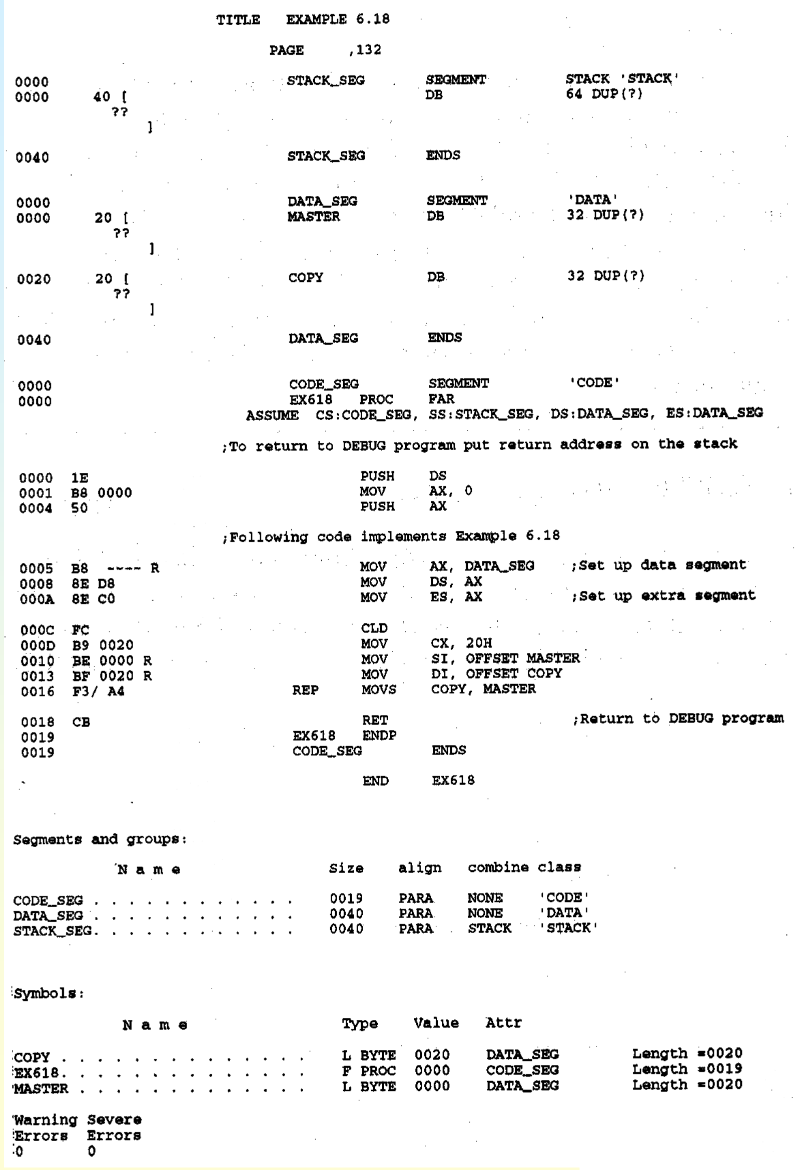
  1. Initialize DS & ES to same value

  2. Load AL with initialization value; DI with block starting address, and CX with the count of elements in the string

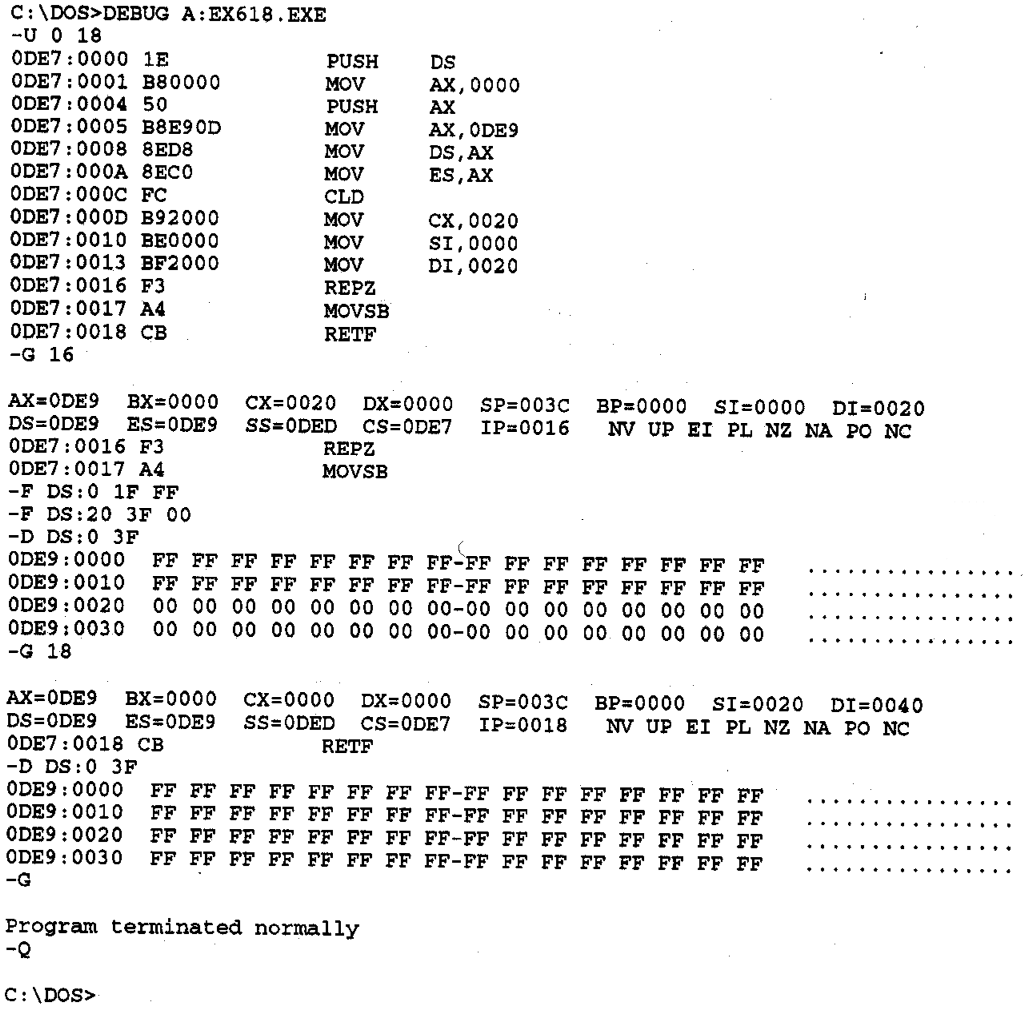
  4. Clear the direction flag for autoincrement mode

  4. Repeat store string until all elements of the string are initialized to 05H

6.6 Strings and String-Handling Instructions- Example String Application



```
                         TITLE    EXAMPLE 6.18

                              PAGE      ,132

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

0040                          STACK_SEG          ENDS

0000                          DATA_SEG           SEGMENT          'DATA'
0000       20 [               MASTER             DB               32 DUP(?)
             ??
                ]

0020       20 [               COPY               DB               32 DUP(?)
             ??
                ]

0040                          DATA_SEG           ENDS

0000                          CODE_SEG           SEGMENT          'CODE'
0000                          EX618    PROC      FAR
                          ASSUME  CS:CODE_SEG, SS:STACK_SEG, DS:DATA_SEG, ES:DATA_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 6.18

0005  B8  ---- R                        MOV      AX, DATA_SEG     ;Set up data segment
0008  8E D8                             MOV      DS, AX
000A  8E C0                             MOV      ES, AX           ;Set up extra segment

000C  FC                                CLD
000D  B9 0020                           MOV      CX, 20H
0010  BE 0000 R                         MOV      SI, OFFSET MASTER
0013  BF 0020 R                         MOV      DI, OFFSET COPY
0016  F3/ A4                  REP       MOVS     COPY, MASTER

0018  CB                                RET                       ;Return to DEBUG program
0019                          EX618     ENDP
0019                          CODE_SEG           ENDS

                                        END      EX618


Segments and groups:

               N a m e                  Size    align   combine class

CODE_SEG . . . . . . . . . . . .        0019    PARA    NONE    'CODE'
DATA_SEG . . . . . . . . . . . .        0040    PARA    NONE    'DATA'
STACK_SEG. . . . . . . . . . . .        0040    PARA    STACK   'STACK'


Symbols:

               N a m e                  Type    Value   Attr

COPY . . . . . . . . . . . . . .        L BYTE  0020    DATA_SEG        Length =0020
EX618. . . . . . . . . . . . . .        F PROC  0000    CODE_SEG        Length =0019
MASTER . . . . . . . . . . . . .        L BYTE  0000    DATA_SEG        Length =0020

Warning Severe
Errors  Errors
0       0
```



```
C:\DOS>DEBUG A:EX618.EXE
-U 0 18
0DE7:0000 1E            PUSH    DS
0DE7:0001 B80000        MOV     AX,0000
0DE7:0004 50            PUSH    AX
0DE7:0005 B8E90D        MOV     AX,0DE9
0DE7:0008 8ED8          MOV     DS,AX
0DE7:000A 8EC0          MOV     ES,AX
0DE7:000C FC            CLD
0DE7:000D B92000        MOV     CX,0020
0DE7:0010 BE0000        MOV     SI,0000
0DE7:0013 BF2000        MOV     DI,0020
0DE7:0016 F3            REPZ
0DE7:0017 A4            MOVSB
0DE7:0018 CB            RETF
-G 16

AX=0DE9  BX=0000  CX=0020  DX=0000  SP=003C  BP=0000  SI=0000  DI=0020
DS=0DE9  ES=0DE9  SS=0DED  CS=0DE7  IP=0016   NV UP EI PL NZ NA PO NC
0DE7:0016 F3            REPZ
0DE7:0017 A4            MOVSB
-F DS:0 1F FF
-F DS:20 3F 00
-D DS:0 3F
0DE9:0000  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
0DE9:0010  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
0DE9:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0DE9:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G 18

AX=0DE9  BX=0000  CX=0000  DX=0000  SP=003C  BP=0000  SI=0020  DI=0040
DS=0DE9  ES=0DE9  SS=0DED  CS=0DE7  IP=0018   NV UP EI PL NZ NA PO NC
0DE7:0018 CB            RETF
-D DS:0 3F
0DE9:0000  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
0DE9:0010  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
0DE9:0020  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
0DE9:0030  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
-G

Program terminated normally
-Q

C:\DOS>
```