Subject: Microprocessors

Program Control Instructions

By:

Dr. Vandana Gandotra

Deptt of Computer Science

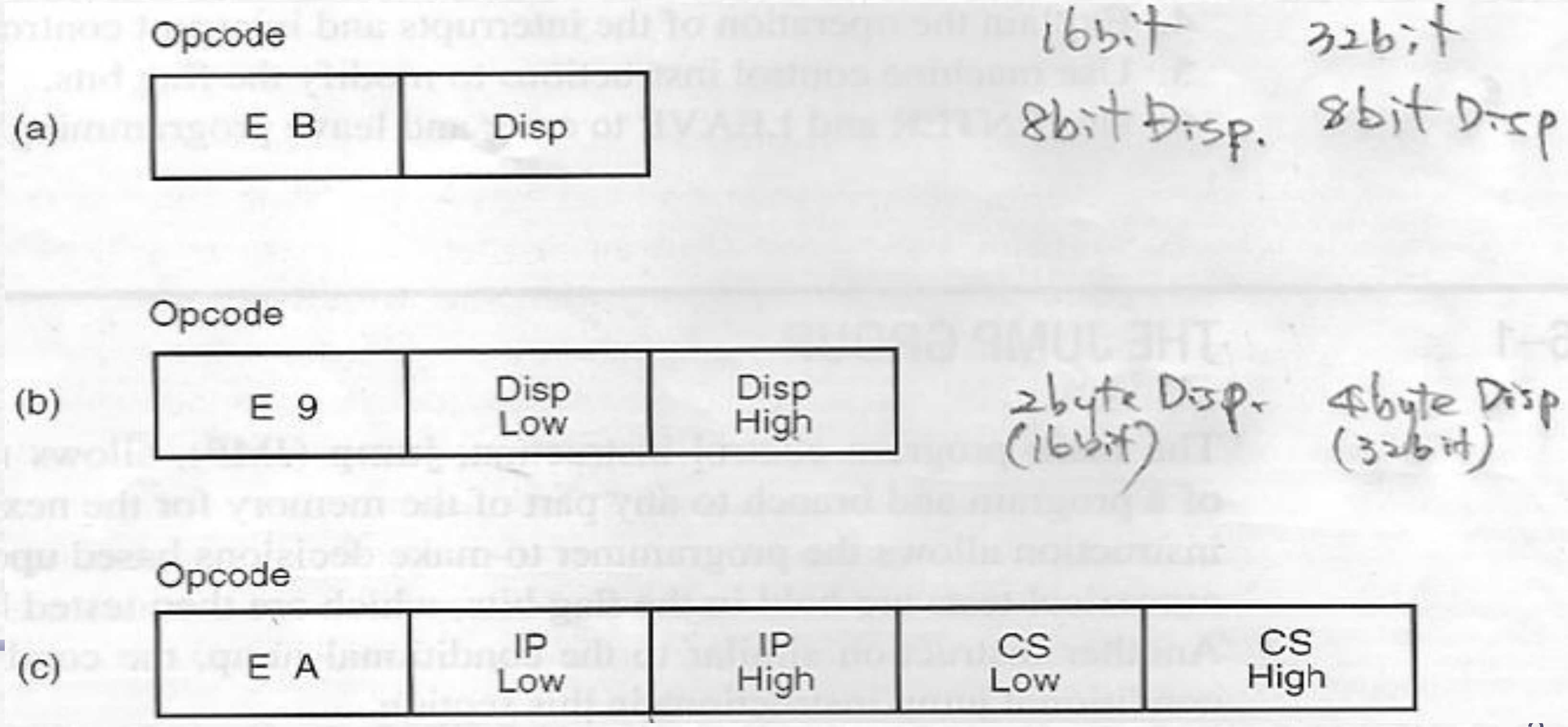
Ram Lal Anand College

Introduction

- Program control instruction :
  - direct the flow of a program, allow the flow to change
  - jumps, calls, returns, interrupts, machine control instructions
- Change in flow :
  - CMP, TEST followed by conditional jump
- Relational assembly language statements :
  - .IF, .ELSE, .ELSEIF, .WHILE, .ENDW, .REPEAT, .UNTIL
  - MASM, TASM Ver.6X ~
  - allow to develop control flow portions of program with C/C++ language efficiency

The Jump Group

- JMP(jump) : allow to skip sections of a program and blanch to any part of memory for next instruction
- unconditional jump, conditional jump
- three type unconditional jump : Fig. 6-1



16bit 32bit

(a) Opcode

E B	Disp

8bit Disp. 8bit Disp

(b) Opcode

E 9	Disp Low	Disp High

2byte Disp. (16bit) 4byte Disp (32bit)

(c) Opcode

E A	IP Low	IP High	CS Low	CS High

Unconditional Jump(JMP)

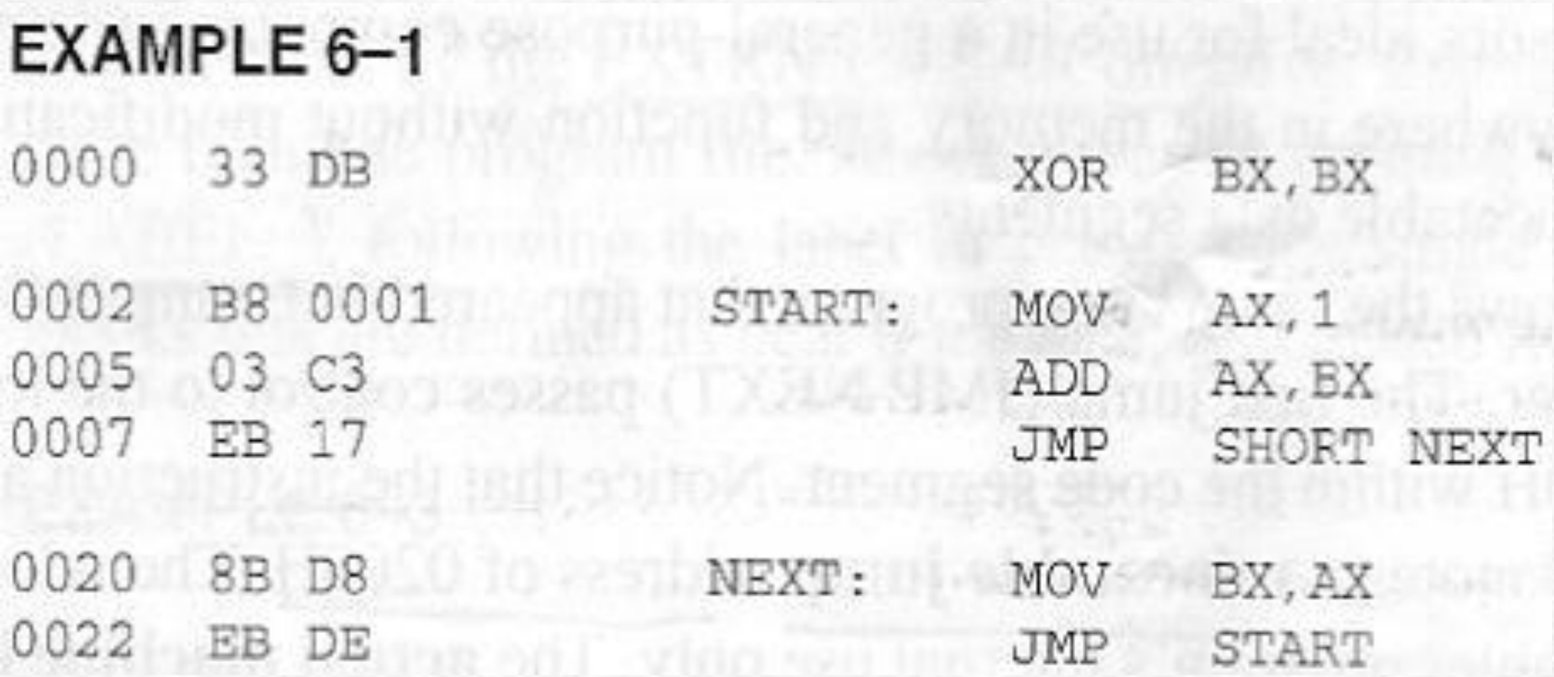
- Intrasegment jump : short, near jump
  - Short jump(2-byte): 1 byte disp.(within +127~-128 byte)
  - Near jump(3-byte) : 2 byte disp.(within ±32K bytes or anywhere in current code segment)
- Segments : cyclic in nature
- Intersegment, far jump(5-byte) :
  - any memory location within the real memory system
- 80386~ (in protected mode)
  - Near(5-byte) : 4 byte displacement(within ±2G bytes)
  - Far(7-byte) : 4 byte(EIP), 2 byte(CS)

Short Jump

- Short jump : relative jump
  - distance or displacement : follow the opcode
- One-byte signed number(+127~-128) :
  - sign-extended and added to IP/EIP
  - to generate the jump address within current code segment
- EX. 6-1 :
- Label : symbolic name for memory address
- SHORT directive : force a short jump
- most assembler : choose best form of jump instruction
- JMP START : assemble as a short jump

Short Jump

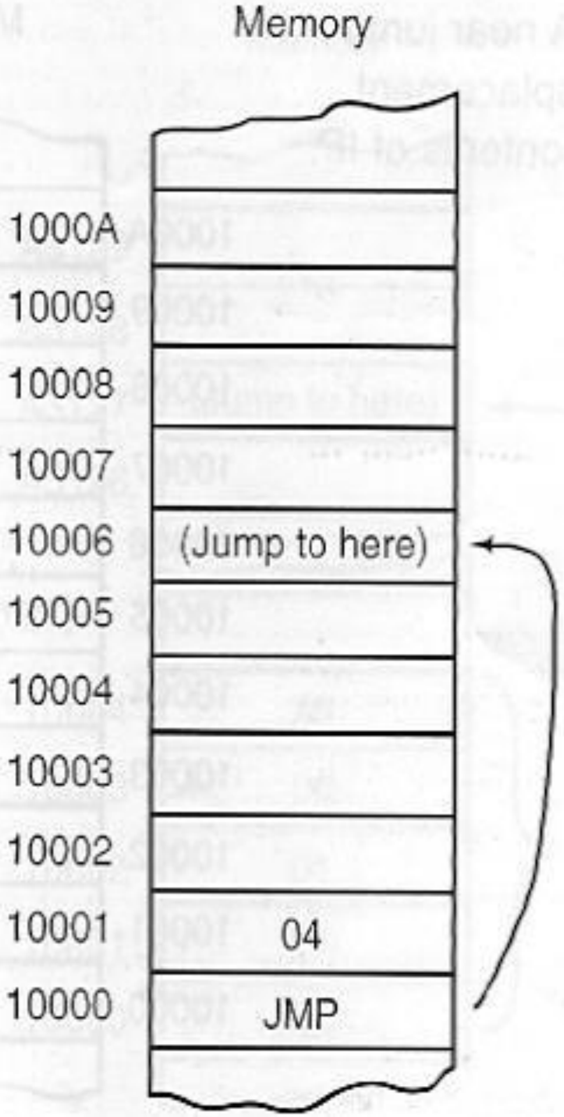
- 1st jump : 0020H – 0009H = 0017(disp. = 17H)
- 2nd jump : 0002H – 0024H = FFDEH(disp. = DEH)



EXAMPLE 6–1

```
0000   33 DB                 XOR    BX,BX

0002   B8 0001      START:   MOV    AX,1
0005   03 C3                 ADD    AX,BX
0007   EB 17                 JMP    SHORT NEXT

0020   8B D8        NEXT:    MOV    BX,AX
0022   EB DE                 JMP    START
```

FIGURE 6–2 A short jump to four memory locations beyond the address of the next instruction.

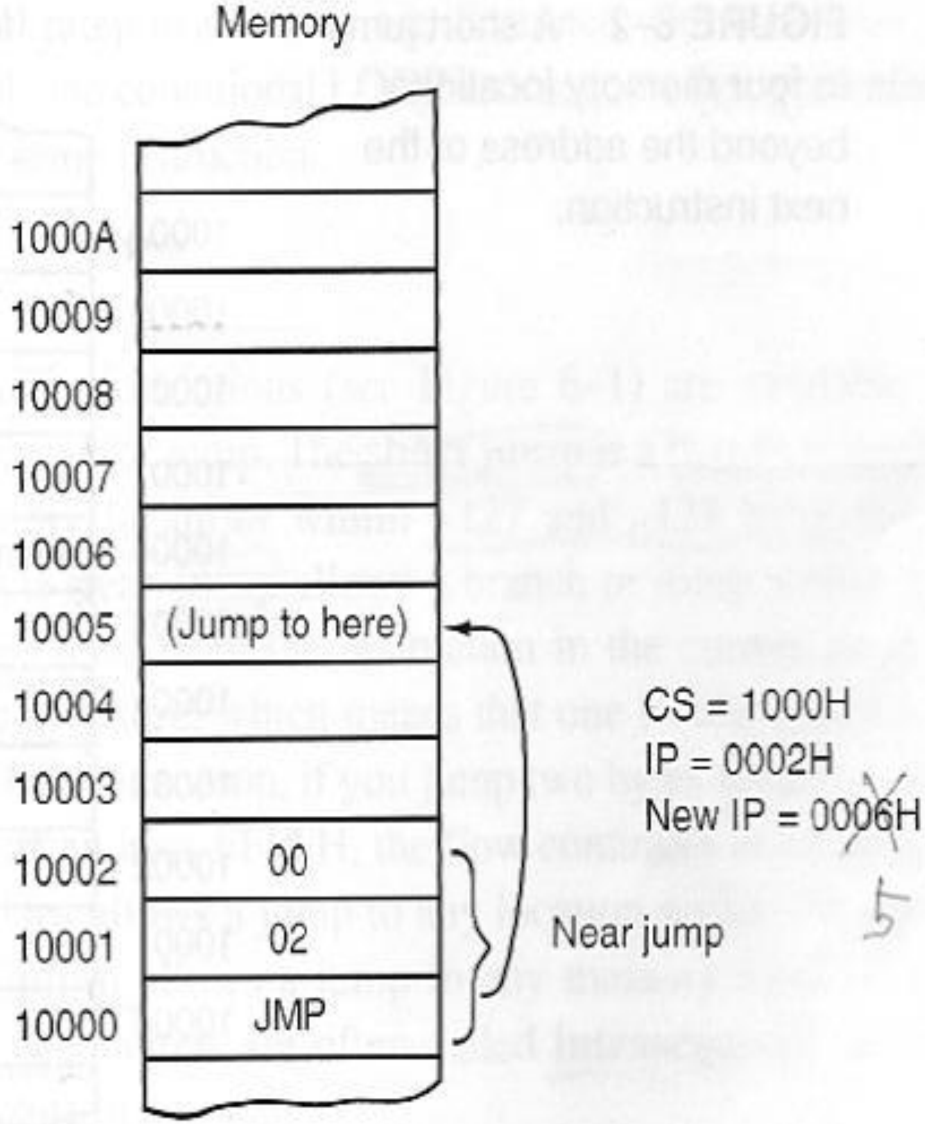


CS = 1000H
IP = 0002H
New IP = IP + 4
New IP = 0006H

Near, Far Jump

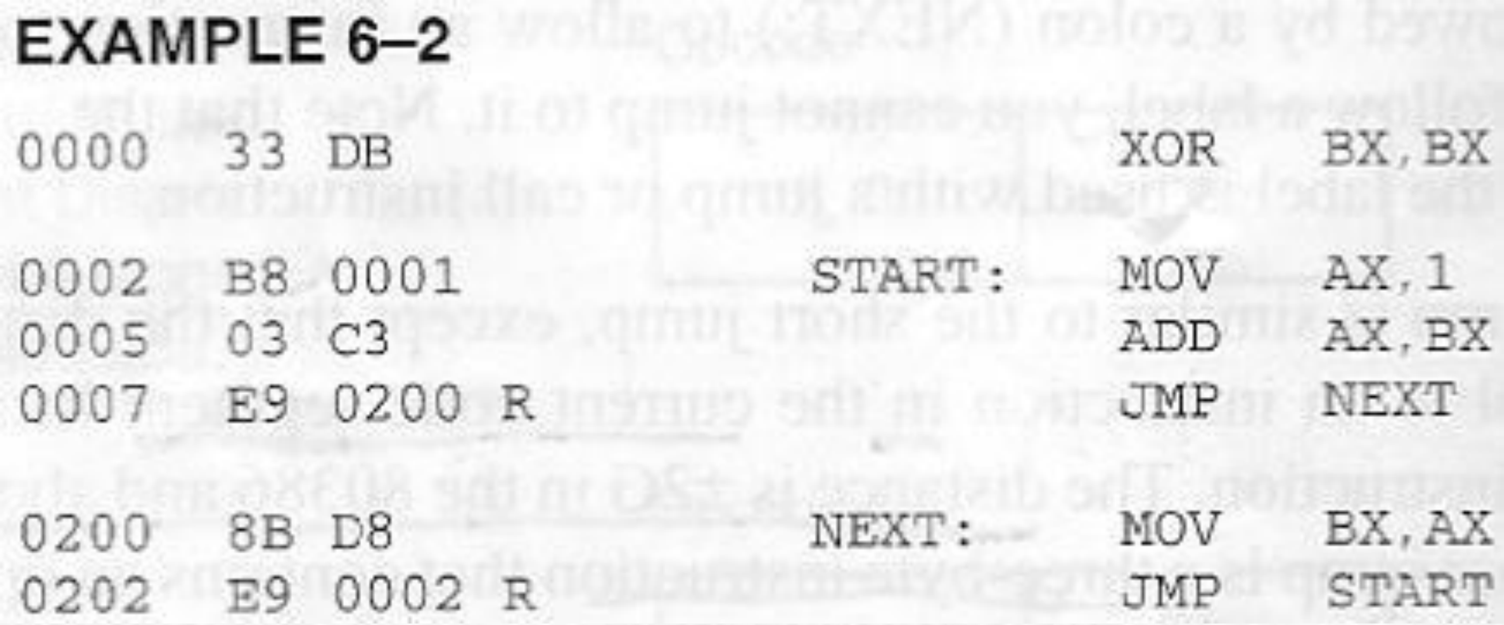
- Near jump : relocatable because relative jump
- signed displacement : added to IP/EIP to generate the jump address
  - 2 byte : ±32K bytes in current code segment
  - 4-byte(386~ in protected mode) : ±2G bytes
- Far jump : 5(7, 80386~) byte instruction
  - new offset address(IP/EIP) : byte 2,3(2~5)
  - new segment address(CS) : byte 4,5(6,7)
- 80286~ in protected mode : CS access a descriptor that contain base address of far jump segment

FIGURE 6–3 A near jump that adds the displacement (0002H) to the contents of IP.



Example: Near Jump

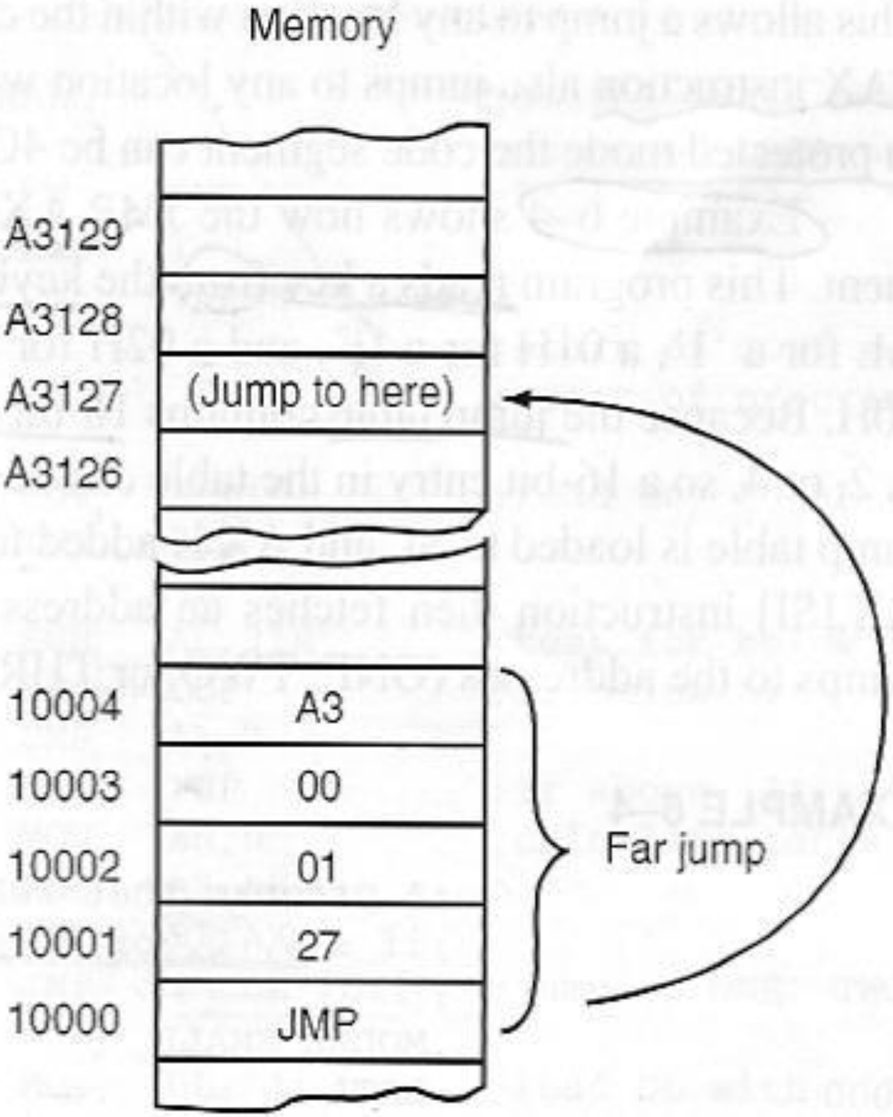
- E9 0200 R   JMP NEXT : only list file
- R : denote a relocatable jump address of 0200H
  - actual machine code : E9 F6 01
  - 0200H - 000AH = 01F6H



EXAMPLE 6–2

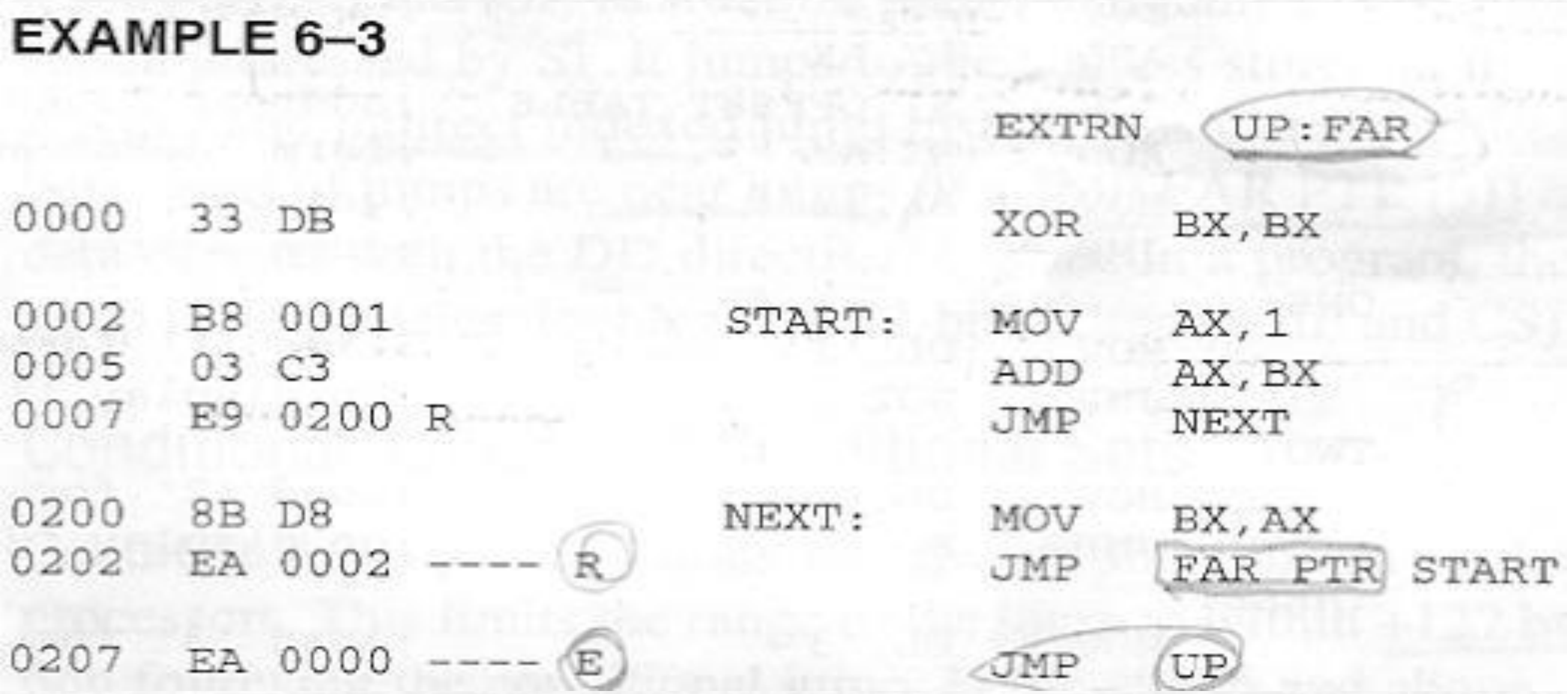
```
0000   33 DB                          XOR    BX,BX

0002   B8 0001              START:    MOV    AX,1
0005   03 C3                          ADD    AX,BX
0007   E9 0200 R                      JMP    NEXT

0200   8B D8                NEXT:     MOV    BX,AX
0202   E9 0002 R                      JMP    START
```

FIGURE 6–4 A far jump instruction replaces the contents of both CS and IP with four bytes following the opcode.



Example

- Far jump : FAR PTR directive, far label
- Far label : external to current code segment
  - EXTRN UP:FAR directive
  - a global label as a double colon(LABEL::)
- ----E : external. filled in by linker when links program files

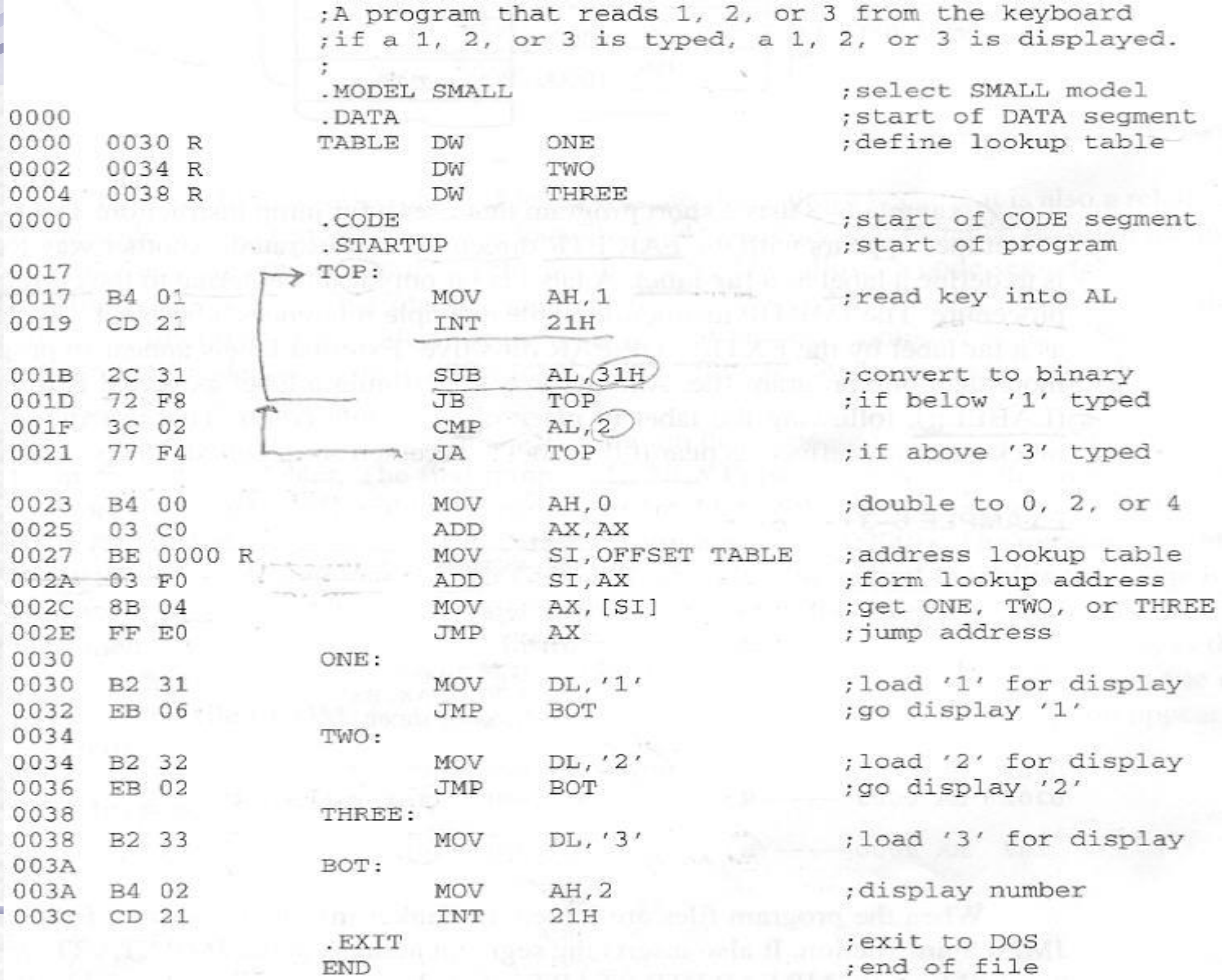


EXAMPLE 6–3

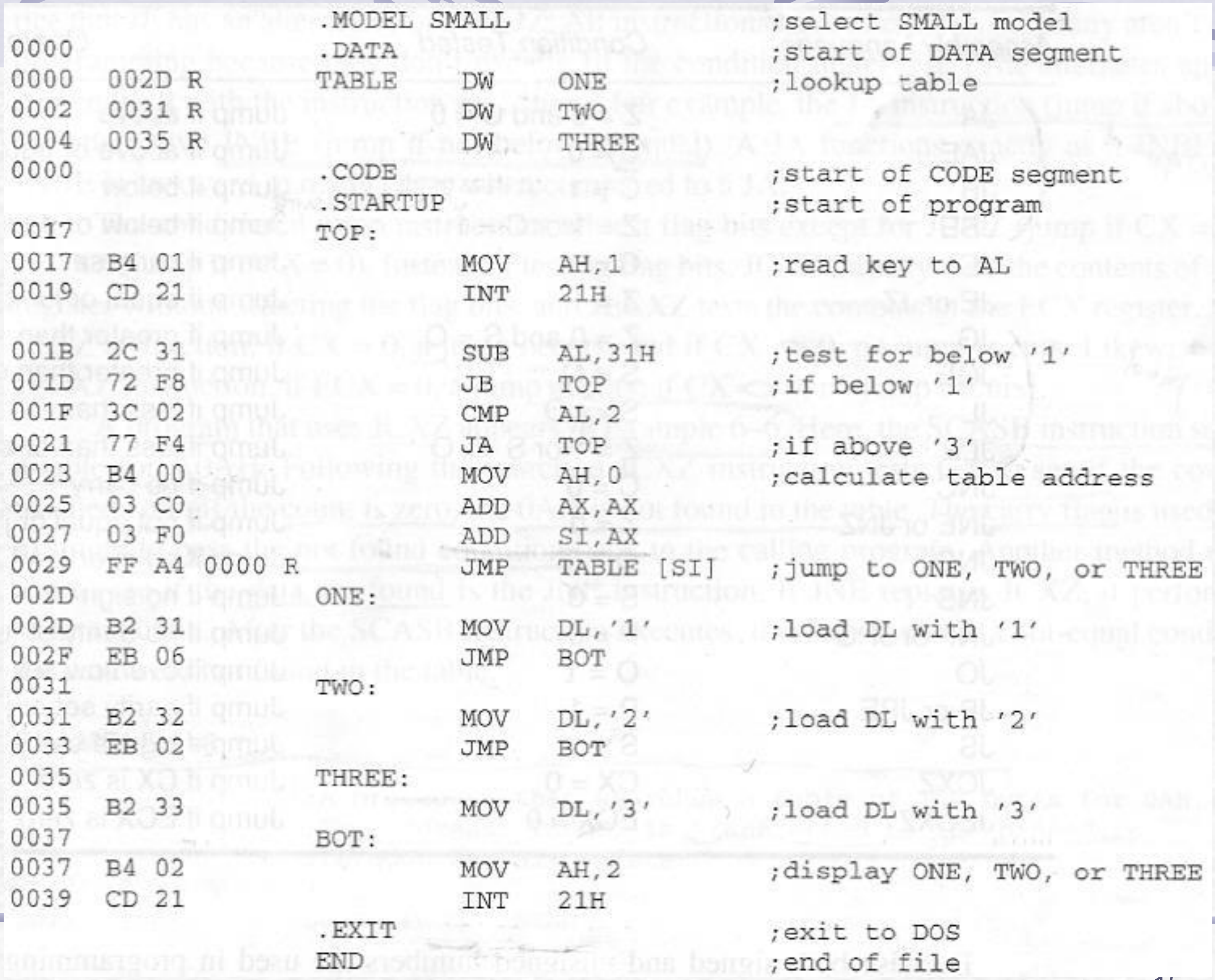
```
                                      EXTRN   UP:FAR

0000   33 DB                          XOR     BX,BX

0002   B8 0001            START:      MOV     AX,1
0005   03 C3                          ADD     AX,BX
0007   E9 0200 R                      JMP     NEXT

0200   8B D8              NEXT:       MOV     BX,AX
0202   EA 0002 ---- R                 JMP     FAR PTR START

0207   EA 0000 ---- E                 JMP     UP
```

Indirect Jump

- Jump with 16-, 32-bit reg. operand : indirect jump
  - contents of reg. : transferred directly into IP/EIP
  - JMP AX : IP ← AX,   JMP EAX : EIP ← EAX
- EX. 6-4 : how JMP AX access jump table
  - read a key, converted ASCII to binary, doubled
  - jump table : 16-bit offset address
- Indirect Jumps using Index : double-indirect jump
  - [ ] form of addressing to directly access jump table
  - near jump JMP TABLE[SI] : IP ← [SI+TABLE]
  - far jump JMP FAR PTR [SI],   JMP TABLE [SI] with TABLE data defined DD directive



```
                    ;A program that reads 1, 2, or 3 from the keyboard
                    ;if a 1, 2, or 3 is typed, a 1, 2, or 3 is displayed.
                    ;
                    .MODEL SMALL                        ;select SMALL model
0000                .DATA                               ;start of DATA segment
0000  0030 R        TABLE   DW    ONE                   ;define lookup table
0002  0034 R                DW    TWO
0004  0038 R                DW    THREE
0000                .CODE                               ;start of CODE segment
                    .STARTUP                            ;start of program
0017                TOP:
0017  B4 01                 MOV   AH,1                  ;read key into AL
0019  CD 21                 INT   21H

001B  2C 31                 SUB   AL,31H                ;convert to binary
001D  72 F8                 JB    TOP                   ;if below '1' typed
001F  3C 02                 CMP   AL,2
0021  77 F4                 JA    TOP                   ;if above '3' typed

0023  B4 00                 MOV   AH,0                  ;double to 0, 2, or 4
0025  03 C0                 ADD   AX,AX
0027  BE 0000 R             MOV   SI,OFFSET TABLE       ;address lookup table
002A  03 F0                 ADD   SI,AX                 ;form lookup address
002C  8B 04                 MOV   AX,[SI]               ;get ONE, TWO, or THREE
002E  FF E0                 JMP   AX                    ;jump address
0030                ONE:
0030  B2 31                 MOV   DL,'1'                ;load '1' for display
0032  EB 06                 JMP   BOT                   ;go display '1'
0034                TWO:
0034  B2 32                 MOV   DL,'2'                ;load '2' for display
0036  EB 02                 JMP   BOT                   ;go display '2'
0038                THREE:
0038  B2 33                 MOV   DL,'3'                ;load '3' for display
003A                BOT:
003A  B4 02                 MOV   AH,2                  ;display number
003C  CD 21                 INT   21H
                    .EXIT                               ;exit to DOS
                    END                                 ;end of file
```



```
                         .MODEL SMALL                       ;select SMALL model
0000                     .DATA                              ;start of DATA segment
0000   002D R            TABLE    DW    ONE                 ;lookup table
0002   0031 R                     DW    TWO
0004   0035 R                     DW    THREE
0000                     .CODE                              ;start of CODE segment
                         .STARTUP                           ;start of program
0017                     TOP:
0017   B4 01                      MOV   AH,1                ;read key to AL
0019   CD 21                      INT   21H

001B   2C 31                      SUB   AL,31H              ;test for below '1'
001D   72 F8                      JB    TOP                 ;if below '1'
001F   3C 02                      CMP   AL,2
0021   77 F4                      JA    TOP                 ;if above '3'
0023   B4 00                      MOV   AH,0                ;calculate table address
0025   03 C0                      ADD   AX,AX
0027   03 F0                      ADD   SI,AX
0029   FF A4 0000 R               JMP   TABLE [SI]          ;jump to ONE, TWO, or THREE
002D                     ONE:
002D   B2 31                      MOV   DL,'1'              ;load DL with '1'
002F   EB 06                      JMP   BOT
0031                     TWO:
0031   B2 32                      MOV   DL,'2'              ;load DL with '2'
0033   EB 02                      JMP   BOT
0035                     THREE:
0035   B2 33                      MOV   DL,'3'              ;load DL with '3'
0037                     BOT:
0037   B4 02                      MOV   AH,2                ;display ONE, TWO, or THREE
0039   CD 21                      INT   21H
                         .EXIT                              ;exit to DOS
                         END                                ;end of file
```

Conditional Jumps

- Conditional jump : short jump
  - ~ 80286(short jump) : +127 ~ -128
  - 80386 ~(short, near jump) : 1, 4 bytes
- Test one flag bit or some more : S, Z, C, P, O
  - if condition under test is true : branch to the label
  - if condition is false : next sequential instruction
- Relative magnitude comparisons :
  - require more complicated conditional jump instructions that test more than one flag bit
- Table 6-1 : conditional jump instructions

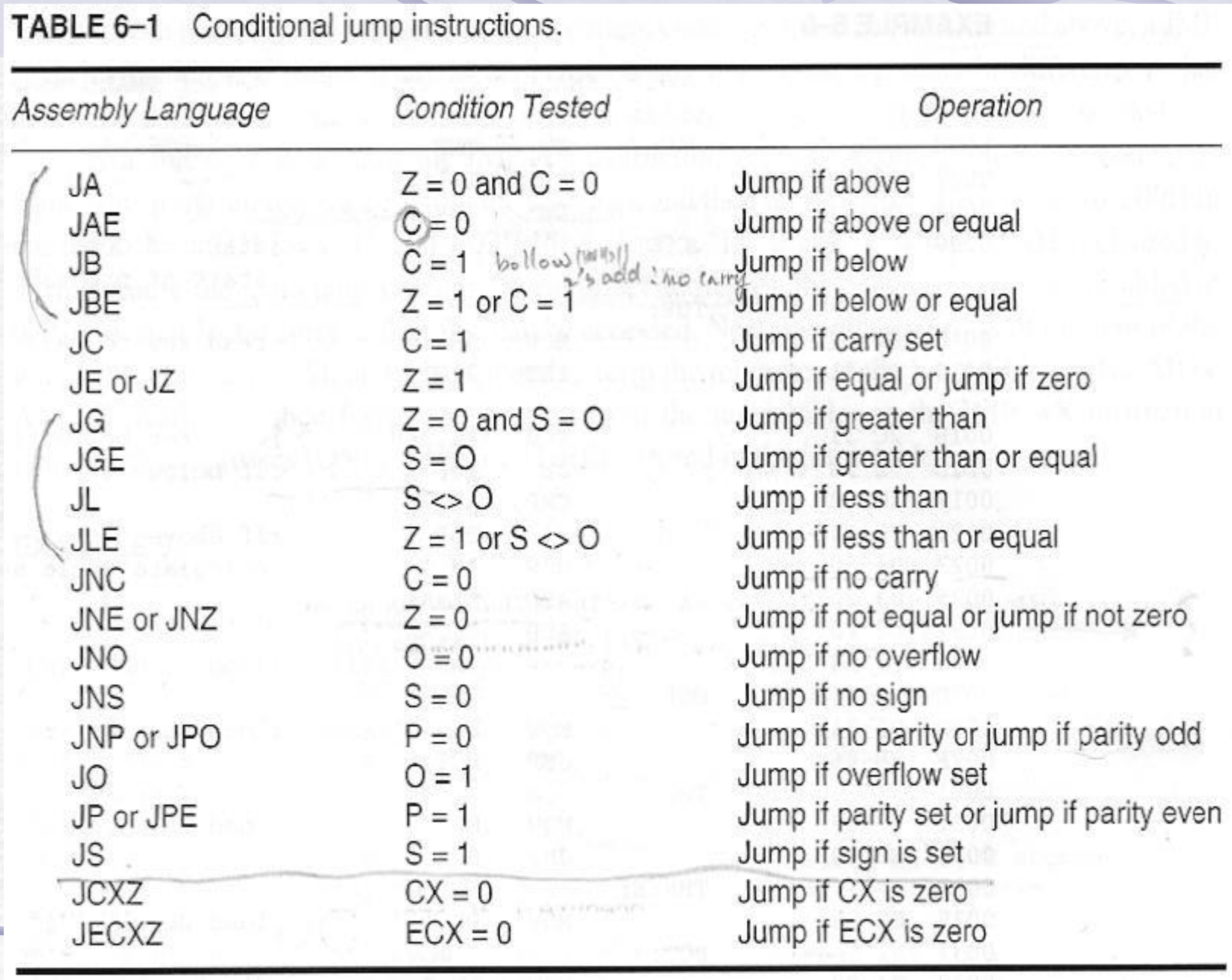
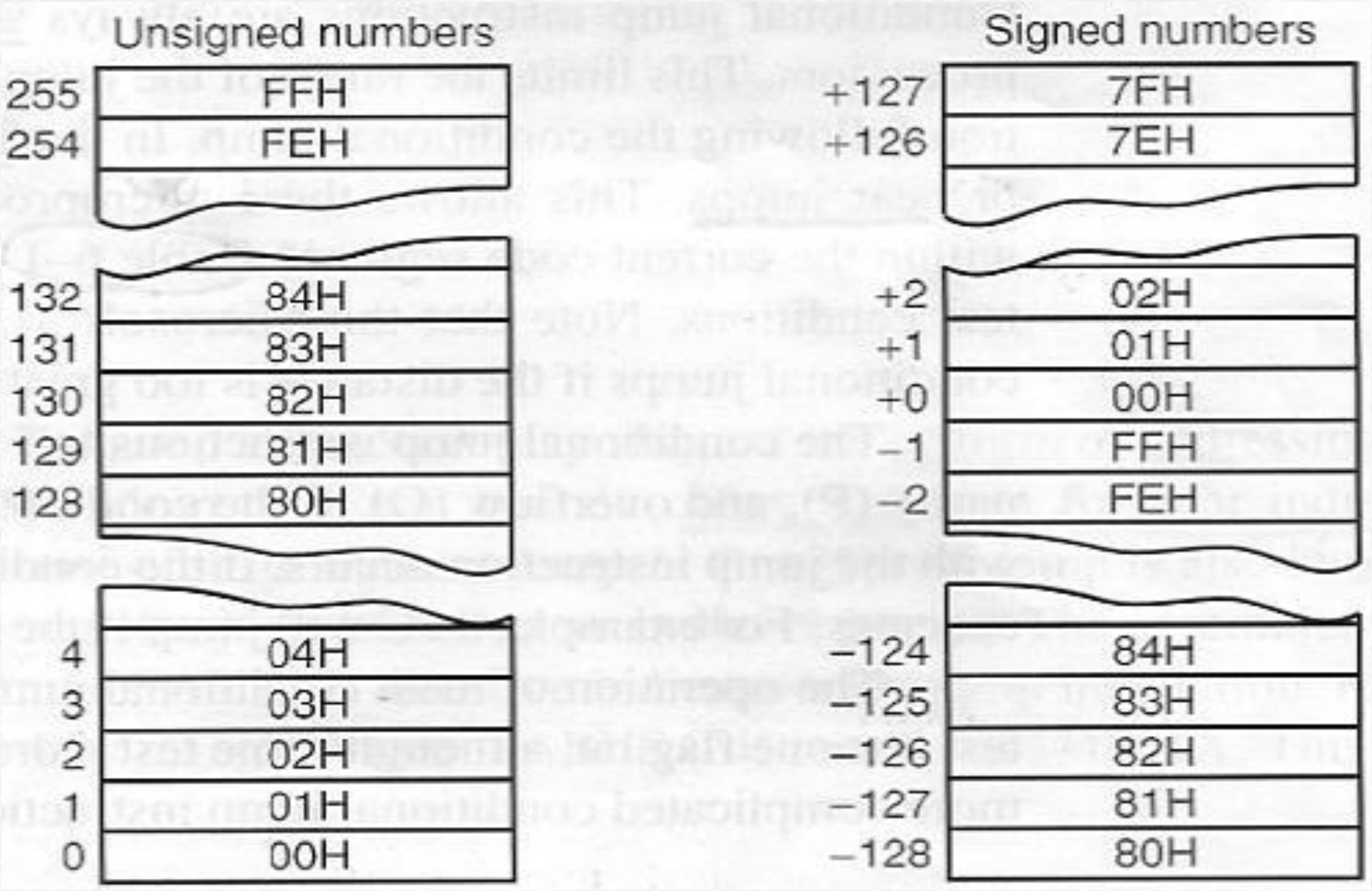


TABLE 6–1 Conditional jump instructions.

Assembly Language	*Condition Tested*	*Operation*
JA	Z = 0 and C = 0	Jump if above
JAE	C = 0	Jump if above or equal
JB	C = 1 bollow [illegible] 2's add : no carry	Jump if below
JBE	Z = 1 or C = 1	Jump if below or equal
JC	C = 1	Jump if carry set
JE or JZ	Z = 1	Jump if equal or jump if zero
JG	Z = 0 and S = O	Jump if greater than
JGE	S = O	Jump if greater than or equal
JL	S <> O	Jump if less than
JLE	Z = 1 or S <> O	Jump if less than or equal
JNC	C = 0	Jump if no carry
JNE or JNZ	Z = 0	Jump if not equal or jump if not zero
JNO	O = 0	Jump if no overflow
JNS	S = 0	Jump if no sign
JNP or JPO	P = 0	Jump if no parity or jump if parity odd
JO	O = 1	Jump if overflow set
JP or JPE	P = 1	Jump if parity set or jump if parity even
JS	S = 1	Jump if sign is set
JCXZ	CX = 0	Jump if CX is zero
JECXZ	ECX = 0	Jump if ECX is zero

- Fig. 6-5 : order of signed, unsigned 8-bit no.s



Unsigned numbers

255	FFH
254	FEH
132	84H
131	83H
130	82H
129	81H
128	80H
4	04H
3	03H
2	02H
1	01H
0	00H

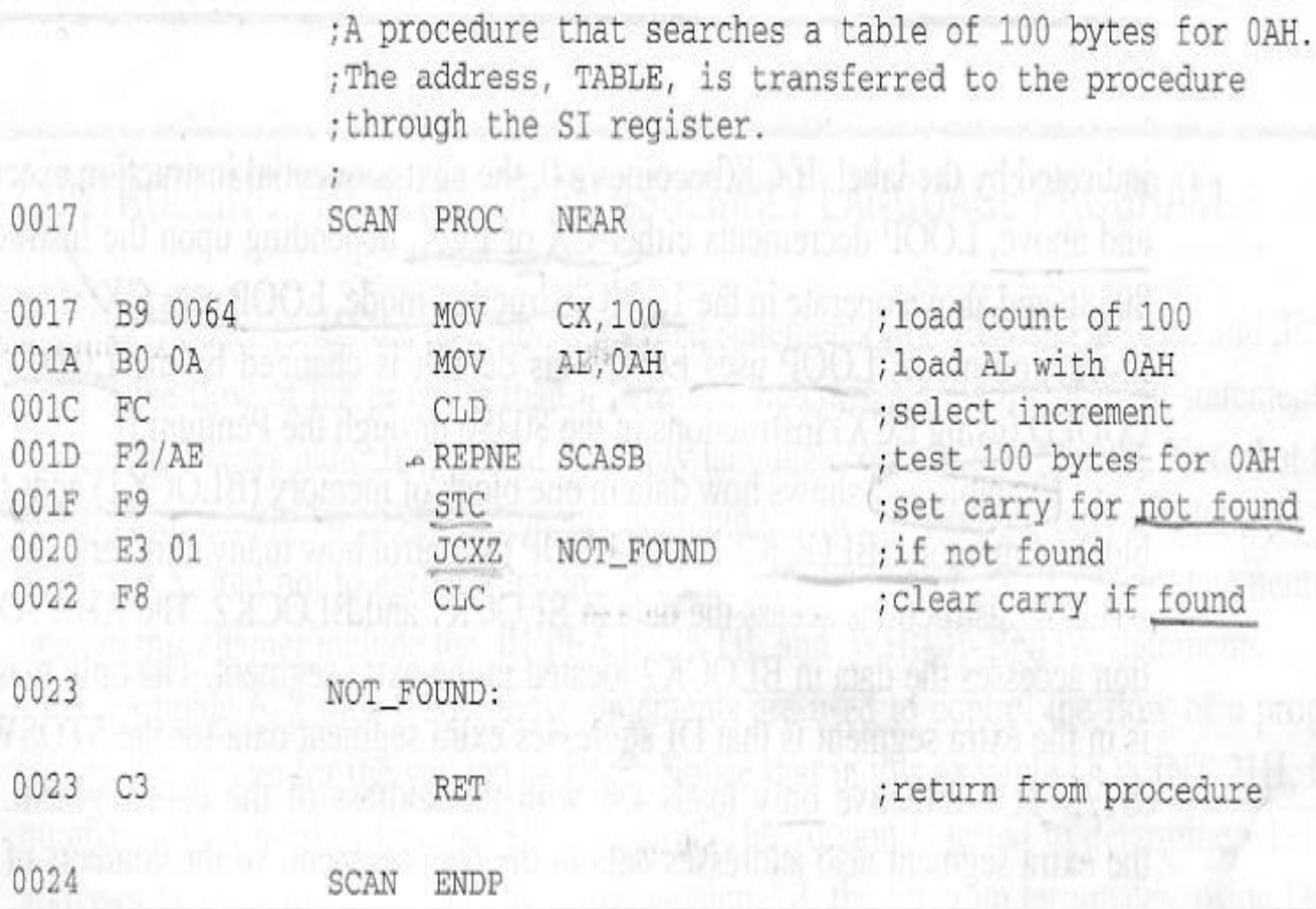
Signed numbers

+127	7FH
+126	7EH
+2	02H
+1	01H
+0	00H
−1	FFH
−2	FEH
−124	84H
−125	83H
−126	82H
−127	81H
−128	80H

Conditional Jumps

- Unsigned : FFH is above 00H, above, below, equal
- Signed : FFH less than 00H, greater, less, zero
- Alternate form :
  - JE = JZ
  - JA(if above) = JNBE(if not below or equal)
- JCXZ(jump if CX = 0), JECXZ(jump if ECX=0)
  - if CX/ECX = 0 : jump occur
  - if CX/ECX <> 0 : no jump occur
- EX. 6-6 : search table for 0AH using SANSB, JCXZ

Example: Conditional Jump



```
                 ;A procedure that searches a table of 100 bytes for 0AH.
                 ;The address, TABLE, is transferred to the procedure
                 ;through the SI register.
                 ;
0017             SCAN  PROC  NEAR

0017  B9 0064          MOV   CX,100              ;load count of 100
001A  B0 0A            MOV   AL,0AH              ;load AL with 0AH
001C  FC               CLD                       ;select increment
001D  F2/AE            REPNE SCASB               ;test 100 bytes for 0AH
001F  F9               STC                       ;set carry for not found
0020  E3 01            JCXZ  NOT_FOUND           ;if not found
0022  F8               CLC                       ;clear carry if found

0023             NOT_FOUND:

0023  C3               RET                       ;return from procedure

0024             SCAN  ENDP
```

Conditional Set Instructions

- Conditional set instructions :
  - 80386~
  - set a byte to either a 01H or clear a byte to 00H
  - useful where a condition must be tested at a point much later in the program
- SETNC MEM :
  - places a 01H into memory location MEM if carry is cleared   and
  - a 00H into MEM if carry is set
- Table 6-2 :

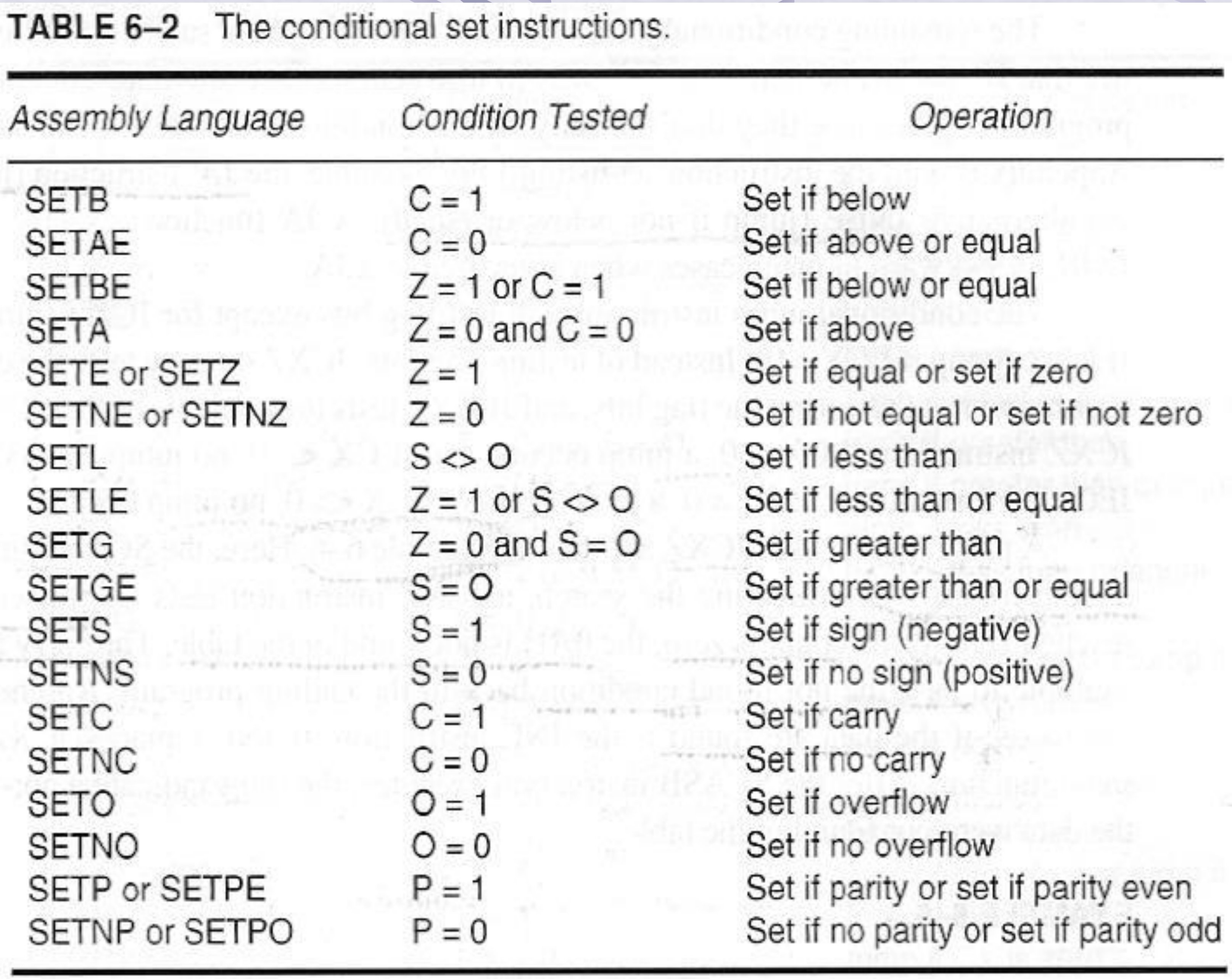
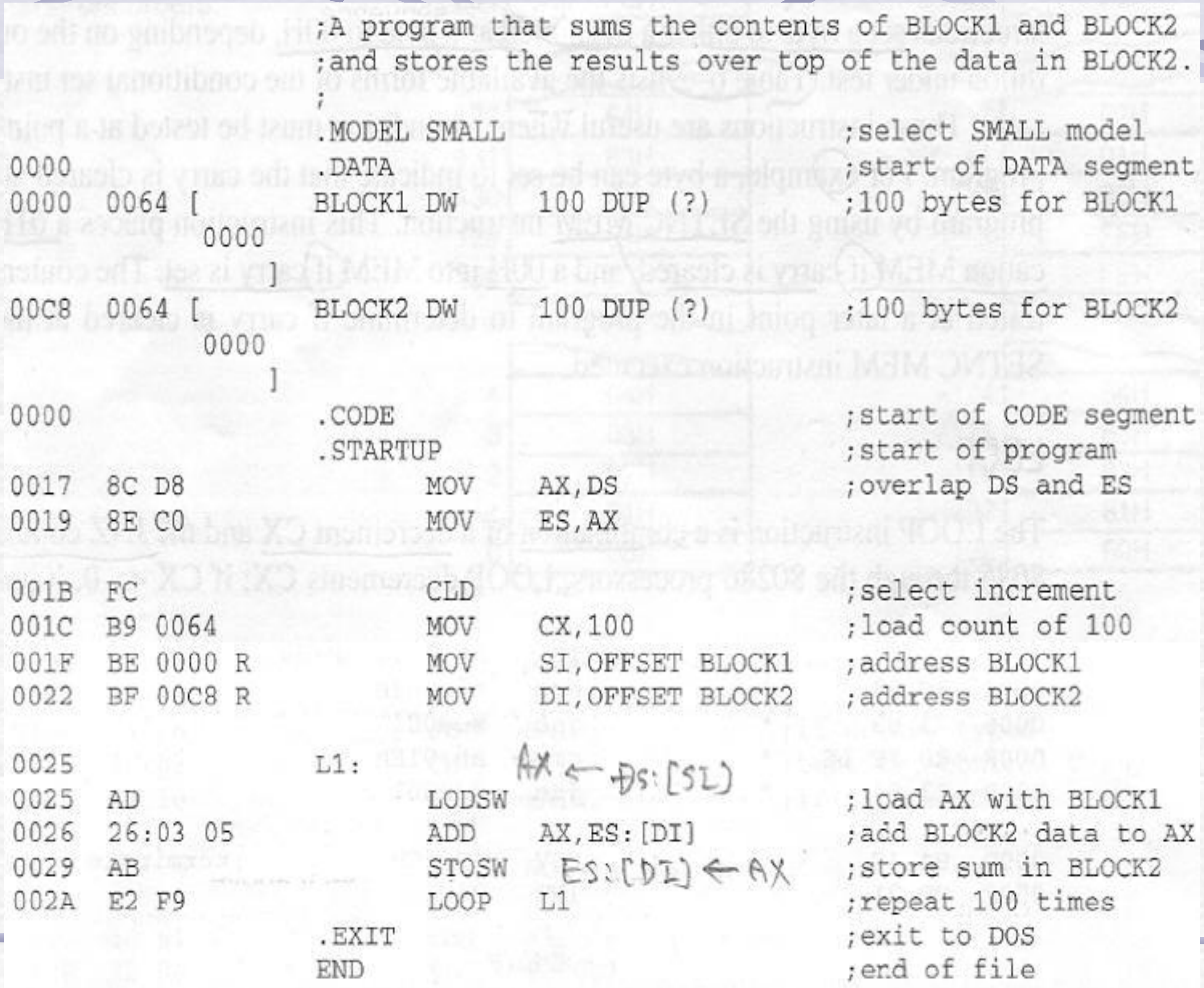


TABLE 6–2 The conditional set instructions.

Assembly Language	*Condition Tested*	*Operation*
SETB	C = 1	Set if below
SETAE	C = 0	Set if above or equal
SETBE	Z = 1 or C = 1	Set if below or equal
SETA	Z = 0 and C = 0	Set if above
SETE or SETZ	Z = 1	Set if equal or set if zero
SETNE or SETNZ	Z = 0	Set if not equal or set if not zero
SETL	S <> O	Set if less than
SETLE	Z = 1 or S <> O	Set if less than or equal
SETG	Z = 0 and S = O	Set if greater than
SETGE	S = O	Set if greater than or equal
SETS	S = 1	Set if sign (negative)
SETNS	S = 0	Set if no sign (positive)
SETC	C = 1	Set if carry
SETNC	C = 0	Set if no carry
SETO	O = 1	Set if overflow
SETNO	O = 0	Set if no overflow
SETP or SETPE	P = 1	Set if parity or set if parity even
SETNP or SETPO	P = 0	Set if no parity or set if parity odd

LOOP, Conditional LOOP

- LOOP : combination of decrement CX and JNZ
  - ~ 80286 : DEC CX ; if CX <> 0, jump to label if CX = 0, execute next sequential instruction
  - 80386 ~ : CX/ECX depending on instruction mode
- LOOPE(loop while equal, LOOPZ) :
  - jump if CX <> 0 while equal condition exist
  - exit the loop if CX = 0 or condition is not equal
- LOOPNE(loop while not equal, LOOPNZ) :
  - jump if CX <> 0 while not-equal condition exist
  - exit the loop if CX = 0 or condition is equal
- LOOPEW/LOOPED,LOOPNEW/LOOPNED:override mode



```
                        ;A program that sums the contents of BLOCK1 and BLOCK2
                        ;and stores the results over top of the data in BLOCK2.
                        ;
                        .MODEL SMALL                           ;select SMALL model
0000                    .DATA                                  ;start of DATA segment
0000  0064 [            BLOCK1 DW     100 DUP (?)              ;100 bytes for BLOCK1
           0000
               ]
00C8  0064 [            BLOCK2 DW     100 DUP (?)              ;100 bytes for BLOCK2
           0000
               ]
0000                    .CODE                                  ;start of CODE segment
                        .STARTUP                               ;start of program
0017  8C D8                    MOV    AX,DS                    ;overlap DS and ES
0019  8E C0                    MOV    ES,AX

001B  FC                       CLD                             ;select increment
001C  B9 0064                  MOV    CX,100                   ;load count of 100
001F  BE 0000 R                MOV    SI,OFFSET BLOCK1         ;address BLOCK1
0022  BF 00C8 R                MOV    DI,OFFSET BLOCK2         ;address BLOCK2

0025                    L1:           AX ← DS:[SI)
0025  AD                       LODSW                           ;load AX with BLOCK1
0026  26:03 05                 ADD    AX,ES:[DI]               ;add BLOCK2 data to AX
0029  AB                       STOSW    ES:[DI] ← AX           ;store sum in BLOCK2
002A  E2 F9                    LOOP   L1                       ;repeat 100 times
                        .EXIT                                  ;exit to DOS
                        END                                    ;end of file
```

Controlling the Flow of an Assembly Language Program

- Relational statements
  - .IF, .ELSE, .ELSEIF, ENDIF, .REPEAT-.UNTIL, .WHILE-.ENDW :
  - easier to control the flow than conditional jump
- EX. 6-8 : testing system for version of DOS
- DOS INT 21H, function no. 30H : read DOS ver.
- (a) : source program, (b) fully expended assembled
- * : assembler-generated and -inserted statements
- && : logical AND
- Table 6-3 : relational operator

Table of Operators and their Functions

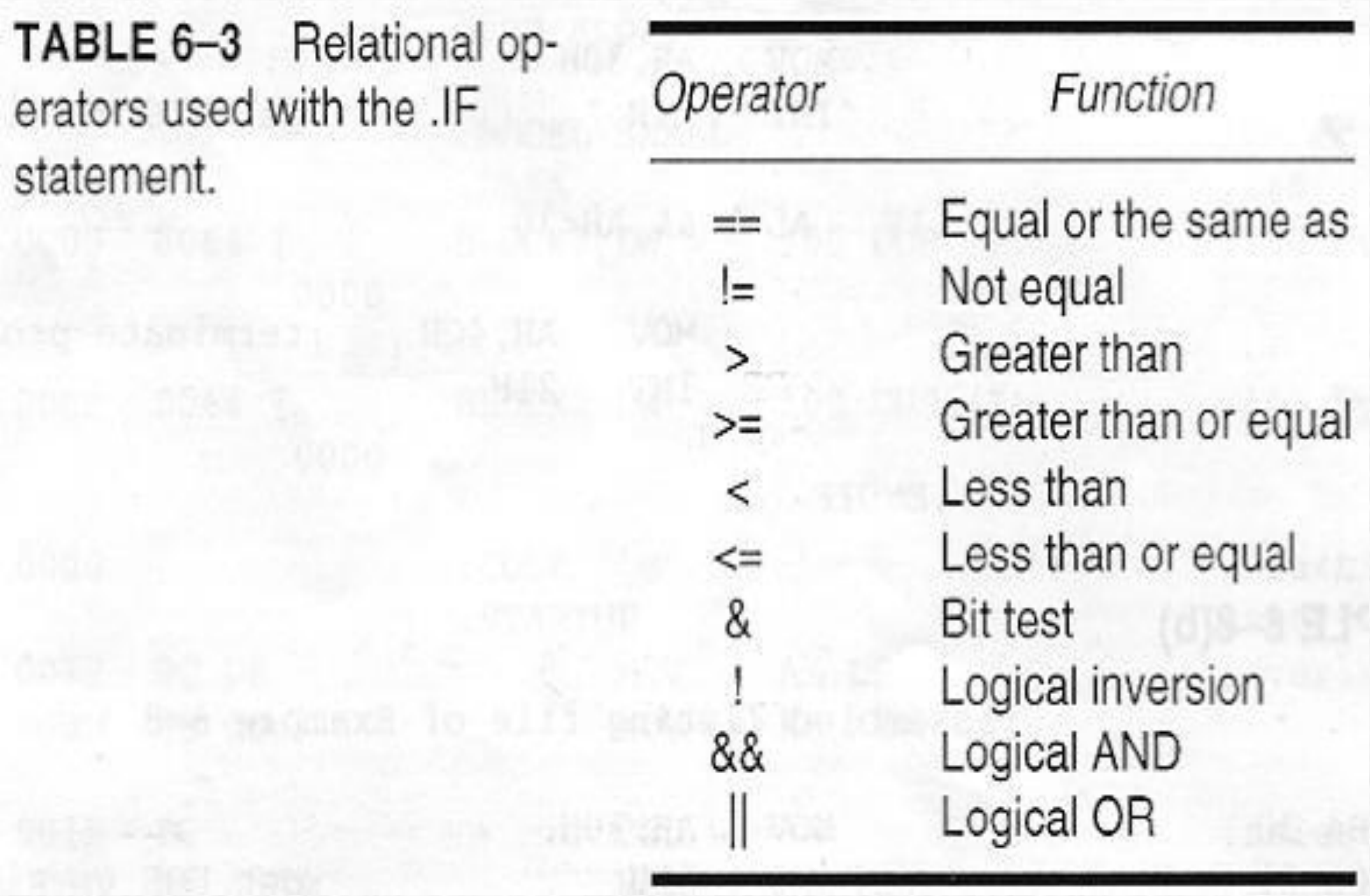
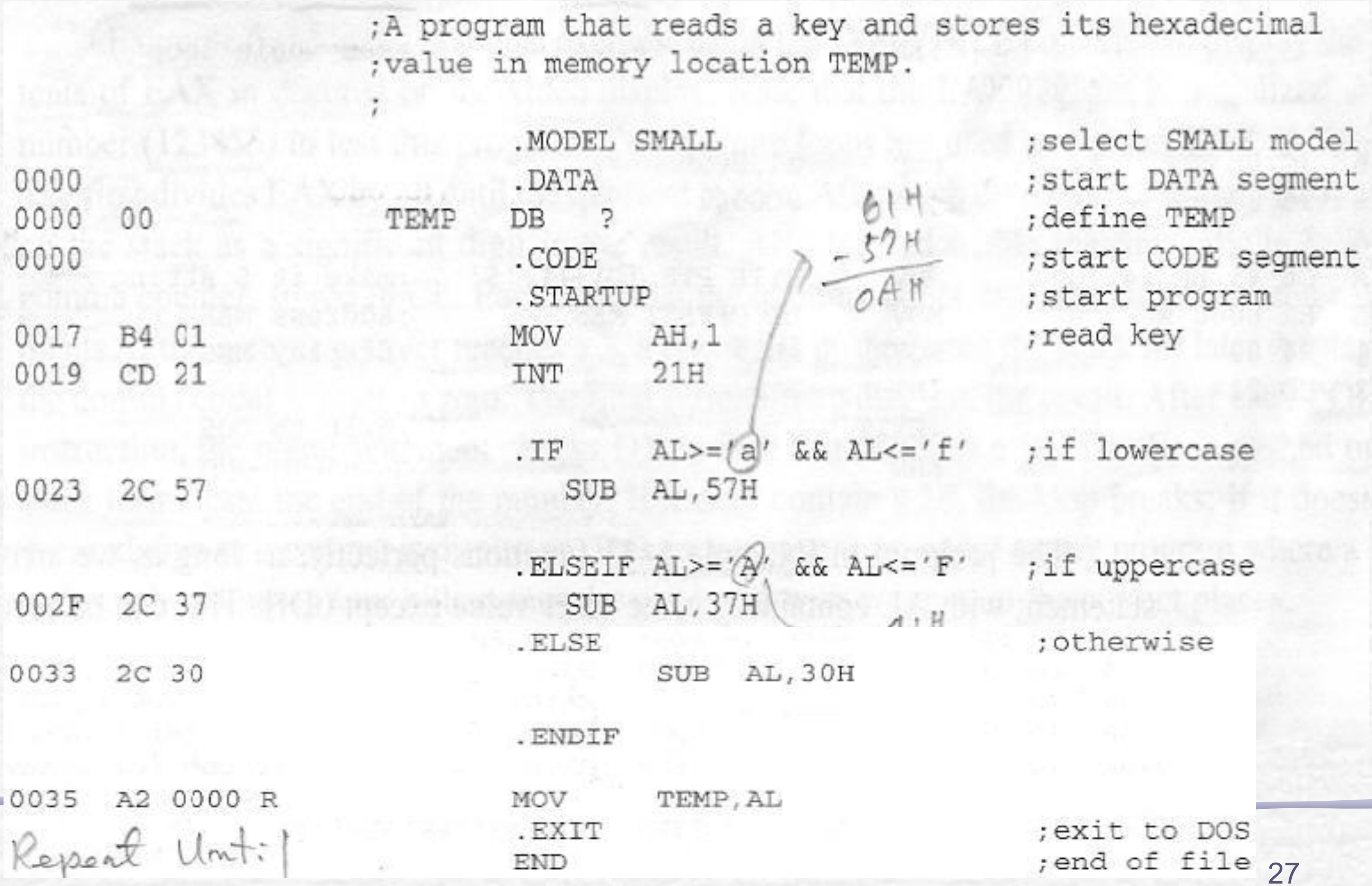


TABLE 6–3 Relational operators used with the .IF statement.

<table>
<tr><th><i>Operator</i></th><th><i>Function</i></th></tr>
<tr><td>==</td><td>Equal or the same as</td></tr>
<tr><td>!=</td><td>Not equal</td></tr>
<tr><td>></td><td>Greater than</td></tr>
<tr><td>>=</td><td>Greater than or equal</td></tr>
<tr><td><</td><td>Less than</td></tr>
<tr><td><=</td><td>Less than or equal</td></tr>
<tr><td>&</td><td>Bit test</td></tr>
<tr><td>!</td><td>Logical inversion</td></tr>
<tr><td>&&</td><td>Logical AND</td></tr>
<tr><td>||</td><td>Logical OR</td></tr>
</table>

Example

- EX. 6-10 : read a key, convert to hexadecimal
- `a`(61H), `A`(41H) : 61H(41H)-57H(37H)=0AH



```
                    ;A program that reads a key and stores its hexadecimal
                    ;value in memory location TEMP.
                    ;
                            .MODEL SMALL                ;select SMALL model
0000                        .DATA                       ;start DATA segment
0000  00            TEMP    DB    ?                     ;define TEMP
0000                        .CODE                       ;start CODE segment
                            .STARTUP                    ;start program
0017  B4 01                 MOV     AH,1                ;read key
0019  CD 21                 INT     21H

                            .IF     AL>='a' && AL<='f'  ;if lowercase
0023  2C 57                    SUB  AL,57H

                            .ELSEIF AL>='A' && AL<='F'  ;if uppercase
002F  2C 37                    SUB  AL,37H
                            .ELSE                       ;otherwise
0033  2C 30                         SUB  AL,30H

                            .ENDIF

0035  A2 0000 R             MOV     TEMP,AL
                            .EXIT                       ;exit to DOS
                            END                         ;end of file
```

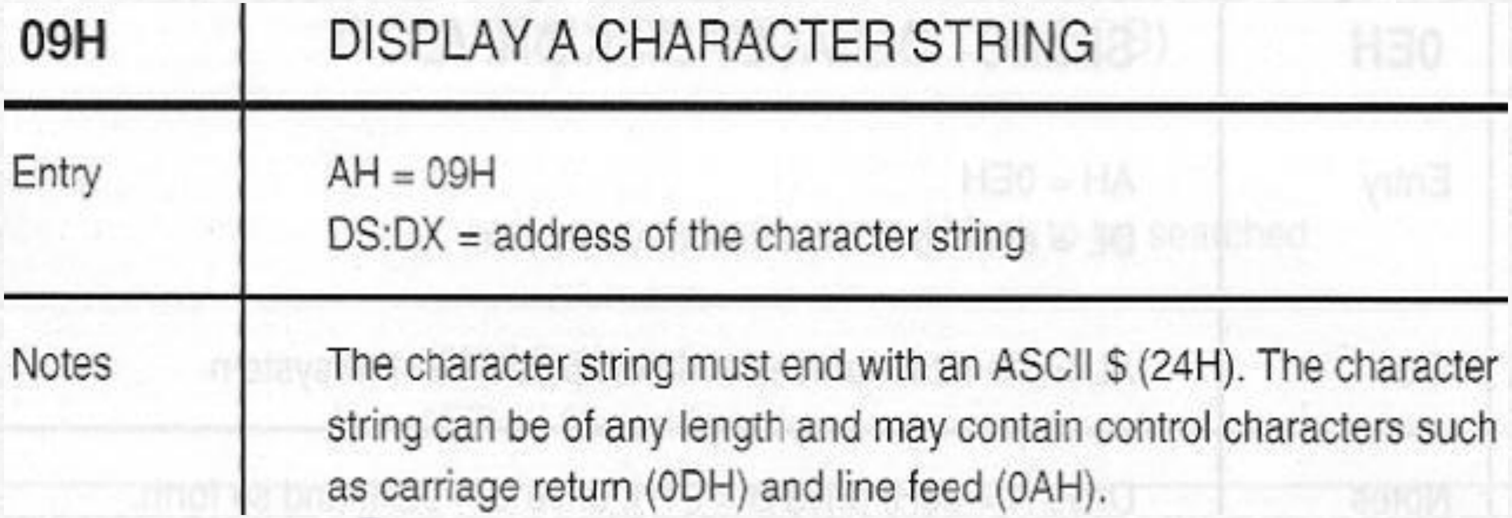
61H
-57H
0AH

41H

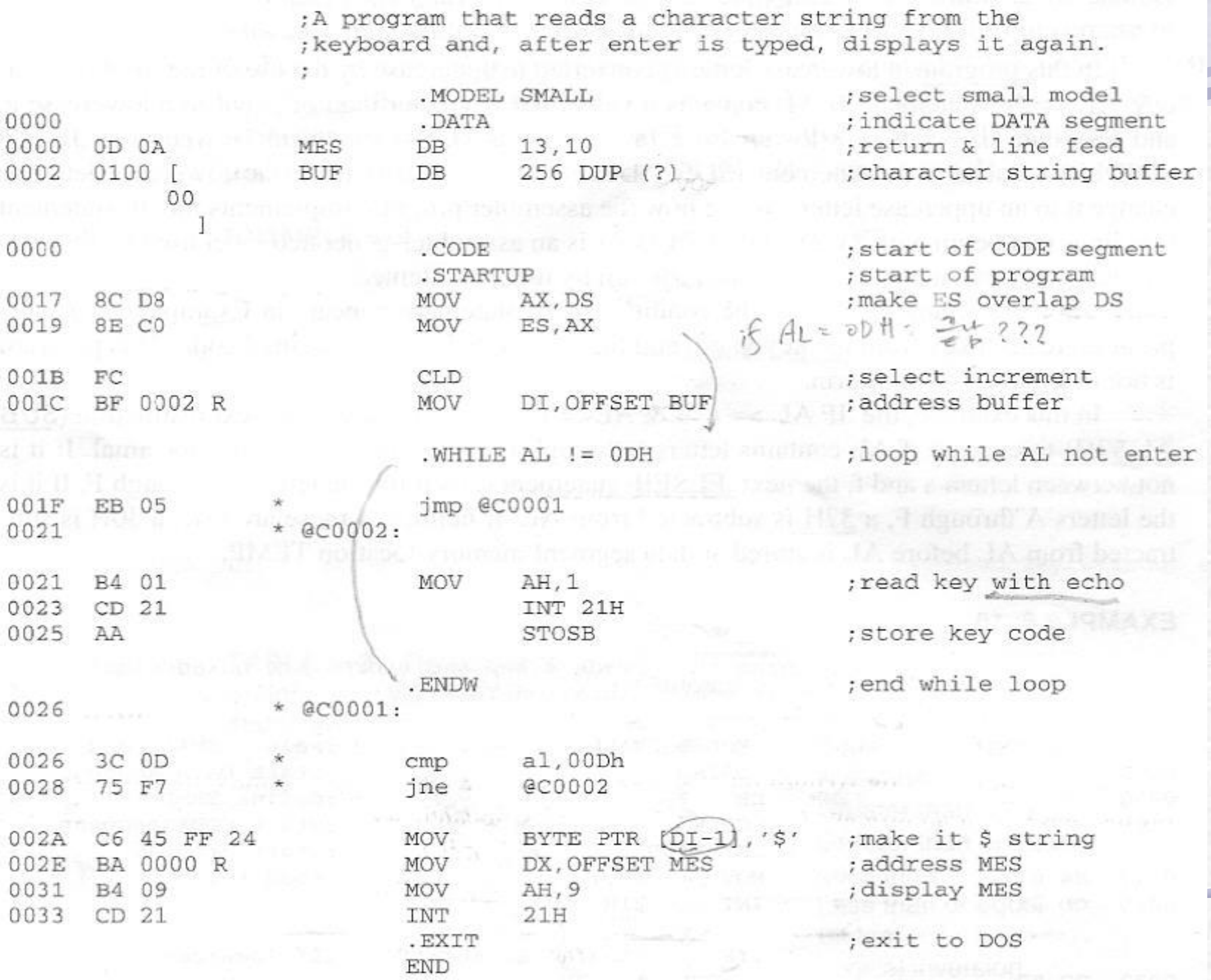
Repeat Until:|

DO-WHILE Loops

- .WHILE statement : used with a condition to begin the loop
- EX. 6-11 : read a key, store into array called BUF until enter key(0DH) is typed
- DOS 21H, fn no. 09H



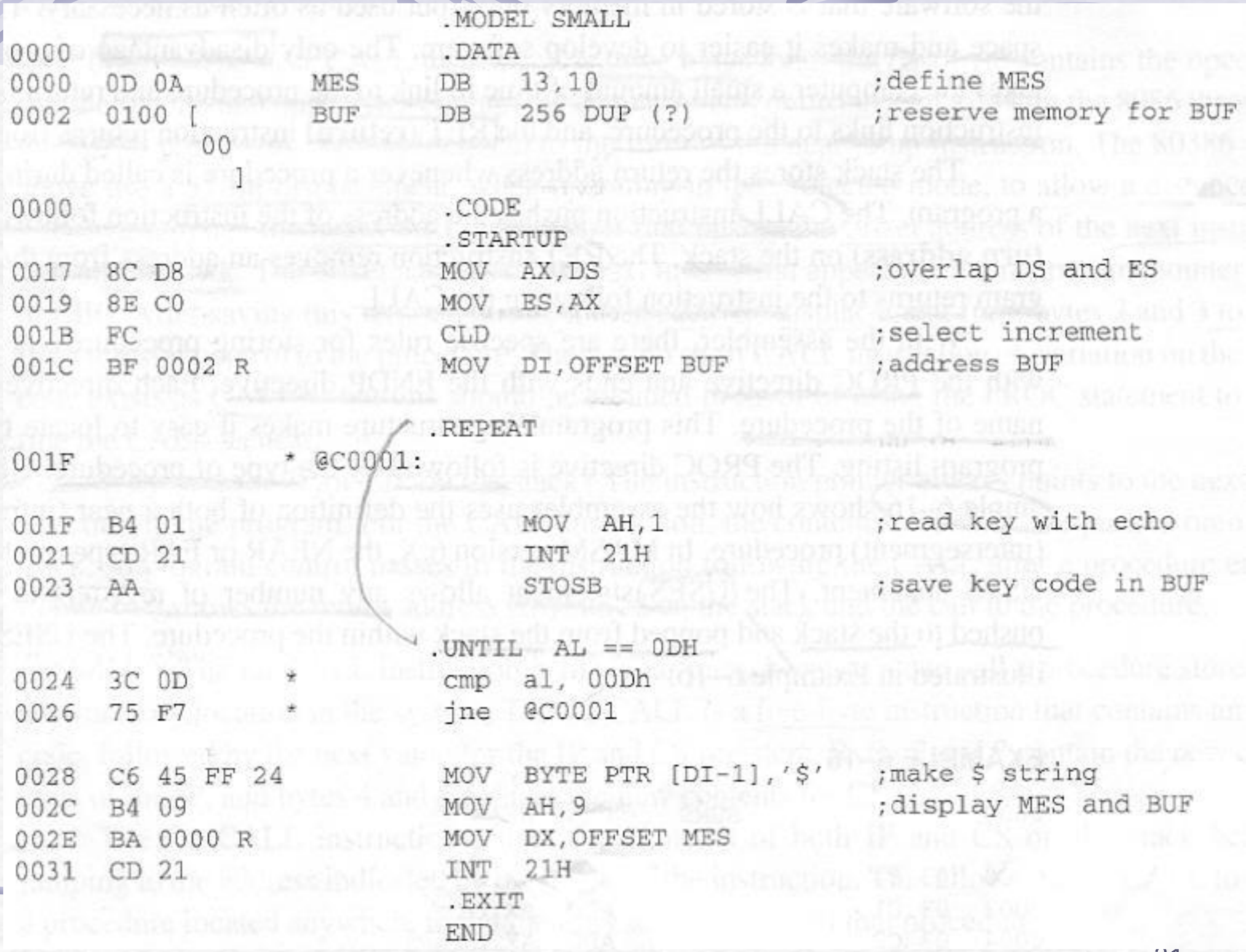
09H	DISPLAY A CHARACTER STRING
Entry	AH = 09H DS:DX = address of the character string
Notes	The character string must end with an ASCII $ (24H). The character string can be of any length and may contain control characters such as carriage return (0DH) and line feed (0AH).



```
                                 ;A program that reads a character string from the
                                 ;keyboard and, after enter is typed, displays it again.
                                 ;
                                           .MODEL SMALL                   ;select small model
0000                                       .DATA                          ;indicate DATA segment
0000  0D 0A                      MES       DB      13,10                  ;return & line feed
0002  0100 [                     BUF       DB      256 DUP(?)             ;character string buffer
            00
              ]
0000                                       .CODE                          ;start of CODE segment
                                           .STARTUP                       ;start of program
0017  8C D8                                MOV     AX,DS                  ;make ES overlap DS
0019  8E C0                                MOV     ES,AX                 if AL = 0DH : 같냐 ???

001B  FC                                   CLD                            ;select increment
001C  BF 0002 R                            MOV     DI,OFFSET BUF          ;address buffer

                                           .WHILE AL != 0DH               ;loop while AL not enter

001F  EB 05                      *         jmp @C0001
0021                             * @C0002:

0021  B4 01                                MOV     AH,1                   ;read key with echo
0023  CD 21                                        INT 21H
0025  AA                                           STOSB                  ;store key code

                                          .ENDW                           ;end while loop
0026                             * @C0001:

0026  3C 0D                      *        cmp      al,00Dh
0028  75 F7                      *        jne      @C0002

002A  C6 45 FF 24                         MOV      BYTE PTR [DI-1],'$'    ;make it $ string
002E  BA 0000 R                           MOV      DX,OFFSET MES          ;address MES
0031  B4 09                               MOV      AH,9                   ;display MES
0033  CD 21                               INT      21H
                                          .EXIT                           ;exit to DOS
                                          END
```

REPEAT-UNTIL Loops

- .REPEAT : defined start of loop
- .UNTIL : defined end of loop, contained condition
- EX. 6-14 : EX. 6-11,12



```
                        .MODEL SMALL
0000                    .DATA
0000  0D 0A       MES   DB   13,10              ;define MES
0002  0100 [      BUF   DB   256 DUP (?)        ;reserve memory for BUF
          00
            ]
0000                    .CODE
                        .STARTUP
0017  8C D8             MOV  AX,DS              ;overlap DS and ES
0019  8E C0             MOV  ES,AX
001B  FC                CLD                     ;select increment
001C  BF 0002 R         MOV  DI,OFFSET BUF      ;address BUF

                       .REPEAT
001F            * @C0001:

001F  B4 01                  MOV  AH,1          ;read key with echo
0021  CD 21                  INT  21H
0023  AA                     STOSB              ;save key code in BUF

                       .UNTIL  AL == 0DH
0024  3C 0D       *     cmp  al, 00Dh
0026  75 F7       *     jne  @C0001

0028  C6 45 FF 24       MOV  BYTE PTR [DI-1],'$'   ;make $ string
002C  B4 09             MOV  AH,9               ;display MES and BUF
002E  BA 0000 R         MOV  DX,OFFSET MES
0031  CD 21             INT  21H
                        .EXIT
                        END
```

Questions

- Q1: Contrast the operation of JMP DI with JMP [DI].
- Q2: What is the purpose of .BREAK directive?
- Q3: Explain how the LOOPE instruction operates.
- Q4: What happens if the .WHILE instruction is placed in a program?
- Q5:When does JCXZ instruction jump?
- Q6: Write a program that reads the keyboard and converts all lowercase data to uppercase before displaying it.
- Q7: Develop a short sequence of instruction that uses
- DO-WHILE Loop

  REPEAT-UNTIL Loop