Chapter-6

Flow Control Instructions

Overview

Flow control instructions are used to make decisions and repeat sections of code. The jump and loop instructions transfer control to another part of the program. This transfer can be unconditional or conditional (dependent on a particular combination of status flags settings)

6.1 An Example of a Jump

```
TITLE    PGM6_1:    IBM CHARACTER DISPLAY
.MODEL  SMALL
.STACK  100H
.CODE
MAIN  PROC
MOV  AH, 2
MOV  CX, 256
MOV  DL, 0
PRINT_LOOP:
INT  21H
INC  DL
DEC  CX
JNZ  PRINT_LOOP
MOV  AH, 4CH
INT 21H
MAIN  ENDP
END  MAIN
```

6.2 Conditional Jumps

JNZ is an example of a **conditional jump instruction**. The syntax is

Jxxx destination_label

Range of a Conditional Jump

The structure of the machine code of a conditional jump requires that destination_label must precede the jump instruction by no more than 126 bytes, or follow it by no more than 127 bytes.

How the CPU Implements a Conditional Jump

To implement a conditional jump, the CPU looks at the FLAGS register. If the conditions for the jump are true, the CPU adjusts the IP to point to the destination label.

Table 6.1 shows the conditional jumps. There are three categories of conditional jumps:

(1) **Signed Jumps:** used when a signed interpretation is being given to results.
(2) **Unsigned Jumps:** used when for an unsigned interpretation.
(3) **Single-Flag Jumps:** operate on settings of individual flags.

Note: the jump instructions themselves do not affect the flags.

The CMP Instruction

The jump condition is often provided by the **CMP** (compare) instruction. It has the form

CMP destination, source

This instruction compares destination and source by computing destination contents minus source contents. The result is not stored, but the flags are affected. The operands of CMP may not both be memory locations. Destination may not be a constant.

Note: CMP is just like SUB, except that destination is not changed.

For example, suppose a program contains these lines:

```
CMP  AX, BX
JG BELOW
```

where, AX=7FFFH, and BX=0001H. Table 6.1 shows that JG is satisfied, because ZF = SF = OF = 0, so control transfers to label BELOW.

Table 6.1: Conditional Jumps

Signed Jumps

Symbol	Description	Condition for Jumps
JG/JNLE	jump if greater than jump if not less than or equal to	ZF = 0 and SF = OF
JGE/JNL	jump if greater than or equal to jump if not less than	SF = OF
JL/JNGE	jump if less than jump if not greater than or equal to	SF <> OF
JLE/JNG	jump if less than or equal Jump if not greater than	ZF = 1 or SF <> OF

Unsigned Jumps

Symbol	Description	Condition for Jumps
JA/JNBE	jump if above	CF = 0 and ZF = 0
	jump if not below	
	or equal to	
JAE/JNB	jump if above	CF = 0
	or equal to	
	jump if not below	
JB/JNAE	jump if below	CF = 1
	jump if not above or equal	
JBE/JNA	jump if below or equal	CF = 1 or ZF = 1
	jump if not above	

Single-Flag Jumps

Symbol	Description	Condition for Jumps
JE/JZ	jump if equal	ZF = 1
	jump if equal to zero	
JNE/JNZ	jump if not equal	ZF = 0
	jump if not zero	
JC	jump if carry	CF = 1
JNC	jump if not carry	CF = 0
JO	jump if overflow	OF = 1
JNO	jump if not overflow	OF = 0
JS	jump if sign negative	SF = 1
JNS	jump if nonnegative sign	SF = 0
JP/JPE	jump if parity even	PF = 1
JNP/JPO	jump if parity odd	PF = 0

Interpreting the Conditional Jumps

In the following example:

```
CMP  AX, BX
JG     BELOW
```

if AX is greater than BX (in a signed sense), then JG transfers to BELOW.

Another example is

```
DEC   AX
JL   THERE
```

Here, if the contents of AX, in a signed sense, is less than 0, control transfers to THERE.

Signed Versus Unsigned Jumps

For example, suppose we are giving a signed interpretation. If AX = 7FFFH, BX = 8000H, and we execute

```
CMP  AX, BX
JA     BELOW
```

then even though 7FFFH > 8000H in a signed sense, the program does not jump to BELOW. The reason is that 7FFFH < 8000H in an unsigned sense, and we are using the unsigned jump JA.

Example 6.1: Suppose AX and BX contain signed numbers. Write some code to put the biggest one in CX.

Solution:

```
MOV  CX, AX
CMP  BX, CX
JLE  NEXT
MOV  CX, BX
NEXT:
```

6.3. The JMP Instruction

The **JMP** (jump) instruction causes an unconditional transfer of control (**unconditional jump**). The syntax is

JMP destination

where the destination is usually a label in the same segment as the JMP itself.

JMP can be used to get around the range restriction of a conditional jump. For example, suppose, we want to implement the following loop:

```
TOP:
; body of the loop
DEC  CX
JNZ  TOP
MOV  AX, BX
```

and the loop body contains so many instructions that label TOP is out of range for JNZ (more than 126 bytes before JNZ TOP). We can do this:

```
        TOP:
        ; body of the loop
        DEC  CX
        JNZ  BOTTOM
        JMP  EXIT
BOTTOM:
        JMP  TOP
EXIT:
        MOV  AX, BX
```

6.4 High-Level Language Structures

6.4.1 Branching Structures

IF-THEN

The pseudocode of IF-THEN structure is as follows:

IF condition is true
THEN
Execute true-branch statements
END_IF

Example 6.2: Replace the number in AX by its absolute value.

Solution:

```
        CMP  AX, 0
        JNL  END_IF
        NEG  AX
END_IF:
```

IF-THEN-ELSE

The pseudocode of IF-THEN-ELSE structure is as follows:

```
IF condition is true
THEN
Execute true-branch statements
ELSE
Execute false-branch statements
END_IF
```

Example 6.3: Suppose AL and BL contain extended ASCII characters. Display the that comes first in the character sequence.

Solution:

```
                MOV  AH, 2

                CMP  AL, BL
                JNBE  ELSE_

                MOV  DL, AL
                JMP  DISPLAY
ELSE_:
                MOV  DL, BL
DISPLAY:
                INT  21H
END_IF:
```

CASE

The pseudocode of IF-THEN-ELSE structure is as follows:

```
CASE  expression
        values_1:  statements_1
        values_2:  statements_2
                :
                :
        values_n:  statements_n
END_CASE
```

Example 6.4: If AX contains a negative number, put -1 in BX; if AX contains 0, put 0 in BX; if AX contains a positive number, put 1 in BX.

Solution:

```
                CMP  AX, 0
                JL  NEGATIVE
                JE  ZERO
                JG  POSITIVE
NEGATIVE:
                MOV  BX, -1
                JMP  END_CASE
ZERO:
                MOV  BX, 0
                JMP  END_CASE
POSITIVE:
                MOV  BX, 1
END_CASE:
```

Example 6.5: If AL contains 1 or 3, display “o”; if AL contains 2 or 4, display “e”.

Solution:

```
        CMP  AL, 1
        JE  ODD
        CMP  AL, 3
        JE  ODD
        CMP  AL, 2
        JE  EVEN
        CMP  AL, 4
        JE  EVEN
        JMP END_CASE
ODD:    MOV  DL, ‘o’
        JMP  DISPLAY
EVEN:   MOV  DL, ‘e’
DISPLAY:
```

Branches with Compound Conditions

AND Conditions

Example 6.6: Read a character, and if it's an uppercase letter, display it.

Solution:

```
        MOV  AH,1
        INT  21H
        CMP  AL, 'A'
        JNGE  END_IF
        CMP  AL, 'Z'
        JNLE  END_IF
        MOV  DL,AL
        MOV  AH,2
        INT  21H
END_IF:
```

OR Conditions

Example 6.7: Read a character. If it's "y"' or "Y", display it; otherwise, terminate the program.

Solution:

```
        MOV  AH,1
        INT  21H
        CMP  AL, 'y'
        JE   THEN
        CMP  AL, 'Y'
        JE   THEN
        JMP  ELSE_
THEN:   MOV  AH, 2
        MOV  DL,AL
        INT  21H
        JMP  END_IF
```

```
ELSE_:   MOV  AH, 4CH
         INT  21H
END_IF:
```

6.4.2 Looping Structures

FOR LOOP

The pseudocode of for loop is as follows:

```
FOR loop_count times DO
Statements
END_FOR
```

The LOOP Instruction:

The **LOOP** instruction can be used to implement a for loop. It has the form:

LOOP destination_label

The counter for the loop is the register CX which is initialized to loop_count. Execution of the LOOP instruction causes CX to be decremented automatically, and if CX is not 0, control transfers to destination_label. If CX=0, the next instruction after LOOP is done.

Using the instruction LOOP, a FOR loop can be implemented as follows:

```
        ; initialize CX to loop_count
TOP:
        ; body of the loop
        LOOP TOP
```

Example 6.8: Write a count-controlled loop to display a row of 80 stars.

Solution:

```
        MOV  CX, 80
        MOV  AH,2
        MOV  DL, '*'
TOP:
        INT  21H
        LOOP  TOP
```

If CX contains 0 when the loop is entered, the loop is then executed FFFFH = 65535 more times! To prevent this, the instruction **JCXZ** (jump if CX is zero) may be used before the loop. Its syntax is

```
JCXZ   destination_label
```

So, a loop implemented as follows its bypassed if CX is 0. rm:

```
        JCXZ  SKIP
TOP:
        ; body of the loop
        LOOP  TOP
SKIP:
```

WHILE LOOP

The pseudocode of while loop is as follows:

```
WHILE  condition DO
Statements
END_WHILE
```

Example 6.9: Write some code to count the number of characters in an input line.

```
        MOV  DX, 0
        MOV  AH, 1
        INT  21H
WHILE_:
        CMP  AL, 0DH
        JE  END_WHILE
        INC  DX
        INT  21H
        JMP  WHILE_
END_WHILE:
```