Flow Control Instructions

Covering topics

- Overview
- Conditional Jumps
- Conditional Jump Range
- How CPU implements a Conditional Jump
- Signed and Unsigned Jumps, Single Flags Jumps
- CMP,JMP, IF THEN, IF THEN ELSE, CASE, Instruction
- Branches and Compound Conditions
- Loop Structures

Overview

- How to make decision and repeat code sections in assembly language.
- How control is transfer with jump and loop.
- Control transfer can be conditional or can depend on a particular combination of status flags.
- How to use jump instructions to implement high-level language decision and looping structure.

Conditional Jumps

- Jumps are used to transfer control to a label and depends on a condition. Means the instruction to be executed is the one at destination label, which may precede or follow the jump instruction itself.
- If condition is false instruction immediately executes the following the jump.

Conditional Jump Range

- Machine code of a conditional jump requires the destination label must precede the jump instruction no more then 126 bytes, or follow it by no more than 127 bytes.
- There are 256 characters set. From 32 to 127 are standard ASCII display characters. From 0-31 and 128-255 are set of graphic characters.

How CPU implements a conditional jump

- Conditional jumps are implemented by looking at the FLAGS registers.
- If the condition for the jump is true; CPU adjust the IP to point to the destination label.
- There are three categories of jumps.
- **Singed Jumps:** Are used when a signed interpretation is being given to results.
- **Unsigned Jumps:** Used for unsigned interpretation.,
- **Single Flag Jumps:** Which operates on setting of individual flags.

Signed Jumps

Symbol	Description	Condition for Jump
JG/JNLE	Jump if greater than Jump if not less or equal	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	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 Jump
JA/JNBE	Jump if above Jump if not below or equal	CF=0 and ZF=0
JAE/JNB	Jump if above of Equal Jump if not below	CF=0
JB/NAE	Jump if below Jump if not below or equal	CF=1
JBE/JNA	Jump if Equal Jump if not above	CF=1 or ZF=1

Single Flag Jumps

Symbol	Description	Condition for Jump
JE/JZ	Jump if Equal Jump if Zero	ZF=1
JNE/JNZ	Jump if not equal Jump if not zero	ZF=0
JC	Jump if carry	CF=1
JNC	Jump if no carry	CF=0
JO	Jump if overflow	OF=1
JNO	Jump if no overflow	OF=0
JS	Jump if signed	SF=1
JNS	Jump if unsigned	SF=0
JP/JPE	Jump if parity even	PF=1
JNP/JP	Jump if parity odd	PF=0

CMP Instruction

- Jump condition is often provided by the CMP instruction having following syntax.

```
        CMP    destination, source
```

- CMP instruction compares destination with source by subtracting destination from source, without changing the destination, but effecting the flags.
- Let us elaborate this by using an example

```
        CMP     AX,BX               ;AX=123d, BX=23d
         JG              DESTINATIONLABEL
```

this statement means 123-23=100 the result is positive means the sign flag is unaffected, the result is not zero, this means SF=ZF which satisfies OF=0 so control will be transfer to the DESTINATIONLABEL

JMP instruction

- JMP instruction is used for unconditional transfer of control. Syntax is

```
      JMP         DESTINATIONLABEL
```

- JMP is basically used to get around the range restriction of a conditional jumps.

```
      TOP:
                  ;body between jump and label
      DEC         CX
      JNZ         TOP
```

instructions between label and jump is out of range for JNZ. We can do this:

```
TOP:
                  ;body between jump and label
      DEC         CX
      JNZ         BOTTOM
      JMP         EXIT
BOTTOM:
      JMP         TOP
EXIT:
      MOV         AH,4CH
      INT         21h
```

IF THEN

- In high level language IF-THEN has following structure.

```
IF condition is true
THEN execute true-branch statements
END-IF
```

- Through programmatically:

```
 IF        AX<0
           THEN
           AX=-AX
END IF
```

- Through assembly it will be as:

```
      ;IF      AX<0              =>        CMP      AX,0
                                           JNL      END_IF
                                           NEG      AX
END_IF:
                                           MOV      AH,4CH
                                           INT      21h
```

IF THEN ELSE

-
  ```
  IF      condition is true
          THEN
          execute the true-branch
          ELSE
          execute the false-branch
          END_IF
  ```

-
  ```
  IF      AL<=BL
          THEN
          display the character in AL
          ELSE
          display the character in BL
          END_IF
  ```

CASE (SWITCH)

- When we have a multiway branch structure that tests a register, variable, or expression for particular values or a range of values.

```
CASE          expression
      value_1:        statement_1
      value_2:        statement_2
      ............3:       ………………..3

                .
                .
      values_n:       statement_n
      END_CASE
```

CASE (SWITCH)

- CASE AX

```
CASE    AX
        <0        put       -1        in        BX
        >0        put        1        in        BX
        =0        put        0        in        BX
        END_CASE
```

- :case AX

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

Branches and Compound Conditions

- Sometime branching condition in case and IF takes the forms

  Conditiion_1 AND Condition_2

  Condtion_1 OR Condition_2

  Where condition_1 and Condtion_2 or either true or either false. We refer to the first as AND Condition and the second as OR Condition.

- AND Condition: It is true only if condition_1 and condition_2 are both true. It either of them is false, then whole condition is false.

  Read a character if it is in lower case display otherwise exit.

  Solution: Read a character into AL

```
        if((AL<=‘z’) AND(AL>=‘a’)
        THEN
        display AL
        END IF
```

Branches and Compound Conditions

```
Read a character
MOV         AH,1
INT         21h
CMP         AL,’a’
JNGE        END_IF
CMP         AL,’z’
JNLE        END_IF
;then display
MOV         AH,2
MOV         DL,AL
INT         21H
END_IF:
```

Branches and Compound Conditions

- **OR Condition:** Condition_1 OR Condition_2 is true if either of the condition is true.

  Read a character. If it is ‘b’ or ‘B’ then display otherwise exit.

  Solution:

```
Read a character into AL
if((AL=‘b’)OR(AL=‘B‘))
THEN
display it
ELSE
Exit
END_IF
```

Branches and Compound Conditions

```
    MOV              AH,1
    INT              21H
     ;if
    CMP              AL , ’b’
    JE               DISPLAY
    CMP              AL,’B’
    JE               DISPLAY
    JMP              END_IF
DISPLAY:
    MOV              AH,2
    MOV              DL,AL
    INT              21H
_ELSE:
    MOV              AH,4CH
    INT              21h
END_IF:
```

Loop Structure

- A way to repeat a sequence of instructions, which will be known in advance or may depend on a specific condition.
- ***For Loop:*** It is used when we know how many times a specific portion of code will be executed.

```
FOR      counter times    DO
Statements
END_FOR
```

Syntax:

```
LOOP         destination_Label
```

Loop instruction is used to implement FOR loop, counter for loop is CX, which must be initialized before encountering the loop instruction and is decremented automatically, when the CX becomes 0 next instruction following the LOOP instruction will be executed.

Loop Structure

```
.MODEL    SMALL
.STACK              100h
.DATA
.CODE
MAIN                PROC
   MOV              CX,'z'
DISPLAY:
   CMP              CX,'a'
JL  END_LOOP
   MOV              DL,CL
   MOV              AH,2
   INT              21H
   LOOP   DISPLAY
END_LOOP:
   MOV              AH,4CH
   INT              21h
MAIN      ENDP
END       MAIN
```

References

- Assembly Language Programming and Organization of the IBM PC (Ytha Yu, Charles Marut)