CSE 331/L-14/28.03.2024/

String Comparisons

- ⊛ Block Data manipulation occurs with MOVS, LODS, STOS, INS and OUTS.

- ⊛ SCAS B/W/D/Q ⇒ String Scan

  - scan within a memory with a register.

⇒ Source and destination operand fixed by default.

- Only the flag bits changed after execution.

⇒ Destination operand:

ES: DI

Source Operand : AL, AX, EAX

D = used for auto increament or decrement

CX = used for bit count to scan or repeat count.

- ⊛ two repeat instruction can be used here.

REPNE ⇒ Repeat if not equal

REPE ⇒ Repeat if equal

$\boxed{\text{slide- 71}}$ → Example

⊛ CMPS B/W/D/Q ⇒ String Compare

- compare two data block in memory

⇒ Source and Destination operand is fixed by default.

- Only the flag bits changed after execution.

⇒ Destination operand:

ES : DI

Source Operand :

DS : SI

D = used for direction

CX = used for repeat count or data block size for compare

⇒ two repeat instruction can be used

⇒ REPE & REPNE

Slide-73 → Example

From eBooks

Ytha Yu & Charles Manut

Chapter - 6

Page - 106

Flow Control Instruction

⇒ Control the execution flow.

⊛ There are two types of jump instruction:

(i) Unconditional Jump ⇒ JMP ⇒ No Restriction

(ii) Conditional Jump ⇒ JXXX/JXX/JX ⇒ Label need to be close to the jump instruction. Limit above - 126 byte bellow - 127 byte

- Unsigned C.J.
- Signed C.J.
- Single flag/bit C.J.

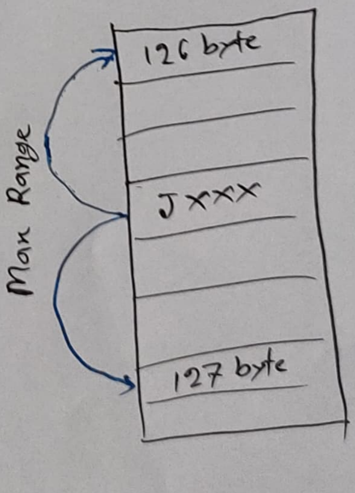
⊛

Unsigned ⇒ above/below ⇒ JA

Signed ⇒ greater/less ⇒ JG

Flag bits ⇒ JO

↳ Overflow flag



⊛ Importance of ~~sing.~~ signed and unsigned:

Let's say,

0111 → Sign bit (+)

AX = 7FFF H ⇒ Positive for signed number

BX = 8000 H ⇒ negative for signed number

1000 → sign bit (−)

⇒

CMP AX, BX ; AX − ~~A~~BX = ? (−)

×JA NSV ; if AX > BX ⇒ for unsigned its false

✓JG NSV ; if $\overset{(+)}{AX} > \overset{(-)}{BX}$ ⇒ for signed its true

⊛ For extended ASCII character:

80 H → FF H

$\underbrace{1000}_{(-)\text{ for signed}}$ $\underbrace{1111}_{(-)\text{ for signed}}$

⇒ we need to use unsigned conditional jump.

⇒ for other case, we can use both signed or unsigned conditional jump.

⊛ Handling the range of conditional jump:

```
TOP:

    ; body of the loop

    DEC CX    ; Decrement Counter - arithmetic operation, flag bits will
                                                         change (Z)
    JNZ TOP   ; if Z flag is not set as 0, jump to TOP

    MOV AX, BX ; if conditional jump is false, it will execute as
                        normal.
```

⇒ This code will work perfectly as soon as the label TOP is within the 126 byte range. What about if the label is far away than the JNZ instruction over 126 byte?

It will not work.

⇒ Solution:

```
TOP:
      ; body of the loop

      DEC CX

      JNZ BOTTOM
      MOV AX, BX
  BOTTOM:
      JMP TOP ; unconditional jump. no restriction can jump
                       any where.
```

⇒ So, the range problem is solved. But another problem is here. mov instruction will execute as normal flow as well as JMP will also enecute in nonmal enecution flow, even aften the JNZ is false, it will enecute. And it will cause an infinite loop enecution.

⇒ Solution:

```
TOP :
      ; body of the loop
      DEC  CX
      JNZ  BOTTOM
      JMP  EXZT  ; in nonmal flow it will jump to the exit.
                   ignone the BOTTOM label.
  BOTTOM:
      JMP  TOP

  EXZT :
      MOV  AX, BX  ; aften that nonmal enecution flow
                     will occune.
```

⊛ High Language Stnucture:

⇒ We have used a lot of funetion, method in high level language ~~like c.~~ (c) like if, if else, switch-case etc.

We alneady see, how loop wonk in low level language. Now we will see the stnuctune of if then, if else, switch-case etc.

6·2/ IF - THEN ⇒ Replace number in Ax by its absolute value
↳ |Modulas|

Algorithm:

```
IF AX < 0
   THEN
      replace AX by -AX
END_IF
```

⇒ we need to inverse the condition for executing 'then'

⇒ in this case ≮ or >=

Code:

```
    CMP AX, 0    ; AX-0
    JGE END_IF   ;   AX ≥ 0 exit ~~loop~~ if else continue normal flow
    NEG AX       ; it will execute as normal flow. when AX < 0
END_IF:
       ; normal execution flow
```

6·3/ IF-THEN-ELSE

⇒ Suppose AL, BL contain $\underbrace{\text{extended ASCII}}_{\text{need to use unsigned conditional jump}}$ character. Display the character which come first in sequence.

Algorithm:

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

⇒ we need to inverse the condition ⇒ ≰ or >

Code:

```
        MOV AH, 2      ; prepare to display

; IF start
        CMP AL, BL     ; AL - BL

        JA ELSE        ; if  AL > BL goto else

; then
        MOV DL, AL     ; mov al for output. normal execution flow if
                       ; above condition false.
        JMP DISPLAY    ; jump to display section for interrupt.
                       ; ignore else statement

ELSE:
        MOV DL, BL     ; mov bl to output register

DISPLAY:
        INT 21H        ; display it. normal execution flow

END_IF:
        ; normal execution flow
```

6.4/ Switch- Case

⇒ 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.

Algorithm:

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

} no need to inverse the condition in switch-case structure

Code:

```
; CASE AX

        CMP AX, 0       ; test ax, AX-0

        JL NEGATIVE     ; AX < 0

        JE  ZERO        ; AX = 0

        JG  POSITIVE    ; AX > 0. we can use the code of
                          positive section here directly. Because,
                          in this case if above two condition
 NEGATIVE:                is false then, AX must be positive.
        MOV BX, -1

        JMP END-CASE ; exit to avoid other statement of
                          another case.

 ZERO :
        mov BX , 0

        JMP END-CASE

 POSITIVE :
        mov BX, 1    ; no ~~end~~ need to exit as it will execute
                          as normal flow.

 END-CASE :

        ; normal execution flow
```

range 41 H → 54 H
we can use signed conditional jump

6.6.1 AND (&&)

⇒ Read a character, and if it's an uppercase letter, display it.

Algorithm:

```
Read a character in AL
IF ('A' <= character && character <= 'Z') ⇒ again we need to
                                             inverse the condition
  THEN
     display character

END_IF
```

Code:

```
; read a character

        MOV AH, 1    ; prepare for read
        INT 21H      ; read char in AL

; if statement

        CMP AL, 'A'  ;  AL -'A' test

        JL END_IF    ;  AL < 'A' go to end_if, as one condition is
                                                        fail

        CMP AL, 'Z'  ;  ~~A~~ compare with 'Z'

        JG END_IF    ;  AL >'Z' go to exit, any one false exit
                                          the if statement

; if both condition false that means its uppercase letter, display
    it by normal execution flow

         MOV DL, AL  ; moving to output register

         MOV AH, 2   ; prepare to display

         INT 21H     ; show output

    END_IF:

         ; normal execution flow as if statement end here.
```

6.7/ OR (ii)

⇒ Read a character if its "y" or "Y", display it,

otherwise terminate it.

Algorithm:

```
Read a character in AL

IF ( character = 'y' || character = 'Y' ) => no need to inverse
                                             for OR structure
  THEN
        display it
 ELSE
      terminate the program
 END-IF
```

Code:

```
; read a character

     MOV AH, 1    ; prepare to read
     INT 21H      ; read character in AL

; if condition

     CMP AL, 'y'   ; comparing with 'y' ; AL-'y'
     JE THEN       ; any one true execute the if statement
     CMP AL, 'Y'   ; comparing with 'Y' ; AL-'Y'
     JE THEN       ; any one true execute the if statement
     JMP ELSE      ; if both are false go to else

 THEN :
     MOV AH ,2     ; prepare to display
     MOV DL, AL    ; moving to output register
     INT 21H       ; display it
     JMP END-IF    ; ignoring the ELSE statement

 ELSE :
     MOV AH, 4C H  ; DOS Exit
     INT 21 H
```

END-IF:

; normal execution flow.

✻ Extra: AND, OR mix

⇒ read a character, if it is a uppercase letter or a number then display it.

Algorithm:

```
Read a character in AL

if ( ~~A~~ ('A' <= AL && AL <= 'Z') || ('0' <= AL && AL <= '9'))

  THEN
     display it

 END-IF
```

Code:

```
; read a character

    MOV AH, 1     ; prepare to read
    INT 21H       ; read character in AL

; 1st condition
     cmp AL, 'A'   ; comparing with 'A'
     JL  SECOND-CONDITION  ; need to check 2nd condition
     cmp  AL, 'Z'  ; comparing with 'Z'
     JG  SECOND-CONDITION  ; need to check 2nd condition
     JMP THEN ; if both false, then 1st condition true and
                need to execute the statement under if.
                   no need to check second condition
```

```
SECOND_CONDITION:           ; we are now in second condition
                              that's means first condition false

    CMP AL, '0'

    JL  END_IF              ; AL <'0', ~~then~~ any one false will make
                              the second condition also False
                              and need to exit if

    CMP AL, '9'

    JG  END_IF              ; AL >'9', ~~fail~~ then goto exit.

    ~~JUMP THEN~~           ; if both fail, then go as normal flow

THEN:     ; normal execution flow ~~if~~ for second condition

    MOV DL, AL              ; moving to output register

    MOV AH, 2               ; prepare for output

    INT 21H                 ; display it

END_IF:

    ; normal execution flow
```

H.W. ⇒ All Example

Upto Page - 117
Done