EXPERIMENT 7

Call, Jumps and Loop Instructions

Objective

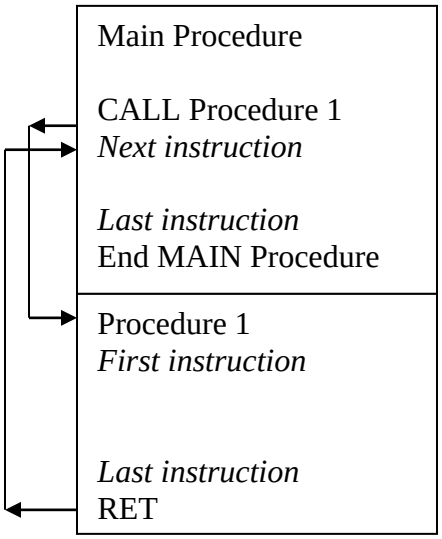
- Understand working of Procedure (CALL instruction) and Stack
- Understand difference between Conditional and Unconditional jump
- Implement LOOP instruction

Theory

The call and jump instructions transfer the flow of the program. The main difference between call and jump instruction is the jump instruction jumps to target and proceed further instructions, whereas call instruction jump to target (called subroutine or procedure) execute the lines until reach to the RET instruction, then return back to the next line from where its called.

CALL and RET instruction

The Subroutines (procedures) in 8086/88 assembly language are design by using CALL and RET instruction. The CALL instruction use to call the procedure and RET instruction jump back the flow to the next line from where it is call.



- When procedure is call the processor first store (PUSH) the address of next line on the stack and jump to the procedure. The procedure execute until RET (return) instruction execute; then processor get back (POP) the address from stack (the address of next line which is store when procedure is call) and jump to the next line of the CALL instruction.

Example: In the following program a procedure of display string function is made by the name of DISPLAY.

```
                MOV   AX, @DATA
                MOV   DS, AX
                LEA   DX, MESSAGE1
                CALL  DISPLAY
                LEA   DX, MESSAGE2
                CALL  DISPLAY
                LEA   DX, MESSAGE3
                CALL  DISPLAY
                JMP   END_PROG
DISPLAY:        MOV   AH, 09H       ; Display function subroutine
                INT   21H
                RET
END_PROG:
```

Jump instruction

Unlikely the CALL instruction jump instruction has same working except that the jump instruction not return. The Jumps instruction can distinguish into two types:

1) Unconditional Jump
2) Conditional Jump

1) Unconditional Jump

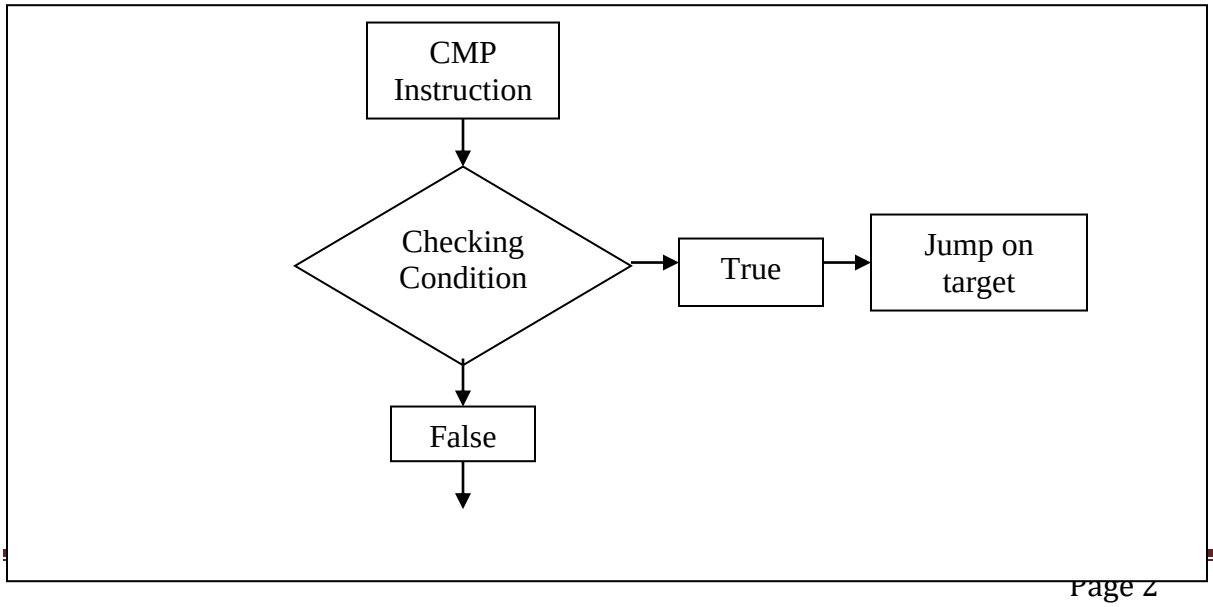
The unconditional jump transfers the control of program without checking any condition. The unconditional jump can go any where in the program and has no limitation.

Unconditional Jump Format

```
JMP    Target          ; The target is the label
```

2) Conditional Jump

The conditional jump transfers the controls of program by first watching the condition. The conditional jump depend on the value of status flag. If condition is true then it jump on target else it goes on the next line. Also all unconditional jumps are short jump means the target must present with in range of -128 byte to +127 byte.



The working of conditional jump

Conditional jump can be categorized into three types:

1) **The Signed Jump**
2) **(a=<2)**

JG / JNLE	*Target* ; Jump if Greater / ; Jump if Not Less or Equal	(ZF = 0 & SF = OF)
JGE / JNL	*Target* ; Jump if Greater or Equal / ; Jump if Not Less	(SF = OF)
JL / JNGE	*Target* ; Jump if Less / ; Jump if Not Greater or Equal	(SF < > OF)
JLE / JNG	*Target* ; Jump if Less or Equal / ; Jump if Not Greater	(ZF = 0 & SF < > OF)

2) **The Unsigned Jump**

JA / JNBE	*Target* ; Jump if Above / ; Jump if Not Below or Equal	(CF = 0 & ZF = 0)
JAE / JNB	*Target* ; Jump if Above or Equal / ; Jump if Not Below	(CF = 0)
JB / JNAE	*Target* ; Jump if Below / ; Jump if Not Above or Equal	(CF = 1)
JBE / JNA	*Target* ; Jump if Below or Equal / ; Jump if Not Above	(CF = 1 or ZF = 1)

3) **The Single Flag Jump**

JZ / JE	*Target* ; Jump if Zero/ Jump if Equal	(ZF = 1)
JNZ / JNE	*Target* ; Jump if Not Zero/ Jump if Not Equal	(ZF = 0)
JC	*Target* ; Jump if Carry	(CF = 1)
JNC	*Target* ; Jump if No Carry	(CF = 0)
JP / JPE	*Target* ; Jump if Parity/ Jump if Parity Even	(PF = 1)
JNP /JPO	*Target* ; Jump if No Parity/ Jump if Parity Odd	(PF = 0)
JO	*Target* ; Jump if Overflow	(OF = 1)
JNO	*Target* ; Jump if No Overflow	(OF = 0)
JS	*Target* ; Jump if *negative*Sign	(SF = 1)
JNS	*Target* ; Jump if No *negative*Sign	(SF = 0)

The difference between Signed jump and Unsigned jump is only status of Sign Flag (SF). The Signed jump watch the status of Signed flag with other flag whereas the Unsigned jump does not watch signed flag. The Single Flag jump only watch the status of the desire flag only.

The Signed Jump example:

```
        CMP   designation, source
        JG    Target        ; Jump if greater
```

The Unsigned Jump example:

```
        CMP   designation, source
        JA    Target        ; Jump if above
```

The above instructions having same working that is jump if destination is greater than source but the Signed jump see SF(signed flag), CF(carry flag) and ZF(
flag) where as Unsigned jump see only CF(carry flag) and ZF(zero flag).

The Single flag jump

```
        CMP   designation, source
        JZ    Target        ; Jump if designation = source
```

and

```
        JC    Target        ; Jump if carry
        JP    Target        ; Jump if parity
```

Example:

1) The following program takes decimal number from keyboard and if user presses button other then any decimal number then it take again.

```
AGAIN:  MOV   AH, 01H       ; Take input function
        INT   21H
        CMP   AL, '0'
        JB    AGAIN         ; Jump if below than 0
        CMP   AL, '9'
        JA    AGAIN         ; Jump if greater than 9
```

2) The following program takes only + sign and – sign from keyboard and jump on that function, and if user presses any other button then it take again.

```
AGAIN:  MOV   AH, 01H       ; Take input function
        INT   21H
        CMP   AL, '+'       ; Compare with + sign
        JZ    ADDITION      ; If + sign enter then jump to ADDITION
        CMP   AL, '-'       ; Compare with - sign
        JE    SUBTRACTION ; If - sign enter then jump to SUBTRACTION
JMP   AGAIN         ; Jump to again if all condition were false
```

The CMP Instruction

In the above examples the unconditional jump depend on the values present in the Status Flag, but how the flag is affect to meet condition. This is done by CMP instruction, the CMP instruction compare the destination with source by subtracting source value from destination value, the result is not store in operand but according to result the bits of status flag register affected. The CMP

Instruction is used to compare Byte or Word, to check status of Bit(s) see TEST instruction. (Note that the CMP is instruction is same as SUB instruction except the answer is not store in the designation in CMP instruction).

The TEST Instruction

The TEST instruction perform logical AND operation to affect the flags but result does not store. The TEST instruction is used to check specific bit(s) are set or clear.
Example:

The only difference between upper case character ASCII and lower case character ASCII is of bit 5 which is '1' in lower case character ASCII and '0' in upper case character ASCII. The following program check the ASCII present in AL register, if it is upper case the program display 'U' character on screen and if it is lower case then display 'L'.

```
        TEST  AL,00100000B
        JE    LOWER
        MOV   DL,'U'
        MOV   AH,02H
        INT   21H
        JMP   END_PROG
LOWER:
        MOV   DL,'L'
        MOV   AH,02H
        INT   21H
END_PROG:
```

The LOOP Instruction

The LOOP instruction repeats the sequence of instructions. The LOOP instruction jumps on target to the number of times the value present in CX register.
The LOOP instruction execute in following sequence.

1. Decrease CX register.(5) -4,3,2,1
2. Compare CX with 0.
3. If CX = 0 then goto next line else jump on Target.

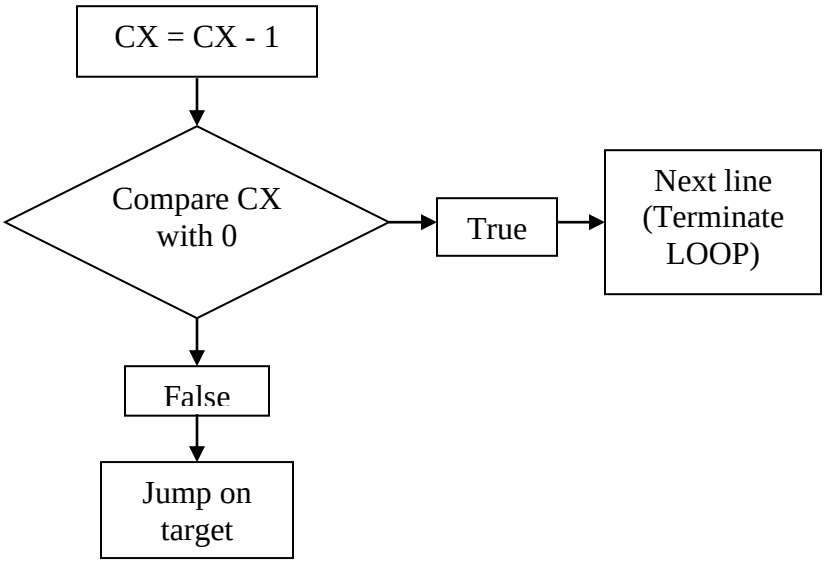
LOOP Instruction format:

```
        LOOP  Target  ; The target should be written before the LOOP instruction
```

Example:

```
        MOV  CX, 05H
        MOV  AL, 00H
        MOV  BL, 09H
HERE:   INC  AL
        DEC  BL
        LOOP HERE
```

After completion of the program values present in register are Al = 05H, = BL = 04H and CX = 00H



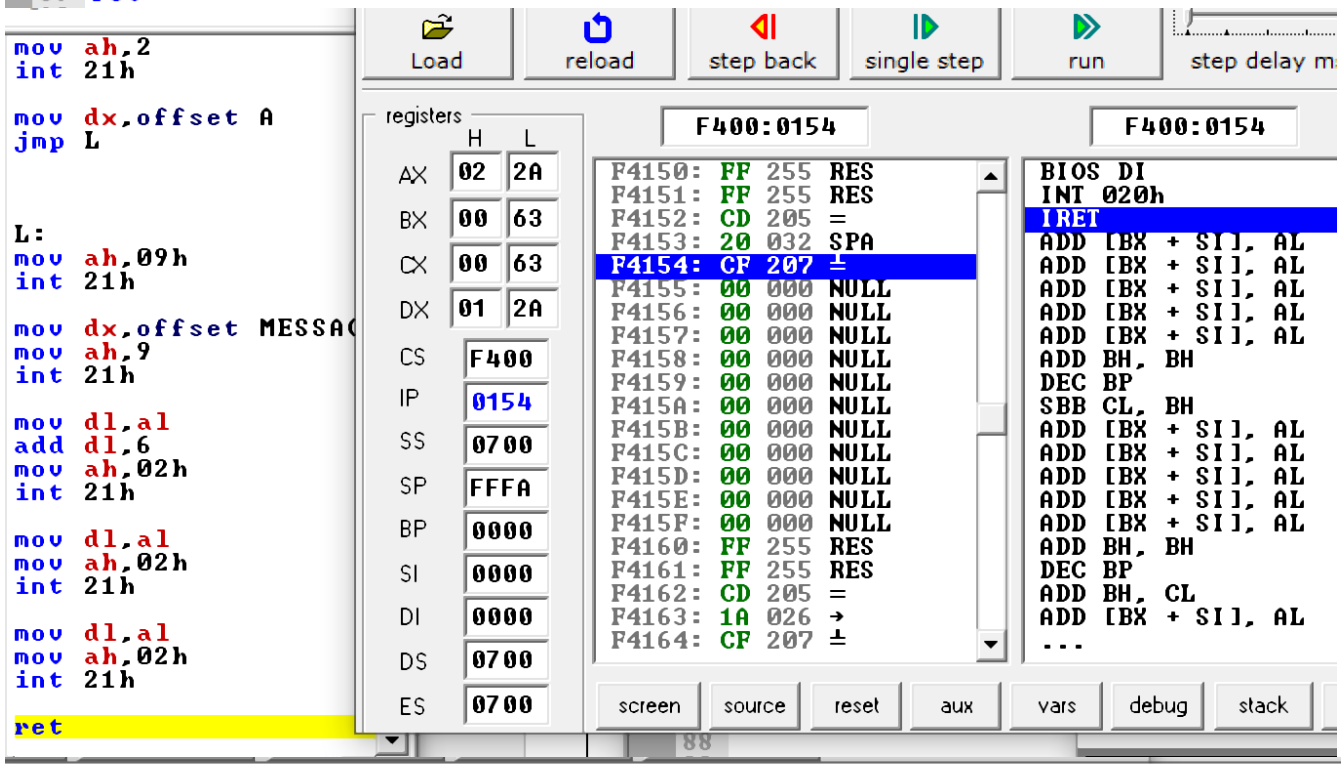
The LOOP instruction flow

Exercise

1) Write a password program which take three character and display '*' on screen. Then compare the input character with the store character and display "Correct Password" if password correctly enter else print "Incorrect password" on screen.

```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.txt

org 100h

; add your code here
.data

A db 'password is correct $'
B db 'password is not correct $'
C db 'Enter your Password : $'
MESSAGE db 0ah , 0dh , 'your input were : $'

.code
main :
mov ax,@data
mov ds,ax

mov dx,offset C
mov ah,09h
int 21h

mov ah,01h
int 21h
int 21h
int 21h

mov dx,'ab'
mov bx,'c'
lea dx,bx
mov cx,dx

cmp al,cl

je L1


mov dl,10
mov ah,2
int 21h

mov dl,13
mov ah,2
int 21h

mov dx,offset B
jmp L




L1:
mov dl,10
mov ah,2
int 21h

mov dl,13
mov ah,2
int 21h

mov dx,offset A
jmp L

```

```
67
68  L:
69  mov ah,09h
70  int 21h
71
72  mov dx,offset MESSAGE
73  mov ah,9
74  int 21h
75
76  mov dl,al
77  add dl,6
78  mov ah,02h
79  int 21h
80
81  mov dl,al
82  mov ah,02h
83  int 21h
84
85  mov dl,al
86  mov ah,02h
87  int 21h
88
89  ret
```



2) Write a program which ask user to enter a digit (from 1 to 9 only) and then print the table of that number.

The program is look like this
Please enter a digit = 3
(User press 3 and then press Enter key)

3 x 01 = 03
3 x 02 = 06
3 x 03 = 09
3 x 04 = 12
3 x 05 = 15
3 x 06 = 18
3 x 07 = 21
3 x 08 = 24
3 x 09 = 27
3 x 10 = 30

```
.model small
.stack 100h
.data
A DB 'enter a number : $'
.code
main proc

    mov ax,@data
    mov ds,ax
    lea dx,A
    mov ah,9
    int 21h

        |

     mov dx,10
    mov ah,2
    int 21h
    mov dx,13
    mov ah,2
    int 21h

    mov ah,01h
    int 21h

   mov ch,0AH
   mov cl,00h


   cmp al,3ah
   sub al,30h
   mov bh,al
   mov bl,01h

    L1:
    mov dl,0dh
    mov ah,02h
    int 21h

    mov dl,0ah
    mov ah,02h
    int 21h

    mov dl,'0'
    mov ah,02h
    int 21h

    mov dl,bh
    add dl,30h
    mov ah,02h
    int 21h

    mov dl,'*'
    mov ah,02h
    int 21h

    mov al,bl
    mul bh

    AAM
```

```
    PUSH ax

    mov ah,00h
    mov al,bl

    AAA

    mov cl,ah
    mov bl,al

    mov dl,cl
    add dl,30h
    mov ah,02h
    int 21h

    mov dl,bl
    add dl,30h
    mov ah,02h
    int 21h

    mov dl,'='
    mov ah,02h
    int 21h

    POP ax

    mov dh,al
    mov dl,ah
    add dl,30h
    mov ah,02h
    int 21h

    mov dl,dh
    add dl,30h
    mov ah,02h
    int 21h



    inc bl
    dec ch
    cmp ch,00h



    jne L1


   mov ah,4ch
   int 21h



    main endp
end main
ret
```