Experiment 5

Addressing modes, CALL , RET , XLAT , Stack , Arrays

5.1 Objective

The objective of this experiment are

- To learn the basic ideas of addressing modes: accessing variables and array elements,
- To get familiarized with procedures and stack: CALL and RET instructions, and
- To understand the XLAT instruction.

5.2 Address Arithmetic

5.2.1 Arrays

Arrays allow us to refer to a collection of similar data by a single name and individual items in the collection with the name and a numeric subscript. An array of 4 bytes might be declared as

```
Arr     DB      0BH , 1BH , 2BH , 3BH
```

Now if we write

```
MOV             AL , Arr +3      ;        sets AL to 3BH
MOV             AL , [Arr + 3]   ;        sets AL to 3BH
```

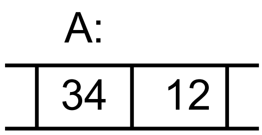
Suppose the following arrays are defined

```
A       DB      0Ah , 1Ah , 2Ah , 3Ah , 4Ah , 5Ah
B       DB      0Bh , 1Bh , 2Bh , 3Bh
C       DB      0Ch , 1Ch
```

Then the following instructions have the results given:

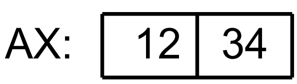
```
MOV     AL , [B+3]      ;  sets AL = 3Bh
MOV     AL , [B+4]      ;  sets AL = 0Ch
MOV     AL , [A+12]     ; sets AL = 1Ch
```

5.2.2 Byte Swapping



8086 stores the low order byte first.

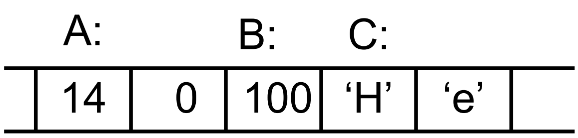
```
A       DW      1234h
```



Is similar to the instruction

```
A       DB      34h , 12h
MOV     AX , A  ; sets AX = 1234h (AH = 12h , AL = 34h)
```

Now see:




```
A       DW      14
B       DB      100
C       DB      'Hello'
MOV     AX , WORD PTR B         ; sets AL to 100, AH to 'H' = 72
```

5.2.3 Examples and applications

Example 1:

```
CODE   SEGMENT
       ASSUME CS: CODE , DS: CODE
; sets up an array of 10 words , with each initialized by 9
; Here we shall copy W to Z and set W[N] = 0 for N = 0 to 9
       ORG    100H
       MOV    CX , 10
       MOV    DI , 0                  ; N = 0
Zero:
       MOV    AX , [W + DI]
       MOV    [Z + DI] , AX
       MOV    [W + DI] , 0            ; W[N] = 0
       ADD    DI , 2                  ; N = N + 1
       DEC    CX
       JNZ    Zero
       HLT

       ORG    150H
W      DW     10     DUP (9)          ; Creates an array of 10 words initialized by 9 each
                                      ; Note : DUP came for duplicate.
Z      DW     10     DUP(?)           ; Creates an array of 10 uninitialized words
CODE ENDS
       END
```

Example 2:

```
CODE   SEGMENT
       ASSUME CS: CODE , DS: CODE
; convert all lowercase letters in the array to the corresponding uppercase letters
;
       ORG    100H
       MOV    SI , OFFSET    B
Upcase:
       MOV    AL , [SI]
       CMP    AL , '$'
       JE     DONE
       CMP    AL , 'a'
       JB     NotLC
       CMP    AL , 'z'
       JA     NotLC
       ADD    BYTE PTR [SI] , 'A'  - 'a'
NotLC:
       INC    SI
       JNZ    Upcase
DONE:
       HLT

B      DB     'HelLo' , '$'
CODE ENDS
       END
```

Example 3:

```
; This program generates the average marks of students' class test
CODE   SEGMENT
```

```
        ASSUME CS: CODE , DS: CODE

        ORG     100H
        MOV     AX , CS
        MOV     DS , AX
        MOV     SI , 3
REPEAT:
        MOV     CX , 4
        XOR     BX , BX
        XOR     AX , AX
FOR:
        MOV     DL , MARK[BX + SI]   ; Based Indexed Addressing mode
  ; Note: as we want to take one byte contents from the
  ; memory location MARK[BX + SI] the destination must be
  ; also an one byte register
        MOV     DH , 0
        ADD     AX , DX              ; accumulating the summation in AX
        ADD     BL , 4
        LOOP    FOR
; end of FOR
        XOR     DX , DX
        DIV     FOUR                 ; Here AH will contain the remainder and AL the result
        MOV     AVG[SI] , AL
        SUB     SI , 1
        JNL     REPEAT
DONE:
        HLT

FOUR            DB      4
        ;Class Test Mark:   T1 : T2 : T3 : T4 : Name
        ;------------------------------------------------
MARK            DB      15 , 20 , 12 , 16 ; Shajib
                DB      14 , 10 , 18 , 20 ; Imran
                DB      12 , 15 , 20 , 17 ; Akash
                DB      14 , 17 , 16 , 11 ; Maisha
AVG             DB      4    DUP (0)

CODE ENDS
        END
```

5.3 Procedures

5.3.1 CALL

To invoke a procedure, the **CALL** instruction is used. Executing a CALL instruction causes the following to happen:

1. The return address to the calling program is saved on the stack. This is the offset of the next instruction after the CALL statement.
2. IP gets the offset address of the first instruction of the procedure.

5.3.2 RET

To return from a procedure the instruction

RET pop_value

is executed. The integer argument pop_value is optional. After executing RET the value at the stack is stored into IP. Remember this value was previously stored in the stack when we called the procedure and this is the address of the next instruction of the CALL statement. If a pop_value is specified, it is added to SP after performing the previous action (the value in the stack to IP).

5.3.3 Examples And Applications

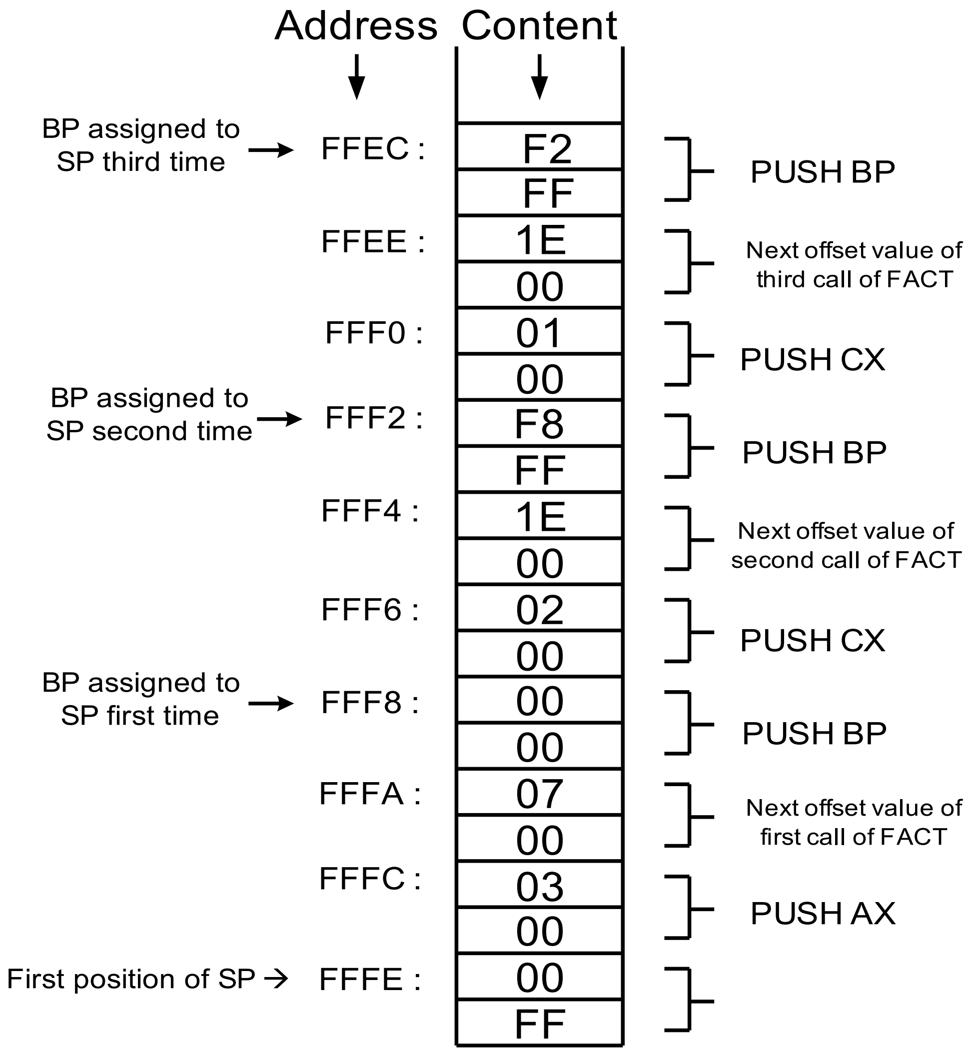
Example : 4

```
;First follow the code in the box.
;If you face problem to find the logic of the
;code below then keep it for home.
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
   ORG 100H
   MOV  AX,3
   PUSH AX
   CALL FACT
     ; AX CONTAINS THE FACTORIAL OF AX
   HLT
FACT:
   PUSH BP
   MOV  BP,SP
IF:
   CMP  WORD PTR[BP+4],1
   JG   ENDIF
THEN:
   MOV  AX,1
   JMP  RETURN
ENDIF:
   MOV  CX,[BP + 4]
   DEC  CX
   PUSH CX
   CALL FACT
   MUL  WORD PTR[BP+4]
RETURN:
   POP  BP
   RET  2            ; increments the stack pointer (SP) by 4. Here 2 increament is for
                     ; popping the stack top to IP and the remaining 2 is that is specified after ; the RET instruction.
CODE ENDS
   END
```

```
CODE SEGMENT
   ASSUME CS:CODE,DS:CODE
   ORG 100H
   MOV  AX,3
   PUSH AX
   CALL FACT
   ; AX CONTAINS THE FACTORIAL OF AX
   HLT
FACT:
   PUSH BP
   MOV  BP,SP
IF:
   CMP  WORD PTR[BP+4],1
   JG   ENDIF
THEN:
   MOV  AX,1
   POP  BP
   JMP  RETURN
ENDIF:
   MOV  CX,[BP + 4]
   DEC  CX
   PUSH CX
   CALL FACT
   MUL  WORD PTR[BP+4]
   POP  CX
   POP  BP
RETURN:
   RET
CODE  ENDS
   END
```

The instruction in the box do the same operation. But here we have equal number of PUSH and POP instructions. The program flow itself maintain the value of stack pointer through PUSH POP instruction. If we have not the same number of PUSH and POP then we have think about the stack. And we may need to insert value after the RET instruction.

For clear understanding the change in stack during the program is shown below:



STACK

5.4 XLAT

The instruction **XLAT** (translate) is a no-operand instruction that can be used to convert a byte value into another value that comes from a table. The byte to be converted must be in AL, and the offset address of the conversion table must be in BX. The instruction does :

Step 1. Adds the contents in AL to the address in BX to produce an address within the table.
Step 2. Replaces the contents of AL by the value found at that address.

5.4.1 Examples And Applications

Example: 5

```
; To read a secret message
CODE    SEGMENT
   ASSUME CS: CODE, DS: CODE
   ;
   ORG 100H
```

```
   LEA BX,ENCRYPT ; LEA stands for Load Effective Address.
   LEA SI,ORIGINAL
   LEA DI,ENCODED
; To convert a message into encrypted version
NCRPT:
   MOV AL,[SI]
   CMP AL,'$'
   JE END1
   XLAT
   MOV [DI],AL
   INC DI
   INC SI
   JMP NCRPT
END1:
   MOV [DI],AL
;To decrypt the encrypted message
   LEA BX,DECRYPT
   LEA SI,ENCODED
   LEA DI,DECODED
DCRPT:
   MOV AL,[SI]
   CMP AL,'$'
   JE  END2
   XLAT
   MOV [DI],AL
   INC DI
   INC SI
   JMP DCRPT
END2:
   MOV [DI],AL
;End of decryption
   HLT
        ;       ALPHABET            'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ENCRYPT    DB 65 DUP ('  '), 'XQPOGHZBCADEIJUVFMNKLRSTWY' ; one space in the bracket
           DB 37 DUP ('  ')   ; Think ! why the 65 blank spaces placed first.
        ;       ALPHABET            'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
DECRYPT    DB 65 DUP ('  '), 'JHIKLQEFMNTURSDCBVWXOPYAZG'
           DB 37 DUP ('  ')
ORIGINAL    DB 'GATHER YOUR FORCES AND ATTACK' , 30 DUP ('$')
ENCODED     DB 80 DUP ('$')
DECODED     DB 80 DUP ('$')
;
CODE ENDS
   END
```

Go to the offset address 0131H for original message , offset address 016CH for encoded message and offset address 01BCH for decoded message. Use your own message and verify to those addresses after running.

Home Task:

- Make a program that will sort an array content descending order and ascending order and put in another arrays. First of all describe the algorithm you want to implement.
- Write an algorithm to convert a binary number into decimal and implement it in assembly language.
- Find out the DECRIPT sequence for the ENCRIPT sequence of :

‘QWERTYUIOPASDFGHJKLZXCVBNM’

And rewrite the last program to detect any secret message according to the upper encryption.

Date: 7 - 7 - 7