Experiment 8

Transfer of Control Instructions (Call and Return)

Introduction:

In this experiment you will be introduced to subroutines and how to call them. You will verify the exchange of data between a main program and a subroutine in the 8086 environment. You will also use Macros, and as applications you will deal to a useful data representation: look-up tables.

Objectives:

To study CALL and RET instructions present in 8088 instruction set.

Subroutine calls:

A procedure is a reusable section of the software that is stored in memory once, but used as often as necessary. The CALL instruction links to the procedure and the RET (return) instruction returns from the procedure. The Stack stores the return address whenever a procedure is called during the execution of a program. The CALL instruction pushes the address of the instruction following the CALL (return address) onto the stack. The RET instruction removes an address from the stack, so the program returns to the instruction following the CALL.

With the Assembler (MASM) there are specific ways for writing, and storing, procedures. A procedure begins with the PROC directive and ends with the ENDP directive. Each directive appears with the name of the procedure. The PROC directive is followed by the type of the procedure: NEAR (intra-segment) or FAR (inter- segment).

In MASM version 6.X, a NEAR or FAR procedure can be followed by the USES statement. The USES statement allows any number of registers to be automatically pushed onto the stack and popped from the stack within the procedure.

Procedures that are to be used by all software (*global*) should be written as FAR procedures. Procedures that are used by a given task (*local*) are normally defined as NEAR procedures.

The CALL Instruction:

The CALL instruction transfers the flow of the program to the procedure. The CALL instruction differs from the jump instruction in the sense that a CALL saves a return address on the stack. The RET instruction return control to the instruction that immediately follows the CALL. There exist two types of calls: FAR and NEAR, and two types of addressing modes used with calls, Register and Indirect Memory modes.

Near CALL:

A near CALL is three bytes long, with the first byte containing the opcode, and the two remaining bytes containing the displacement or distance of ±32 K. When a NEAR CALL executes, it pushes the offset address of the next instruction on the stack. The offset address of the next instruction appears in the IP register. After saving this address, it then adds the displacement from bytes 2 and 3 to the IP to transfer
control to the procedure. A variation of NEAR CALL exists, CALLN, but should be avoided.

Far CALL:

The FAR CALL can call a procedure anywhere in the system memory. It is a five-byte instruction that contains an opcode followed by the next value for the IP and CS registers. Bytes 2 and 3 contain the new contents of IP, while bytes 4 and 5 contain the new contents for CS. The FAR CALL instruction places the contents of both IP and CS on the stack before jumping to the address indicated by bytes 2 to 5 of the
instruction. This allows a call to a procedure anywhere in memory and return from that procedure. A variant of the FAR CALL is CALLF but should be avoided.

CALLs with register operand:

CALLs can contain a register operand. An example is CALL BX, in which the content of IP is pushed into the stack, and a jump is made to the offset address located in register BX, in the current code segment. This type of CALL uses a 16-bit offset address stored in any 16-bit register, except the segment registers.

Program 7.1 illustrates the use of the CALL register instruction to call a procedure that begins at offset address DISP. The offset address DISP is placed into the BX register, then the CALL BX instruction calls the procedure beginning at address DISP. This program displays “OK” on the monitor screen.

CALLs with Indirect Memory Address:

A CALL with an indirect memory address is useful when different subroutines need to be chosen in a program. This selection process is often keyed with a number that addresses a CALL address in a lookup table.

ASSEMBLER PROGRAM

program that uses a CALL lookup table to access one of three different procedures:

```
.MODEL SMALL

.DATA                                        ;  indicate start of DATA segment
  TABLE DW          ONE                      ;  define lookup table
        DW          TWO
        DW          THREE

.CODE                                        ;  indicate start of CODE segment

  ONE  PROC  NEAR
        MOV    AH, 2
        MOV    DL, 'A'                              ;  display a letter A
               INT
               RET
ONE ENDP
```

```
TWO PROC NEAR
            MOV     AH, 2
            MOV   DL, 'B'                                           ;  display letter B
                       INT
           RET
TWO          ENDP

;  THREE PROC  NEAR
MOV AH, 2                          ;  display letter C
MOV  DL, 'C'
           INT
           RET
THREE ENDP



;   Start of Main Program
            .STARTUP
TOP:
  MOV  AH, 1                       ;  read key into AL
                       INT
  SUB    AL, 31H                    ;  convert from ASCII to 0,  1,  or 2
  JB
 CMP   AL, 2
JA TOP                             ;  if above 2
 MOV  AH, 0             lookup address

       MOV
ADD          BX, BX

                 TABLE [BX]      ;  call procedure ONE,  TWO,  or THREE
                                              exit to;DOS
.EXIT
END                                           endof;file

CALL  TABLE [BX]        ;  call procedure ONE,  TWO,  or THREE
                           exit to;DOS
                           endof;file
```

OBSERVATIONS

By using single stepping observe the contents of registers AX, BX, CX, DX

EXERCISE

Write a program, which produce a beep after a certain time delay. The program repeats itself 10 times. Produce the delay using a subroutine. Also draw the flow chart of the program. (use the onboard buzzer to produce a beep).