Computer System Engineering Department

ENCS 511

Computer Lab

Section No: 1

Report for Experiment No.6

Interrupts

Prepared for

Dr. EMAD HAMADA

ENG. Anas Abdelraziq

By

MU'EZ ZABEN 1061287

MAHMOUD QUR’AN 1060095

Procedure:

A. Activating the interrupt service routine automatically:

To check the interrupt that relates to the error "Division on zero" which is an example of the interrupts ,we used the commands like that on figure (1) :

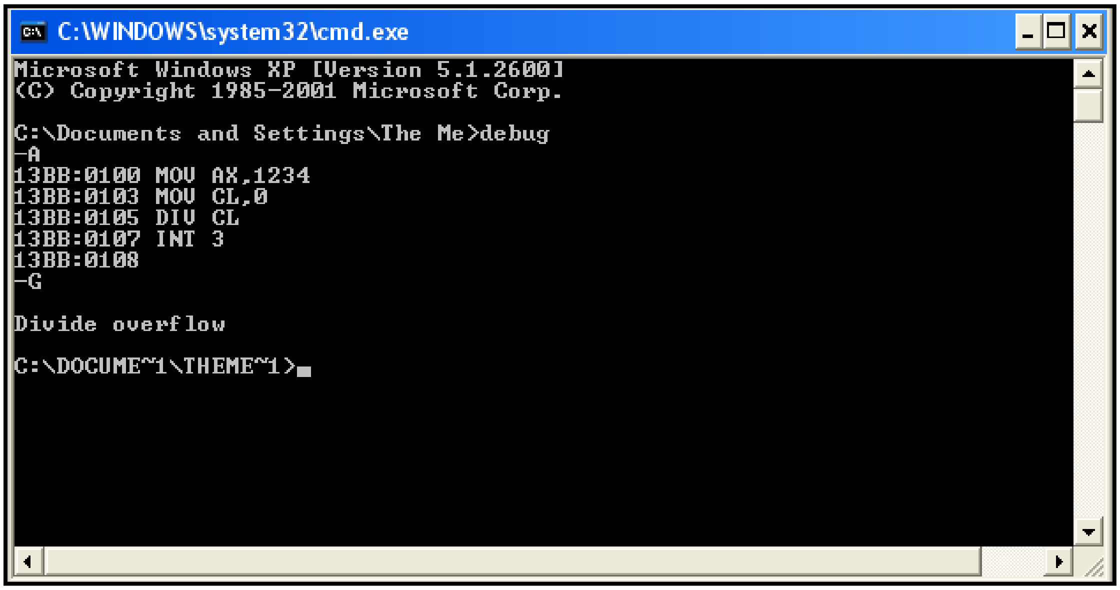

Figure 1

When we executed the command by the order "G" the output was "Divide overflow", because in the code we divided some number on zero .

B.Activating the interrupt service routine manually:

We can test the interrupts each one by it self by typing it then using interrupt 3 as a break point, like what we did in figure (2) :

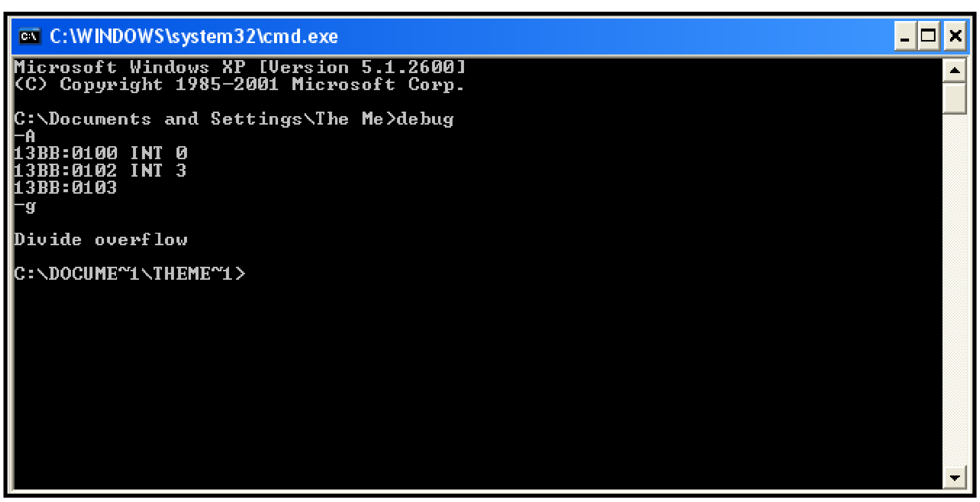

Figure 2

C.Activating the interrupt service routine manually (another way):

Another way to use the interrupt is to call it by it's IP and CS ,which are in the figure (3) :

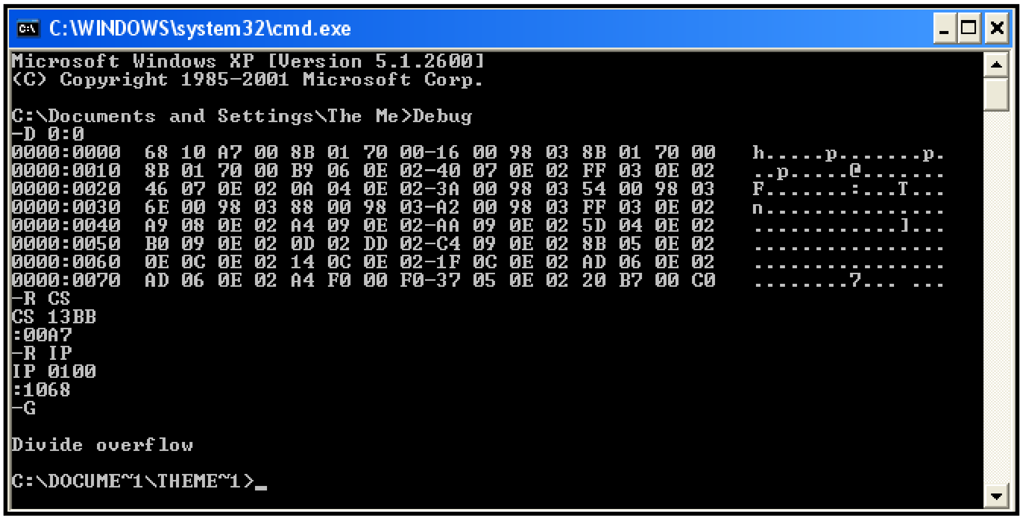

Figure 3

To call an interrupt we must inter it's IP and CS as shown in figure (3)m then we can execute it by the order "G", **and the address on INT F in IVT is (F*4)H=(60)H=3CH.**

D. Writing my own interrupt service routine :

We can define any interrupt that we want .

In this part we changed the message that appears when interrupt zero executes .

At first the message which was "AAAA" was defined ,then we relate the interrupt zero which we know it's IP and CS to the IP and CS of the message "AAAA" then we run some code that had the error that relates to interrupt zero as shown in figure (4) :

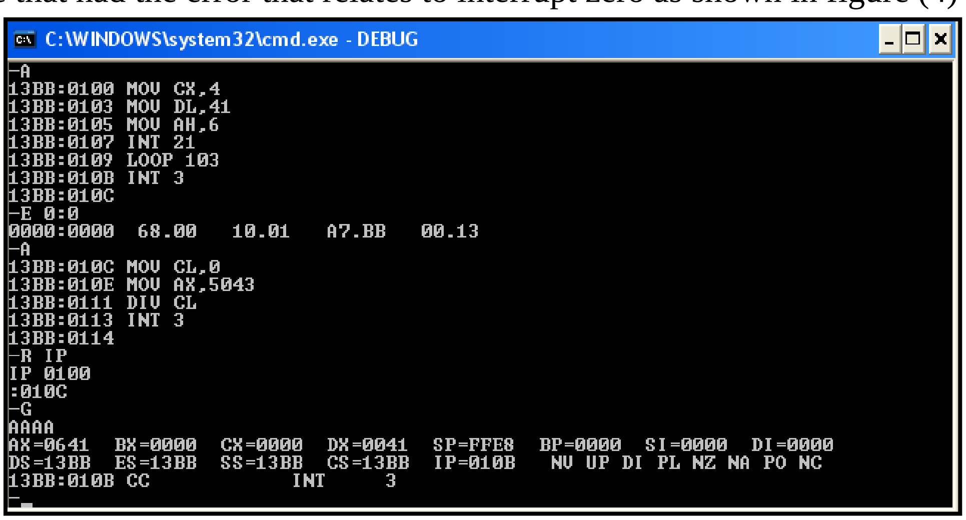

Figure 4

E-Installing a interrupt service routine using TASM

On this example, a new interrupt service routine (62) was defined, but to understand this code we went back to interrupt 21H function that depends on the value stored on the AH register, on this code we need the following function numbers which are shown on table.1, which was taken from reference[2].

Int 10H	**Function 02H "select cursor position"**
Entry	AH = 02H BH=page number(usually 0) DH =row number(beginning with zero) DL= column number(beginning with zero)
Exit	Change cursor to new position
Int 10H	**Function 09H " write attribute character/at current cursor position"**
Entry	AH =09H AL=ASCI character code BH=page number BL= character attribute CX=number of characters to write
notes	This function call normally displays data on the video display
Int 21H	**Function 25H " set interrupt vector"**
Entry	AH =25H AL= interrupt vector number DS:DX address of new interrupt procedure
Int 21H	**Function 31 H "Terminate and stay resident TSR"**
Entry	AH =33H AL = The DOS return code BH = Number of paragraphs to reserve for program

Table 1

- The code of figure. code was saved as 2.asm, and then executed by writing the following command on the ms-dos window:

C:\>**TASM MATRIX_2.ASM**; open tasm assembler
C:\>**TLINK MATRIX_2.OBJ**; convert from .asm to .exe
C:\>**exe2bin MATRIX_2.EXE**; convert from.exe to bin
C:\>**bin2hex MATRIX_2.BIN MATRIX_1.HEX**; convert from bin to hex

```
.MODEL TINY
.CODE
.STACK 100h; set CS=100h
MOV BX,OFFSET MESS     ; get offset address of the messege
MOV DL,20

;*****////////*******

L1:
MOV AL,CS:[BX]; move the 1st charachter of the string messege into AL register
CMP AL,'$';
JZ L2

;*****////////*******

PUSH BX ;WE WILL NEED IT
MOV BH,0 ; 10H, AH=2. set crusure position
MOV AH,2
MOV DH,20; row number
INC DL; column number
INT 10H

;*****////////*******

MOV AH,9H; write attribute character/at current cursor position, INT 10 FUNCTION AH=9
MOV BH,0; page number
MOV BL,84H; charachter attribute
MOV CX,1;number of charachters to write
INT 10H
POP BX; get back the value of BX
INC BX; to point to 2nd charachter
JMP INSTALL

;*****////////*******

MESS DB "WELCOME TO 511 LAB $"; the messege we want to display
MYINT PROC
JMP L1
L2:
IRET
MYINT ENDP

;*****////////*******
```

```
;INSTALL NEW INTERRUPT VECTOR.

INSTALL:
MOV AH, 25H ; INT 21H WITH AH =25 FUNCTION :SET VECTOR
MOV AL, 62H ; interrupt vector number
MOV DX, CS   ;DS:DX is the address of the interrupt number
MOV DS, DX
MOV DX, OFFSET MYINT
INT 21H; call interrupt 21H to do the specified function set vector table

; INT 21H WITH AH =31H FUNCTION :Terminate and stay resident.
MOV AX, 3100H
INT 21H
END
```

Figure 5

After that we write the following to execute it following these steps:

C:\>**TASM MATRIX_3.ASM**; open tasm assembler
C:\>**TLINK MATRIX_3.OBJ**; convert from .asm to .exe

And the result was that it prints the string "WECLOME TO 511 LAB" in red color, this is what happens on the lab, but when I try this at home, it prints only the character W, although I used the same code, I tried to find error, but I didn't find.

- On the next part we have written the following code and then executed it, and the result was the same as the previous code.

```
.model small
.stack 100
.code
int 62h
mov ah,4ch
int 21h
end
```

PARTB:

- **User defined interrupt:**

On the first part of this experiment, the interrupt 21H was adjusted to do a summation process instead of its original function, this is done by defining our special routine that add the content of two register, this was done by storing this ISR address on the IVT corresponding to INT 21H, so when we call it, it will point to our code.

The code is shown below on figure.1; the comments clarify how this code works.

```
CODE     SEGMENT
     ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
     ;
V_TAB    EQU 21H*4  ;addrees locatio of int21h ISR  on the IVT
SEG_D    EQU 0000H  ; initialize segment
     ;
     ORG 1000H;
     MOV AX,SEG_D
     MOV DS,AX     ; initialize DS
     MOV BX,V_TAB
     MOV AX,OFFSET INT_SER

     ;****** move the adreess of INT_SER to 21H*4 and replce it with the original ISR of int 21H****

     MOV WORD PTR [BX],AX; move offset address (IP)  to 1st 2 bytes of IVT 21h*4 address[1]
     ;
     INC BX
     INC BX  ; increment by 2 to store CS value since IP takes 2 bytes
     ;
     MOV DX,0
     MOV WORD PTR [BX],Dx;move the base address (CS)to to IVT 21h*4 addressto 2nd 2 bytes of IVT 21h*4 address[1]
     ;
     MOV AX,1234H;insert ist number
     MOV BX,6789H;insert second number
     INT 21H; call int 21h to add thes numbers
     NOP
     NOP
     INT 3; terminate and  stop execution
     ;
INT_SER:  ADD     AX,BX; routine to add 2 numbers stored in AX,BX
     IRET
     ;
CODE     ENDS
     END
```

Figure 6

- This code was saved as 1.asm, and then executed by writing the following command on the ms-dos window:

C:\>**TASM MATRIX_1.ASM**; open tasm assembler
C:\>**TLINK MATRIX_1.OBJ**; convert from .asm to .exe
C:\>**exe2bin MATRIX_1.EXE**; convert from.exe to bin
C:\>**bin2hex MATRIX_1.BIN MATRIX_1.HEX**; convert from bin to hex

After that, the WinCom program was opened, and the following instructions were follows:
L command was typed, after that we go to File >> send program and then choosing the hex file1.hex the G command was typed and the program started.

- **The result was that it added the content of AX and BX and stored the result in AX.**

Note: this procedure of compilation was repeated for all parts of the experiments

- 8259A INTERRUPT CONTROL:

In this part of the experiment, we will use PIC 8259a in a program that controls the lighting sequence of 4 LED's ,but befor that we were ordered in the **prelab** to review the intel 8259a PIC and do the following:

1. **what are the Modes of operation for 8259a?**

a) **Fully nested mode: (it's the default mode)**

IR0 has highest priority and IR7 is the lowest.[1]

b) **Rotating priority mode.**

c) **Special masked mode.**
xddc

d) **Polled mode: (this mode will be used in the experiment)**

the INT output is not used, the μp checks the status of interrupt request by issuing a poll command, the microprocessor reads content of 8259A after issuing poll command, during this read operation, the 8259A provides polled word and sets ISR bits of highest priority active interrupt request format.[1].

2. **What would be the I/O ports for the 8259 if direct addressing mode is used with only 8086 A4 being "1" and 8086 A1 being connected to A0 of 8259?**

Direct I/O address → 8 bit address, A1 is connected to A0 of 8259

A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	1	0	0	0/1	0

Address of I/O ports **10H** and **12H**.

3. Study the To Do items and write down the values for ICW1, ICW2, and ICW4?

In all to do in the experiment, the 8259 initialized on the same way, the data sheet we used to set the ICW's is reference [2 which is from Intel data sheet the information was given to us was:

The 8259 is initialized with the following features:

1. ICW4 is needed---ICW1
2. Edge triggered mode---ICW1
3. An address interval of 8----ICW1
4. Single mode---ICW1
5. Interrupt vector of 40H ----ICW2
6. Normal end of interrupt---- ICW4 -D1
7. Non-buffered mode---ICW4 - D3
8. Not SFNM ----ICW4

ICW1 = 13H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	0	0	1	1

ICW2 = 40H

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	0	0	0	0	0

ICW4 =01H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1

Note: ICW3 is note needed since there is no cascaded 8259a's.

4. What values of OCWs are needed?

OCW1 = FBH

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	1	0	1	1

OCW2 = 20

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	0	0	0	0

OCW3 = 0CH

D7	D6	D5	D4	D3	D2	D1	D0
0	0	**0**	0	1	1	0	0

the interface of 8289a on the system:

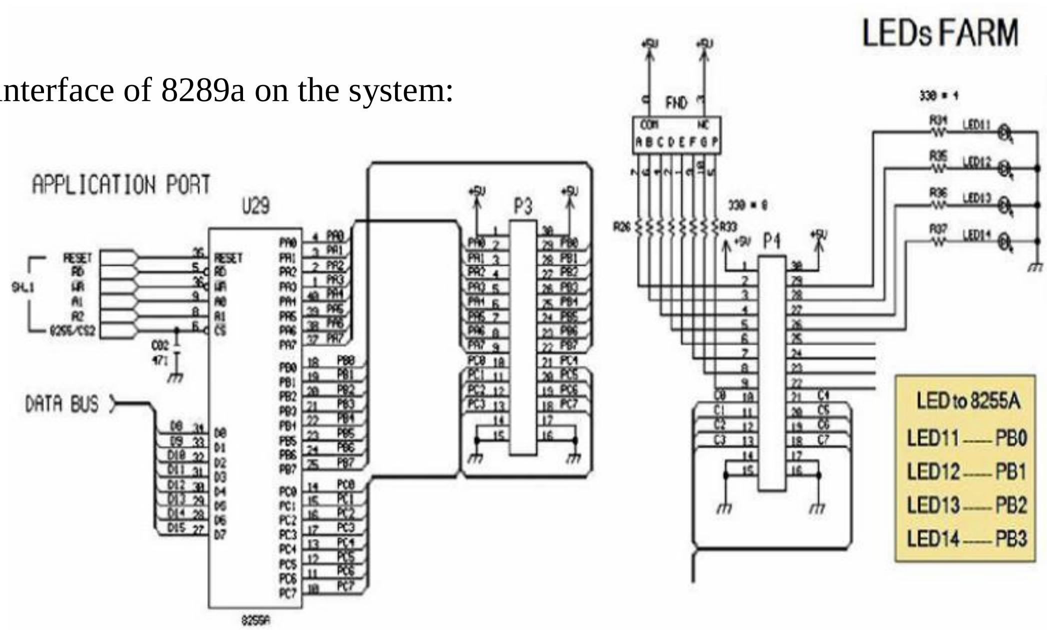

Figure 8

The LED's is connected to ports PB0-PB3, as shown on figure.2, so we must use PPI port B to send data, and the others are in active.

Note: it's observed that IR2 pin is connected to a push button to generate an interrupt.

After understanding the interface of 8259 and 8255 on the system, and after writhing the appropriate command words and knowing its address, the code below can be completed and under stood well.

The comments clarify how this code works.

```
CODE    SEGMENT
ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ;
PPIC_C      EQU 1FH     ;control word address of the PPI
PPIC        EQU 1DH     ;port C address
PPIB        EQU 1BH     ;port B address
PPIA        EQU 19H     ;port A address
        ;
INTA        EQU 10h     ;1st address of PIC 8259 command word
INTA2       EQU 12h     ;2nd address of 8259 command word
        ;
        ORG 1000H
        ;
        CALL    INIT            ; go to the procedure that initializes
                                ; the command word of the 8259
        ;
        MOV AL,10000000B
        OUT PPIC_C,AL           ; define all the ports as output ports
        ;
        MOV AL,11111111B        ;disable port A , not used
        OUT PPIA,AL;
        ;
        MOV AL,00000000B        ;disable port C, not used
        OUT PPIC,AL
        ;
        MOV AH,11110001B
        MOV AL,AH
        OUT PPIB,AL             ; enable the 1st LED by sending 1 to PB0
        ;
L2:     MOV AL, 0ch             ; Enable Poll command on interrupts (OCW3)
        OUT INTA,AL             ;this is explained on the prelab at the 1st of the report
        IN  AL,INTA             ;read the status of 8259, this is usually done after making OCW3 in polled mode
        TEST AL, 10000000B      ;from intel 8259 data sheet D7 mus be 1 wheh there is an interrupt request
        JZ  L2                  ;keep loop here while there is not interrupt
```

```
    ;
    SHL AH,1                  ; shift the on bit to the left to light the next LED
    TEST    AH,00010000B  ;keep shift until rech portPB5
    JNZ L1                    ; if shifted bit reach PB4 go to L1 to reset the coun again and start fromPB0
    OR  AH,11110000B          ;because bits of PB4-PB7 must be 1's since they are not connected
    JMP L3                    ;LED out

L1: MOV AH,11110001B
L3: MOV AL,AH
    OUT PPIB,AL             ; send the shifted value to PPB to light next LED
                            ;EOI command
    MOV AL,20H              ;send non-specific EOI (OCW2)
    OUT INTA,AL             ; interrupt request must be reset
    JMP L2
    ;
INIT    PROC    NEAR      ;this  procedure initialize the 8259
    MOV AL,13H            ;send the value of ICW1
    OUT INTA , AL

    MOV AL,40H            ;send the value  ICW2 interrupt vector
    OUT INTA2 , AL

    MOV AL,01H            ;send the value of ICW4
    OUT INTA2 , AL
    MOV AL,00000100B      ;interrupt mask, since IR2 is connected to  the push button which creats the interrupt
    OUT INTA ,AL
    RET
INIT    ENDP
    ;
CODE    ENDS
    END
```

Figure 9

After writing this code, it was compiled following the same steps of the previous code, but the MDA 8086 kit is turned on, after that it's observed that when we pressed the push button, the next LED is on, and after when LED#4 is on, LED #1 is on …and so forth, and so 8259a used to control the lighting sequence, through a software interrupts.

Note: although the interrupt is generated by push button, its considered a software interrupt, because within the code the 8259 is programmed in polled mode, i.e.it checks all the pins until an interrupt occurs at one of these pins, so the software controls the interrupts.

- On the next part, a program will be written to control the count of a seven segment display from 0-9 using a push button also.

The code is shown below, and the comments clarify how this code works.

```
CODE    SEGMENT
ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
        ;
PPIC_C        EQU 1FH        ;control word address of the PPI
PPIC          EQU 1DH        ;port C address
PPIB          EQU 1BH        ;port B address
PPIA          EQU 19H        ;port A address
        ;
INTA          EQU 10h        ;1st address of PIC 8259 command word
INTA2         EQU 12h        ;2nd address of 8259 command word
        ;
INT_V   EQU 42H*4            ;adderss of  INT 42H ISR on the IVT
        ;
STACK   EQU 540H             ; since we need apermanent storage...we define our stack
        ;
        ORG 1000H
        ;
        XOR BX,BX            ;set BX to zero, Xor is used because it requiers less time than mov operatoin
        MOV ES,BX            ;set ES to zero, Xor is invalid for segment regisiters
        MOV DS,AX            ;set DS to zero, Xor is invalid for segment regisiters
        MOV SS,BX            ;set SS to zero, Xor is invalid for segment regisiters
        MOV SP,STACK         ; make the stack pointer SP point to the top of the stack i.e the starting of the stack
        ;

        ;************here we will define the address of the ISR in the memory to locate it  wehen calling interrept 42H***


        MOV AX,OFFSET INT_SER    ;get the offset address of the routine (porcedure) that we want INT 42H to perform
        MOV BX,INT_V             ;move the caculated address of INT 42H to BX
        MOV WORD PTR ES:[BX],AX  ;store the offset address ((IP)) of the INT 42H ISR on the location
                                 ; pointed by  the content of BX (( 42H*4)) which is usually done
                                 ;when we call an interrupt
        ;
        XOR AX,AX                    ; reset AX
        MOV WORD PTR ES:[BX+2],AX    ;stor the base address ((CS=0 in this code))
        ;*********************************************************************************************************
        CALL INIT                    ; call the procedure that initializes the PIC 8259a

        MOV AL,80H
        OUT PPIC_C,AL                ; set all PPI ports as output ports

        MOV AL,0F0H
        OUT PPIB,AL                  ;turn off PB0-PB3 which connected to LED we dont need them.
        MOV AL,00H
        OUT PPIC,AL                  ;turn off port C
        ;
        MOV SI,OFFSET DATA       ;get offset address of DATA , SI is used as apointer to this array
```

```
    MOV AL,BYTE PTR CS:[SI]; point to the array that contains the code for #'s from 0-9 on 7-seg display

    OUT PPIA,AL; send the values to port A which is connected to 7-seg display
    ;
    STI ; set interrupt flag
L2: NOP
    JMP L2          ; stay in this loop until interrupt came
    ;
    ;
INT_SER:
    MOV AL,BYTE PTR CS:[SI]; move the 7-seg code, addresses by Cs and the
                            ;content of SI to AL
    CMP AL,00H              ;keep in moving data until reach #9 then we must stop and count again
                            ;so we copare the last element of the array with it self then we repeat again
    JNE L3
    MOV SI,OFFSET DATA
    JMP INT_SER

L3: OUT PPIA,AL ; send the code to the 7-seg through port A
    INC SI; increment SI to point to next code
    ;

    ; ****EOI command****
    ;
    MOV AL,00100000B        ;send non-specific EOI (OCW2)
    OUT INTA,AL             ; interrupt request must be reset
    STI
    IRET     ; return t
    ;
INIT    PROC    NEAR

    MOV AL,13H  ;this procedure initialize the 8259
    MOV AL,13H  ;this procedure initialize the 8259
    OUT INTA,AL         ;send the value of ICW1

    MOV AL,40H          ;send the value  ICW2 interrupt vector
    OUT INTA2,AL

    MOV AL,01H          ;send the value of ICW4
    OUT INTA2,AL

    MOV AL,0FBH         ;interrupt mask bet #3 = 0, since IR2 is connected to  the push button which creats the interrupt
    OUT INTA2,AL
    ;interrupt mask

    RET
INIT    ENDP
    ;

DATA: DB 11000000B;#0 on seven segment display
      DB 11111001B;#1 on seven segment display
      DB 10100100B;#2 on seven segment display
      DB 10110000B;#3 on seven segment display
      DB 10011001B;#4 on seven segment display
      DB 10010010B;#5 on seven segment display
      DB 10000010B;#6 on seven segment display
      DB 11111000B;#7 on seven segment display
      DB 10000000B;#8 on seven segment display
      DB 10010000B;#9 on seven segment display
      DB 00H        ; all segmnets are on
CODE    ENDS
    END
```

Figure 10

After writing this code, it was compiled following the same steps of the previous code, after that it's observed that when we pressed the push button, the seven segment display will be incremented until it reached 9 it will go to zero again, and so 8259a used to control the lighting sequence, through software interrupts.

- After that we were ordered to write a C code to implement the same previous function but, it will count from A-F, the code is shown below, and the comments clarified how this code works.

```
#include        "mde8086.h"

#define         INT_V    0x42

int     data[6] = {0x88,0xff,0xc6,0xc0,0x86,0x8c};/*data to view chrachters from A-F respectively */
int     index = 0;

void    wait(long del)/* time delay*/
{
    while( del-- );
}


/* Process Interrupt Routine */
void    int_ser(void)
{
    INTERRUPT_IN;

    index ++;
    if( index >= 6  )  index = 0;/* increment until  reach the sixth charachter F then reset and start from A*/
    outportb( PPI1_A, data[index] );

    /* eoi command */
    outportb( INTA, 0x20);/*send non-specific EOI (OCW2)*/
                          /* interrupt request must be reset*/

    /*restor the values of regesters after finishing interrupts because program mus return to routine befor interrupt*/
    asm  pop  ds;
    asm  pop  es;
    asm  pop  dx;
    asm  pop  cx;
    asm  pop  bx;
    asm  pop  ax;
    asm  pop  di;
    asm  pop  si;
    asm  iret;

}


void    main(void)
{
    unsigned long far *intvect_ptr;

    intvect_ptr = ((unsigned long far *)0);

    /* Init 8259 */
    asm CLI;/* clear interrupt flag*/
```

```
    outportb( INTA, 0x13 );         /* ICW1            */
    outportb( INTA2, 0x40 );        /* ICW2 interrupt Vector */
    outportb( INTA2, 0x01 );        /* ICW4            */
    outportb( INTA2, 0xFB );        /* interrupt mask     */

    /* 8255 Initial */
    outportb( PPI1_CR, 0x80 );/*send control word all ports are output ports*/
    outportb( PPI1_B, 0xF0 );/* turn off port B*/
    outportb( PPI1_C, 0x00 );/*not used...turn off*/
    outportb( PPI1_A, 0x00);/* */

    /* Define Interrupt Vector Table */
    *(intvect_ptr+INT_V) = ( unsigned long )int_ser;

    asm     STI;/* set interrupt flag*/

    while(1);
}
```

Figure 11

- After writhing the previous code ,we choose send a program from wincom program, and the file 1.c was located after insuring that we are on the "C code" mode not " assembly 8086", these steps will generate the exe file, after that we used the exe2bin and bin2hex soft ware to get the hex file by writing the following command on the MS-DOS window:

  C:\>exe2bin 1.EXE; convert from.exe to bin
  C:\>bin2hex 1.BIN 1.HEX; convert from bin to hex

  Then the code was downloaded to the board using WinCom program, and it's observed that the 7 segment counts from A-F successfully.

Note: I lost the code that I was did on the lab, so there might be errors that I couldn't check

Conclusion:

- In this experiment, we introduced the two types of Interrupts; Hardware and software interrupts and there characteristics.

- Hardware interrupts are not included in this excrement, but they occurre in fault cases such as power failure

- We learned how to make a pre- defined interrupt to do another function that we want it to do, by simply exchanging the address stored on the IVT with our routine address.

- The system interrupts are loaded by the OS at the beginning of the program, and the address of this IVT is the same to all computers.

- In this experiment PIC 8259A was used as a controller for simple functions like lighting LED or incrementing 7 segment displays.

- Soft ware polling minimizes time to handle an interrupt, especially in case when the CPU is executing a simple function, and has no other thing to do like what happens on this experiment.

References:

[1]MICROPROCESSORS, GODSE

[2]THE INTEL MICROPROCESSOR, BARRY BRAY