Laboratory Report

Laboratory Exercise No.:	7	**Date Performed:**	11/19/2022
Laboratory Exercise Title:	Parallel I/O Interfacing		
Name of Student:	Paul John Toral	**Document Version:**	1.0

Activity #1

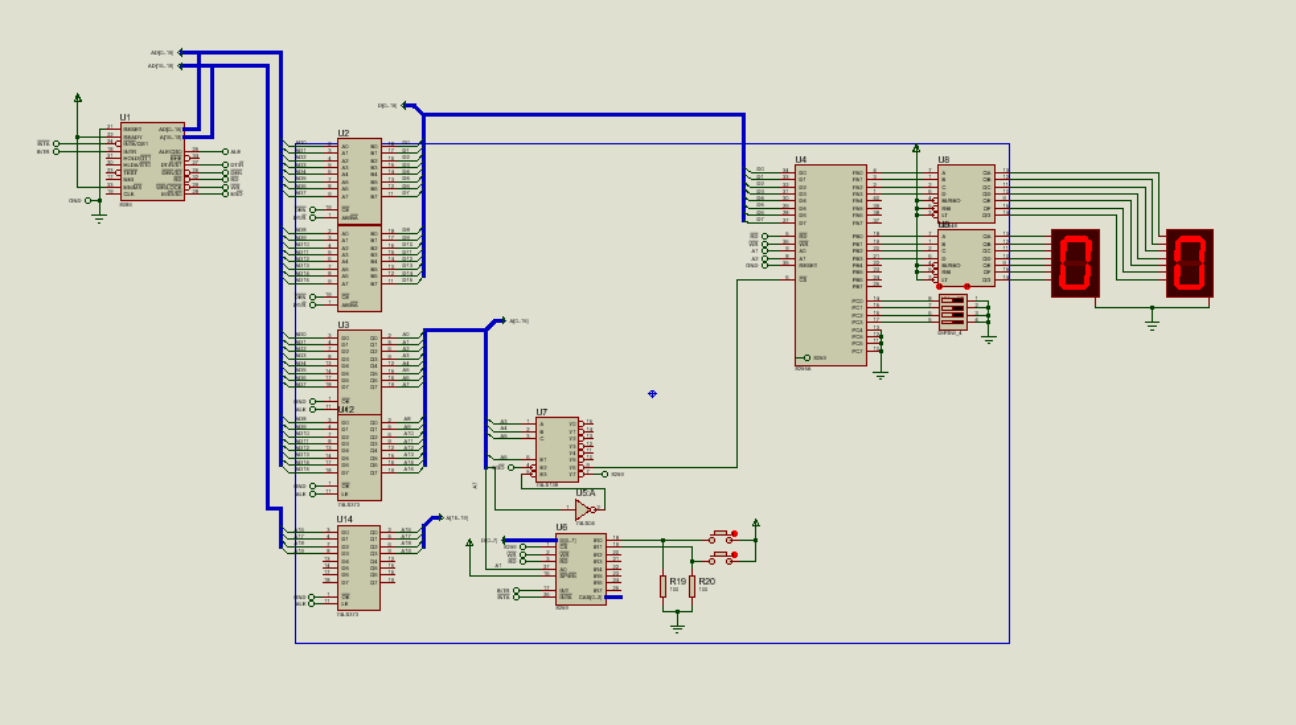

Figure 1: Schematic Diagram for Activity 1

ICW1 = 00010011b

1; A7-A0 = 000; 1 ; edge triggered=0 ; interval of 8 = 0 ; single = 1 ; ICW4 needed = 1

ICW2 = 10000000b

8086 vector address 80h-87h = 10000000b

ICW4 = 00000011b

000 ; not specially fully nested = 0 ; non buffered mode = 00 ; auto EOI = 1 ; 8086 mode = 1

OCW1 = 11111100b

Mask only D1 and D0 = 11111100b

Activity #2

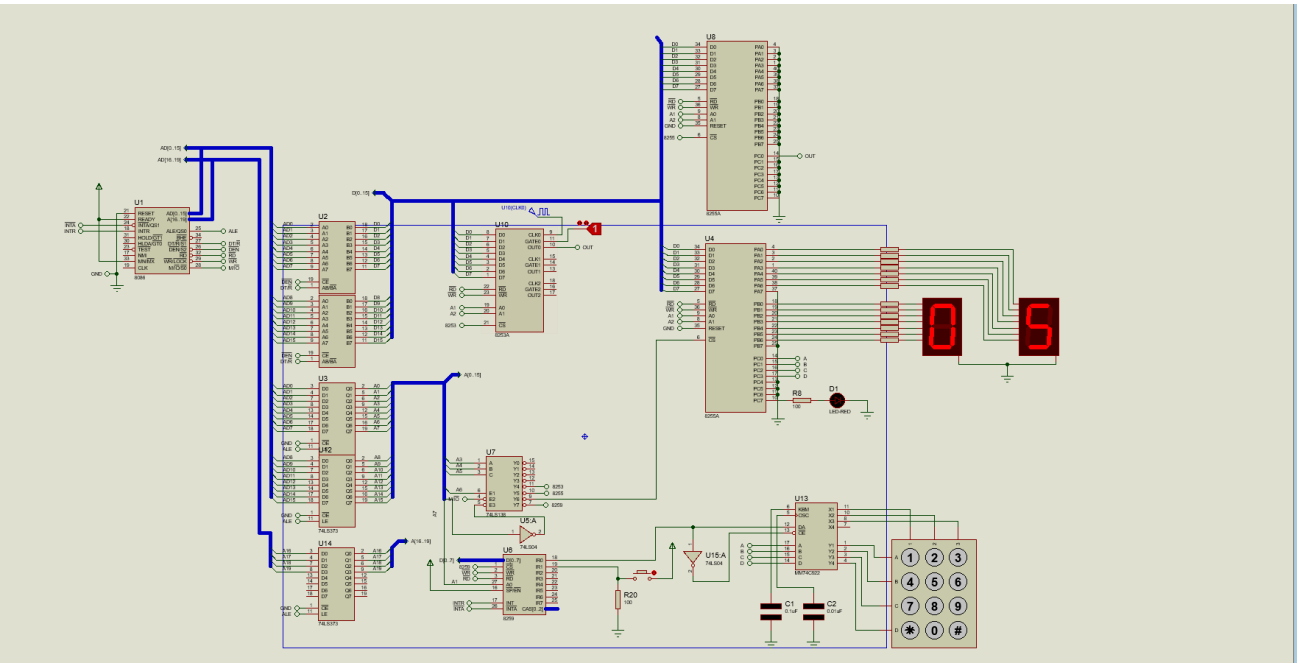

Figure 2: Schematic Diagram for Activity 2

Why do you think the LED is blinking steadily while other activities are going on?

Interrupts allow us to utilize and execute functions while a background process is being run which is why it adds to the efficiency of the whole circuitry. Added to this, there isn't any other functions that actually hinder the LED from blinking steadily.

What do you think is the ultimate advantage of using interrupts especially involving I/O devices?

The CPU may be used more effectively and under better control thanks to interruptions. Interrupts circumvent this while involving I/O or even other higher priority activities and prevent the CPU from becoming idle. This guarantees that the interruption will be handled right away.

Source Code for Activity #1

```
PROCED1 SEGMENT
   ISR1 PROC FAR
      ASSUME CS:PROCED1, DS:DATA
      ORG 01000H ; write code within below starting at address 08000H
            PUSHF ; push 16-bit operands
            PUSH AX ; save program context
            PUSH DX
           ; <write the ISR code here>
              MOV DX, PORTA
              MOV AL, 00001001b
              OUT DX, AL
            POP DX ; retrieve program context
            POP AX
            POPF ; pop 16-bit operands
            IRET ; return from interrupt
   ISR1 ENDP ; end of procedure
 PROCED1 ENDS

 PROCED2 SEGMENT
   ISR2 PROC FAR
      ASSUME CS:PROCED2, DS:DATA
      ORG 02000H ; write code within below starting at address 09000H
            PUSHF ; push 16-bit operands
            PUSH AX ; save program context
            PUSH DX
           ; <write the ISR code here>
              MOV DX, PORTA
              MOV AL, 00000000B
              OUT DX, AL
            POP DX ; retrieve program context
            POP AX
            POPF ; pop 16-bit operands
            IRET ; return from interrupt
   ISR2 ENDP ; end of procedure
 PROCED2 ENDS
```

```
DATA SEGMENT
ORG 03000H
  PORTA EQU 0F0H ; PORTA address
  PORTB EQU 0F2H ; PORTB address
  PORTC EQU 0F4H ; PORTC address
  COM_REG EQU 0F6H      ; Command Register Address
  PIC1 EQU 0F8H     ; A1 = 0
  PIC2 EQU 0FAH     ; A1 = 1
  ICW1 EQU 13H      ; refer to #4
  ICW2 EQU 80H      ; refer to #4
  ICW4 EQU 03H      ; refer to #4
  OCW1 EQU 0FCH  ; refer to #4
DATA ENDS

STK SEGMENT STACK
  BOS DW 64d DUP(?) ; stack depth (bottom of stack)
  TOS LABEL WORD ; top of stack
STK ENDS

CODE SEGMENT PUBLIC 'CODE'
        ASSUME CS:CODE, DS:DATA, SS:STK

        ORG 04000H ; write code within below starting at address 0E000H

START:
  MOV AX, DATA
  MOV DS, AX ; set the Data Segment address
  MOV AX, STK
  MOV SS, AX ; set the Stack Segment address
  LEA SP, TOS ; set address of SP as top of stack

  CLI ; clears IF flag
  ;program the 8255
  MOV DX, COM_REG        ; set port to Command Register
  MOV AL, 10001001B      ; set command byte
```

```
OUT DX, AL              ; send data in AL to Command Register
MOV DX, PORTA
MOV AL, 0000000B
OUT DX, AL
;program the 8259
MOV DX, PIC1      ; set I/O address to access ICW1
MOV AL, ICW1
OUT DX, AL              ; send command word
MOV DX, PIC2      ; set I/O address to access ICW2,ICW4 and OCW1
MOV AL, ICW2
OUT DX, AL              ; send command word
MOV AL, ICW4
OUT DX, AL              ; send command word
MOV AL, OCW1
OUT DX, AL              ; send command word
STI               ; enable INTR pin of 8086
;storing interrupt vector to interrupt vector table in memory
MOV AX, OFFSET ISR1         ; get offset address of ISR1 (IP)
MOV [ES:200H], AX           ; store offset address to memory at 200H
MOV AX, SEG ISR1            ; get segment address of ISR1 (CS)
MOV [ES:202H], AX           ; store segment address to memory at 202H

MOV AX, OFFSET ISR2         ; get offset address of ISR2 (IP)
MOV [ES:204H], AX           ; store offset address to memory at 204H
MOV AX, SEG ISR2            ; get segment address of ISR2 (CS)
MOV [ES:206H], AX           ; store segment address to memory at 206H

;foreground routine
HERE:
  MOV DX, PORTC ;
  IN AL, DX
  AND AL, 0FH

  CMP AL, 09H
  JLE DISP
  MOV AL, 00H
```

```
  DISP:
    MOV DX, PORTB
    MOV AH, 00H
    MOV SI, AX
    MOV AL, S_COUNT[SI]
    OUT DX, AL

  JMP HERE

; 0-9 array for 7 segment display
S_COUNT DB 00000000B;0
        DB 00000001B;1
        DB 00000010B;2
        DB 00000011B;3
        DB 00000100B;4
        DB 00000101B;5
        DB 00000110B;6
        DB 00000111B;7
        DB 00001000B;8
        DB 00001001B;9


CODE ENDS
END START
```

Source Code for Activity #2

```
PROCED1 SEGMENT
   ISR1 PROC FAR
     ASSUME CS:PROCED1, DS:DATA
     ORG 01000H ; write code within below starting at address 08000H
          PUSHF ; push 16-bit operands
          PUSH AX ; save program context
          PUSH DX
         ; <write the ISR code here>
                    XOR AX, AX
                    MOV DX, PORTC
                    IN AL, DX

                    AND AL, 0FH
                    CMP AL, 0CH              ; check if key pressed is *
                     JE PRNT_AST
                    CMP AL, 0EH              ; check if key pressed is #
                    JE PRNT_OCTO
                    CMP AL, 0DH              ; check if key pressed is 0
                    JE PRNT_0
                    CMP AL, 00H              ; check if key pressed is 1
                    JE PRNT_1
                    CMP AL, 01H              ; check if key pressed is 2
                    JE PRNT_2
                    CMP AL, 02H              ; check if key pressed is 3
                    JE PRNT_3
                    CMP AL, 04H              ; check if key pressed is 4
                    JE PRNT_4
                    CMP AL, 05H              ; check if key pressed is 5
                    JE PRNT_5
                    CMP AL, 06H              ; check if key pressed is 6
                    JE PRNT_6
                    CMP AL, 08H              ; check if key pressed is 7
                    JE PRNT_7
                    CMP AL, 09H              ; check if key pressed is 8
```

```
JE PRNT_8
CMP AL, 0AH                ; check if key pressed is 9
JE PRNT_9

        PRNT_AST:
          MOV DX, PORTA
          MOV AL,  01000000B
          JMP ENDPRINT
        PRNT_OCTO:
          MOV DX, PORTA
          MOV AL,  01000000B
          JMP ENDPRINT
        PRNT_0:
          MOV DX, PORTA
          MOV AL, 00111111B      ;0
          JMP ENDPRINT
        PRNT_1:
          MOV DX, PORTA
          MOV AL, 00000110B      ;1
          JMP ENDPRINT
        PRNT_2:
          MOV DX, PORTA
          MOV AL, 01011011B      ;2
          JMP ENDPRINT
        PRNT_3:
          MOV DX, PORTA
          MOV AL, 01001111B      ;3
          JMP ENDPRINT
        PRNT_4:
          MOV DX, PORTA
          MOV AL, 01100110B      ;4
          JMP ENDPRINT
        PRNT_5:
          MOV DX, PORTA
          MOV AL, 01101101B      ;5
          JMP ENDPRINT
```

```
                PRNT_6:
                  MOV DX, PORTA
                  MOV AL, 01111101B      ;6
                  JMP ENDPRINT
                PRNT_7:
                  MOV DX, PORTA
                  MOV AL, 00000111B      ;7
                  JMP ENDPRINT
                PRNT_8:
                  MOV DX, PORTA
                  MOV AL,  01111111B     ;8
                  JMP ENDPRINT
                PRNT_9:
                  MOV DX, PORTA
                  MOV AL, 01100111B      ;9
                  JMP ENDPRINT

        ENDPRINT:
        MOV NUM, AL
        OUT DX, AL
        POP DX ; retrieve program context
        POP AX
        POPF ; pop 16-bit operands
        IRET ; return from interrupt
  ISR1 ENDP ; end of procedure
PROCED1 ENDS

PROCED2 SEGMENT
  ISR2 PROC FAR
    ASSUME CS:PROCED2, DS:DATA
    ORG 02000H ; write code within below starting at address 09000H
        PUSHF ; push 16-bit operands
        PUSH AX ; save program context
        PUSH DX
       ; <write the ISR code here>
         MOV DX, PORTB
```

```
            MOV AL, NUM
            OUT DX, AL
          POP DX ; retrieve program context
          POP AX
          POPF ; pop 16-bit operands
          IRET ; return from interrupt
   ISR2 ENDP ; end of procedure
PROCED2 ENDS

DATA SEGMENT
ORG 03000H
  PORTA EQU 0F0H ; PORTA address
  PORTB EQU 0F2H ; PORTB address
  PORTC EQU 0F4H ; PORTC address
  COM_REG EQU 0F6H      ; Command Register Address
  PORTA2 EQU 0E8H       ; PORTA address
  PORTB2 EQU 0EAH       ; PORTB address
  PORTC2 EQU 0ECH       ; PORTC address
  COM_REG2 EQU 0EEH     ; Command Register Address
  PIC1 EQU 0F8H     ; A1 = 0
  PIC2 EQU 0FAH     ; A1 = 1
  ICW1 EQU 13H      ; refer to #4
  ICW2 EQU 80H      ; refer to #4
  ICW4 EQU 03H      ; refer to #4
  OCW1 EQU 0FCH  ; refer to #4
  LOAD_CTR0 EQU  0E0H   ; counter 0 address
  TIMER_REG EQU 0E6H    ; 8253 Command register address
  LED dw 0
  NUM DB 00111111B      ;0
DATA ENDS

STK SEGMENT STACK
  BOS DW 64d DUP(?) ; stack depth (bottom of stack)
  TOS LABEL WORD ; top of stack
STK ENDS
```

```
CODE SEGMENT PUBLIC 'CODE'
        ASSUME CS:CODE, DS:DATA, SS:STK

        ORG 04000H ; write code within below starting at address 0E000H

START:
  MOV AX, DATA
  MOV DS, AX ; set the Data Segment address
  MOV AX, STK
  MOV SS, AX ; set the Stack Segment address
  LEA SP, TOS ; set address of SP as top of stack

  CLI ; clears IF flag

  ;program the 8255
  MOV DX, COM_REG       ; set port to Command Register
  MOV AL, 10000001B     ; set command byte
  OUT DX, AL            ; send data in AL to Command Register
  MOV DX, PORTA
  MOV AL, 00111111B     ;0
  OUT DX, AL
  MOV DX, PORTB
  MOV AL, 00111111B     ;0
  OUT DX, AL

  MOV DX, COM_REG2      ; set port to Command Register
  MOV AL, 10001001B     ; set command byte
  OUT DX, AL            ; send data in AL to Command Register

  ;program the 8259
  MOV DX, PIC1      ; set I/O address to access ICW1
  MOV AL, ICW1
  OUT DX, AL            ; send command word
  MOV DX, PIC2      ; set I/O address to access ICW2,ICW4 and OCW1
  MOV AL, ICW2
  OUT DX, AL            ; send command word
```

```
MOV AL, ICW4
OUT DX, AL              ; send command word
MOV AL, OCW1
OUT DX, AL              ; send command word
STI                     ; enable INTR pin of 8086

;8253
MOV DX, TIMER_REG
MOV AL, 00111000B
OUT DX, AL


;storing interrupt vector to interrupt vector table in memory
MOV AX, OFFSET ISR1     ; get offset address of ISR1 (IP)
MOV [ES:200H], AX       ; store offset address to memory at 200H
MOV AX, SEG ISR1        ; get segment address of ISR1 (CS)
MOV [ES:202H], AX       ; store segment address to memory at 202H

MOV AX, OFFSET ISR2     ; get offset address of ISR2 (IP)
MOV [ES:204H], AX       ; store offset address to memory at 204H
MOV AX, SEG ISR2        ; get segment address of ISR2 (CS)
MOV [ES:206H], AX       ; store segment address to memory at 206H

;foreground routine
HERE:

  MOV CX , LED
  CMP CX, 0
  JE LED_ON
  JNE LED_OFF
  LED_ON:
      MOV LED, 1
      MOV DX, PORTC
      MOV AL, 00000000b
      OUT DX, AL
      CALL DELAY_1S
```

```
    JMP HERE
    LED_OFF:
        MOV LED, 0
        MOV DX, PORTC
        MOV AL, 10000000b
        OUT DX, AL
        CALL DELAY_1S
    JMP HERE
  JMP HERE


;==========1 second timer ============;
DELAY_1S:
   MOV DX, LOAD_CTR0
    MOV AL, 0A0H
    OUT DX, AL
    MOV AL, 0FH
    OUT DX, AL

    TIMER:
        MOV DX, PORTC2
        IN AX, DX
        MOV AH, 00H
        AND AL, 00000001B ; MASK OFF
        CMP AL, 00H
        JNE TIMER
RET
CODE ENDS
END START
```