Laboratory Exercise #7
Hardware Interrupt Interfacing

Exercise Objectives:

1. Interface hardware interrupt source to the 8086 microprocessor.
2. Implement hardware interrupt interfacing using the 8259 Programmable Interrupt Controller IC.
3. Setup hardware interrupt interfacing using 8259 IC in Proteus.

Tools Required:

The following will be provided for the students:
1. PC with Proteus ISIS v8.6 or later installed
2. Programming IDE + 8086 assembler (Emu8086)

Delivery Process:

The instructor will conduct a short briefing about the exercise. Knowledge from Unit VIII (Interrupt Interfacing) will be applied in this exercise, thus the instructor will also provide a short review of the recent chapter.

Note:
Some of the contents in this manual are derived with permission from other sources. This document is a guide only. Your answers are to be submitted in a separate document.

Part I. Theory

Programmable Interrupt Controller

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors. The initial part was 8259, a later A suffix version was upward compatible and usable with the 8086 or 8088 processor.

The main signal pins on an 8259 are as follows: eight interrupt input request lines named IRQ0 through IRQ7, an interrupt request output line named INTR, interrupt acknowledgment line named $\overline{\text{INTA}}$, D0 through D7 for communicating the interrupt level or vector offset. Other connections include CAS0 through CAS2 for cascading between 8259s. Up to eight slave 8259s may be cascaded to a master 8259 to provide up to 64 IRQs. 8259s are cascaded by connecting the INT line of one slave 8259 to the IRQ line of one master 8259.

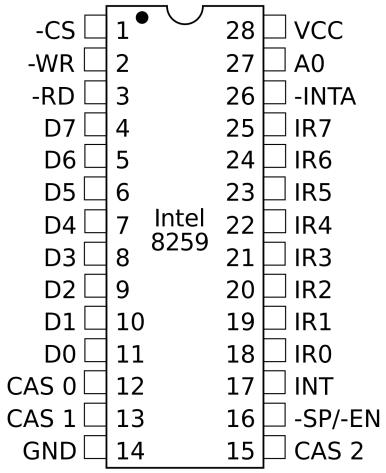

Fig. 1. 8259 Pinout.

There are three registers, an Interrupt Mask Register (IMR), an Interrupt Request Register (IRR), and an In-Service Register (ISR). The IRR maintains a mask of the current interrupts that are pending acknowledgement, the ISR maintains a mask of the interrupts that are pending an EOI, and the IMR maintains a mask of interrupts that should not be sent an acknowledgement.

End Of Interrupt (EOI) operations support specific EOI, non-specific EOI, and auto-EOI. A specific EOI specifies the IRQ level it is acknowledging in the ISR. A non-specific EOI resets the IRQ level in the ISR. Auto-EOI resets the IRQ level in the ISR immediately after the interrupt is acknowledged.

Edge and level interrupt trigger modes are supported by the 8259A. Fixed priority and rotating priority modes are supported.

Programming the 8259

The 8259A is programmed by initialization and operation command words. Initialization command words (ICWs) are programmed before the 8259A is able to function in the system and dictate the basic operation of the 8259A. Operation command words (OCWs) are programmed during the normal course of operation. The OCWs control the operation of the 8259A.

Initialization Command Words (ICW)

There are four initialization command words (ICWs) for the 8259A that are selected when the A0 pin is a logic 1. When the 8259A is first powered up, it must be sent ICW_1, ICW_2, and ICW_4. If the 8259A is programmed in cascade mode by ICW1, then we also must program ICW_3. So if a single 8259A is used in a system, ICW_1, ICW_2, and ICW_4 must be programmed. If cascade mode is used in a system, then all four ICWs must be programmed.

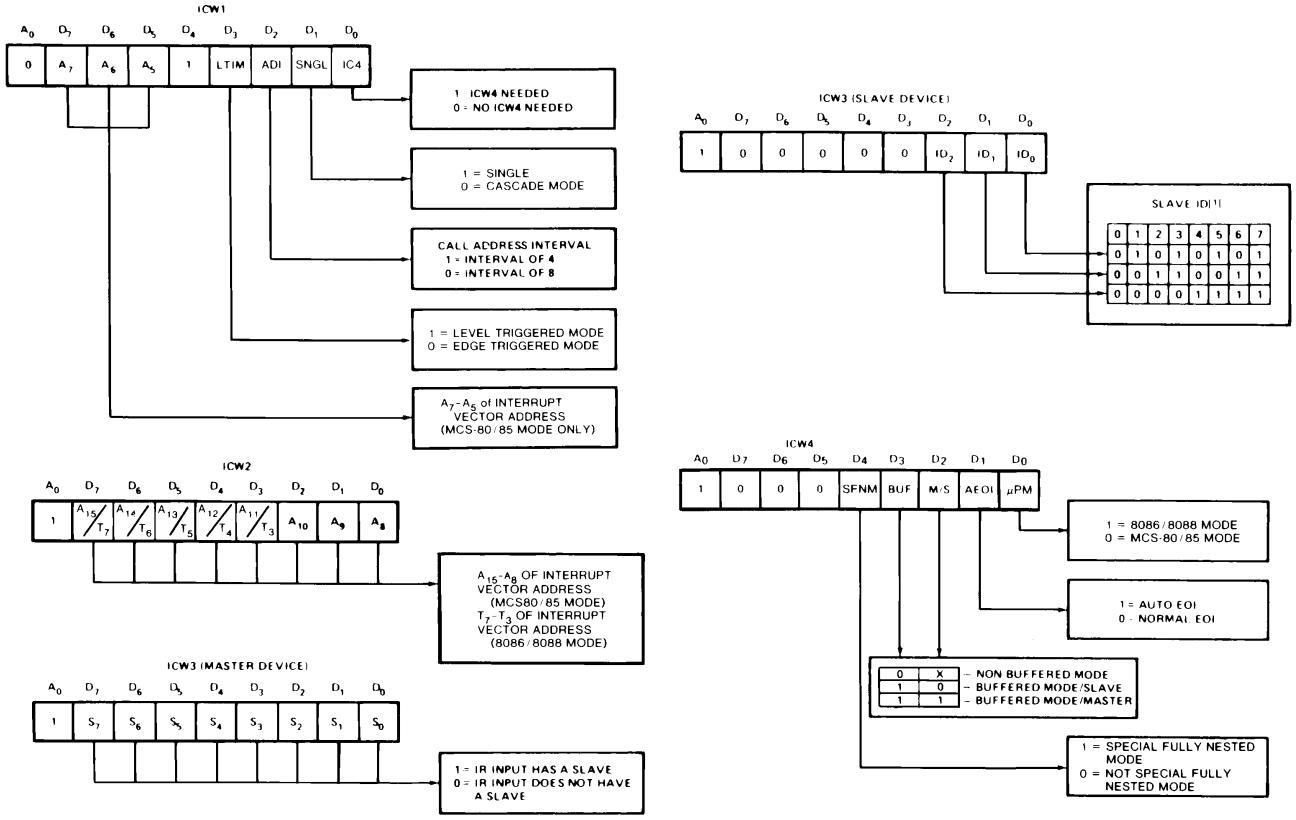

Fig. 2. Initialization Command Word (ICW) Format

Operation Command Words

The operation command words (OCWs) are used to direct the operation of the 8259A once it is programmed with the ICW. The OCWs are selected when the A0 pin is at a logic 0 level, except for OCW_1, which is selected when A0 is a logic 1. Figure 3 lists the binary bit patterns for operation command words #1 (OCW_1) and #2 (OCW_2). For more information about the other OCW, refer to the datasheet.

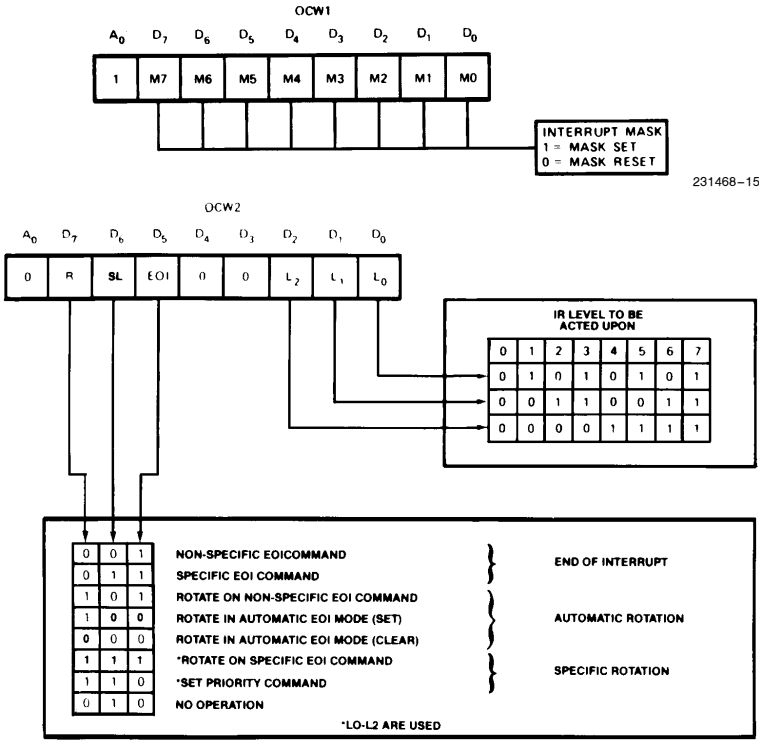

Fig. 3. Operation Command Word (OCW) Format for OCW_1 and OCW_2.

Interfacing the 8259 to 8086

The 8259 will be interfaced to the 8086 microprocessor via the data bus (D_0-D_7), address bus (A_0), INTR, $\overline{INTA}$, $\overline{RD}$, $\overline{WR}$ as well as the I/O address decoder (see Figure 4). The connection to the data bus allows the microprocessor to send commands either ICW or OCW to the 8259 in a similar fashion with the 8255. Therefore the 8259 has to have an I/O address for accessing the ICW and another address for accessing I/O for the OCW.

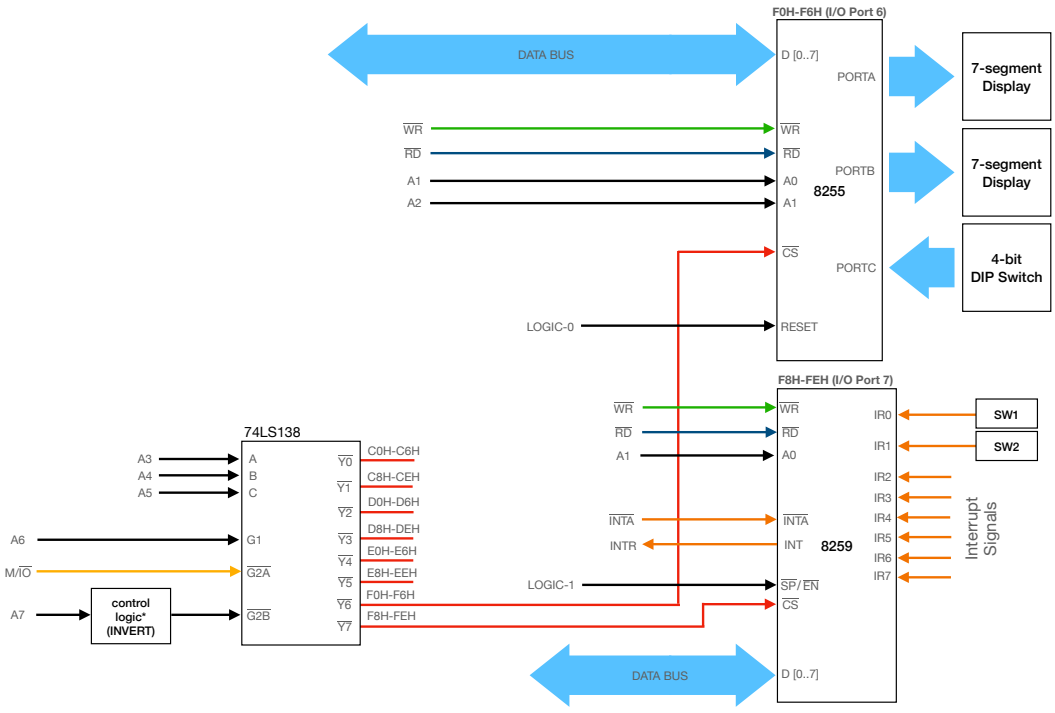

Fig. 4. Interfacing 8259 to 8086 with the I/O address decoder.

To access the ICW_1, OCW_2 and OCW_3 the I/O address should be F8H (A1 is logic-0) while the address FAH (A1 is logic-1) will be used to access the ICW_2, ICW_3, ICW_4 and OCW_1.

Part II. Setting Up 8259 in Proteus

Before starting the activity, make sure you have already completed the circuit used in Laboratory Exercise #5, Activity#3.

Activity #1:

Instructions:

1. Open the circuit in Laboratory Exercise #5 activity #3 ("<last name>_LE5-3.pdsprj") and save it as "<last name>_LE7-1.pdsprj". Replace the push-button switches in PORTC with a 4-bit DIP* switch and connect it to PC0-PC4 of PORTC.

   Important: Change the internal memory size of the 8086 model to 0xF0000.

   ** must be biased properly to connect the I/O pin to ground when the switch is open*

2. Connect the 8259 to the rest of the circuit . Follow the connections as shown in Figure 4.

3. Connect a normally open, active-high* push button switch to IR0 and IR1 pins of the 8259.
   ** must be biased properly to connect the I/O pin to ground when the switch is open*

4. Determine the value of the following command words (in hexadecimal) based on the requirements. Refer to Figures 2 and 3 for the command word format.
   - ICW4 needed, Single Mode, Call Address Interval of 8, Edge Triggered — ICW_1 = _____
   - Interrupt Vector Address: 80H-87H — ICW_2 = _____
   - 8086 Mode, Auto EOI — ICW_4 = _____
   - only IR0 and IR1 are unmasked — OCW_1 = _____

   ** since the configure is single mode, ICW_3 will not be configured*

5. The assembly code below will program the 8259 based on the command words in #4.

```
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 <command word>    ; refer to #4
       ICW2 EQU <command word>    ; refer to #4
       ICW4 EQU <command word>    ; refer to #4
       OCW1 EQU <command word>    ; 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, DATA:DS, SS:STK
ORG 08000H                  ; 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
        <insert code here to program the 8255>

        ;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
        <insert code to store interrupt vector to interrupt vector table>

        ;foreground routine
HERE:
        <insert foreground routine code here>

        JMP HERE
CODE ENDS
END START
```

Important: Insert the code to program the 8255 on its current I/O connections before proceeding.

6. Since there are two interrupt sources IR0 and IR1 (push-button switches), we will need to write the interrupt service routines (ISR) for IR0 and IR1. The code below is a procedure which is outside of the “CODE” segment thus using “PROC FAR”. Note that the segment and procedure names are different (and should be). They should be on different part of the memory. ISR1 will be the interrupt handler (ISR) for IR0 while ISR2 for IR1.

```
PROCED1 SEGMENT
ISR1 PROC FAR
ASSUME CODE: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>
        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 CODE: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>
        POP DX      ; retrieve program context
        POP AX
        POPF        ; pop 16-bit operands
        IRET        ; return from interrupt
ISR2 ENDP           ; end of procedure
PROCED2 ENDS
```

7. After defining the ISRs, we need to store its interrupt vector to the interrupt vector table in memory. Since the interrupt vector is 80H-87H, we need to store the address of the ISRs in this part of memory. IR0 and IR1 are at address 80H and 81H respectively. To find the effective address, we will multiply the interrupt vector address by 4 since there are 4 bytes allocated to each interrupt vector which results into:

   80H x 4 = 200H
   81H x 4 = 204H

   The following program will store the ISR address to memory (via the Extra Segment). The CS:IP of the ISR will be stored. Insert this code in #5.

```
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
```

   After this code is executed, the address of the ISRs are stored in memory. When an interrupt signal happens at IR0, the CPU will execute the ISR at vector address 200H (IP) and 202H (CS). When adding another interrupt source, always calculate the address based on the interrupt vector address in 8259. For example INT5, its vector address in 8259 is 85H and its location in the vector table is 85H x 4 = 214H.

8. Modify the ISRs in #6 where:
   - ISR1: will display ‘9’ on the 7-segment in PORTA
   - ISR2: will display ‘0’ on the 7-segment in PORTA

   Write a program in the “foreground routine” in #5 that will read the data in PORTC (lower nibble) and display the equivalent 7-segment display in PORTB. For example if the data read from PORTC is “00000110” then the display is ‘6’ (mask the upper nibble). Check if the value is greater than 09H. If true, then then display ‘0’ instead.

9. Put the codes altogether in the code editor in Proteus (ISRs, DATA, STK and CODE). Compile and fix errors if any. Before running, check the properties of the 8086 model to make sure the internal memory size is set to 0xF0000 otherwise you will encounter errors during simulation. Run the simulation and observe the response of the microprocessor to the switches connected in IR0 and IR1 of 8259 by observing the 7-segment display in PORTA. Check also the foreground routine by changing the values of the DIP switch in PORTC and observe the display in PORTB.

10. Revise the program or fix the circuit if the desired responses are not met.

11. Save the assembly source code as “<last name>_LE7-1.asm”.

Activity #2:

For the this activity, use the design file in Activity #1. Do not proceed if the previous activity is not done completely.

Instructions:

1. Open the circuit in Activity #1 (“<last name>_LE7-1.pdsprj”) and save as using the filename format “<last name>_LE7-2.pdsprj”. Remove the DIP switch in PORTC and interface a 3x4 keypad with a 74C922 encoder. Connect the DAVBL pin of 74C922 to IR0 of 8259 and the keypad data to PC0-PC4 of PORTC. Connect also an LED in PC7 of PORTC. Retain the push-button switch in IR1.

2. In the foreground routine, write a program that will blink the LED in PC7. For IR0 (keypad), write an ISR that will display the key pressed* on the 7-segment display in PORTA. The number will

be latched until a new key is pressed. For IR1, display the same number displayed in PORTA to the 7-segment display in PORTB.

** for keys '*' and '#', just display '-' (only segment g is on)*

3. Compile and simulate. Observe the output and response from pressing a key in the keypad or pressing the push-button switch. The LED should be blinking steadily while the keypad or switch are pressed.
4. Save the assembly source code as "<last name>_LE7-2.asm".
5. Revise the program or fix the circuit if the desired responses are not met.
6. Why do you think the LED is blinking steadily while other activities are going on?
7. What do you think is the ultimate advantage of using interrupts especially involving I/O devices?

Submission instructions:

- Accomplish the laboratory report. Include the solutions, data, source codes and answers to questions. Save the report as PDF with the filename "<last name>_LE7.pdf".
- Submit the following in Canvas in the correct order:
    i. <last name>_LE7.pdf
    ii. <last name>_LE7-1.asm
    iii. <last name>_LE7-1.pdsprj
    iv. <last name>_LE7-2.asm
    v. <last name>_LE7-2.pdsprj

Assessment

Criteria	1.0 **Outstanding**	2.0 **Competent**	3.0 **Marginal**	4.0 **Not Acceptable**
Circuit Integrity	The circuit was properly constructed.	-	Circuit has issues in terms of construction.	Circuit was constructed poorly.
Circuit Simulation	Simulation results are accurate with no issues.	Simulation results are partially correct.	-	Simulation results are inaccurate.
Correctness of Program	The program is correct with no issues.	The program has minor issues.	The program has several issues but manage to achieve some of the functions required.	Program did not work as intended in the exercise.

End of Laboratory Exercise #7

Copyright Information

Author: Van B. Patiluna (vbpatiluna@usc.edu.ph)
Contributors: none
Date of Release: November 7, 2021
Version: 2.0.3

References

- *Brey, Barry B. The Intel microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit extensions: architecture, programming, and interfacing / Barry B. Brey—8th ed (2009).*
- *Stallings, W. Computer Organization and Architecture, 6th edition, Pearson Education, Inc. (2003).*
- *https://pdos.csail.mit.edu/6.828/2010/readings/hardware/8259A.pdf*

Special thanks to **Prof. Nenad Petrovic** from the Faculty of Electronic Engineering, University of Niš, Serbia for the technical assistance provided.

Change log:

Date	Version	Author	Changes
August 4, 2020	1.0	Van B. Patiluna	- Initial draft.
August 5, 2020	1.0.1	Van B. Patiluna	- Corrected the exercise objectives.
November 23, 2020	2.0	Van B. Patiluna	- Removed the theory and activity involving the 8254 PIT. - Added connection diagram for the 8259 PIC. - Added sample codes for ISRs and storing its address to the interrupt vector table. - Changed the activities to be performed.
November 23, 2020	2.0.1	Van B. Patiluna	- Fixed factual errors.
November 24, 2020	2.0.2	Van B. Patiluna	- Changed INT0 and INT1 to IR0 and IR1.
November 7, 2021	2.0.3	Van B. Patiluna	- Fixed some typographical errors. - Changed the memory address for the data and code segments and interrupt service routines.
November 9, 2021	2.0.4	Van B. Patiluna	- Fixed internal memory size for 8086 VSM model in Activity #1, Item #9 to 0xF0000. - Revised Figure 4.