COLEGIO DE MUNTINLUPA

DEPARTMENT OF COMPUTER ENGINEERING

COEN 3211 - Microprocessors Lab

IO Interfacing with 8255 (Programmable Peripheral Interface)

Laboratory Experiment No. 5

Grade

STUDENT NAME	: **DAVID, Raven A.**
STUDENT NUMBER	: **20202011637**
DATE PERFORMED	: **01 March 2024**
DATE SUBMITTED	: **01 March 2024**

Engr. Ricrey E. Marquez, PCpE

(Lab Instructor)

PRINCIPLES

8255 (Programmable Peripheral Interface) is a very popular low - cost I/O interfacing component found in most microprocessor and microcontroller applications. It has 24 input/output pins, programmable in groups of 12 pins that are used in three (3) separate mode of operation, and can be interface with any Transistor-Transistor Logic (TTL) compatible device to the microprocessor and microcontroller. Figure 5-1 shows the pin configuration of 8255 dual-inline package IC.

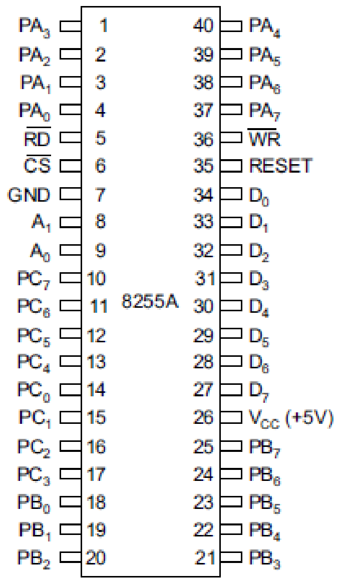


Figure 5-1. 8255 pin diagram (Source: Intel Corporation)

Basically, there are three ports in 8255, namely: **Port A**, **Port B**, and **Port C**, each having pins as illustrated in the block diagram shown in Figure 5-2. Furthermore, the Port C can be divided into **Port C_{upper}** and **Port C_{lower}** - each having four pins (or *nibble*). Thus, 8255 can be viewed to have four ports such as Port A, Port B, Port C_{upper} and Port C_{lower}.

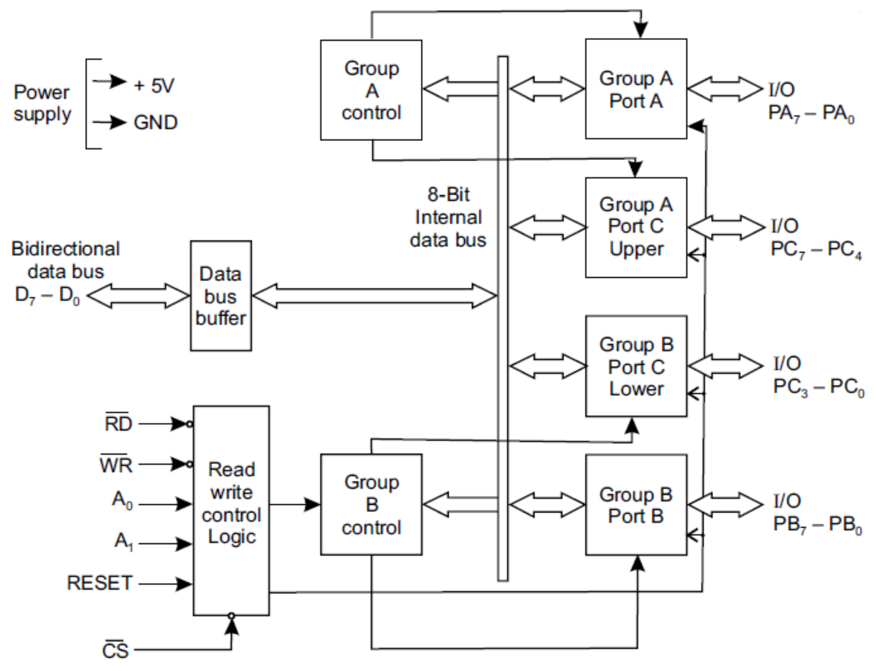


Figure 5-2. Block diagram of 8255 (Source: Intel Corporation)

The three ports are divided into two groups such as Groups *A* and *B*. **Group A** consists of *Port A* and *Port C_U*. Port A can be operated in any of the modes (0, 1, or 2). While **Group B** consists of *Port B* and C_L. Here Port B can be operated in either mode 0 or 1.

Aside from IO ports, 8255 has six pins associated with Read/Write control logic block. These are CS, WR, RD, A0 & A1, and RESET signals.

- **Chip Select (CS)’** - use to select for the programming, and reading and writing to a port.
- **Address 0 & Address 1 (A1 & A0)** - register selection that selects an internal register for programming or operation
- **Read (RD)’** - use to select the read (input) operation of PPI.
- **Write (WR)’** - use to select the write (output) operation of PPI.
- **Reset (RESET)’** - this input pin use to initialize the device.

Parts of control byte of PPI

A single control word determines the operating mode of 8255 is called *command* or *control byte* as shown in Figure 5-3. The two address lines, along with CS signal, determine the selection of a particular port or control register.

Table 5-1. Selecting different ports and control words/registers are in 8255

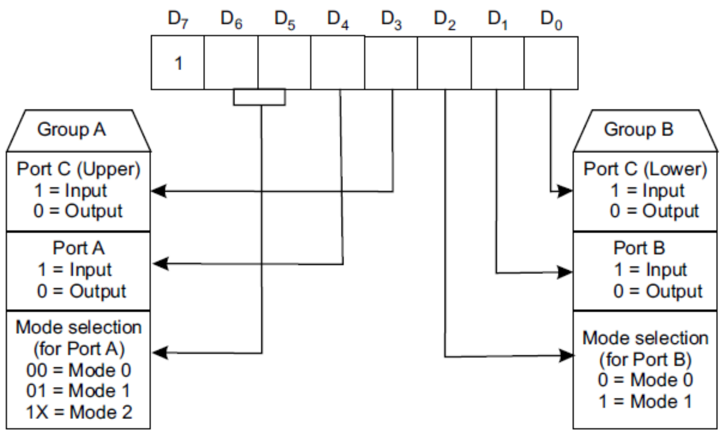
$\overline{CS}$	A_1	A_0	*Selection*
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	X	X	8255 not selected (because $\overline{CS}$ = 1)

For example, if CS signal is made 0 by choosing **A_7** = **1** and **A_6** though **A_2** = **0**. Thus,

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0			
1	0	0	0	0	0	0	0	=	80_H	Port A selected
1	0	0	0	0	0	0	1	=	81_H	Port B selected
1	0	0	0	0	0	1	0	=	82_H	Port C selected
1	0	0	0	0	0	1	1	=	83_H	Control Register selected

PPI Mode of Operations

The PPI 8255 can operate in three modes for IO operations namely Mode 0, Mode 1 and Mode 2. These are I/O operations and selected only if **D_7** bit of the control word register is put as **1**.



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1
Group A
Port C (Upper) 1 = Input 0 = Output
Port A 1 = Input 0 = Output
Mode selection (for Port A) 00 = Mode 0 01 = Mode 1 1X = Mode 2
Group B
Port C (Lower) 1 = Input 0 = Output
Port B 1 = Input 0 = Output
Mode selection (for Port B) 0 = Mode 0 1 = Mode 1

Figure 5-3. The control register byte in the I/O mode

Mode 0: This is a basic or simple input/output mode, whose features are:

- Outputs are latched.
- Inputs are not latched.
- All ports (A, B, C_U, C_L) can be programmed in either input or output mode.
- Ports don't have handshake or interrupt capability.
- Sixteen possible input/output configurations are possible.

Mode 1: In this mode, input or outputting of data is carried out by taking the help of handshaking signals, also known as *strobe signals*. The basic features of this mode are:

- Ports A and B can function as 8-bit I/O ports, taking the help of pins of Port C.
- Inputs and output pins are latched.
- Interrupt logic is supported.
- Handshake signals are exchanged between CPU and peripheral prior to data transfer.
- In this mode, Port C is called status port.
- There are two groups in this mode—group A and group B. They can be configured separately. Each group consists of an 8-bit port and a 4-bit port. This 4-bit port is used for handshaking in each group.

Mode 2: In this mode, Port A can be set up for bidirectional data transfer using handshake signals from Port C. Port B can be set up either in mode 0 or mode 1.

The basic operations of the three modes are shown in Figure 5-4.

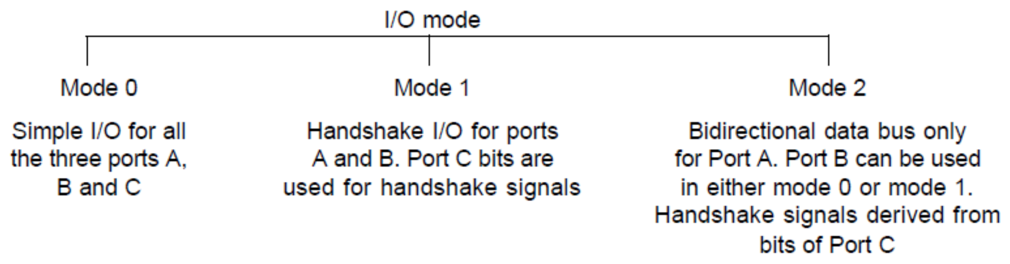

I/O mode
Mode 0
Simple I/O for all the three ports A, B and C
Mode 1
Handshake I/O for ports A and B. Port C bits are used for handshake signals
Mode 2
Bidirectional data bus only for Port A. Port B can be used in either mode 0 or mode 1. Handshake signals derived from bits of Port C

Figure 5-4. Basic operations of the three modes in IO operations

For example, write down the mode 0 control or command byte for the following two cases:

a) Port A = Input port, Port B = not used, Port CU = Input port and Port C_L = Output port.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	0	0	1	1	0	0	0	=	98_H
I/O mode	Mode 0 for Port A		Port A input	Port C_U input	Port B not used		Port C_L output		

Thus, the control or command byte value is **98H**

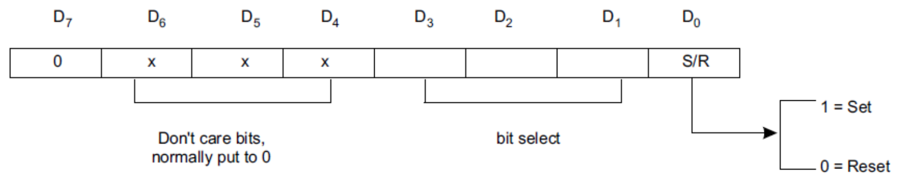
b) Port A = Output port, Port B = Input port, Port C = Output port

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	0	0	0	0	0	1	0	=	82_H
I/O mode	Mode 0 for Port A		Port A output	Port C_U output	Mode 0 for Port B	Port B input	Port C_L output		

Thus, the control or command byte value is **82H**

Apart from the above PPI modes, there is another mode called **BSR mode** (**B**it **S**et/**R**eset mode). The characteristics of BSR mode are:

- BSR mode is selected only when **D_7** = **0** of the control or command register.
- Concerned with bits of Port C.
- Individual bits of Port C can either be Set or Reset.
- At a time, only a single bit of port C can be Set or Reset.
- Is used for control or on/off switch.
- BSR control word doesn’t affect ports A and B functioning.



D_3	D_2	D_1	*Particular bit of Port C selected*
0	0	0	Bit 0
0	0	1	Bit 1
0	1	0	Bit 2
0	1	1	Bit 3
1	0	0	Bit 4
1	0	1	Bit 5
1	1	0	Bit 6
1	1	1	Bit 7

Figure 5-5. The control register byte in BSR mode

A. Syntax of initializing PPI's control register:

```
MOV DX, CTRL_REG_ADDRESS
MOV AL, CTRL_BYTE_VALUE
OUT DX, AL
```

B. Syntax of reading data on PPI's input port:

```
MOV DX, INPUT_PORT_ADDRESS
IN AL, DX
```

C. Syntax of sending/writing data on PPI's output port:

```
MOV DX, OUTPUT_PORT_ADDRESS
MOV AL, 8-BIT_DATA
OUT DX, AL
```

OBJECTIVES AND MATERIALS

Objectives:

After this lab experiment, student should be able to:

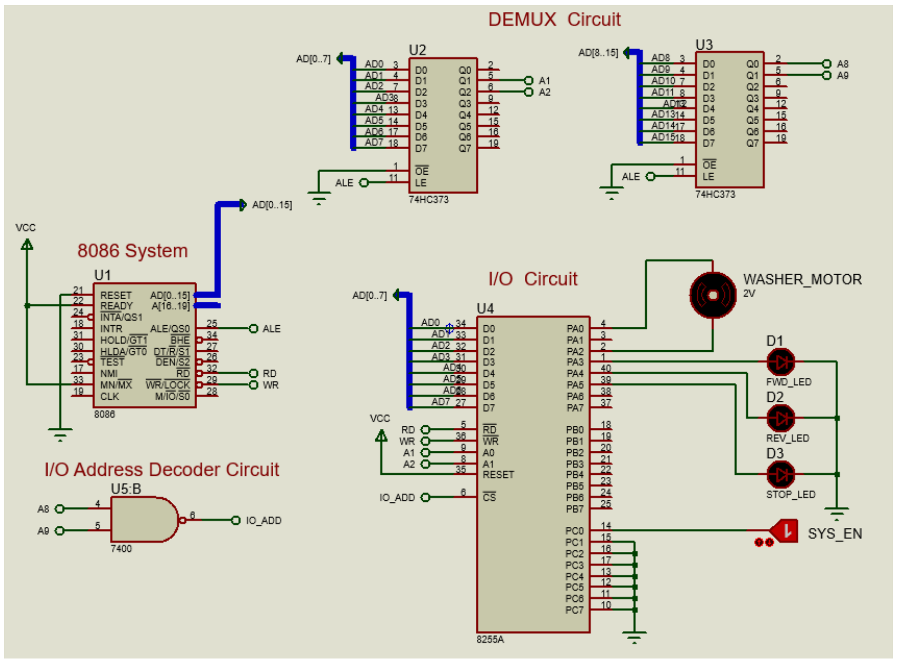
1. familiarize 8255 (PPI) mode of operations as well as its I/O ports,
2. configure control byte register of 8255,
3. discuss how to interface 8255 to an MPU based systems, and
4. write an assembly program control I/O connected to 8255.

Materials:

QUANTITY	PART NUMBER	DESCRIPTION
1	-	PC/Laptop with Proteus ISIS software installed (or optional Emu8086 assembler software)

DRILL EXERCISES

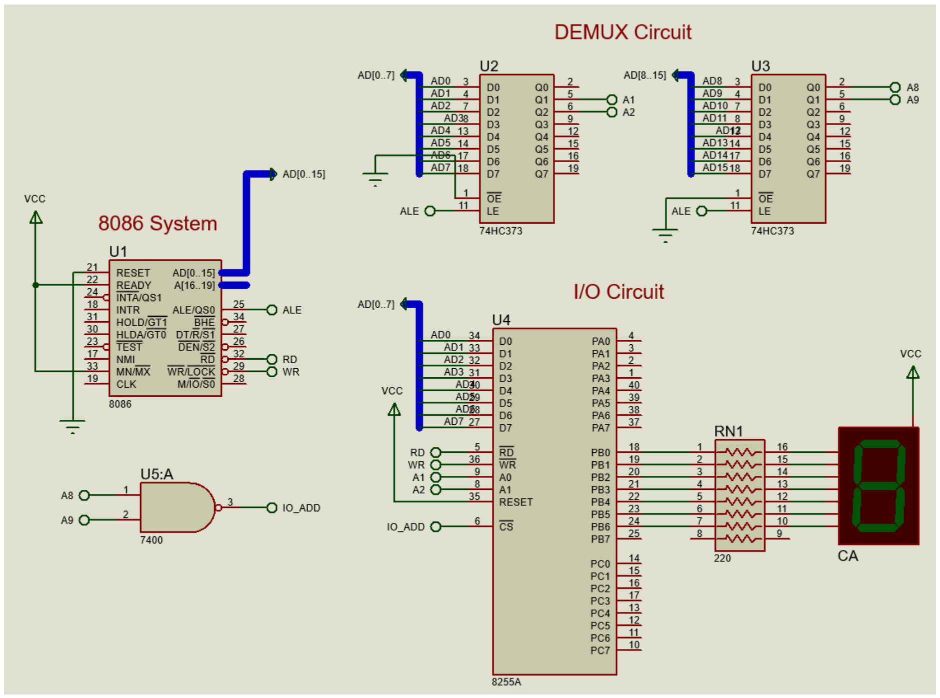
Drill Exercise 5-1 – Given the schematic diagram and assembly language source code illustrated in Figure 5-6, re-type and test the program. Save as **`COEN3211-3x-x_Drill5_1.pdsprj`**.



```
DATA SEGMENT
   PORTA EQU 0300H
   PORTC EQU 0304H
   PORT_CON EQU 0306H
   CON_BYTE EQU 89H
   STP EQU 20H
   FWD EQU 09H
   REV EQU 14H
   OFF_STATE EQU 00H
   ON_STATE EQU 01H
DATA ENDS

CODE    SEGMENT
        MOV AX, DATA
        MOV DS, AX
        JMP START
        ;initialized PPI's ports
START:  MOV DX, PORT_CON
        MOV AL,  CON_BYTE
        OUT DX,  AX
        ;check switch status
CHK_SW: MOV DX, PORTC
        IN AL, DX
        CMP AL, OFF_STATE
        JE STP_MODE
        CMP AL, ON_STATE
        JE RUN_MODE
        JMP CHK_SW

STP_MODE:   MOV DX, PORTA
            MOV AL, STP
            OUT DX, AL
            CALL DELAY
            JMP CHK_SW

RUN_MODE:   MOV DX, PORTA
            MOV AL, FWD
            OUT DX, AL
            CALL DEL30
            MOV AL, STP
            OUT DX, AL
            CALL DELAY
            MOV AL, REV
            OUT DX, AL
            CALL DEL30
            MOV AL, STP
            OUT DX, AL
            CALL DELAY
            JMP CHK_SW
;30s DELAY
DEL30  PROC NEAR
        MOV BX, 30
   D1:  CALL DELAY
        DEC BX
        CMP BX, 0
        JNE D1
        RET
DEL30  ENDP
;APPROX. 1 SEC. DELAY
DELAY  PROC NEAR
        MOV CX, 5FFFH
   D2:  NOP
        NOP
        NOP
        LOOP D2
        RET
DELAY  ENDP
CODE    ENDS
END START
```

Figure 5-6. Schematic diagram and assembly source of Drill Exercise 5-1

Drill Exercise 5-2 – Given the schematic diagram and assembly language source code illustrated in Figure 5-4, re-type and test the program. Save as **`COEN3211-x_Drill5_2.pdsprj`**.



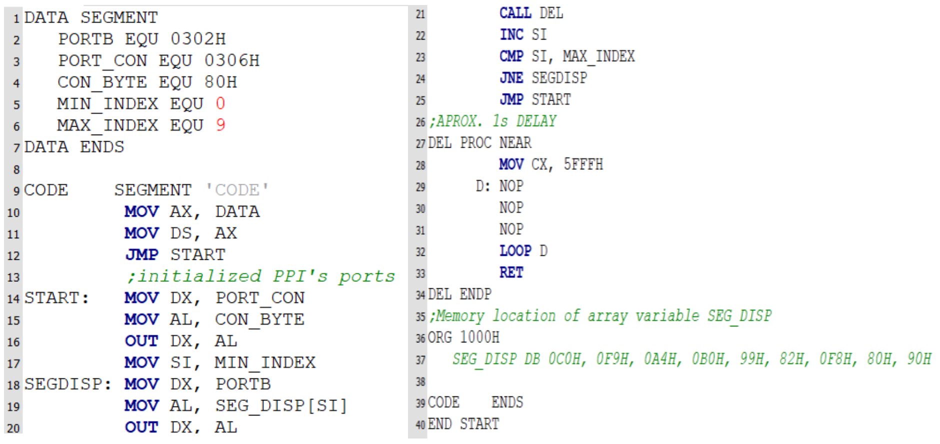

```
DATA SEGMENT
   PORTB EQU 0302H
   PORT_CON EQU 0306H
   CON_BYTE EQU 80H
   MIN_INDEX EQU 0
   MAX_INDEX EQU 9
DATA ENDS

CODE      SEGMENT 'CODE'
           MOV AX, DATA
           MOV DS, AX
           JMP START
           ;initialized PPI's ports
START:     MOV DX, PORT_CON
           MOV AL, CON_BYTE
           OUT DX, AL
           MOV SI, MIN_INDEX
SEGDISP:   MOV DX, PORTB
           MOV AL, SEG_DISP[SI]
           OUT DX, AL
           CALL DEL
           INC SI
           CMP SI, MAX_INDEX
           JNE SEGDISP
           JMP START
;APROX. 1s DELAY
DEL PROC NEAR
           MOV CX, 5FFFH
        D: NOP
           NOP
           NOP
           LOOP D
           RET
DEL ENDP
;Memory location of array variable SEG_DISP
ORG 1000H
   SEG_DISP DB 0C0H, 0F9H, 0A4H, 0B0H, 99H, 82H, 0F8H, 80H, 90H

CODE    ENDS
END START
```

Figure 5-7. Schematic diagram and assembly source of Drill Exercise 5-2

For both applications, the design address of PPI's port address is shown below:

PPI Address in Protues ISIS when **A9** & **A8** is part of decoder circuit for **CS'**:

															A1	**A0**	***RSV***	**REGISTERS**	**EQV. ADDRESS**	
CPU Address (AD)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**0**	**0**	**0**	PORTA	**0300H**	`PORTA EQU 0300H`
	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**0**	**1**	**0**	PORTB	**0302H**	`PORTB EQU 0302H`
	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**	**0**	**0**	PORTC	**0304H**	`PORTC EQU 0303H`
	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**	**1**	**0**	CREG	**0306H**	`CNTL_REG EQU 0306H`
	0				**3**				**0**											

PROGRAM EXERCISES

Program Exercise 5-1. 3. Design IO interfacing circuit with PPI with a port register addresses of (0500H, 0502H, 0504H, and 0506H). Write an assembly program that will rotate two LEDs at time from right to left (LED_0 – LED_7) that is connected to 8255 port C, and repeat. Save as **`COEN3211-3x-x_ProgExer5_1.pdsprj`**.

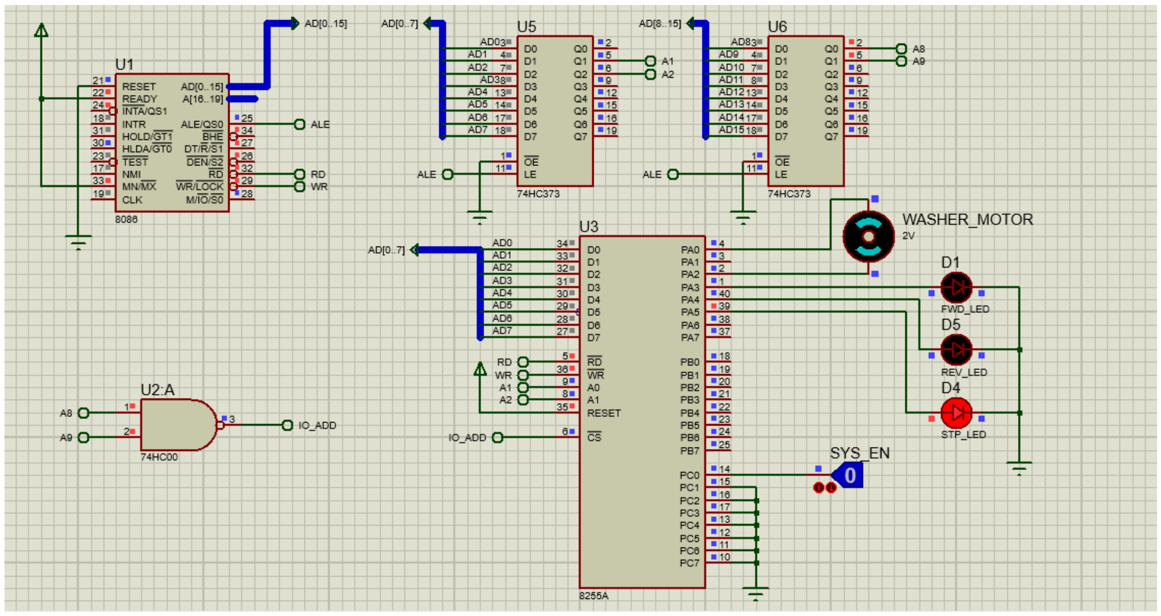
Program Exercise 5-2. Design IO interfacing circuit with PPI with a port register address of (0C00H, 0C02H, 0C04H, and 0C06H) respectively. Create an assembly code that will read status of the switches connected at lower port C of 8255, then display the sequence of LEDs connected at 8255 Port A that is listed below. Save as **`COEN3211-3x-x_COEN3211_ProgExer5_2.pdsprj`**.

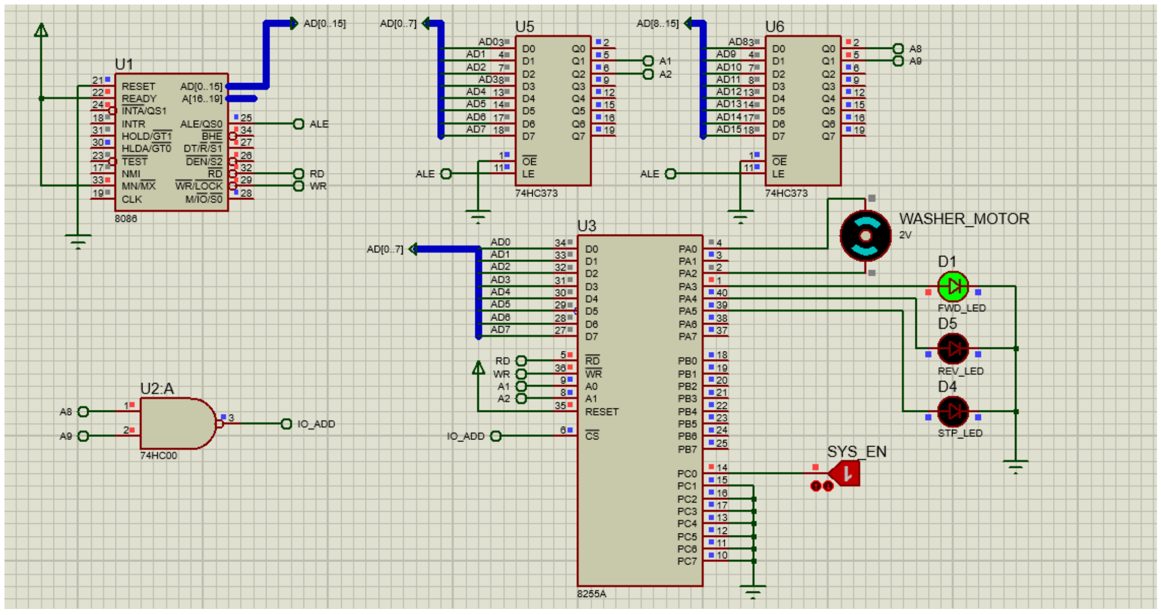
Table 5-3. Switch status and LEDs patterns for program exercise 5-2

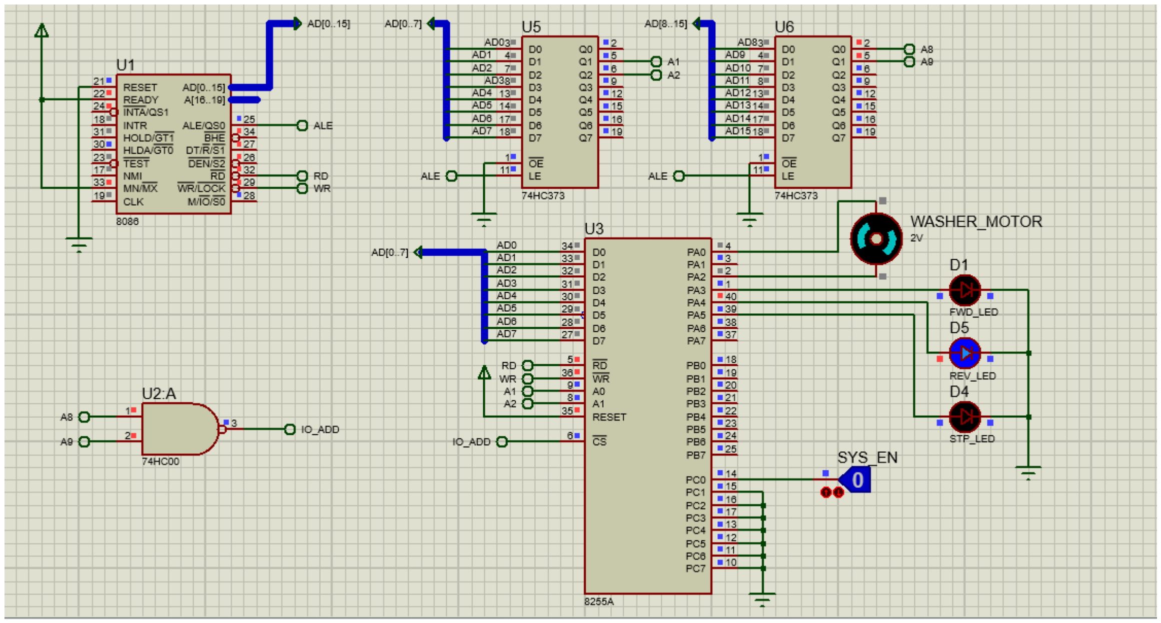
INPUTS				OUTPUTS							
PC3	**PC2**	**PC1**	**PC0**	**PA0**	**PA1**	**PA2**	**PA3**	**PA4**	**PA5**	**PA6**	**PA7**
SW3	**SW2**	**SW1**	**SW0**	**LED0**	**LED1**	**LED2**	**LED3**	**LED4**	**LED5**	**LED6**	**LED7**
OFF	*OFF*	*OFF*	*ON*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*ON*	*ON*
OFF	*OFF*	*ON*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*ON*	*ON*	*OFF*	*OFF*
OFF	*ON*	*OFF*	*OFF*	*OFF*	*OFF*	*ON*	*ON*	*OFF*	*OFF*	*OFF*	*OFF*
ON	*OFF*	*OFF*	*OFF*	*ON*	*ON*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*
OTHER COMBINATIONS				*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	……

DATA RESULTS

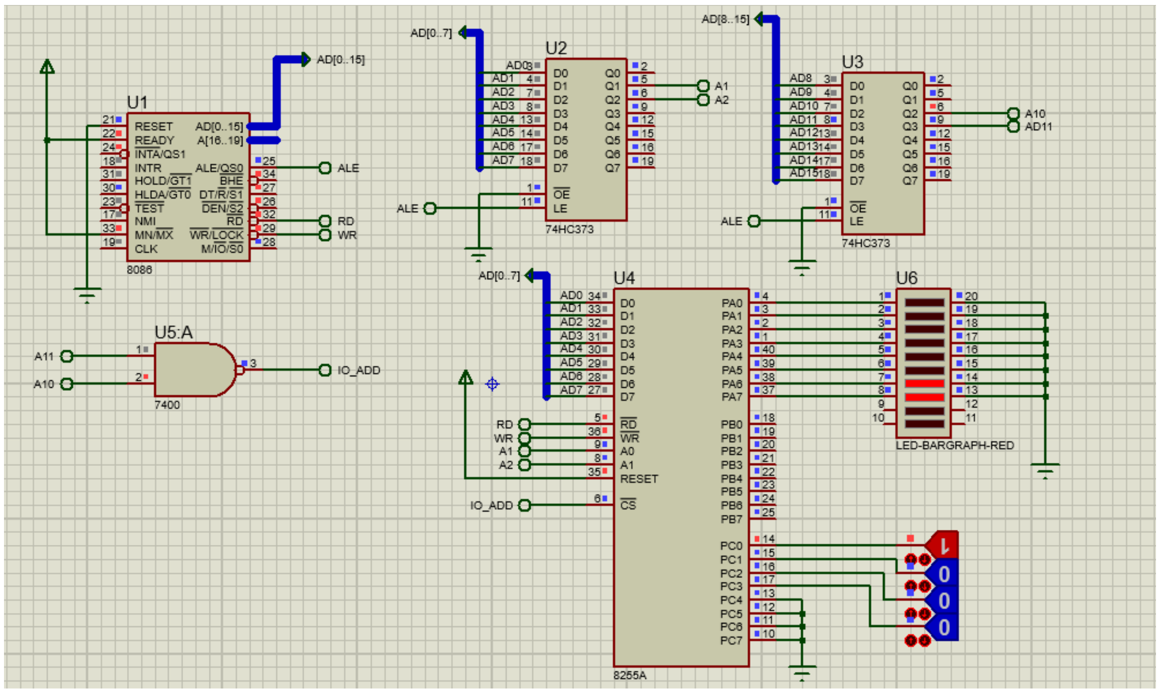
Drill Exercise 5-1. Test output

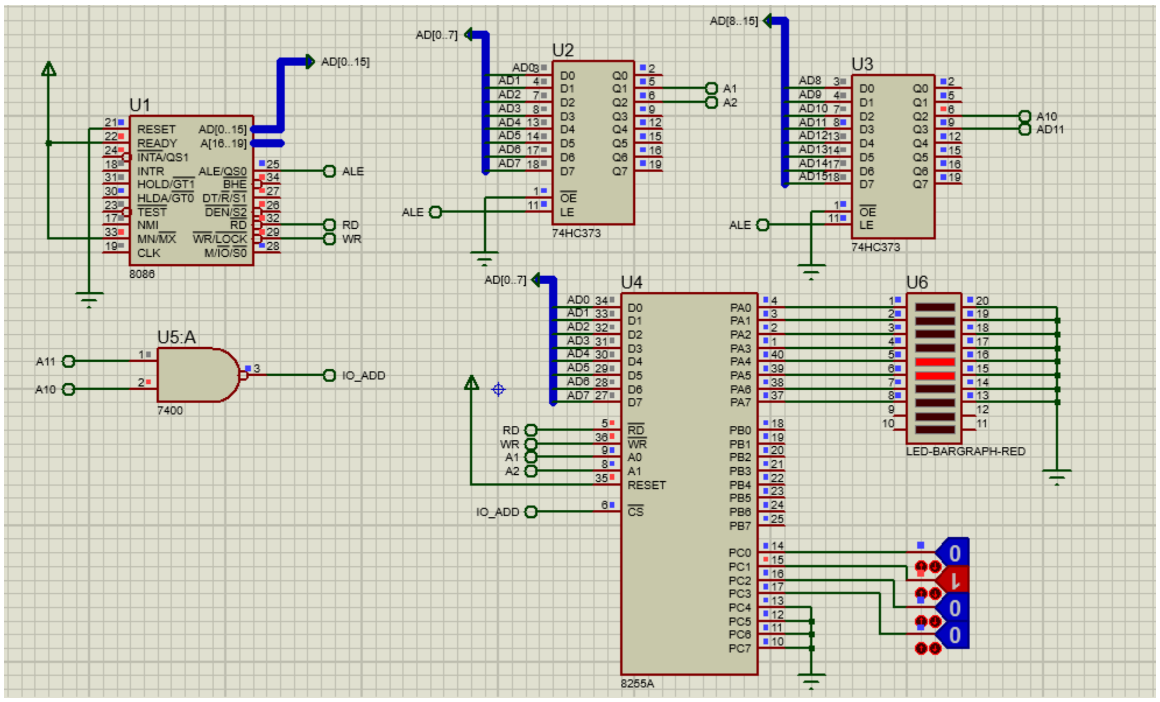


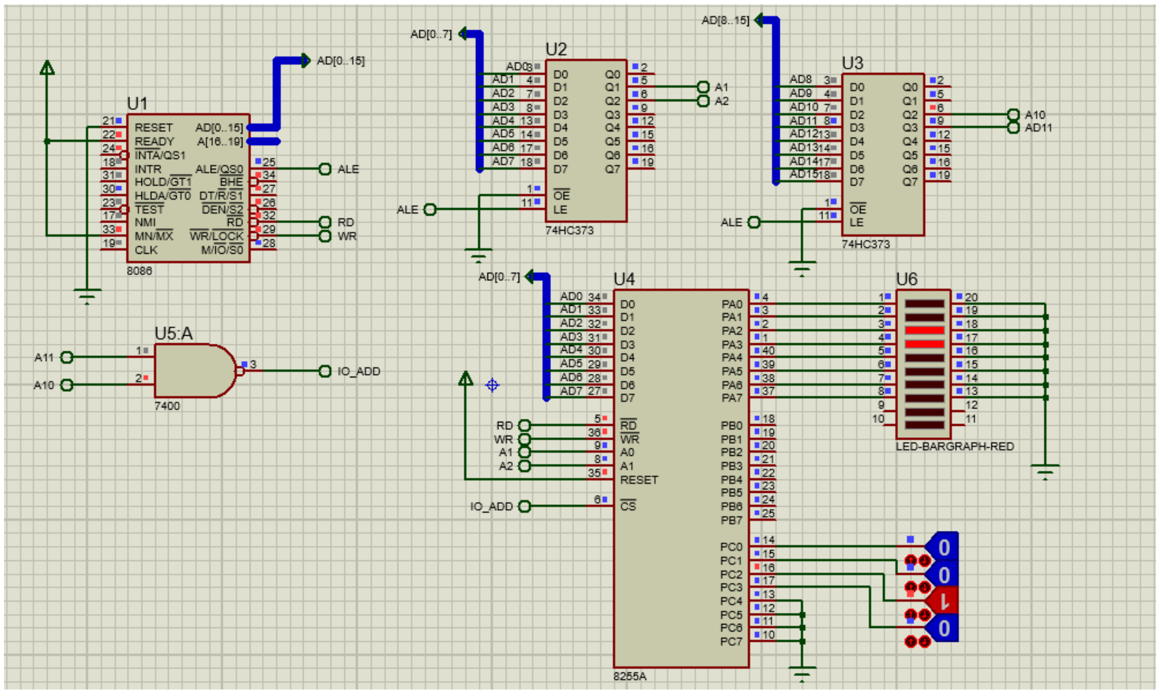


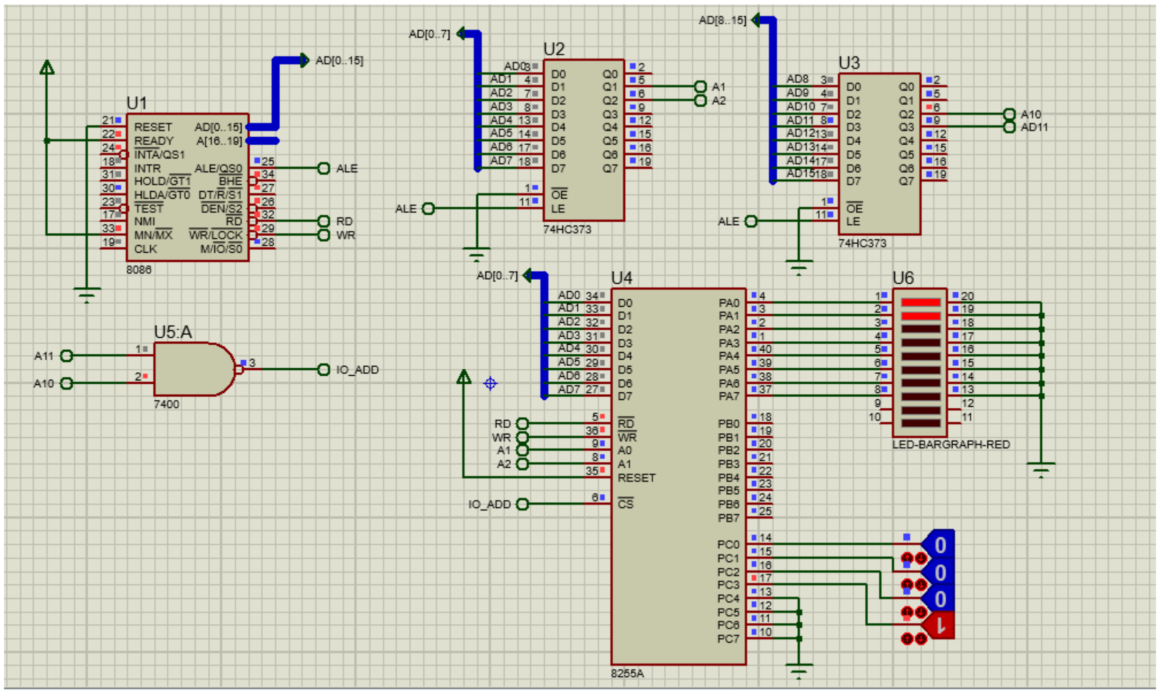

U1
8086
AD[0..15]
ALE
RD
WR
U5
74HC373
AD[0..7]
A1
A2
U6
74HC373
AD[8..15]
A8
A9
U3
8255A
AD[0..7]
IO_ADD
WASHER_MOTOR
2V
D1
FWD_LED
D5
REV_LED
D4
STP_LED
SYS_EN
U2:A
74HC00
A8
A9
IO_ADD

Drill Exercise 5-2. Test output



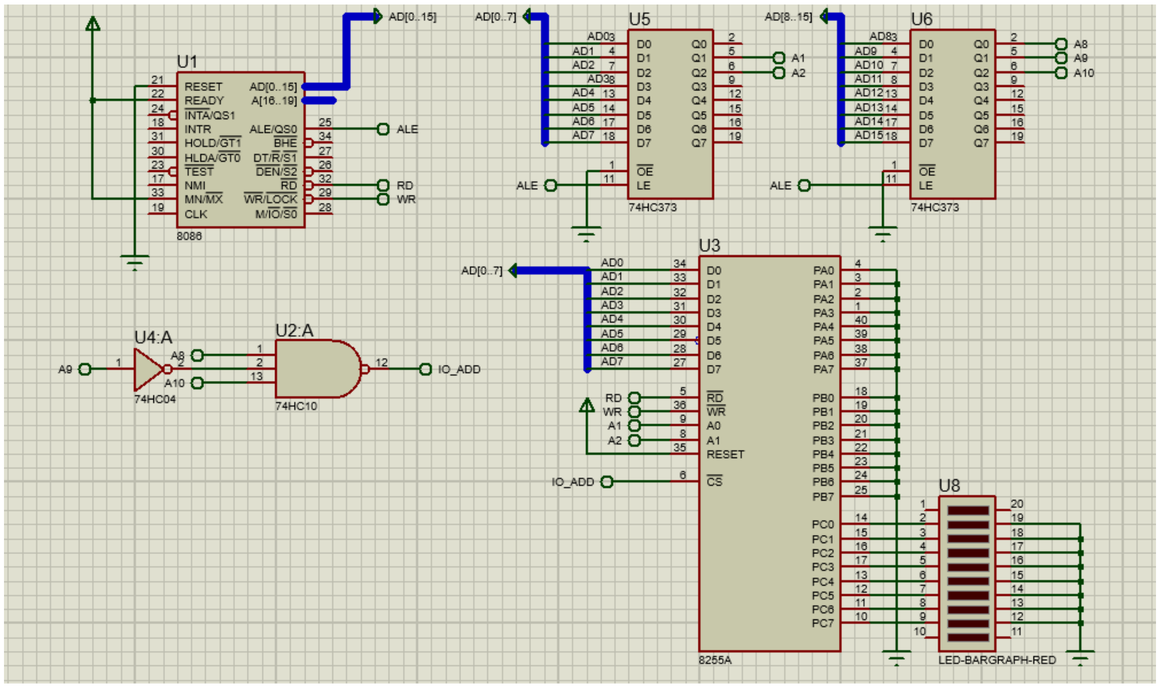



U1
8086
U2
74HC373
U3
74HC373
U4
8255A
U5:A
7400
U6
LED-BARGRAPH-RED
AD[0..15]
AD[0..7]
AD[8..15]
ALE
RD
WR
A1
A2
A10
A11
IO_ADD


U1
8086
U2
74HC373
U3
74HC373
U4
8255A
U5:A
7400
U6
LED-BARGRAPH-RED
AD[0..15]
AD[0..7]
AD[8..15]
ALE
RD
WR
A1
A2
A10
A11
IO_ADD

Program Exercise 5-1. Schematic diagram, program listing, and outputs

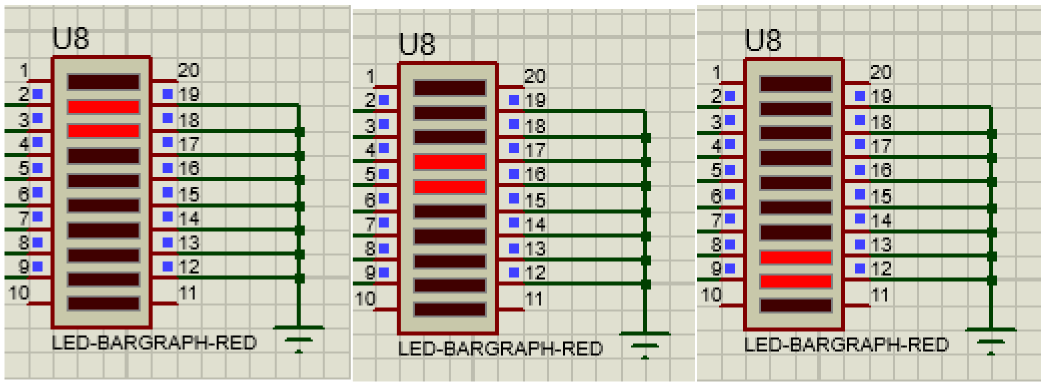
Schematic diagram:



Program Listing:

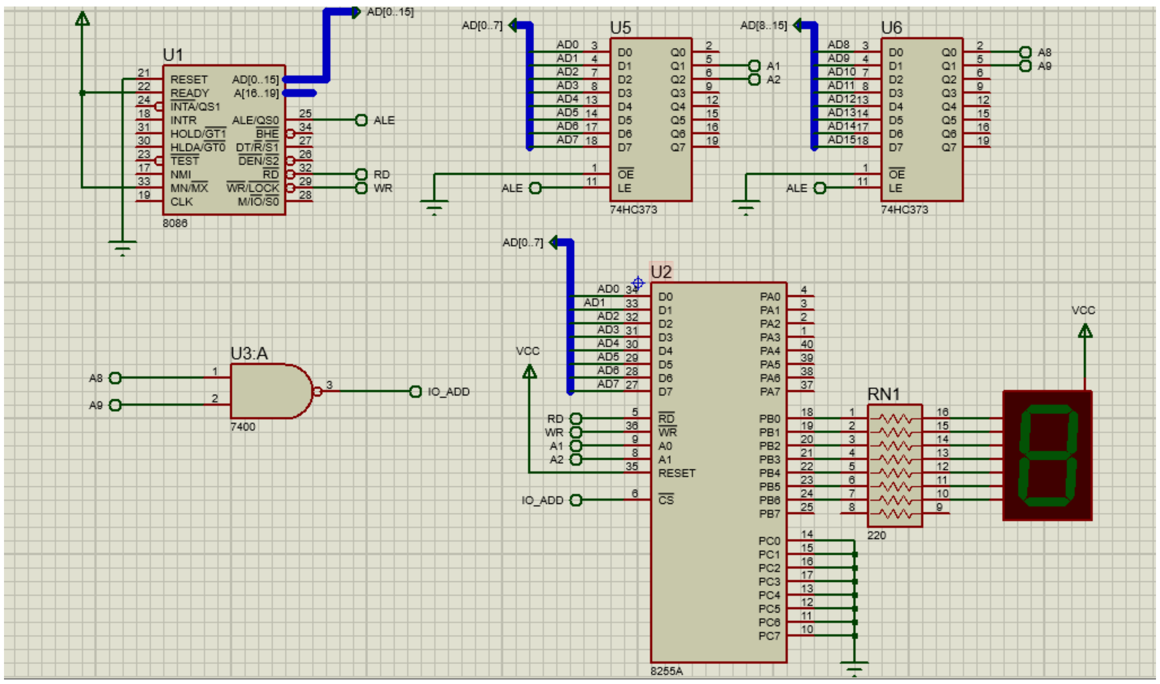
```
DATA SEGMENT
      PORTA       EQU 0500H
      PORTB       EQU 0502H
      PORTC       EQU 0504H
      PORT_CON    EQU 0506H
      CON_BYTE    EQU 80H
DATA ENDS

CODE    SEGMENT PUBLIC
      MOV AX, DATA
      MOV DS, AX
      JMP START

START:    MOV DX, PORT_CON
          MOV  AL, CON_BYTE
          OUT DX, AX
OUTPUT:   MOV DX, PORTC
          MOV AL, 03H
ROTATE:   OUT DX, AL
          CALL DELAY

          SHL AL, 01H
          CMP AL, 00H
          JNE ROTATE
          CALL DELAY
          JMP START

DELAY PROC NEAR
      MOV CX, 5FFFH
   D2: NOP
      NOP
      NOP
      LOOP D2
      RET
DELAY ENDP

CODE    ENDS
END START
```

Output:



Program Exercise 5-2. Schematic diagram, program listing, and outputs

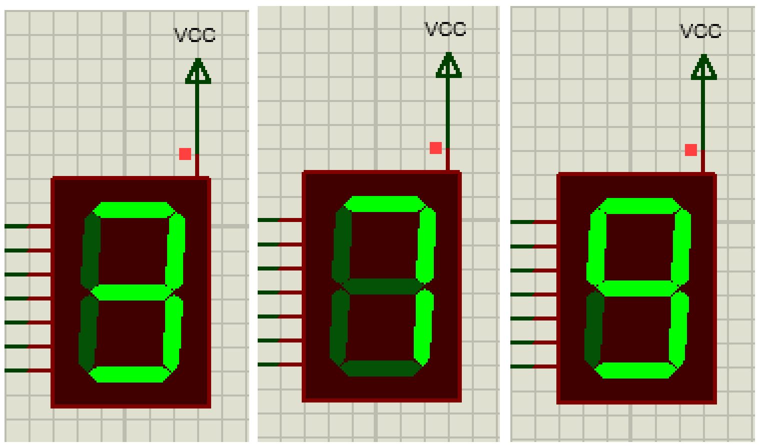
Schematic diagram:



Program Listing:

```
DATA SEGMENT
        PORTB EQU 0302H
        PORT_CON EQU 0306H
        CON_BYTE EQU 80H
        MIN_INDEX EQU 0
        MAX_INDEX EQU 9
DATA ENDS

CODE SEGMENT 'CODE'
        MOV AX, DATA
        MOV DS, AX
        JMP START
        ;initialized PPI'S ports
START:    MOV DX, PORT_CON
          MOV AL, CON_BYTE
          OUT DX, AL
          MOV SI, MIN_INDEX
SEGDISP: MOV DX, PORTB
           MOV AL, SEG_DISP[SI]
           OUT DX, AL

           CALL DEL
           CALL DEL
           INC SI
           CMP SI, MAX_INDEX
           JBE SEGDISP
           JMP START
;APROX. 1s DELAY
DEL PROC NEAR
            MOV CX, 5FFFH
       D:   NOP
            NOP
            NOP
            LOOP D
            RET
DEL ENDP
;Memory location of array variable SEG_DISP
ORG 1000H
     SEG_DISP DB 0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H

CODE    ENDS
END START
```

Output:



DATA ANALYSIS

In Drill Exercise 5-1, The program controls a stepper motor through an 8085 microprocessor, establishing memory locations to regulate motor operation and defining constants to represent different states. It continuously monitors a switch, pausing if the switch remains off, and initiating motor movement when the switch is activated. Initially, the motor advances forward for a predetermined duration before halting. Subsequently, it reverses direction, travels for the same duration, and halts once more. This forward-reverse sequence continues cyclically until the switch is deactivated. The program employs two delay subroutines to precisely manage the duration of the motor's movement in each direction, ensuring smooth and controlled operation.

In Drill Exercise 5-2, The setup comprises an 8086 microprocessor and an 8255A programmable peripheral interface (PPI) chip, meticulously crafted to exhibit a series of numbers on seven-segment displays. Orchestrated by assembly code, the operation unfolds seamlessly. Initially, the code delineates memory locations to govern the 8255A and sets parameters like delay timing and the sequence of digits to be displayed. Subsequently, it delves into a loop, incessantly traversing a data array housing binary code representing digits 0 through 9 for the seven-segment displays. Within each iteration, the code dispatches the current digit code to the 8255A, orchestrating the displays' behavior, before orchestrating a brief pause to transition to the subsequent digit in the sequence. This rhythmic cycle perpetuates, etching the digit sequence onto the seven-segment displays with unfaltering continuity.

In Program Exercise 5-1, The circuit showcases an I/O interfacing setup utilizing an 8255A Programmable Peripheral Interface (PPI) chip. Tasked with controlling LEDs linked to its port C, the 8255A operates under the directives of an assembly program. The initialization phase entails configuring port A and port C as outputs before entering an infinite loop. Within this loop, the program initiates by illuminating the first LED (LED0) and subsequently introduces a brief delay. It then cyclically shifts the LED pattern one bit to the left, extinguishing LED0 while activating LED1. This rotational process persists until all LEDs have sequentially illuminated, upon which the loop recommences, perpetuating a continuous leftward progression of lit LEDs. To visualize this phenomenon, simulating the circuit in Proteus enables a graphical representation of the LED pattern. It's imperative to note that the program necessitates a delay routine to regulate the rotational speed, and the schematic likely integrates components for managing clock signals to ensure proper circuit functionality.

In Program Exercise 5-2, The circuit illustration showcases an I/O interfacing arrangement employing an 8255A chip. Operating as a bridge between the system and peripherals, the 8255A interfaces LEDs linked to its port A and switches connected to the lower bits of port C. The accompanying assembly program framework commences by initializing port A for output, thereby illuminating all LEDs. Subsequently, it configures port B for input to monitor switch states. Within the primary loop, the program perpetually monitors switch states, employing the logic provided in Program Exercise 4-2 to interpret these states. Through conditional statements and bitwise operations, the LED pattern is dynamically updated in response to changes in switch configurations, facilitating interactive control and visualization within the circuit.

QUESTIONS AND ANSWERS

Questions:

1. Discuss the mode 2 of PPI 8255 in brief and cite an example when to use this mode.
2. Explain what happens when RESET pin of 8255 is made high?
3. Design an IO interfacing circuit with PPI (8255) with port address shown below:
   - **Port A - 0700H,**
   - **Port B - 0702H,**
   - **Port C – 0704H,** and
   - **Control Port – 0706H.**

   To check the functionality of the circuit, write an assembly language program that will read the status of the two switches and display the sequence of seven-segment below sequence below. Save as **`SURNAME_COEN3211_QA3.pdsprj`**.

SWITCH STATUS		**SEVEN-SEGMENT SEQUENCE**
SW2	**SW1**	
Off	Off	0
Off	On	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, then repeat
On	Off	9, 8, 7, 6, 5, 4, 3, 2, 1, 0, then repeat
On	On	E

Answers:

1. **MODE 2 OF PPI 8255:**

Mode 2 of the Programmable Peripheral Interface (PPI) 8255 is commonly known as the Bidirectional Mode. In this mode, Port A and Port B of the 8255 operate as bidirectional I/O ports, while Port C can be used either as two separate 4-bit ports (in Mode 0) or as a single 8-bit port (in Mode 1).

In Mode 2, the 8255 can function as a bidirectional data transfer device, allowing both input and output operations on Port A and Port B simultaneously. This bidirectional capability is particularly useful in applications where data needs to be transferred bidirectionally between a microprocessor and peripheral devices.

An example scenario where Mode 2 of the PPI 8255 could be employed is in the implementation of a parallel data transfer between a microprocessor and an external memory module. In this scenario, Port A and Port B of the 8255 can be used to interface with the data bus of the microprocessor and the data lines of the memory module. The microprocessor can write data to the memory module by outputting it through Port A, while simultaneously reading data from the memory module through Port B. This bidirectional data transfer capability simplifies the interface between the microprocessor and the memory module, enhancing the overall efficiency of the system.

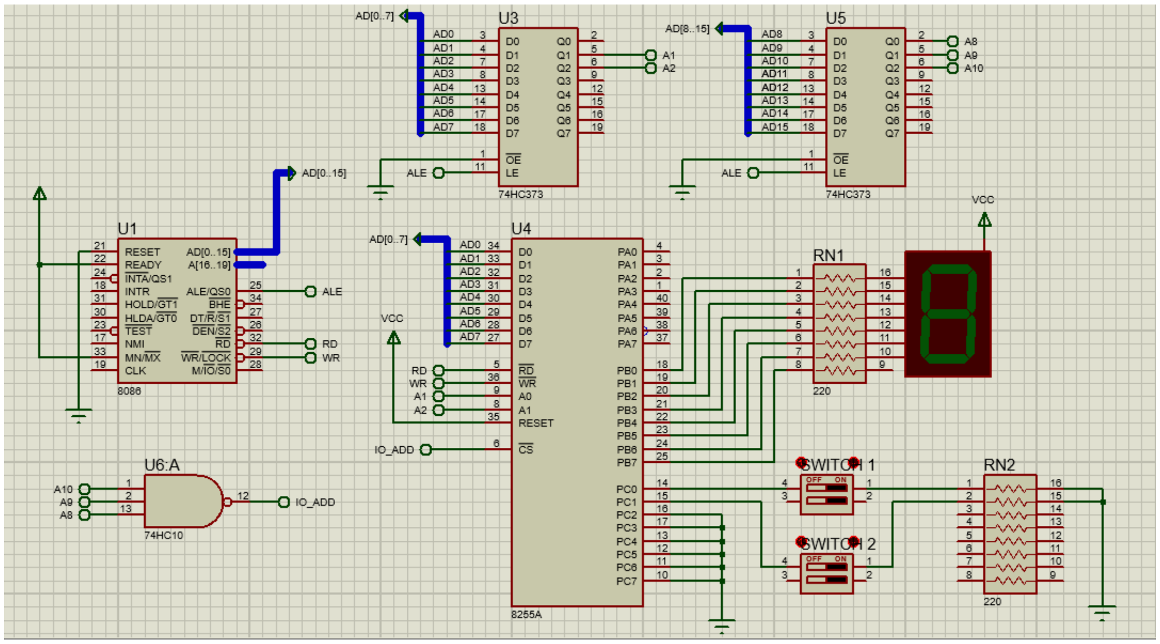
2. RESET PIN OF 8255 IN HIGH STATE:

When the RESET pin of the 8255 is set high, it triggers a reset operation, initializing the device to its default state. This entails resetting the internal control registers, including the control word register, to their default values, ensuring a known starting point for configuration. Additionally, all I/O ports (Port A, Port B, and Port C) are typically reset, along with any output latches, clearing them of any previous data. This process also involves resetting internal logic circuits and registers to prepare the device for subsequent commands and operations.

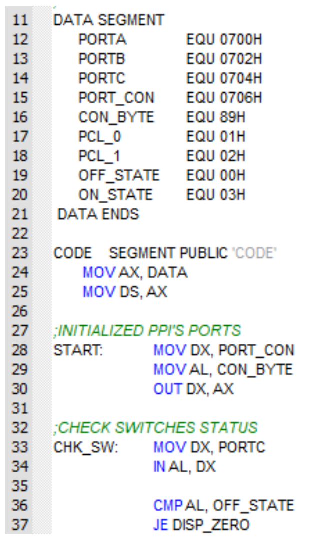
Simultaneously, activating the RESET pin ensures that the 8255 begins in a clean and initialized state, ready for further configuration and integration within the system. This comprehensive reset operation ensures that the device is primed for proper functionality, providing a reliable foundation for subsequent operations and interactions within the microprocessor-based system.

3. SCHEMATICS, PROGRAM LISTING AND OUTPUTS:

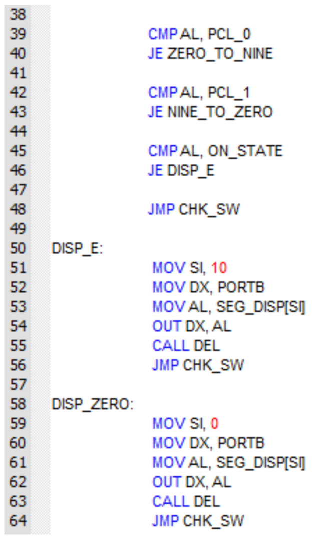
SCHEMATIC DIAGRAM:



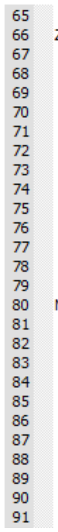
PROGRAM LISTING:



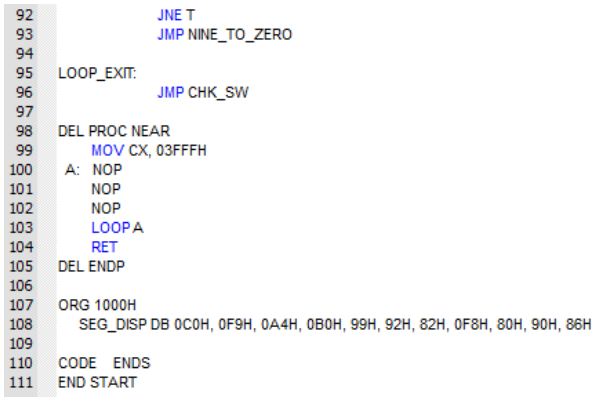
```
DATA SEGMENT
    PORTA          EQU 0700H
    PORTB          EQU 0702H
    PORTC          EQU 0704H
    PORT_CON       EQU 0706H
    CON_BYTE       EQU 89H
    PCL_0          EQU 01H
    PCL_1          EQU 02H
    OFF_STATE      EQU 00H
    ON_STATE       EQU 03H
 DATA ENDS

CODE    SEGMENT PUBLIC 'CODE'
     MOV AX, DATA
     MOV DS, AX

;INITIALIZED PPI'S PORTS
START:          MOV DX, PORT_CON
                MOV AL, CON_BYTE
                OUT DX, AX

;CHECK SWITCHES STATUS
CHK_SW:         MOV DX, PORTC
                IN AL, DX

                CMP AL, OFF_STATE
                JE DISP_ZERO
```



```

                CMP AL, PCL_0
                JE ZERO_TO_NINE

                CMP AL, PCL_1
                JE NINE_TO_ZERO

                CMP AL, ON_STATE
                JE DISP_E

                JMP CHK_SW

DISP_E:
                MOV SI, 10
                MOV DX, PORTB
                MOV AL, SEG_DISP[SI]
                OUT DX, AL
                CALL DEL
                JMP CHK_SW

DISP_ZERO:
                MOV SI, 0
                MOV DX, PORTB
                MOV AL, SEG_DISP[SI]
                OUT DX, AL
                CALL DEL
                JMP CHK_SW
```



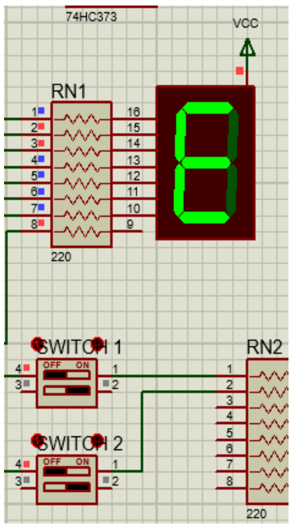
```
ZERO_TO_NINE:
                MOV SI, 0
  D:
                MOV DX, PORTB
                MOV AL, SEG_DISP[SI]
                OUT DX, AL
                CALL DEL
                CALL DEL
                CALL DEL
                INC SI
                CMP SI, 10
                JNE D
                JMP ZERO_TO_NINE

NINE_TO_ZERO:
                MOV SI, 9

  T:
                MOV DX, PORTB
                MOV AL, SEG_DISP[SI]
                OUT DX, AL
                CALL DEL
                CALL DEL
                CALL DEL
                DEC SI
                CMP SI, 0
```



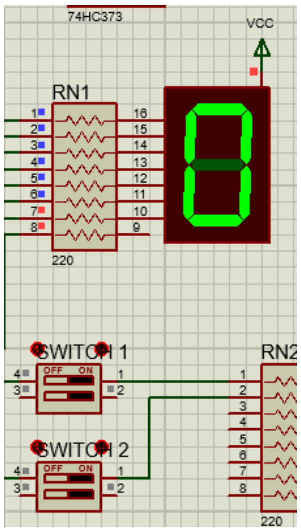
```
                JNE T
                JMP NINE_TO_ZERO

LOOP_EXIT:
                JMP CHK_SW

DEL PROC NEAR
      MOV CX, 03FFFH
  A:  NOP
      NOP
      NOP
      LOOP A
      RET
DEL ENDP

ORG 1000H
   SEG_DISP DB 0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H, 86H

CODE    ENDS
END START
```

OUTPUT:

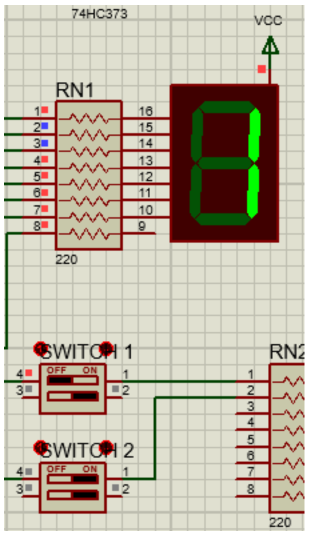
SW2 (ON) – SW1 (ON):



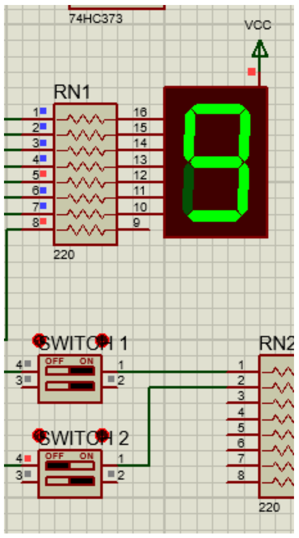
SW2 (OFF) – SW1 (OFF):



SW2 (OFF) – SW1 (ON):



SW2 (ON) – SW1 (OFF):



CONCLUSION

Throughout this laboratory experiment, I gained a deeper understanding of the 8255 (PPI) mode of operations and its I/O ports. By actively engaging with the hardware and observing its functionalities firsthand, I familiarized myself with the intricacies of the 8255 and its significance in digital electronics. Interacting with the 8255's input and output operations provided valuable insight into its role within microprocessor-based systems.

One of the primary objectives of this lab was to configure the control byte register of the 8255. Through experimentation and manipulation of this register, I learned how to control the mode of operation and directionality of the 8255's ports. This hands-on experience enhanced my understanding of digital I/O interfacing concepts, empowering me to effectively utilize the 8255 in future projects and applications.

Moreover, this lab facilitated discussions on interfacing the 8255 to MPU-based systems. By exploring the theoretical principles and practical implementation techniques, I gained valuable insights into the protocols and connections necessary for seamless integration. Understanding the intricacies of interfacing is essential for anyone working with embedded systems and digital design, and this lab provided a solid foundation in this area.

Additionally, I had the opportunity to write an assembly program to control I/O connected to the 8255. By translating my understanding of the 8255's operation into code, I honed my programming skills and reinforced my comprehension of digital I/O control. This practical programming exercise equipped me with valuable skills applicable to real-world scenarios, where efficient control of peripheral devices is crucial.

REFERENCES

Online Sources:

8255A - Programmable Peripheral Interface. (n.d.). Tutorialspoint.com. Retrieved February 24, 2024, from https://www.tutorialspoint.com/microprocessor/microprocessor_intel_8255a_programmable_peripheral_interface.htm

Agarwal, T. (2023, January 16). 8255 Microprocessor : Architecture, Working, Interfacing & its uses. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/8255-microprocessor/

Programmable peripheral interface 8255 - GeeksforGeeks. (n.d.). Geeksforgeeks.org. Retrieved February 24, 2024, from https://www.geeksforgeeks.org/programmable-peripheral interface-8255/amp/

(N.d.). Github.Io. Retrieved February 24, 2024, from https://it gecbh.github.io/2020/resources/ MICROPROCESSOR/series2/8255.ppt