COLEGIO DE MUNTINLUPA

DEPARTMENT OF COMPUTER ENGINEERING

COEN 3211 - Microprocessors Lab

Design of Parallel IO Circuit with Latch and Buffer

Laboratory Experiment No. 4

Grade

STUDENT NAME : **DAVID, Raven A.**
STUDENT NUMBER : **20202011637**

DATE PERFORMED : **21 February 2024**
DATE SUBMITTED : **22 February 2024**

Engr. Ricrey E. Marquez, PCpE
(Lab Instructor)

PRINCIPLES

In 8086 system, there are two schemes employed for I/O addressing: Isolated I/O and Memory I/O. The comparison is shown in Table 4-1.

Table-1. Comparison between isolated and memory mapped I/O

Isolated I/O	Memory mapped I/O
1. I/O devices are treated separate from memory.	1. I/O devices are treated as part of memory.
2. Full 1 MB address space is available for use as memory.	2. Full 1 MB cannot be used as memory since I/O devices are treated as part of memory.
3. Separate instructions are provided in the instruction set to perform isolated I/O input-output operations. These maximise I/O operations.	3. No separate instructions are needed in this case to perform memory mapped I/O operations. Hence, the advantage is that many instructions and addressing modes are available for I/O operations.
4. Data transfer takes place between I/O port and AL or AX register only. This is certainly a disadvantage.	4. No such restriction in this case. Data transfer can take place between I/O port and any internal register. Here, the disadvantage is that it somewhat slows the I/O operations.

In isolated I/O scheme, memory and I/O are treated separately. The 1 MB address space can be treated as memory address space ranging from 00000H to FFFFFH, while the address range from 00000H to 0FFFFH (i.e., 64 KB I/O addresses) can be treated as I/O address space as shown below in Figure 4-1. It should be remembered that two consecutive memory or I/O addresses could be accessed as a word-wide data.

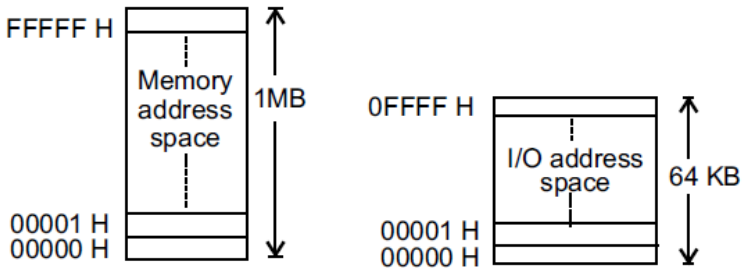


Figure 4-1. Isolated I/O (a) memory address space (b) I/O address space

The 8086 based **MIN** mode I/O interface is shown below in Figure 4-2. It is seen that the lower two bytes AD_0 – AD_{15} are used for input/output data transfers. The interface circuitry performs the following tasks:

- Selecting the particular I/O port,
- Synchronize data transfer,
- Latch the output data,
- Sample the input data, and
- Voltage levels between the I/O devices and 8086 are made compatible.

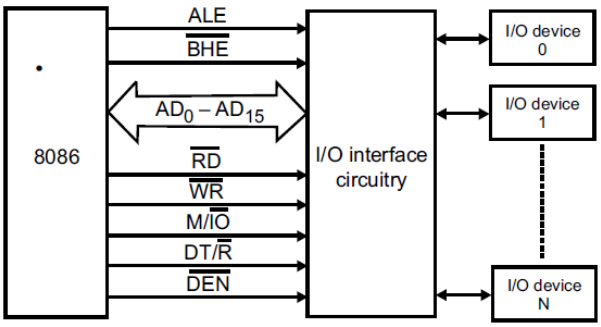


Figure 4-2. The Minimum mode 8086 system I/O interface

For 8086 based systems, isolated I/O is used with **IN** and **OUT** instructions. The **IN** and **OUT** instructions are of two types: direct I/O instructions and variable I/O instructions. The different types of instructions are tabulated in Table 4-2.

Table 4-2. Input/output instructions

Mnemonic	Meaning	Format	Operation
IN	Input direct	IN Acc, Port	(Acc)←(Port) Acc = AL or AX
	Input indirect (variable)	IN Acc, DX	(Acc)←((DX))
OUT	Output direct	OUT Port, Acc	(Port)←(Acc)
	Output indirect (variable)	OUT DX, Acc	((DX))←(Acc)

Only **AL** (for 8-bits) or **AX** (for16-bits) register is involved in data transfer involving the 8086's CPU and I/O devices, thus they are also known as *accumulator I/O*.

The differences between the two types of instructions are tabulated below in Table 4-3.

Table 4-3. Differences between direct and variable I/O instructions

Direct I/O	Variable I/O
1. Involves 8-bit address as part of instruction	1. Involves 16-bit address as part of instruction. This resides in DX register. It must be borne in mind that the value in DX register is not an offset, but the actual port address.
2. Can access a maximum of 2^8 = 255 byte addresses.	2. Can access a maximum of 2^{16} = 64 KB of addresses.

Example of direct I/O instruction:

For IN instruction:

```
IN AL, 0200H
```

On execution, the contents of the byte wide I/O port at address location 0200H will be put into AL register.

For OUT instruction:

```
OUT 0300H, AL
```

Example of variable I/O instruction:

For IN instruction:

```
MOV DX, 0200H
IN AL, DX
```

On execution, at first DX register is loaded with the input port having address 0200FH. The second instruction ensures that the port content is moved over to AL register.

For OUT instruction:

```
MOV DX, 0300H
MOV AL, 65H
OUT DX, AL
```

OBJECTIVES AND MATERIALS

Objectives:

After this lab experiment, student should be able to:

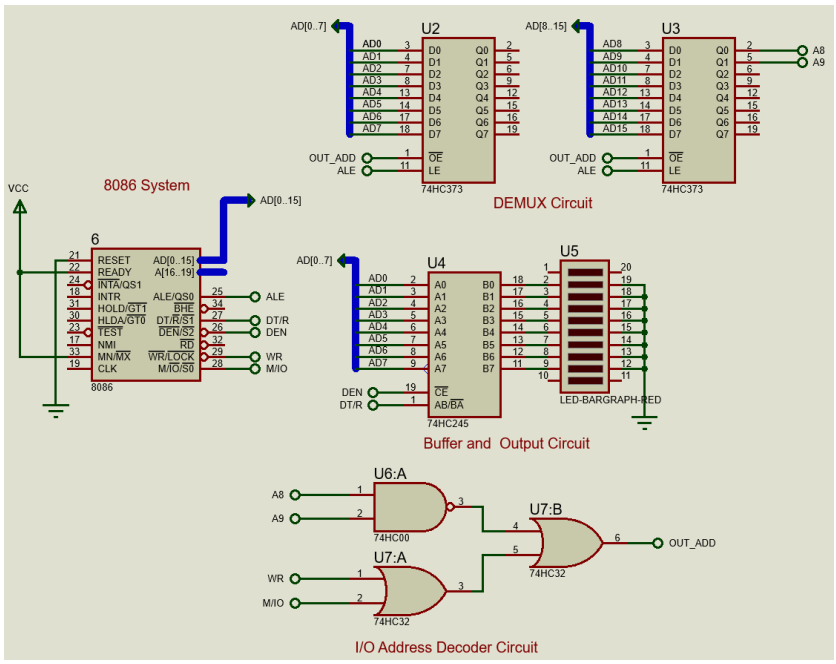
1. understand the way the CPU (8086) send data to an output device,
2. demonstrate the way the CPU (8086) receives data to an input device;
3. design an 8-bit parallel input and output port circuit; and
4. create a program that will enable the CPU to send data to an output device or received data from input device.

Materials:

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

DRILL EXERCISES

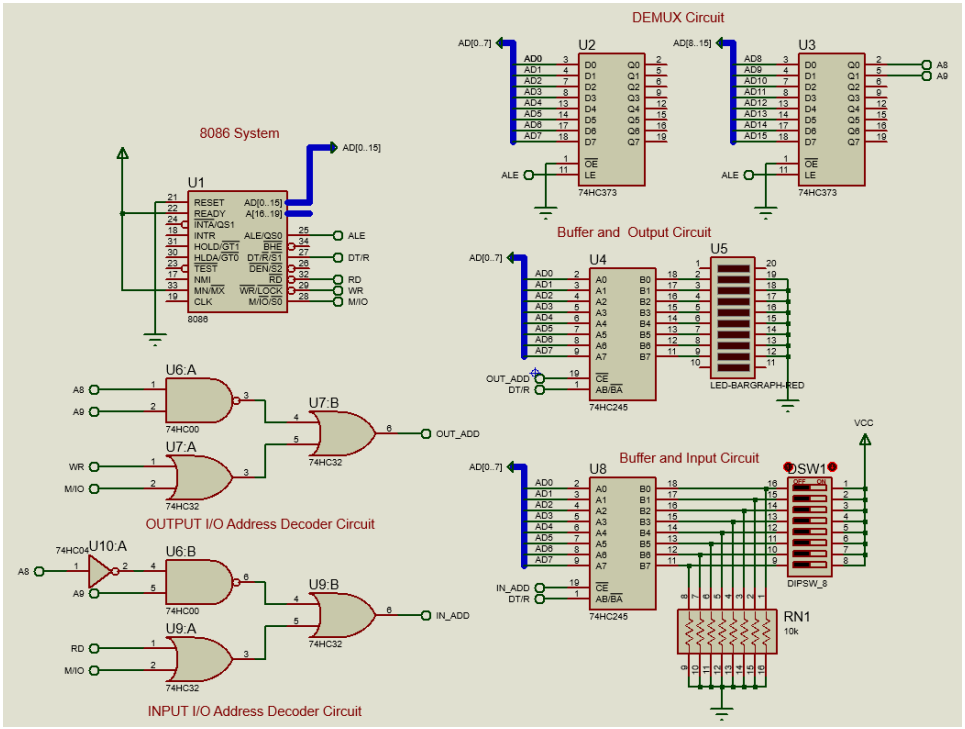
Drill Exercise 1 – Given the schematic diagram and assembly language source code illustrated in Figure 4-3, re-type and test the program. Save as **`COEN3211-x_Drill4_1`**




```
;===============================================================
; Main.asm file generated by New Project wizard
;
; Created:   Thu Apr 22 2021
; Processor: 8086
; Compiler:  MASM32
;
; Before starting simulation set Internal Memory Size
; in the 8086 model properties to 0x10000
;===============================================================
DATA SEGMENT
   OUT_ADD equ 0300H
   LED_SEQ1 equ 0CEH
   LED_SEQ2 equ 65H
DATA ENDS

CODE    SEGMENT PUBLIC

          MOV AX, DATA
          MOV DS, AX
          CALL clr_regs

START:     MOV DX, OUT_ADD
AGAIN:     MOV AL, LED_SEQ1
           OUT DX, AL
           CALL delay
           MOV AL, LED_SEQ2
           OUT DX, AL
           CALL delay
           JMP AGAIN

clr_regs PROC NEAR
      XOR AX, AX
      XOR BX, BX
      XOR CX, CX
      XOR DX, DX
      RET
clr_regs endp

delay PROC NEAR
         MOV CX, 000FH
      x:  NOP
          NOP
          NOP
          LOOP x
          RET
delay ENDP

CODE    ENDS

END START
```

Figure 4-3. Schematic diagram and assembly source of Drill Exercise 1

Drill Exercise 2 – Given the schematic diagram and assembly language source code illustrated in Figure 4-4, re-type and test the program. Save as **`COEN3211-x_Drill4_2`**




```
;===================================================================
; Main.asm file generated by New Project wizard
;
; Created:   Thu Apr 22 2021
; Processor: 8086
; Compiler:  MASM32
;
; Before starting simulation set Internal Memory Size
; in the 8086 model properties to 0x10000
;===================================================================
DATA SEGMENT

   OUT_ADD equ 0300H
   IN_ADD equ 0200H

DATA ENDS

CODE    SEGMENT PUBLIC
        CALL clr_regs
        MOV AX, DATA
        MOV DS, AX

START: MOV DX, IN_ADD
        IN AL, DX
        MOV DX, OUT_ADD
        OUT DX, AL
        CALL delay
        JMP START

clr_regs PROC NEAR
        XOR AX, AX
        XOR BX, BX
        XOR CX, CX
        XOR DX, DX
        RET
clr_regs ENDP

delay PROC NEAR
        MOV CX, 000FH
    x: NOP
        NOP
        NOP
        LOOP x
        RET
delay ENDP

CODE    ENDS

END START
```

Figure 4-4. Schematic diagram and assembly source of Drill Exercise 2

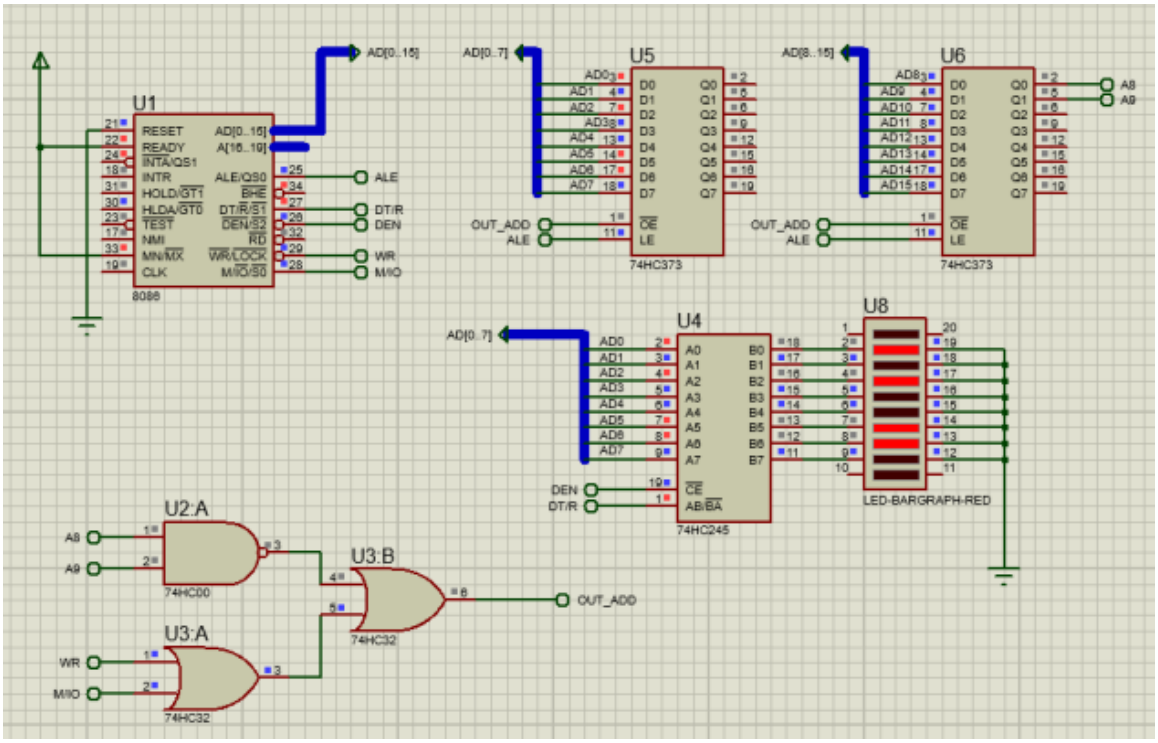
PROGRAM EXERCISES

Program Exercise 4-1. Design a parallel output circuit with an address of 0700H and create an assembly program that will turn on one LED at time from LSB to MSB. Save as **`COEN3211-x_ProgExer4_1`**

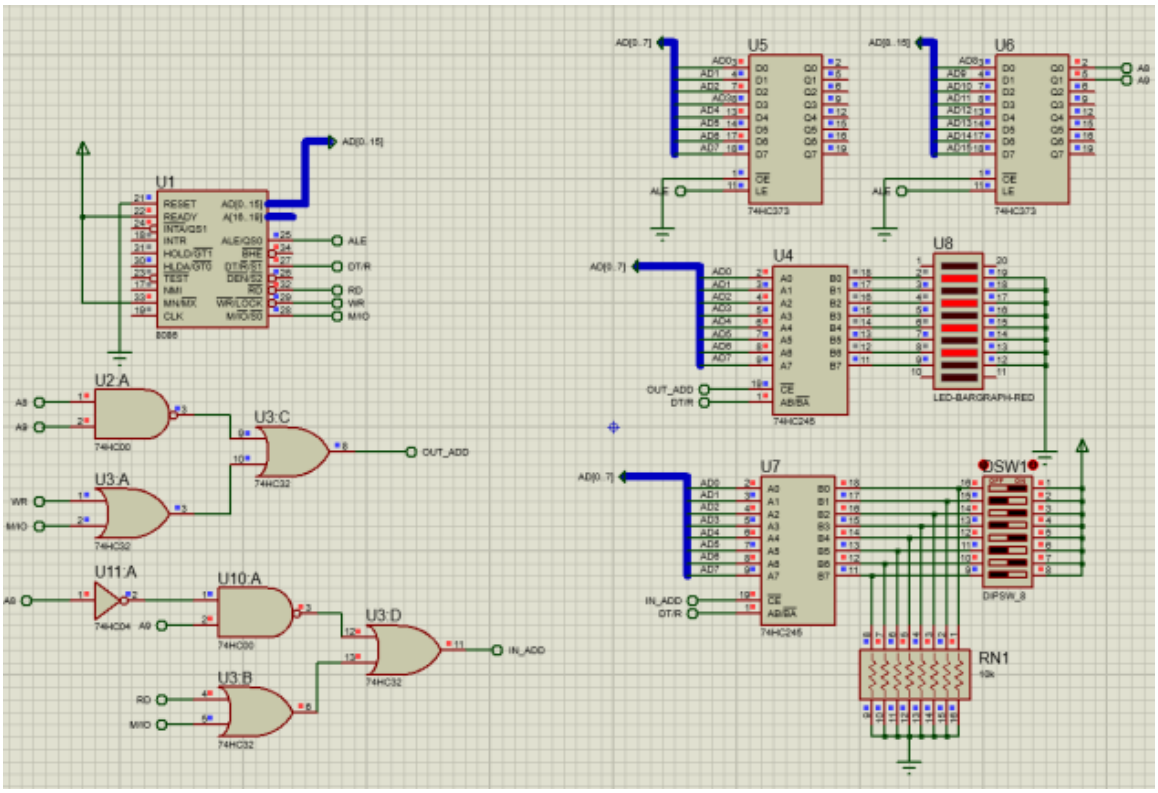
Program Exercise 4-2. Design a parallel input and output circuit with an address of 0600H and 0700h respectively. Create an assembly program that will read the status of the switches then display the 1's complements of the input data. Save as **`COEN3211-x_ProgExer4_2`**

DATA RESULTS

Drill Exercise 4-1. Test output

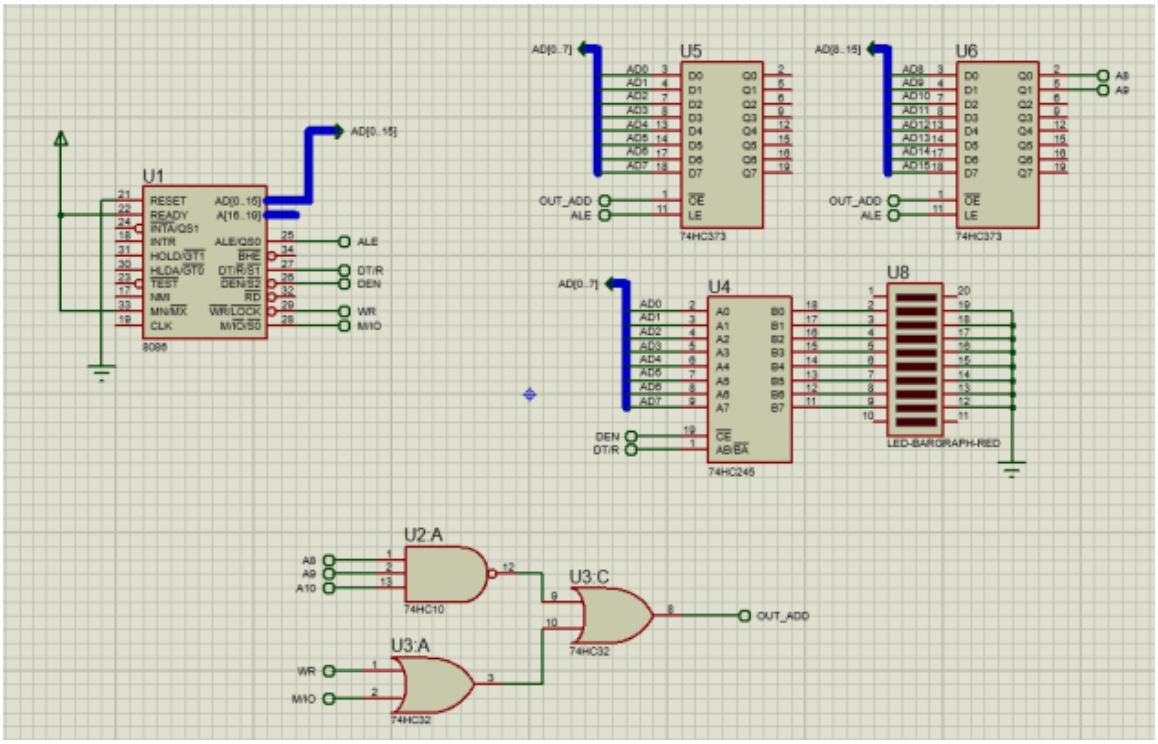


Drill Exercise 4-2. Test output



Program Exercise 4-1. Schematic diagram, program listing, and outputs

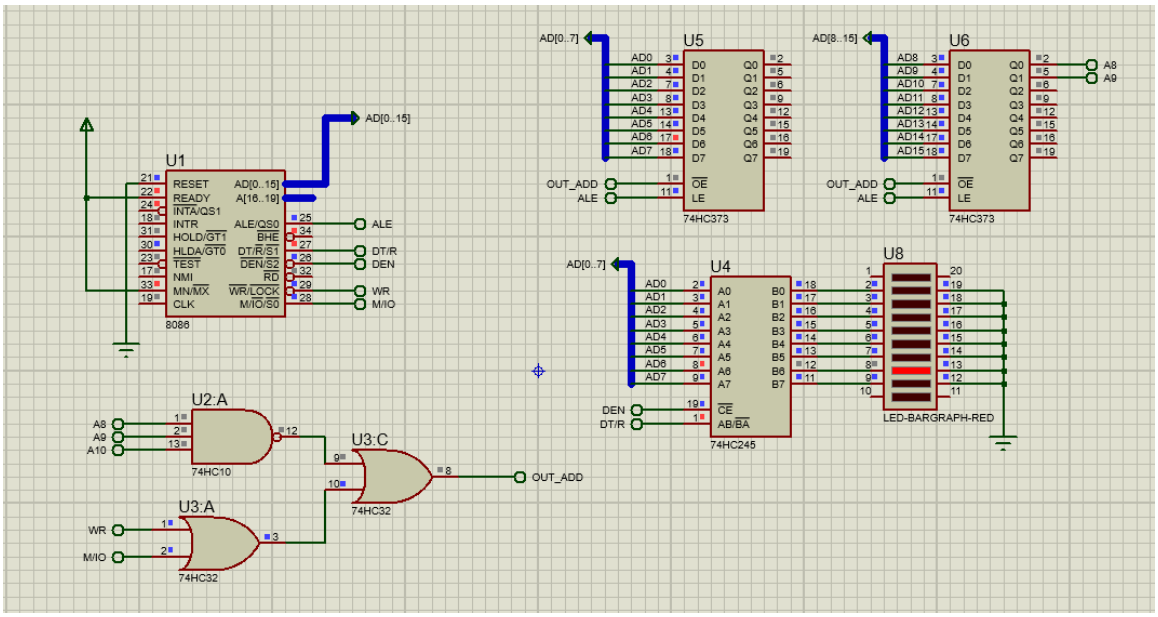
Schematic diagram:



Program Listing:

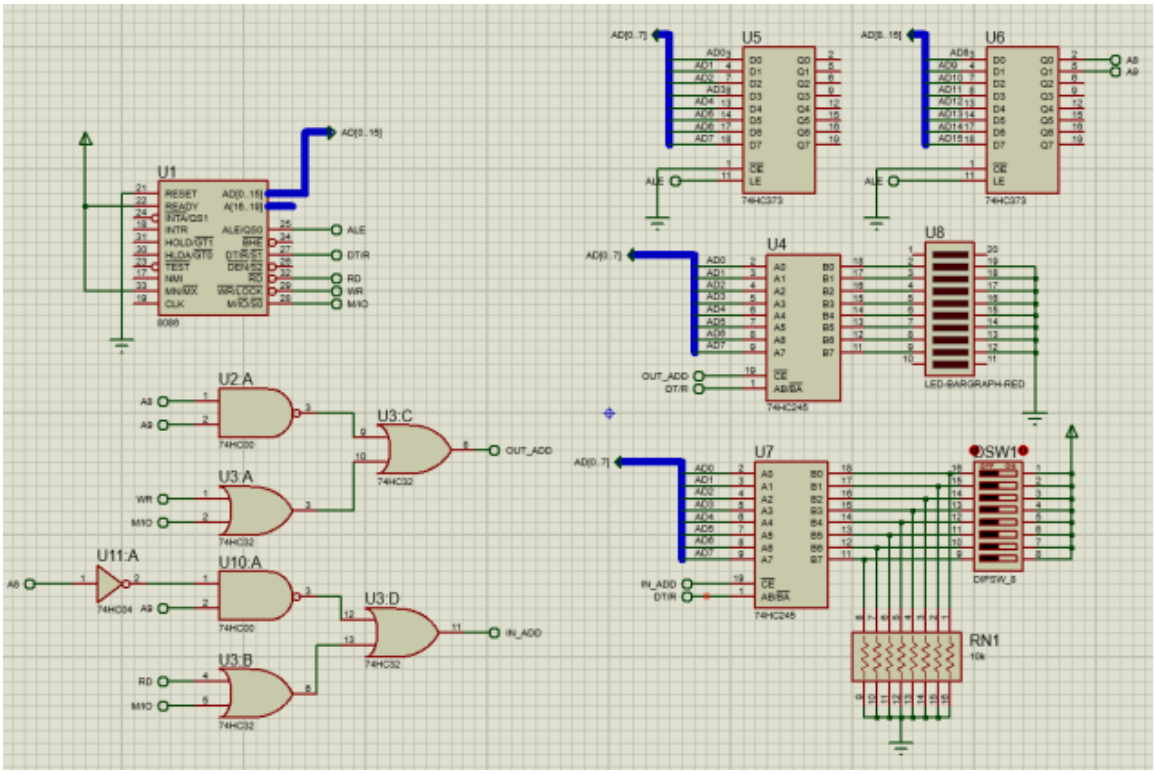
```
DATA SEGMENT
      OUT_ADD    equ  0700H
      LED_SEQ1   equ  01H
DATA ENDS

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


START:  MOV DX,  OUT_ADD
AGAIN:  MOV AL, LED_SEQ1
X:      OUT DX,  AL
        CALL DELAY

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

CLR_REGS PROC NEAR
      XOR AX, AX
      XOR BX, BX
      XOR CX, CX
      XOR DX, DX
      RET
CLR_REGS ENDP

DELAY PROC NEAR
      MOV CX, 000FH
   Y: NOP
      NOP
      NOP
      LOOP Y
      RET
DELAY ENDP

CODE   ENDS
END START
```

Output:



Program Exercise 4-2. Schematic diagram, program listing, and outputs

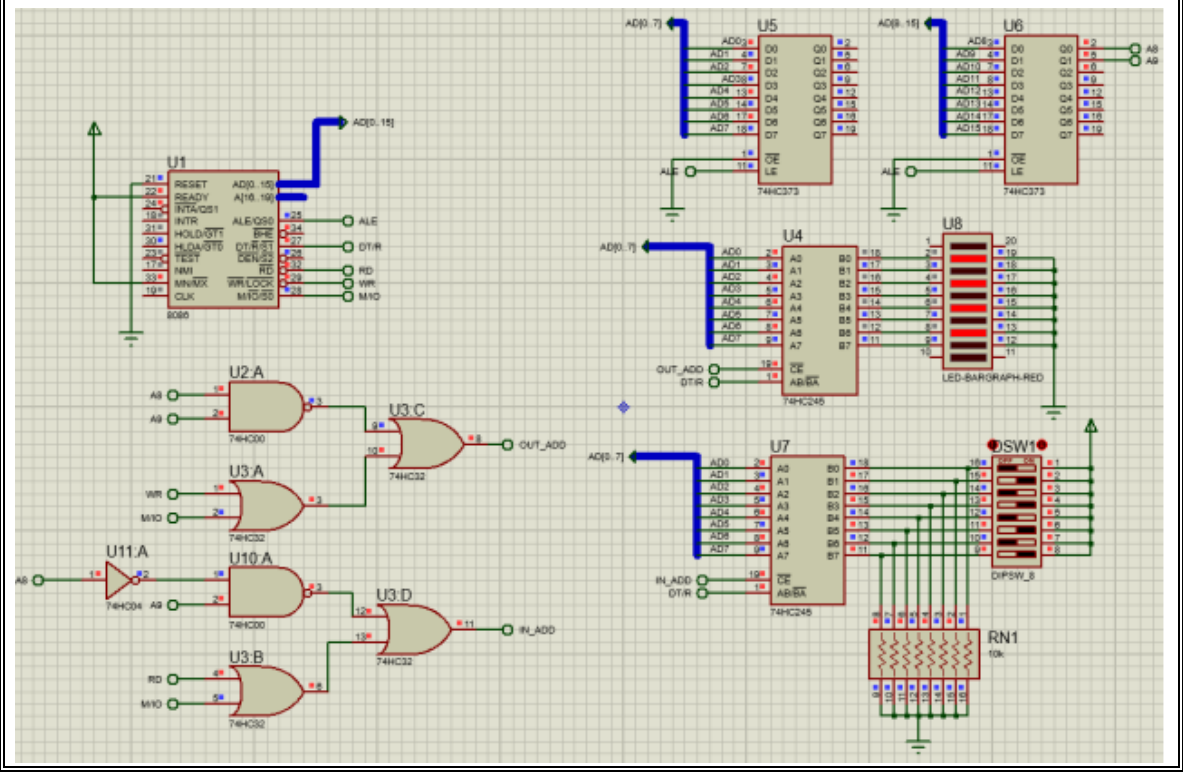
Schematic diagram:



Program Listing:

```
DATA SEGMENT
      OUT_ADD     equ  0700H
      IN_ADD      equ  0600H
DATA ENDS

CODE   SEGMENT PUBLIC
       CALL CLR_REGS
       MOV AX, DATA
       MOV DS, AX


START:   MOV DX, IN_ADD
         IN AL,  DX
         MOV DX, OUT_ADD
         NOT AL
         OUT DX, AL
         CALL DELAY
         JMP START

CLR_REGS PROC NEAR
       XOR AX, AX
       XOR BX, BX
       XOR CX, CX
       XOR DX, DX
       RET
CLR_REGS ENDP

DELAY PROC NEAR
       MOV CX, 000FH
    X: NOP
       NOP
       NOP
       LOOP X
       RET
DELAY ENDP

CODE   ENDS
END START
```

Output:



DATA ANALYSIS

In Drill Exercise 4-1, we successfully constructed and simulated a Parallel Output Circuit using Proteus ISIS. The design incorporated a DEMUX circuit (74373) linked to the AD pins 0-7 of the 8086 system. Another 74373 facilitated full connection of the AD pins of the 8086 system, with the second 74373 connecting AD pins 8-15, utilizing a bus for organized connectivity. Additionally, a BUF Circuit was integrated, linked to the IO device, enabling the test output of the IO circuit's address decoder, set at 0300H. With the provided machine code, we generated a test output for Drill Exercise 4-1.

Drill Exercise 4-2 differed from the first drill, as it required designing both Parallel input and output circuits in Proteus ISIS. While the design resembled Drill Exercise 4-1, it included a Parallel input for an additional input device, necessitating another IO circuit address decoder with an address of 0200h. With control over the test output, facilitated by the input device (a switch), the test output for this exercise indicated that when the switch was off (or active low), the output LED would illuminate.

Program Exercise 4-1 closely mirrored the first drill exercise, focusing on producing a schematic of a Parallel Output circuit. Utilizing the schematic from the initial drill exercise, we aimed to meet the criteria of having an IO address of 0700H, necessitating modifications to the provided assembly code. Additionally, we needed to ensure that one LED turned on at a time from LSB to MSB. The primary modification involved adjusting the assembly language and utilizing a 7410 IC (3-input NAND gate) to meet these requirements.

Program Exercise 4-2 entailed designing a similar circuit to Drill Exercise 4-2, involving a Parallel Input and Output Circuit. Given the design provided in Drill Exercise 4-2, we focused on designing parallel input and output circuits with addresses of 0600H and 0700H, respectively. We created an assembly program to read the status of the switches and display the 1’s complement of the input data. The primary modification here was adding a NOT command in the code to compute the 1’s complement of the input data.

QUESTIONS AND ANSWERS

Questions:

1. Draw the input port and output port bus cycle of 8086 and discuss briefly.
2. Design a parallel input and output circuit using latch and buffer having address of 0C00H and 0E00H respectively that will read the switch from parallel input circuit and display the sequence of LEDs sequence below. Save as **`SURNAME_COEN3211_QA2.asm`**.

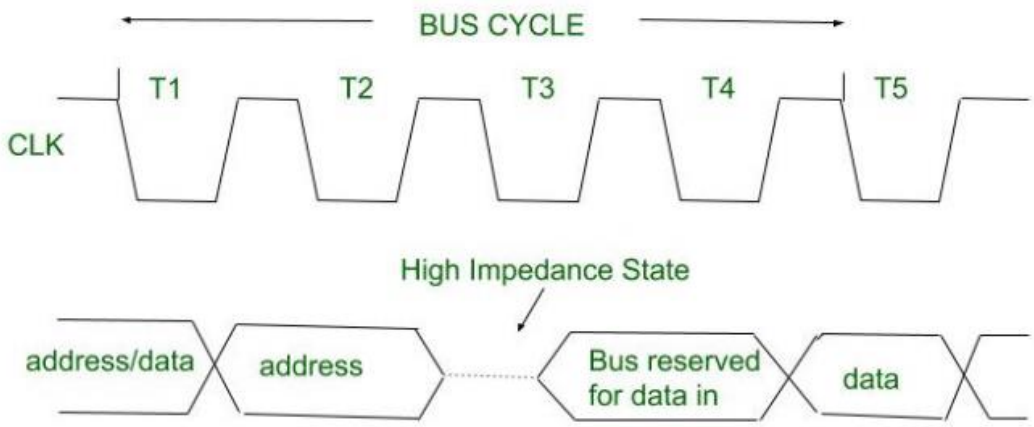
SWITCH STATUS	LED SEQUENCE
SW	
Off	Turn on LED one at a time from MSB to LSB
On	Turn on LED one at a time from LSB to MSB

Answers:

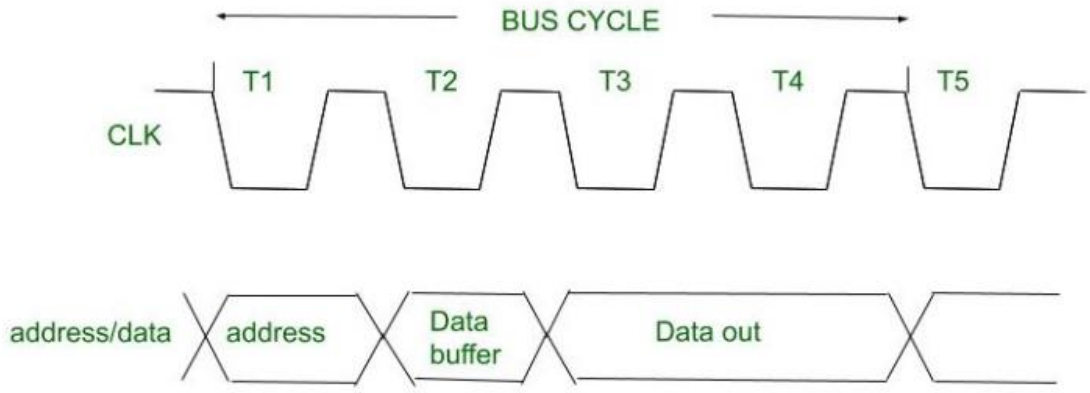
1. Bus Cycles of 8086 Microprocessor

   The bus cycle, also termed as the machine cycle, is integral to the functioning of the 8086 microprocessor, enabling access to memory, input/output (I/O) devices, and the Interrupt controller. This cycle entails a sequence of events initiated by outputting an address onto the system address bus, followed by a read or write data transfer, while the microprocessor generates control signals to regulate bus direction and timing. Comprising four distinct clock periods - T1, T2, T3, and T4 states - this cycle in a 5-MHz 8086 system results in a duration of 800 nanoseconds (ns), calculated by multiplying the standard period of one clock cycle (200 ns) by the four clock states. Understanding this structured framework is paramount for efficient data transfer and system operation across diverse computing applications.

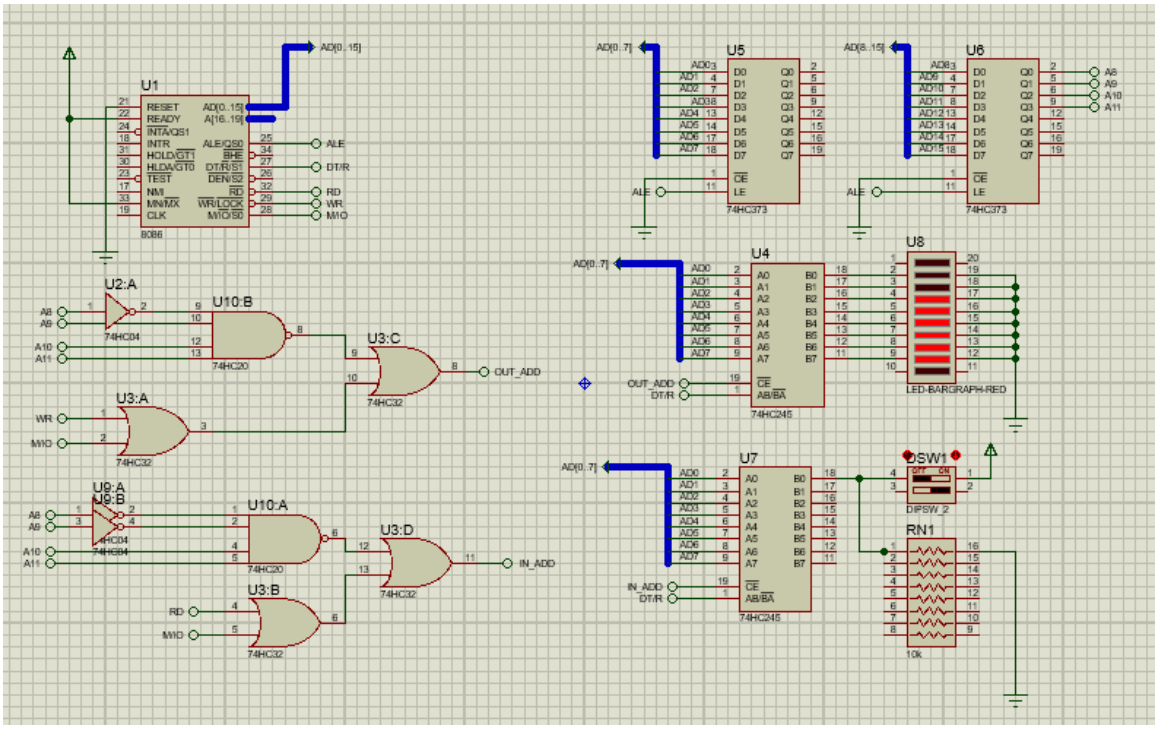
During a read cycle, the 8086 microprocessor begins by placing the desired address onto the address bus in T1. Subsequently, in T2, the bus transitions into a high impedance state. During T3 and T4, the data intended for reading must be present on the bus. In T3, the bus is designated as "reserved for data in," preparing it to receive the incoming data. Finally, in T4, the actual data is read from the bus. This structured sequence ensures efficient and synchronized data retrieval within the read cycle of the 8086 microprocessor, facilitating smooth communication between the processor and memory or peripheral devices.



In a write memory cycle, the 8086 microprocessor initiates the process by placing the desired address onto the address bus in the T1 state. Subsequently, during T2, the CPU transfers data onto the data bus, maintaining this data throughout the T3 and T4 states. This data is then written out to either memory or I/O devices. This structured sequence ensures the proper synchronization of data transfer within the write cycle of the 8086 microprocessor, facilitating effective communication between the CPU and memory or I/O devices.



2. Schematic Diagram:

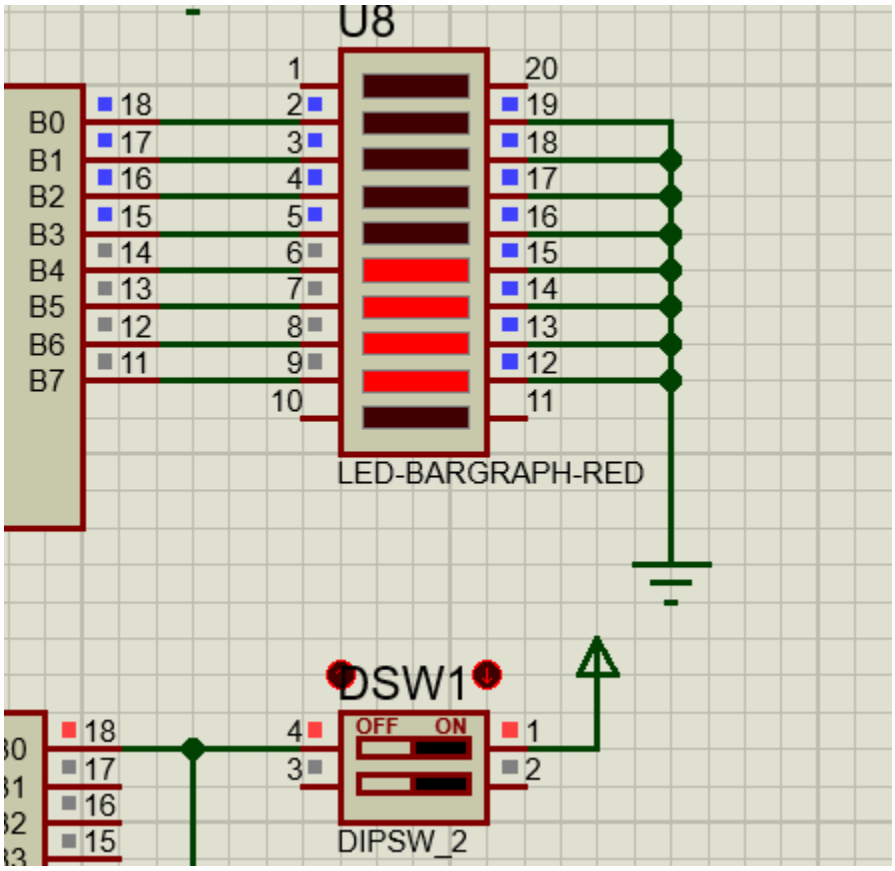


Program Listing:

```
DATA SEGMENT
      OUT_ADD     equ  0E00H
      IN_ADD      equ  0C00H
      LED_SEQ     equ  80H
      LED_SEQ2    equ  00H
DATA ENDS

CODE   SEGMENT PUBLIC
       CALL CLR_REGS
       MOV AX, DATA
       MOV DS, AX


START:   MOV DX, IN_ADD
         IN AL,  DX

         CMP AL, 00H
         JNE LSB

         JMP MSB

MSB:     IN AL, LED_SEQ
         MOV DX, OUT_ADD
         NOT AL
OUT_1:   OUT DX, AL
         CALL DELAY

         SHR AL, 01H
         CMP AL, 00H
         JNE  OUT_1

         JMP START

LSB:     IN AL, LED_SEQ2
         MOV DX, OUT_ADD
         NOT AL
OUT_2:   OUT DX, AL
         CALL DELAY

         SHL AL, 01H
         CMP AL, 80H
         JNE  OUT_2

         JMP START

CLR_REGS PROC NEAR
       XOR AX, AX
       XOR BX, BX
       XOR CX, CX
       XOR DX, DX
       RET
CLR_REGS ENDP

DELAY PROC NEAR
       MOV CX, 000FH
    X: NOP
       NOP
       NOP
       LOOP X
       RET
DELAY ENDP

CODE   ENDS
END START
```

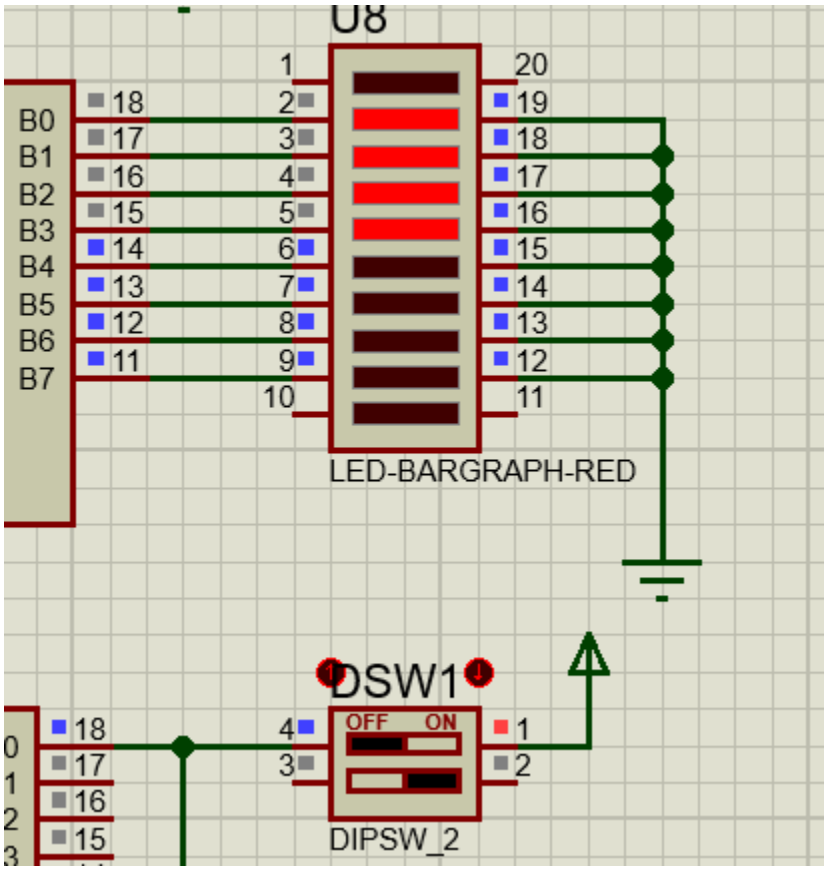
Output (OFF- STATE): MSB TO LSB:

Pin 9 to 2:



Output (ON - STATE): LSB TO MSB:

Pin 2 to 9:



CONCLUSION

Students now possess a comprehensive understanding of how the CPU, specifically the 8086, sends data to output devices. Through practical application and simulation, they have gained insights into the underlying mechanisms of data transmission within digital circuits. This knowledge forms a crucial foundation for further exploration into CPU-device interactions and broader concepts within computer architecture.

Through hands-on experimentation, students have demonstrated proficiency in showcasing how the CPU, particularly the 8086, receives data from input devices. By interfacing with input devices and analyzing the data flow, they have deepened their understanding of signal processing and data reception mechanisms. This practical exposure equips them with valuable skills applicable to various hardware interfacing scenarios.

One of the key outcomes of this lab is the ability for students to design 8-bit parallel input and output port circuits. By integrating components such as DEMUX circuits and addressing the intricacies of parallel data transfer, students have honed their circuit design skills. This hands-on experience empowers them to tackle real-world challenges in developing robust and efficient input-output systems.

Lastly, students have developed the capability to create programs facilitating CPU-device interaction. By writing assembly programs tailored to send data to output devices or receive data from input devices, they have translated theoretical knowledge into practical implementation. This skill is fundamental in software-hardware integration, enabling students to bridge the gap between software logic and hardware functionality effectively.

REFERENCES

Online Articles:

Advance Computer Architecture Computer Science. (n.d.). Retrieved February 18, 2024, from Zeepedia.com website: https://www.zeepedia.com/read.php?designing_parallel_input_output_ports_sad_nuxi_address_decoder_delay_interval_&b=1&c=24

Microprocessor - I/O Interfacing Overview. (n.d.). Retrieved February 18, 2024, from Tutorialspoint.com website: https://www.tutorialspoint.com/microprocessor/microprocessor_io_interfacing_overview.htm

The 8086 Input/output Interface. (n.d.). Retrieved February 18, 2024, from Edu.iq website: https://uomustansiriyah.edu.iq/media/lectures/5/5_2019_10_30!11_07_31_PM.pdf

(N.d.). Retrieved February 18, 2024, from Wsu.edu website: https://eecs.wsu.edu/~ee314/lectures/lecture21.pdf