LABORATORY 1

INTRODUCTION TO 8085 MICROPROCESSOR DEVELOPMENT SYSTEM BOARD

1. INTRODUCTION TO 8085 MICROPROCESSOR DEVELOPMENT SYSTEMS.

The basic components of the 8085 Microprocessor Development System board consist of

- 8085 8-bit Intel Microprocessor
- 8K Byte memory EPROM - 2764
- 8K Byte memory RAM - 6264
- I/O Device - 8255
- Serial Communication (UART) – 8251
- Monitor Program

The circuit diagram for this system is as Appendix 1. The board is connected to the computer through its serial port. Overall system operation is controlled by the monitor program which resides in the EPROM.

1.1 Memory and I/O Map

This Development System has two different address maps; Memory Map and I/O Map. The Development System can address up to 64K of memory devices and 255 addresses for I/O devices.

1.1.1 Memory Map

Figure 1 shows the memory map for the Development System. The first 8K of memory space 0000H - 1FFFH is allocated for EPROM which contains the monitor program code for the system. Whilst the memory space 2000H - 3FFFH is allocated for RAM which is used for user program, data and stack (Default Setting). User can changed the setting to other memory address by configuring the jumper (J16) on the board.

Address	Memory Use
0000H - 1FFFH	EPROM (8K x 8)
2000H - 3FFFH	RAM (8K x 8)
4000H - 5FFFH	RESERVED
4000H - 5FFFH	RESERVED
6000H - 7FFFH	RESERVED
8000H - 9FFFH	RESERVED
A000H - BFFFH	RESERVED
C000H - DFFFH	RESERVED
E000H - FFFFH	RESERVED

Figure 1 : Memory Map

1.1.2 I/O Map

The Development Board is provided with two input and output devices; Programmable Peripheral Interface (PPI) 8255 and Universal Synchronous Asynchronous Transmitter and Receiver (USART) 8251. Both devices have a unique address where it is decoded as isolated I/O. Figure 2 shows the I/O map for the 8085 Development System.

Address	I/O Use
10H	RESERVED
20H	RESERVED
40H 41H	Data Register USART 8251 Control/Status Register USART 8251
80H 81H 82H 82H	Port A 8255 Port B 8255 Port C 8255 Control Port 8255
90H To FFH	NOT USED

Figure 2 : I/O Map

2. INPUT AND OUTPUT

Programmable Peripheral Interface (PPI) 8255 and Universal Synchronous Asynchronous Transmitter and Receiver (USART) 8251 is an input and output device for the Development System to communicate with the outside world.

2.1 Programmable Peripheral Interface 8255

The 8255 Programmable Peripheral Interface (PPI) is a general purpose interface device which is widely used in microprocessor design. It contains three independent 8 bit ports named Port A, B and C. Port A and B can be programmed as either input or output (all eight line must be same), while port C is split into two 4 bit halves (Port C upper (PC4-PC7) and Port C lower (PC0-PC3)) that can be separately programmed as input or output. Figure 3 shows the internal architecture and pin-out for the 8255 Programmable Peripheral Interface (PPI).

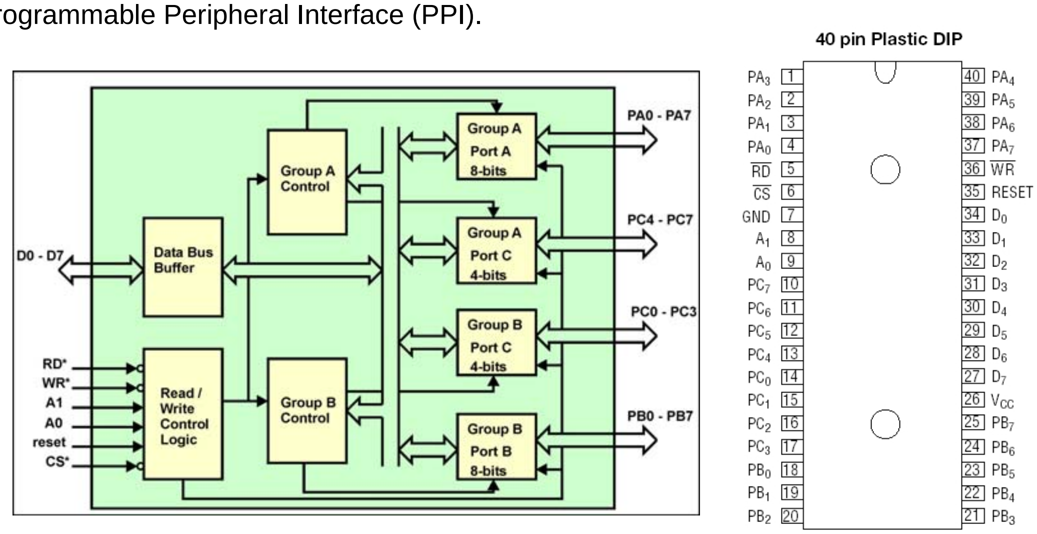


Figure 3 : PPI 8255

There are four registers that control the operations of the PPI and there are mapped to four address locations in the 8085 Development System as shown below:

Port	Address
Port A Register	80H
Port B Register	81H
Port C Register	82H
Control Register	83H

Table 1 shows the details of the signals involved in controlling the PPI.

Table 1 : Signals Controlling PPI.

A0	A1	CS	RD=0	WR=0
0	0	0	Port A to Data bus	Data bus to Port A
0	1	0	Port B to Data bus	Data bus to Port B
1	0	0	Port C to Data bus	Data bus to Port C
1	1	0	-	Data bus to Control Register

The control register is used to configure the PPI into a variety of operation modes. There are three basic modes:

- mode 0 : basic input/output
- mode 1 : strobe input/output
- mode 2 : bidirectional bus

We will only concentrate at mode 0. Further descriptions about mode 1 and 2 can be referred to 'Intel Microsystems Component Handbook'. Mode 0 provides for simple input output operations with no handshaking. This means that data either to or from the port does not depend on other signal for data transfer. Basically, to configure the PPI, a control word must first be sent to the control register. Figure 4 summarized the control word format.

bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
1	**Port A and upper half of Port C**				**Port B & lower half of Port C**		
	MODE		**Port A**	**Port C_{UPPER}**	**MODE**	**Port B**	**Port C_{LOWER}**

Mode Definition Control Byte: Indicate by bit b7=1.

Port A and upper half of Port C (Group A)
Bit b3 to bit b6 control the mode and direction of Group A

b6, b5	mode operation
0 0	mode 0
0 1	mode 1
1 0	mode 2

b4	Port A direction, 0=output, 1=input
b3	upper half of Port C direction, 0=output, 1=input

Port B and lower half of Port C (Group B)
Bit b0 to bit b2 control the mode and direction of Group B

b2	mode operation, 0=mod 0, 1=mod 1
b1	Port B direction, 0=output, 1=input
b0	lower half of Port C direction, 0=output, 1=input

Figure 4 : Control Word Format

The input and output line are connected to a 24 screw type connector, J3, whose pin out is given in Figure 5.

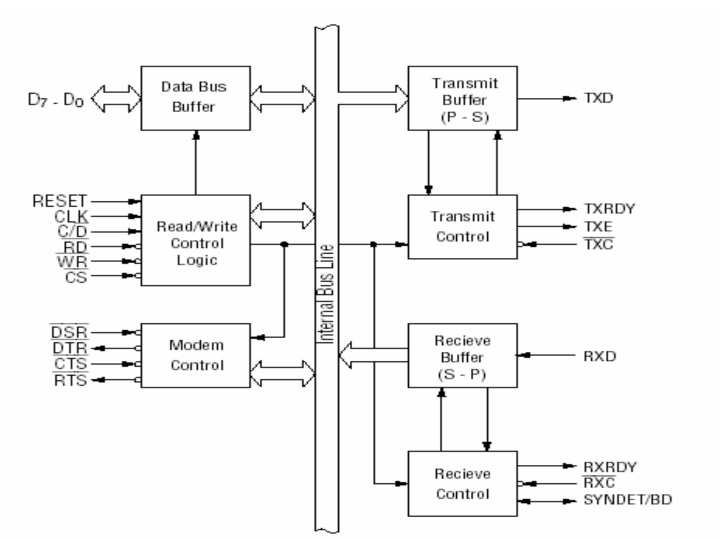
J3 PIN	USE
1	Port A, Bit 0
2	Port A, Bit 1
3	Port A, Bit 2
4	Port A, Bit 3
5	Port A, Bit 4
6	Port A, Bit 5
7	Port A, Bit 6
8	Port A, Bit 7
9	Port B, Bit 0
10	Port B, Bit 1
11	Port B, Bit 2
12	Port B, Bit 3
13	Port B, Bit 4
14	Port B, Bit 5
15	Port B, Bit 6
16	Port B, Bit 7
17	Port C, Bit 0
18	Port C, Bit 1
19	Port C, Bit 2
20	Port C, Bit 3
21	Port C, Bit 4
22	Port C, Bit 5
23	Port C, Bit 6
24	Port C, Bit 7

Figure 5 : J3 Pin Connection

2.2 Universal Synchronous / Asynchronous Receiver and Transmitter (USART) 8251

The RS232 serial communications channel is implemented using a single chip 8251 Universal Synchronous / Asynchronous Receiver and Transmitter (USART). The chip accepts data characters from microprocessor in parallel format and then converts them into a continuous serial data. It also receives serial data and converts them into parallel data for the CPU.

FUNCTIONAL BLOCK DIAGRAM



The 8251 functional configuration is programmed by software. Operation between the USART 8251 and a CPU is executed by program control. Table 2 shows the operation and the signals between a CPU and the USART.

Table 2 : Operation and Signals between CPU and USART

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

The addresses of 8251 in the Development System as shown below

Data Register (Input /output)	40H
Control/Status Register	41H

Before starting data transmission or reception, the 8251 must be initiated with a set of control words generated by the CPU. These control words define the function of 8251. The control words include

- Mode instruction
- Command instruction

Mode instruction has two different formats: one for asynchronous transmission and the other for synchronous transmission. Since in this Development System asynchronous transmission is established, more attention is given to asynchronous format. Bit Configuration of Mode Instruction (Asynchronous) is shown in Figure 6.

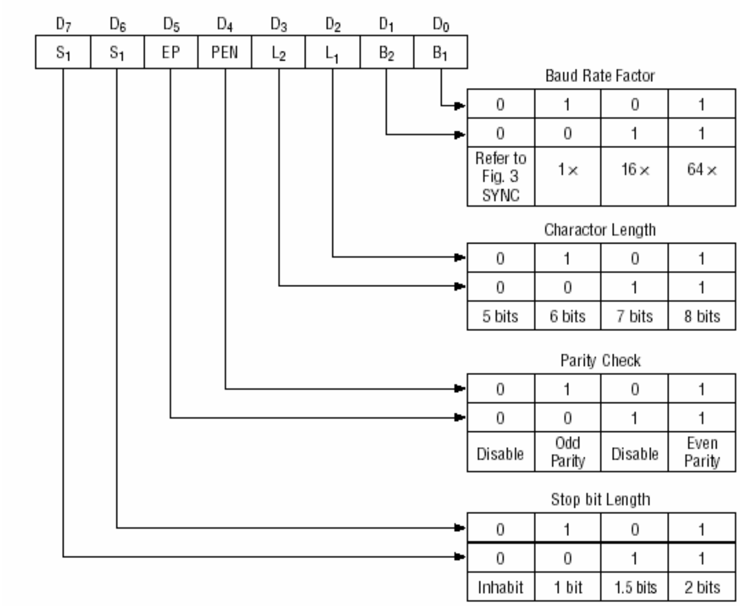


Figure 6 : Bit Configuration of Mode Instruction (Asynchronous) Format

The Command Instruction controls the actual operation of the 8251 by configuring the bit of control register. The functions of command instruction are to enable transmit or receive, reset error flag and configure modem control signal. Figure 7 below shows the bit configuration of Command instruction format.

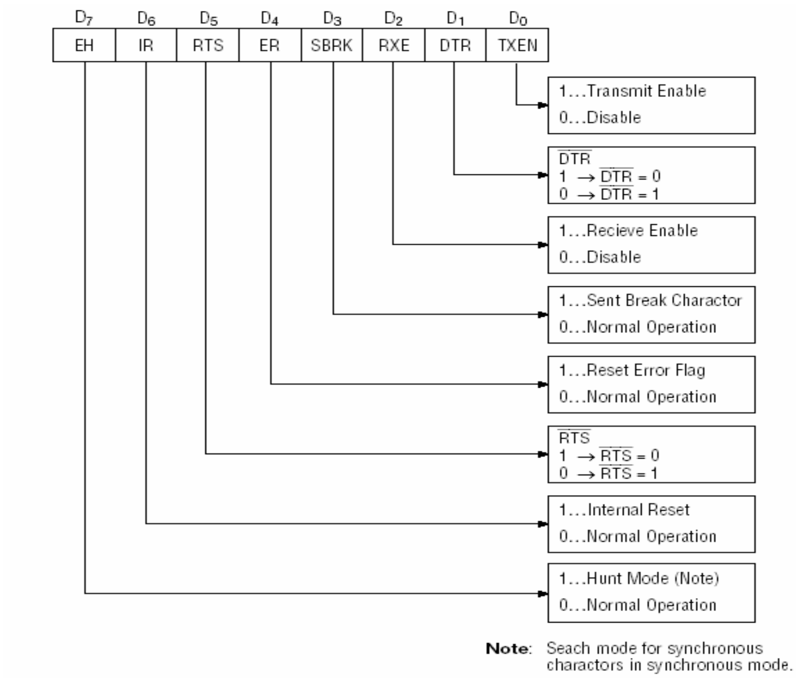


Figure 7 : Bit Configuration of Command Instruction Format

In some applications there is a need to test the status of the 8251 during its operation. The user can read the status of 8251 using standard input operation. The bit configuration of status format is shown in Figure 8.

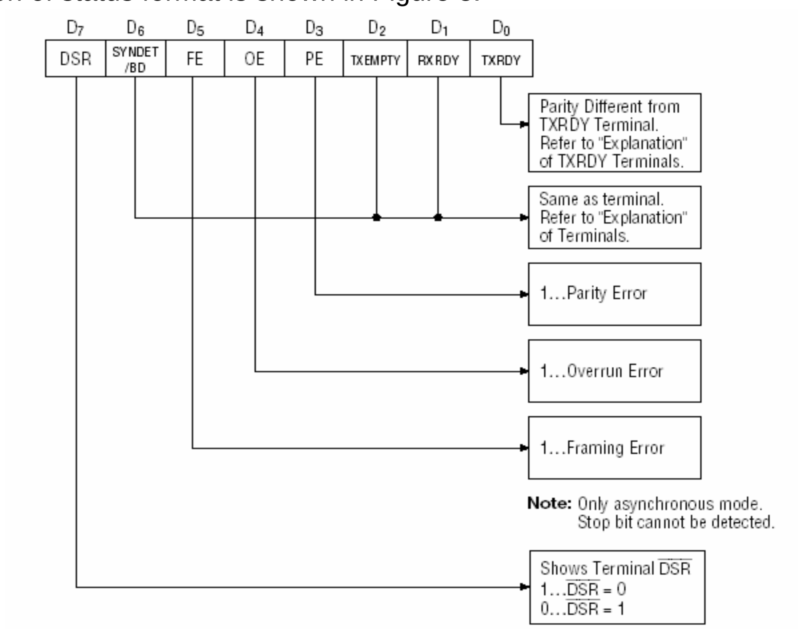


Figure 8 : Bit Configuration of Status Format

3. COMMUNICATION BETWEEN 8085 DEVELOPMENT SYSTEM AND HOST COMPUTER (TERMINAL)

8085 Development System communicates with the computer through its serial port (RS232). In essence, the computer functions as dump terminal. A specific software known as Termulator is used to enable the computer to operate in terminal mode. There is also other software such as Procomm, Windows Hyper Terminal etc. that can be used for this purpose. In this lab, Termulator software is recommended. Therefore, to start the terminal session type the following commands at DOS prompt.

D :\uplab> termultr <press Enter>

Switch on the power supply for the 8085 Development System Board and you will find the following menu as shown below appears on the computer screen.

```
8085 Microprocessor Development System
Monitor Program V2.5
Engineering Centre
Northern M'sia University College of Eng.
Menu

L - Load User Program - Intel hex file
S - Substitute Memory Contents
E - Execute Program
D - Display Memory Contents
R - Display & Modify Register Contents
M – Menu

KUKUM>

Press F10 for HELP
```

If the menu fails to appear on the terminal screen, press again the Reset button on the 8085 Development System. Make sure the power supply is already switched on. Contact the technician in charge if any problem occurs.

Serial communication parameters are as follows:

9600 b/s, 8 data bit, 1 stop bit and none parity.

This communication parameter can be set in terminal mode by pressing ‘ALT’ and ‘P’ simultaneously.

3.1 Monitor Program Commands

Monitor Program provides basic operating system commands to enable interaction between 8085 Development System and the user. It is written using 8085 Assembly Language and programmed into the EPROM 2764. The commands provided are as followed:

- Download user program code to 8085 Development System.
- Substitute memory contents
- Execute program
- Display memory contents
- Display and substitute register contents

3.1.1 Download user program code to 8085 System ('L' Command)

This command is to download a file which contains user program code in Intel Hex format from the computer to the memory location of 8085 Development System through its serial port. Press 'L' and the following information appear.

KUKUM > L
Press alt-v !!

Then press 'Alt' and '**V**' simultaneously and the following message will appear.

Enter the name of file to be viewed:

Enter the name of file to be sent (e.g. example. hex) and press **<enter>**

Enter the name of file to be viewed: example. hex

The following message will then appear and press 'Y'.

Transmit file while viewing? (Y/N)

Press any key for the following messages,

When the process is done, the user codes is transferred to the memory of 8085 Development System

3.1.2 Substitute Memory Contents ('S' Command)

The contents of memory (RAM) can be changed using this command. Type 'S' and the following message is displayed.

```
KUKUM > S
Type memory location address
Type Q – quit
 -
```

At the cursor position, type the address of memory location which the value is to be changed. For example, the location 2000H - 2004H is to be changed. Enter address 2000 and the next message will show address 2000 and the original value of the data. Enter new value; 00 and the address will be increased to 2001. Enter new values if you wish to continue changing the contents, and last of all finish the process with 'Q' command.

```
          -2000
Address   2000-FF   Original data
          00        New data
          2001-FF
          01
          2002-FF
          02
          2003-FF
          03
          2004-FF
          04
          2005-FF
          Q
          KUKUM >
```

Confirm that all data is changed by using display memory contents command.

3.1.3 Execute program ('E' command)

This command can be used after user code is downloaded using 'L' command. The function of this command is to execute the program.

```
KUKUM > E
Type program address: 2000
Program running…
```

3.1.4 Display memory content ('D' command)

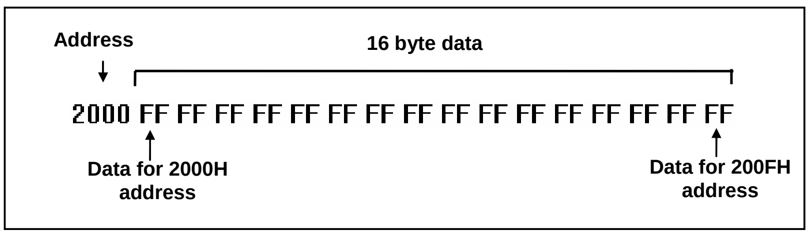
This command enables you to see the memory contents in the 8085 Development System. Press 'D' at prompt 'KUKUM>' and the following message is displayed.

```
KUKUM > D
ROM address:0000 - 1FFFH
RAM address:2000 - 3FFFH

Type Starting Address(Hex) :
```

Enter the address of memory location that you wish to see. Ex. 2000 value is entered and information of memory contents will be displayed as follows.

```
Type Starting Address(Hex) :2000
2000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
<press any key to continue or Q to quit>
```

Each row contains 2 byte representing the address of memory and this is followed by 16 byte which represents data.



Press any key except ‘Q” enables another 128 byte to be displayed beginning with 2080H address. To go back to ‘KUKUM>’ prompt press ‘Q’.

3.1.5 Displaying and substituting register content (‘R’ command)

This command enables you to check and modify the contents of register A, B, C, D, E, H, L, SP and Flag in 8085 Development System. By typing ‘R’, the following message will appear.

```
KUKUM > R
DISPLAY & MODIFY REGISTER


FLAG-FF
ACC-FF
REG_C-FF
REG_B-FF
REG_E-00
REG_D-00
REG_L-00
REG_H-00
SPL-00
SPH-00


MODIFY REGISTER : Y/N
```

LABORATORY 2

INTRODUCTION TO ASSEMBLY LANGUAGE PROGRAMMING

1. OBJECTIVES

1.1 To understand the assembly language programming and Cross Assembler (XASM)
1.2 To download user code (HEX File) to 8085 Development System.
1.3 To execute user program code.

2. INTRODUCTION

Assembly Language is a low level language which is used to program microprocessor based system so that it can function according to the user's desire. Generally, a microprocessor system executes instruction in the form of machine codes which is represented by logic 1 and 0. However, to write a program based on the machine codes will be very complicated and takes a long time. So, assembly language is the solution to solve this problem. Each program in an assembly language will be translated automatically to machine codes by software named 'assembler'.

The structure of an assembly language is very simple. Each instruction is written on a separate line and the format of a line is shown as follows:

SUM: ADI 12 ; add the contents of Acc with 12

The example given above consists of four different fields.

- Label ('SUM')
- Mnemonic ('ADI')
- Operand ('12')
- Comments ('add the contents of Acc with 12')

Notes: Please see the used of notation ':' after label and also notation ';' before comments.

A complete program may consist of instructions as above and every instruction is executed sequentially. In this lab, you will be writing program in assembly language and using assembler to produce machine codes. Refer to Appendix 4 for 8085 instruction set details.

3. CREATING SOURCE CODE

Type in the following program example using **NOTEPAD** text editor:

Lab 2 program example ;

```
        CPU "8085.TBL"      ; Processor declaration

        ORG 2000H           ; User code starting address

        LXI SP,3FF0H        ; Initialize stack pointer
        MVI A,05H           ; Copy immediate value 05H to accumulator (register A)
        MOV B,A             ; Copy the contents of Acc to register B
        MOV C,B             ; Copy the contents of register B to register C
        MOV D,C             ; Copy the contents of register C to register D
        MOV E,D             ; Copy the contents of register D to register E
        INR A               ; Increment the contents of Acc
        STA 2050H           ; Store the contents of Acc to memory location 2050H
        INR A               ; Increment the contents of Acc
        LXI H,2051H         ; Load register pair HL with value 2051H
        MOV M,A             ; Store the contents of Acc to memory
        LDA 2050H           ; Load acc with value from memory location 2050H
        INR A               ; Increment the contents of Acc
        INX H               ; Increment register pair HL
        MOV M,A             ; Store the contents of Acc to memory
        RST 1
        END
```

Your program should be named as follows: filename.ASM. This is to enable the Assembler to recognize the program as source file or source code. After the program is written, it must be translated to machine codes in the binary form of 1 and 0.

4. ASSEMBLING SOURCE CODE

When your program is completed, save your program and exit from NOTEPAD text editor to DOS command. Use the following command to assemble the program into machine codes.

D:\UPLAB> C16 filename.ASM -L filename.LST -H filename.HEX

By executing the above command, another two files will be created:

- Listing File ie. filename.lst
- Hex File ie. filename.hex

If there is any error in the program, you can check the error inside the Listing File and then modify the error in the source file. The Listing File shows the address and machine codes (opcode and operand) for each line of the instruction which has been translated by X-ASM. The Hex File contains machine codes information in ASCII format. Further details can be referred to the lecture notes.

5. DOWNLOADING HEX FILE (MACHINE CODE) TO 8085 DEVELOPMENT SYSTEM AND RUNNING THE PROGRAM

In this section, you will be shown how to transfer your code to 8085 Development System. This can be done with the help of Monitor Program which resides in the system memory (ROM). This monitor program controls the whole system operation by providing the following facilities (as discussed in Lab 1 session).

- Display and Modify register contents
- Display memory contents
- Modify memory contents
- Download user code (HEX file)
- Execute user code

Now you can transfer your machine codes (HEX file) to 8085 Development System. Run the PC terminal program (TERMULTR) and switch on the power supply for the 8085 Development System. From the main menu, select ‘L’ command to download your program code and execute it using ‘E’ command.

After the execution process, use ‘R’ command to check the register contents and check memory location 2050H until 2052H by using ‘D’ command. Complete Table 1:

Table 1 : Register / Memory Contents

Register/Memory	Content
A	
B	
C	
D	
E	
H	
L	
SP	
2050H	
2051H	
2052H	

6. EXERCISE

1. Using LDA and STA instructions, write a program that will transfer five byte of memory from location 3000H through 3004H to location 3200H through 3204H

2. Write a program to exchange the contents of HL register pair with DE register pair using MOV instruction.

3. Write a program to swap lower 4 bit nibble with upper 4 bit nibble of 8 bit data at memory location 2100H and place a result to location 2101H.

4. Write a program using the ADI instruction to add the two hexadecimal numbers 3AH and 48H and store the result in memory location 2100H.

5. Write a program to subtract the number in the D register from the number in the E register. Store the result in register C.

6. Write an assembly language program that AND, OR and XOR together the contents of register B, C and E and place the result into memory location 3000H, 3001H and 3002H.

7. Write a program that store 00H into memory location 2500H through 2510H.

8. Write an assembly language program to add two 8-bit numbers, the sum may be of 16-bits.

9. Write an 8085 assembly language program using minimum number of instructions to add the 16 bit number in BC, DE & HL. Store the 16 bit result in DE.

10. Develop a program in assembly that subtracts the number in the DE register pair from the number in the HL register. Place the result in BC register.

11. Sixteen bytes of data are stored in memory locations at 3150H to 315FH. Write a program to transfer the entire block of data to new memory locations starting at 3250H.

12. Write an 8085 assembly language program, which adds two three-byte numbers. The first number is stored in memory locations 3800H, 3801H & 3802H and the second number is stored in memory location 3803H, 3804H & 3805H. Store the answer in memory locations 3810H upwards.

13. Write an 8085 assembly language program, which checks the number in memory location 2800H. If the number is an even number, then put ‘FF’ in memory location 2810H, otherwise put ‘00’.

14. Write a program to count the data byte in memory that equal to 55H starting at memory location 2800H through 280FH. Place the count value in B register.

15. Write an 8085 assembly language program to find the smallest value between two number in memory location 2800H and 2801. Store the value in memory location 3000H.

LABORATORY 3

EXERCISE ON ASSEMBLY LANGUAGE PROGRAMMING

1. Write a program to calculate the sum of a series of numbers. The length of the block is in memory location 2102H and the series itself begins in memory location 2103H. Store the sum in memory locations 2100H and 2101H (MSB byte in 2101H).

2. Write a program to find the largest element in a block of data. The length of series is in memory location 2501H and the block itself begins in memory location 2502H. Store the largest value in memory locations 2500H. Assume that the numbers in the block are all 8-bit unsigned binary numbers.

3. The following block of data is stored in memory locations from 3055H to 305AH. Write a program to transfer the block of data in reverse order at same memory location.

   DATA (HEX): 22, A5, B2, 99, 7F, 37

4. Read one byte data from memory location 2200H. Determine the number of bit 1's in the data and store the result at memory location 2201H.

5. Write a program to multiply two 8-bit unsigned numbers. Store the result in memory locations 2A00H and 2A01H (MSB byte in 2A01H).

LABORATORY 4

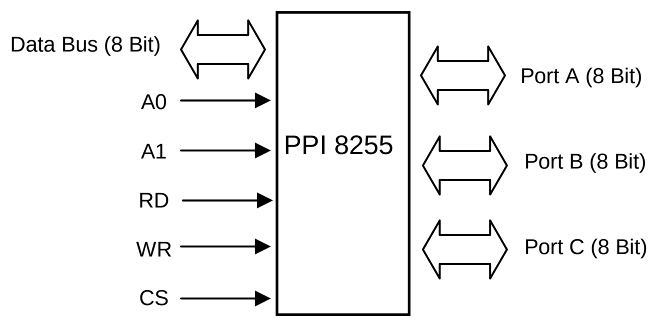
INTRODUCTION TO PROGRAMMABLE PERIPHERAL INTERFACE 8255

1. OBJECTIVES

1.1 To understand the Programmable Peripheral Interface 8255.
1.2 To use the 8255 in mode 0 (Basic I/O).
1.3 To perform simple interfacing with LED.
1.4 To write and understand the software delay routine.

2. INTRODUCTION TO PROGRAMMABLE PERIPHERAL INTERFACE 8255

The 8255 Programmable Peripheral Interface (PPI) is a general purpose interface device which is widely used in microprocessor design. It contains three independent 8 bit ports named Port A, B and C. Port A and B can be programmed as either input or output (all eight line must be same), while port C is split into two 4 bit halves (Port C upper (PC4-PC7) and Port C lower (PC0-PC3)) that can be separately programmed as input or output.



There are four registers that control the operations of the PPI and they are mapped to four address locations in the 8085 Development System as shown below:

Port	Address
Port A Register	80H
Port B Register	81H
Port C Register	82H
Control Register	83H

Table 1 shows the details of signals involved in controlling the PPI.

Table 1 : Signals Controlling PPI

A0	A1	CS	RD=0	WR=0
0	0	0	Port A to Data bus	Data bus to Port A
0	1	0	Port B to Data bus	Data bus to Port B
1	0	0	Port C to Data bus	Data bus to Port C
1	1	0	-	Data bus to Control Register

The control register is used to configure the PPI into a variety of operation modes. There are three basic modes:

- mode 0 : basic input/output
- mode 1 : strobe input/output
- mode 2 : bidirectional bus

We will only concentrate at mode 0. Further descriptions about mode 1 and 2 can be referred to 8255 datasheet. Mode 0 provides for simple input output operations with no handshaking. This means that data either to or from the port does not depends to other signal for data transfer. Basically, to configure the PPI, a control word must first be sent to the control register. Figure 1 summarized the control word format.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	Port A and upper half of Port C				Port B & lower half of Port C		
	MODE		Port A	Port C_{UPPER}	MODE	Port B	Port C_{LOWER}

Mode Definition Control Byte. Indicate by bit b7=1.

Port A and upper half of Port C (Group A)
Bit b3 to bit b6 control the mode and direction of Group A

b6, b5	mode operation
0 0	mode 0
0 1	mode 1
1 0	mode 2

b4	Port A direction, 0=output, 1=input
b3	upper half of Port C direction, 0=output, 1=input

Port B and lower half of Port C (Group B)
Bit b0 to bit b2 control the mode and direction of Group B

b2	mode operation, 0=mod 0, 1=mod 1
b1	Port B direction, 0=output, 1=input
b0	lower half of Port C direction, 0=output, 1=input

Figure 1 : Control Word Format

Therefore to enable these ports to function as input or output, the microprocessor needs to configure the PPI by sending the control word as describe above to the control register.

The following program is used to configure the PPI so that port A functions as input and port B & C as output.

```
MVI A, 10010000B    ; Control word setting where Port A= I/P and Port B & C=O/P.
OUT 83H             ; Send control word to control register.
```

When the control port is configured, the data can be sent to any of the output port using OUT instruction. For example, if a data is to be sent to the Port B. The data must first be loaded into the accumulator and OUT instruction is used as below.

```
MVI A, 55H
OUT 81H
```

To read a data from Port A to the accumulator, IN instruction is used.

IN 80H (where 80H is the address of Port A)

To further understand the I/O concept, perform the following experiment. Connect one LED to bit 0 of Port A (PA0), and one switch to bit 0 Port B (PB0). Configure Port A as output and Port B as input. Write and test the following program:

```
; Example of Lab 4 program 1

        CPU  "8085.TBL"
PORTA:          EQU   80H
PORTB:          EQU   81H
PORTC:          EQU   82H
CTRLPORT:       EQU   83H

            ORG 2000H
            LXI SP, 3FF0H
            MVI A, 82H
            OUT CTRLPORT
            IN PORTB
            OUT PORTA
            RST 1
            END
```

Before executing the program, set the switch to logic ‘1’. Execute the program and see what happen to the LED. Set the switch to logic ‘0’, execute the program once again and see what happen to the LED. Change the LED to PA7 and execute the program once again. Discuss your result.

Connect one LED to Port C_0, write the following program and execute the program. Give a short description about the program.

```
; Example of Lab 4 program 2
        CPU  "8085.TBL"
PORTA:          EQU   80H
PORTB:          EQU   81H
PORTC:          EQU   82H
CTRLPORT:       EQU   83H

            ORG 2000H
            LXI SP, 3FF0H
            MVI A, 80H
            OUT CTRLPORT
            MVI A, 0
REPEAT:     CMA
            OUT PORTC
            CALL DELAY
            JMP REPEAT
```

```
DELAY:    MVI B, 255
LOOP:     DCR B
          JNZ LOOP
          RET
          END
```

3. SOFTWARE DELAY

The following program is given:

```
         MVI C, 255
LOOP:    DCR C
         JNZ LOOP
```

This program forms a loop which will decrease the contents of register C by 1 until it contents becomes 0. Any instructions after JNZ will be execute after register C equal to zero. This loop will create a delay that can be used for certain microprocessor application. The time delay is depended on instruction set which determine the T State for each instruction. (Please refer to Appendix 3: 8080A/8085A INSTRUCTION SET INDEX). The following example shows the calculation of the software delay routine.

```
              MVI C, 255    ; 7 states
254 [ LOOP:   DCR C         ; 4 states
              JNZ LOOP      ; 7/10 states (10 states if condition is met or
                            ; 7 states if the condition not met)
```

The total number of state, $\sum T_s = 7 + 254(4 + 10) + 1(4 + 7) = 3574$.

Where

T state, T_s=1 / (0.5 x crystal frequency)

For 6.144 MHz crystal,

T state, T_s= 325 ns. Time delay, td=3574 x 325 ns = 1.16 ms.

For longer delay, use the following subroutine;

DELAY:	MVI B, ***x_1***		DELAY:	LXI B, ***m***
LOOP:	MVI C, ***x_2***		LOOP:	DCX B
LOOP1:	DCR C			MOV A, B
	JNZ LOOP1	**OR**		ORA C
	DCR B			JNZ LOOP
	JNZ LOOP			RET
	RET			

Now write a program to make the LED blinked at rate of 500ms. Show the calculation to determine the value of ***x_1*** and ***x_2*** or ***m*** for the delay routine.

4. DESIGN PROBLEM

Using eight LED and one switch, write a program to design a simple running light system. The following table shows the operation of the system.

Switch	LED's
OFF (Logic "0")	STOP running
ON (Logic "1")	START running

LABORATORY 5

EXERCISE ON INPUT / OUTPUT (I/O) INTERFACE

1. EXERCISE

1. Figure 1 shows the connection between LED and switch using 8255 I/O port. Write a program to make the LED blink when the switch is ON and stop blinking when the switch is OFF.

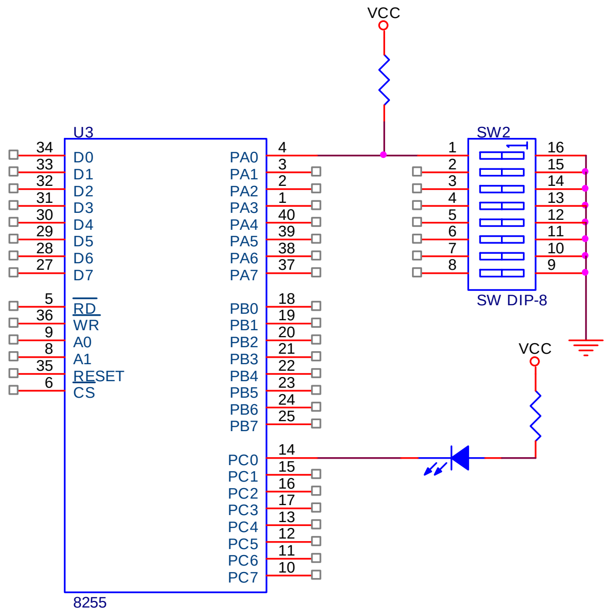


Figure 1 : Connection between LED and Switch with 8255 I/O Port

2. Figure 2 below shows the connection interface of four switches and LED with Port A and Port C. The switches represent a 4-bit data input. Write a program to read 4 bit data input from the switches and the value will be used to determine how many time the LED should blink.

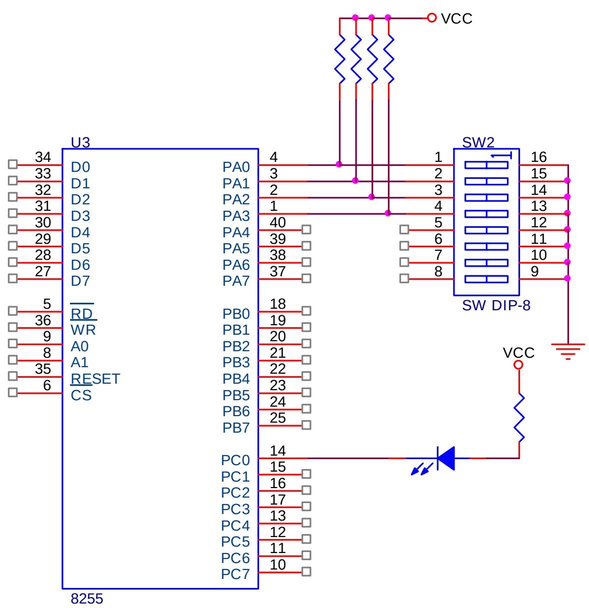
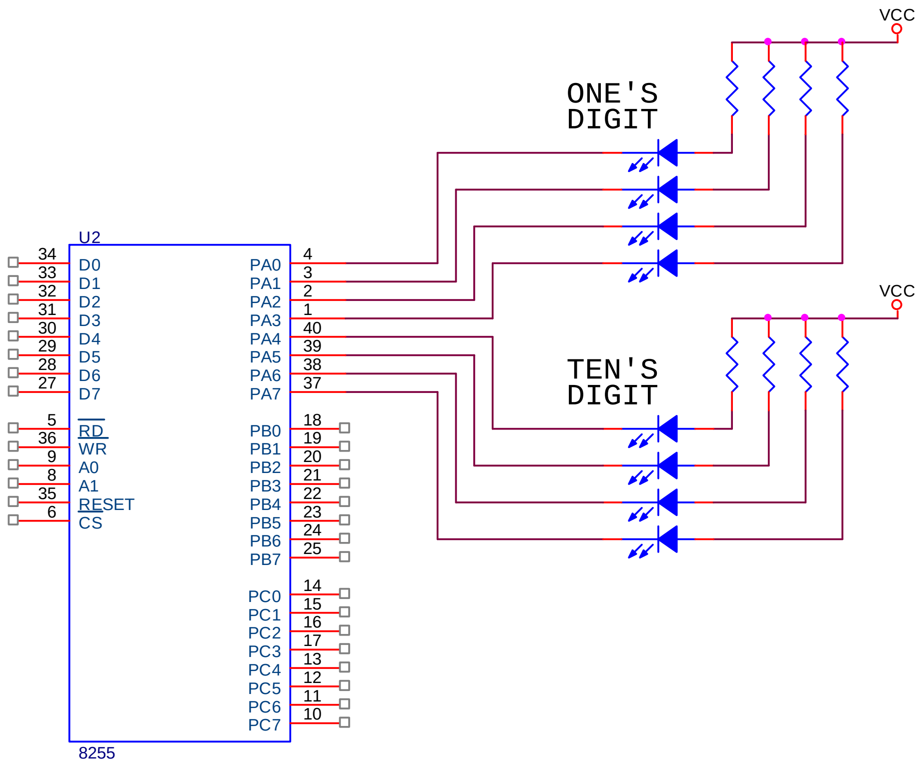


Figure 2 : Connection Interface of Four Switches and LED with Port A and Port C

3. Write a program to implement a BCD counter on the LEDs attached to the 8255. The count on the LEDs is displayed from 0 to 99 using the four LSB leds as the 'ones' digit and the four MSB leds as the 'tens' digit. The count may be allowed to overflow to 0 after 99 or may simply halt.



LABORATORY 6
INTERFACING WITH SEVEN SEGMENT DISPLAY

1. OBJECTIVES

1.1 To perform interfacing with Seven Segment Display
1.2 To design the Up/Down Counter

2. INTRODUCTION

In this lab session, you will interface the 8085 Microprocessor System with seven segment display through its programmable I/O port 8255. Seven segment displays (as shown in Figure 1) is often used in the digital electronic equipments to display information regarding certain process.

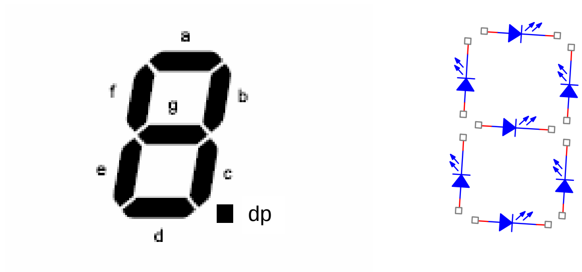
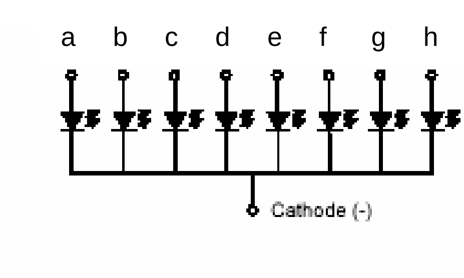


Figure 1 : Seven Segment Display

There are two types of seven segment display; common anode and common cathode. The differences between these two displays are shown in Figure 2a and 2b. The internal structure of the seven segment display consist of a group of Light Emitting Diode (LED)



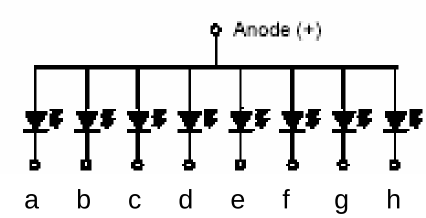


Figure 2a - Common Cathode

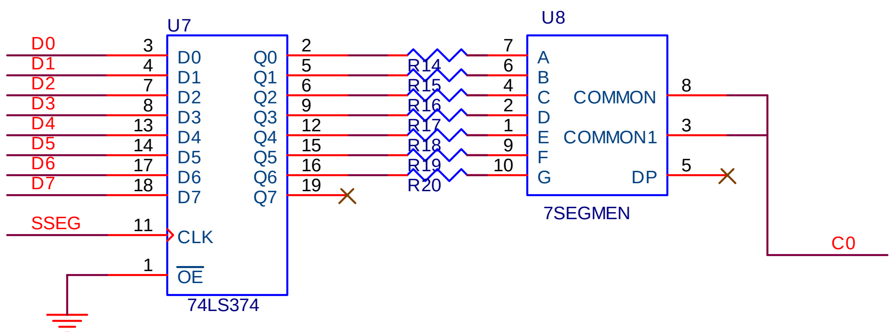
Figure 2b - Common Anode

For common cathode, the segment will light up when logic ‘1’ (+V) is supplied and it will light off when logic ‘0’ (OV) is supplied. While for common anode, logic ‘1’ will light off the segment and logic ‘0’ will light up the segment. Therefore to display number ‘0’ on the seven segment display, segment a, b, c, d, e and f must light up. For common cathode, logic ‘1’ should be given to the related segment whereas in the case of common anode, logic ‘0’ should be given to the necessary segment. Based on the above explanation, complete the Table 1 below:

Table 1 : Common Anode and Common Cathode Logic Assignment

	Common Anode							Common Cathode						
No	a	b	c	d	e	f	g	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
1														
2														
3														
4														
5														
6														
7														
8														
9														

The following diagram shows the interface for common cathode 7 segments in the I/O board. The 7 segments is connected through 8 bit latch (74LS374) and a resistor (range from 100Ω to 220Ω) in series to each segment.



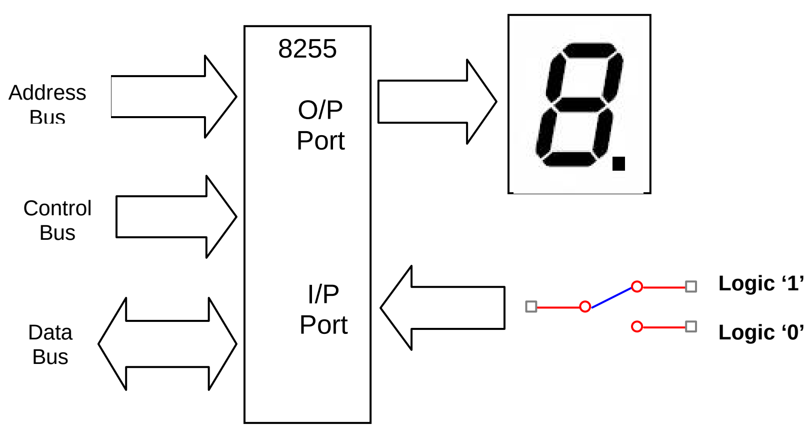
Make a connection for this seven segment display with microprocessor through programmable I/O port 8255. D0-D7 connected to Port A, SSEG to PC0 and C0 to PC1. Write and test the following program.

```
          CPU  "8085.TBL"
PORTA:      EQU   80H
PORTB:      EQU   81H
PORTC:      EQU   82H
CTRLPORT:   EQU   83H

            ORG 2000H
            LXI SP, 3FF0H
            MVI A, 80H
            OUT CTRLPORT
START:      MVI B, 10
            LXI H, TABLE
REPEAT:     MOV A, M
            OUT PORTA
            MVI A, 00000000B
            OUT PORTC
            MVI A, 00000001B
```

```
            OUT PORTC
            CALL DELAY
            INX H
            DCR B
            JNZ REPEAT
            JMP START

DELAY:      PUSH B
            MVI B, 255
LOOP:       MVI C, 255
LOOP1:      DCR C
            JNZ LOOP1
            DCR B
            JNZ LOOP
            POP B
            RET
TABLE:      DFB    ___, ___, ___, ___, ___, ___, ___, ___, ___, ___
                   (Common cathode data pattern from Table 1)
```

3. DESIGNING THE UP/DOWN COUNTER

In this part, you will be modifying the program you have written in Part A to design an up/down counter. The decision whether to count up or down depends on the input from a switch connected to another I/O port. If the switch is set at logic '1', the counter will count up. The counter will count down when the switch is set to logic '0'. The block diagram is as follows:



LABORATORY 7

EXERCISE USING SEVEN SEGMENT DISPLAY (WASHING MACHINE DESIGN)

1. EXERCISE

Design a Washing Machine Spin Timer. Refer to the following algorithm and the block diagram below:

- Set the time using the dip-switch (Maximum value is 99 second)
- The setting value will be displayed on the seven segment
- The spin process will begin when the start button is press and then
    - o START the Motor (LED ON)
    - o The display value on the seven segments start counting down.
- When the value is zero, STOP the motor (LED OFF)

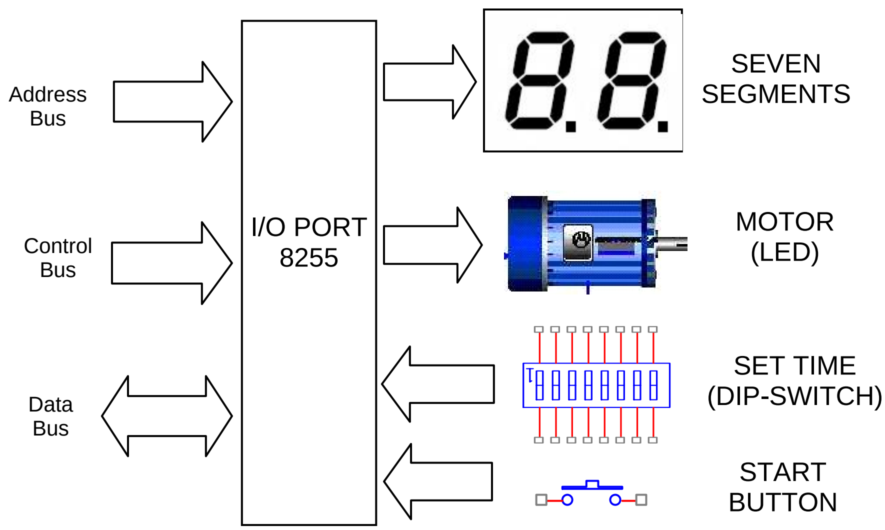


Figure 1 : Block Diagram

LABORATORY 8
INTERFACING WITH LCD DISPLAY

1. OBJECTIVE

1.1 To perform interfacing to Liquid Crystal Display (LCD) module

2. INTRODUCTION

Pin Assignment

The pin assignment shown below is an industry standard for small (80 characters or less) alphanumeric LCD - modules.

Pin number	Symbol	I/O	Function
1	Vss	-	Power supply (GND)
2	V_{DD}	-	Power supply (+5V)
3	Vee	-	Contrast adjust
4	RS	I	0 = Command input/output 1 = Data input/output
5	R/W	I	0 = Write to LCD module 1 = Read from LCD module
6	E	I	Enable signal (Data strobe)
7	DB0	I/O	Data bus line 0 (LSB)
8	DB1	I/O	Data bus line 1
9	DB2	I/O	Data bus line 2
10	DB3	I/O	Data bus line 3
11	DB4	I/O	Data bus line 4
12	DB5	I/O	Data bus line 5
13	DB6	I/O	Data bus line 6
14	DB7	I/O	Data bus line 7 (MSB)

LCD Background

The LCD module requires 3 control lines and either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used, the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used, the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus). The three control lines are referred to as E, RS, and R/W.

The E line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should first set this line high (1) and then set the other two control lines (RS & RW) and put data on the data bus (DB0-DB8). When the other lines are completely ready, bring E low (0) again. The 1→ 0 transition tells the LCD to take the data currently found on the other control lines and on the data bus and to treat it as a command.

The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the LCD screen. For example, to display the letter "T" on the screen you would set RS high.

The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7.

Initializing the LCD

Before the LCD can be used, the LCD module controller needs to be initialized using a few instructions. This is accomplished by sending a number of initialization instructions to the LCD. The following example shows a suggested sequence of instructions to initialize the LCD with 8 bit operation, 2 lines, 5x7 font and automatically incremented display.

RS=0, RW=0, 38H, 38H, 38H, 0CH, 06H

The initialization procedure is shown below.

1. Send Init Command - 38H 3 times (8 bit data, 2 lines and 5x7 fonts)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL=1	N=1	F=0	0	0

DL = Data Length
0 → 4 - bit operation
1 → 8 - bit operation

N = Number of `lines'
0 → for 1/8 duty cycle -- 1 line
1 → for 1/16 duty cycle – 2 line

F = Font,
1 → not used
0 → for 5x7 dot matrix

<Wait 40us or till BF=0>

2. Send Init Command - 0CH (Display on, Cursor off, blink off)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D=1	C=0	0

D = Display ON/OFF
0 → display OFF
1 → display ON

C = Cursor ON/OFF
0 → Cursor OFF
1 → cursor ON

<Wait 40us or till BF=0>

3. Send Init Command - 06H (Increment cursor to the right when writing, don't shift screen)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D=1	S=0

I/D = Set cursor direction
0 → decrement
1 → increment

S= Display shift
0 → disable
1 → enable

<Wait 40us or till BF=0>

Cursor Positioning

The LCD module contains a certain amount of memory which is assigned to the display. All the text we write to the LCD module is stored in this memory, and the LCD module subsequently reads this memory to display the text on the LCD itself. This memory can be represented with the following diagram.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Line 1																
Line 2																
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

As you can see, it measures 16 characters per line by 2 lines. The numbers on top and bottom of each box is the memory address that corresponds to that screen position. Thus, the first character in the upper left-hand corner is at address 00h. The following character position (character #2 on the first line) is address 01h, etc. This continues until we reach the 16th character of the first line which is at address 0Fh. However, the first character of line 2 is at address 40h. Thus we need to send a command to the LCD that tells it to position the cursor on the second line. The "Set Cursor Position" instruction is 80h and we must add it with the address of the location where we wish to position the cursor. For example to start display character at line 2 address 41h, we must send a "Set Cursor Position" instruction--the value of this command will be 80h (the instruction code to position the cursor) plus the address 41h ; 80h + 41h = C1h. Thus sending the command C1h to the LCD will position the cursor on the second line at the second character position.

Exercise

Write the following sample program to display a message string text on the LCD module.

```
; CONNECTION
;   All routines are written for controlling the module in a 8-bit mode.
;   DB0 –DB7  connect to Port A
;   RS        PC0
;   R/W       PC1
;   E         PC2
;-----------------------------------------------
            cpu "8085.tbl"
            org .2000h
            lxi sp,3ff0h

;* Initialization routine for the 8255
;-----------------------------------------------
            mvi a,80h       ;all ports as o/p
            out 83h

;* Initialization routine for the LCD-module
;-----------------------------------------------
            mvi a,38h               ;refer to LCD data sheet for
            call wr_cmd             ;initialization sequences
            mvi a,38h               ;8 bits, 2 rows, 5 x 7 dots
            call wr_cmd
            mvi a,38h
            call wr_cmd
            mvi a,0ch
            call wr_cmd
            mvi a,06h
            call wr_cmd
            mvi a,80h               ;Set address to 80h (home position)
            call wr_cmd             ;of the first line
            lxi h,msg_line1         ;start write char to the first line
line1:      mov a,m                 ;of the LCD
            cpi 0
            jz next_line
            call wr_char

            inx h
            jmp line1
next_line:  mvi a,0c0h              ;set address to C0h (home position)
            call wr_cmd             ;of the second line
            lxi h,msg_line2         ;start write char to the second line
line2:      mov a,m                 ;of the LCD
            cpi 0
            jz exit
            call wr_char
            inx h
            jmp line2
exit:       RST 1                   ;entry point to monitor program
```

```
;* wr_cmd writes an instruction-code to the LCD
;* The accumulator contains the instruction-code.
;----------------------------------------------------
wr_cmd:     out 80h              ;send instruction code data through port A
            mvi a,00000100b      ;RS=0,R/W=0,E=1
            out 82h
            mvi a,00000000b      ;RS=0,R/W=0,E=0
            out 82h
            call delay2          ;internal executing time for LCD
            ret

;* wr_char writes a single character to the LCD.
;* The accumulator contains the character.
;----------------------------------------------------
wr_char:    out 80h              ;send character through port A
            mvi a,00000101b      ;RS=1,R/W=0,E=1
            out 82h
            mvi a,00000001b      ;RS=1,R/W=0,E=0
            out 82h
            call delay1          ;internal executing time for LCD
            ret

delay1:     mvi c,40
loop_1:     dcr c
            jnz loop_1
            ret

delay2:     mvi c,255
loop_2:     dcr c
            jnz loop_2
            ret

msg_line1:  dfb   "8085 Development",0
msg_line2:  dfb   "System – KUKUM ",0
            end
```

3. DESIGN PROBLEM

Write a program to display the following text and count value 0 → 9 on the LCD module.

		C	O	U	N	T	E	R		=		0			
		M	I	C	R	O	-	P		L	A	B			

LABORATORY 9

INTERFACING I/O WITH MATRIX TYPE KEYPAD

1. OBJECTIVE

1.1 To perform interfacing I/O with Matrix Type Keypad

2. INTRODUCTION

Most of microprocessor applications require a keypad for users to enter numbers and commands. In this lab you will be shown how to interface a matrix type keypad to 8085 development system. Figure 1 below is the block diagram of the keypad.

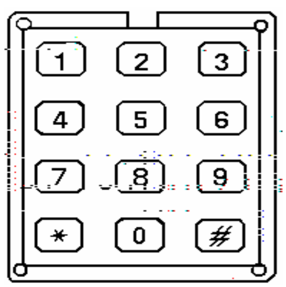


Figure 1 : Block Diagram of Keypad.

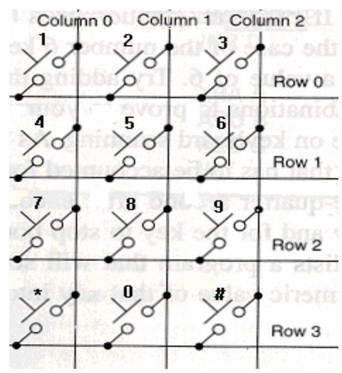


Figure 2 : Internal Structure of 4 x 3 Matrix Keypad.

Figure 2 shows the internal structure of the 4 x 3 matrix keypad. The keypad consists of an array of momentary pushbuttons switch or key. Each row and each column of the pushbutton are connected to a common line. There are 3 column line and 4 row line. Each pushbutton has two terminals; one is connected to a column line and other to a row line. When the key is pressed, the adjacent row and column are connected. For example if key '8' is pressed, row 2 and column 1 will connect to each other. Table 1 below shows the combinations of other key pressed.

Table 1 : Key Combinations

	Row				Column		
Key	R0	R1	R2	R3	C0	C1	C2
1	x				x		
2	x					x	
3	x						x
4		x			x		
5		x				x	
6		x					x
7			x		x		
8			x			x	
9			x				x
*				x	x		
0				x		x	
#				x			x

The usual way to interface a keypad to a microprocessor is by connecting input/output (I/O) port bits to row and column connections on the keypad. The keypad is wired in a matrix arrangement so that when a key is pressed one row is shorted to one column. It's relatively easy to write a routine to scan the keypad, detect key presses, and determine which key was pressed. Another alternative is to use the 74C922 keypad encoder chip. This device accepts input from a 16- keypad, performs all of the required scanning and debouncing, and outputs a "data available" (DA) bit and 4 output bits representing the number of the key pressed from 0 to 15. Figure 3 show the connections between 4 x 3 matrix keypad with 74C922.

Circuit Diagram

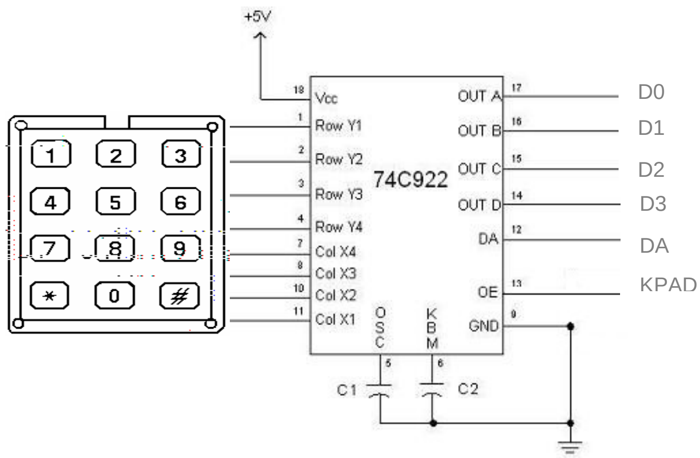


Figure 3 : Connection between 4 x 3 Matrix Keypad with 74C922

3. EXAMPLE

Write the following program to read the key pressed and store it to the memory location 3000h.

```
; CONNECTION
;   All routines are written for controlling the module in a 8-bit mode.
;   D0 –D3      connect to PB0 – PB3
;   DA          PB4
;   KPAD        connect to GND
;-----------------------------------------------
                CPU "8085.tbl"
                ORG 2000h
                LXI SP, 3FF0h

;* Initialization routine for the 8255
;-----------------------------------------------
                MVI A, 82h      ; Set Port B as input
                OUT 83H
```

```
WAIT:       IN 81H              ; key press?
            ANI 00010000B       ; DAV =1 if key press.
            JZ WAIT

WAIT1:      IN 81H              ; wait until key release
            ANI 00010000B
            JNZ WAIT1
            IN 81H              ; read data in from keypad decoder chip
            ANI 00001111B
            STA 3000H           ; store to memory location 3000H
            RST 1
            END
```

4. DESIGN PROBLEM

Connect the circuit as shown in Figure 4. Write a program to read the key pressed and display its number on the Seven Segment display. For example, if key number '5' is pressed, the digit number '5' will be displayed on the Seven Segment.

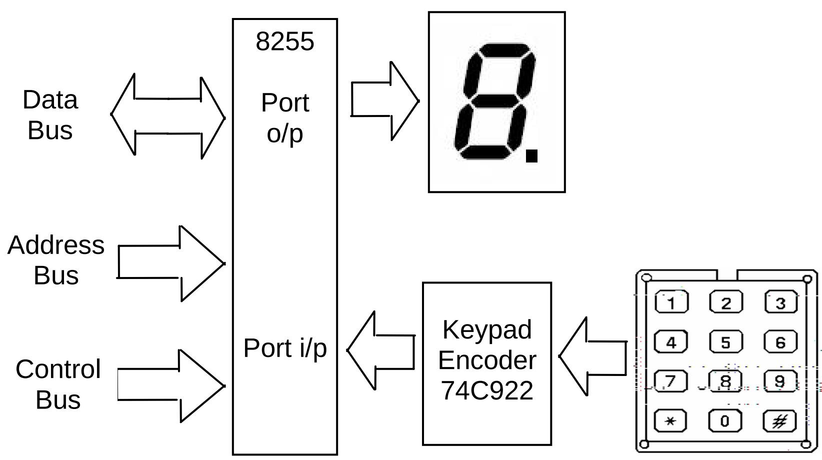


Figure 4 : Connection for Design Problem

LABORATORY 10

INTERFACING WITH ANALOG TO DIGITAL CONVERTER (ADC)

1. OBJECTIVE

1.1 To understand the function of Analog to Digital Converter (ADC)
1.2 To interface the ADC to the Microprocessor

2. INTRODUCTION

Analog to Digital Converter (A/D) is used to convert current signal or voltage into a digital word form, which is represented by logic '1' and '0' to be used either in the computer, digital equipment or any other circuits which involve microprocessor. Generally in a control and data acquisition system, measurement on a few physical parameters such as pressure, temperature, speed etc is done by the transducer. The output of the transducer is in the form of analogue either voltage or current which is proportional with the physical parameter measured. It is impossible to use directly the output signal from the transducer to the Microprocessor System. This is because the information that is to be processed by the microprocessor must be in the digital form. In this situation, using Analog to Digital Converter will fulfill the space between these two systems.

Analogue to Digital Converter provides digital information that is equivalent with analogue value that is to be processed and to be analyzed for certain purposes. So, it is important for you to understand the principle of Analogue to Digital Converter and how it is interfaced with the microprocessor.

Most ADC has two main control line signals:

- Start conversion signal (SC)
- End of conversion signal (EOC)

Other signals are data bus (8, 10 or 12 bit) and chip select. The ADC usually doesn't do anything until an external device such as a microprocessor generates start conversion signal. This signal will start the conversion process of analogue voltage or current input of ADC to the digital form until the process ended. At this point, the ADC will generate a signal through its End of Conversion signal line to show that the conversion process is completed. Now the data produced by ADC is valid and proportional to analogue voltage or current input.

3. ANALOG TO DIGITAL CONVERTER (ADC0804)

Analog to Digital Converter used in this lab is the 8-bit ADC0804. Figure 1 shows the configuration pin for this device. The /WR Signal (input) represents Start Conversion signal while the /INT signal (output) represents End of Conversion signal. The /RD signal (input) enables the digital data converted by the ADC to the data bus (DB0-DB7). Figure 2(a) and 2(b) shows the timing diagram of the control signal involved.

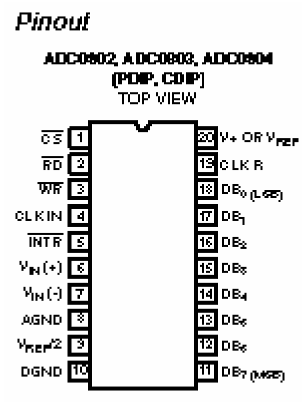


Figure 1 : ADC0804 Pin Configuration’

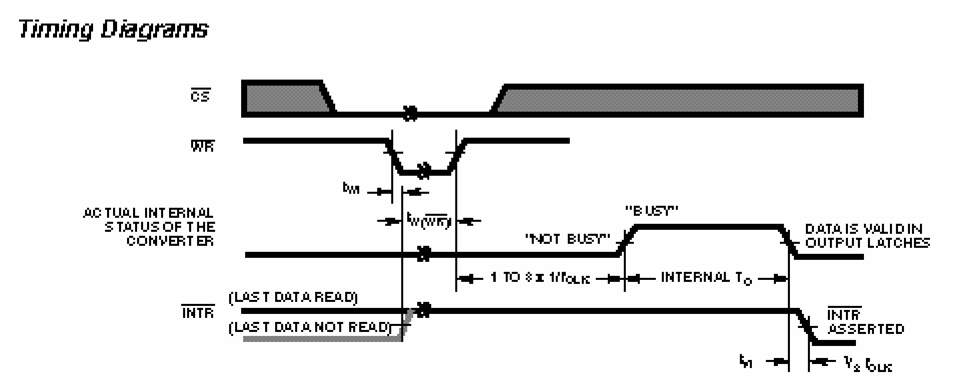


Figure 2(a) : Timing Diagram.

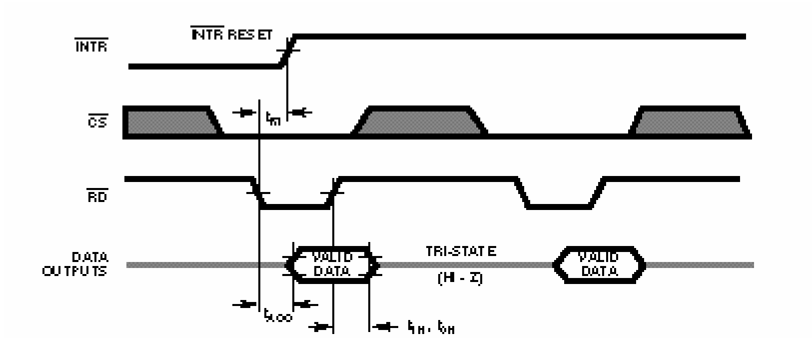


Figure 2(b) : Timing Diagram

4. DESIGN PROBLEM

Figure 3 shows the connection of the ADC to the 8085 Development System through its I/O port. Based on the flowchart below (Figure 4), write a program to implement the conversion of analogue voltage to digital data. Set the input of analogue voltage to a different value that ranges between 0 to 5 volt and record the digital value.

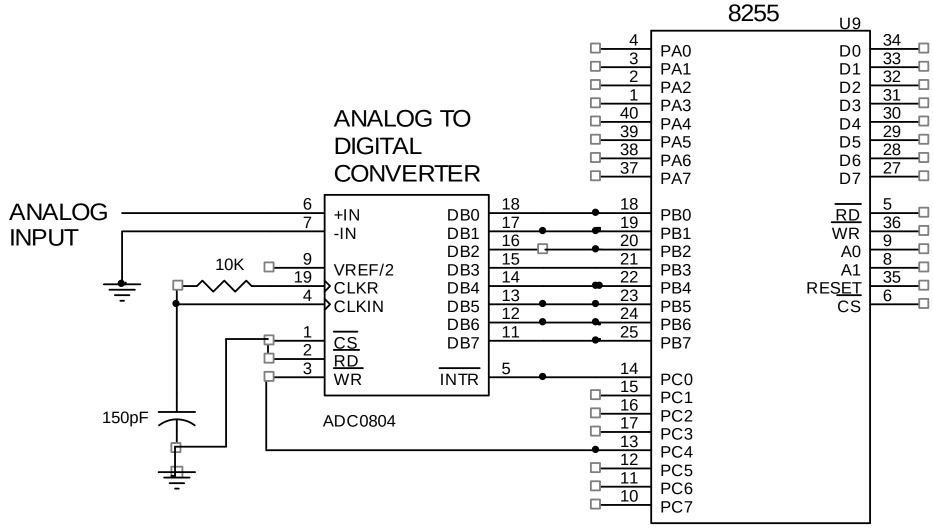


Figure 3 : Connection of ADC to 8085

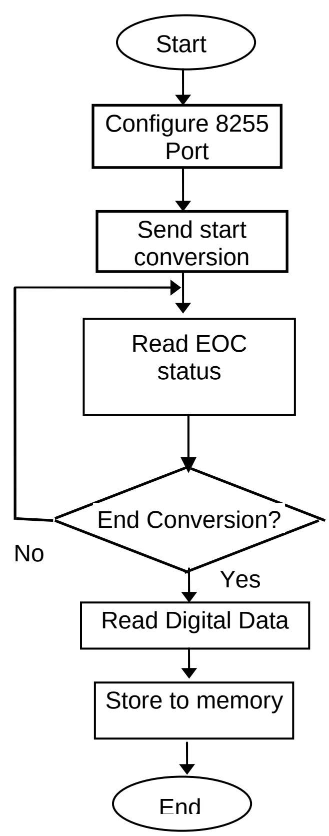


Figure 4 : Flowchart

LABORATORY 11

INTERFACING WITH DIGITAL TO ANALOG CONVERTER (DAC)

1. OBJECTIVES

1.1 To understand the function of Digital to Analog Converter (DAC)
1.2 To use DAC to generate waveform

2. INTRODUCTION

Waveform signal produced by analog voltage changes with time. Generally, several physical parameter such as speed, pressure etc can be controlled by analog voltage. For example the speed of direct current motor can be controlled by supplying analog voltage to the driver circuit which consists of a group of transistors to control the flow of current to the motor. Therefore the speed of the motor will be proportional with the flow of current through it.

In the digital systems, generating analog signal can be done using Digital to Analog Converter devices (DAC). This device will received digital input and convert it to analog voltage or current signal.

2.1 Circuit Diagram

Figure 1 shows the circuit diagram of the DAC converter using 8-bit DAC0832 chip. If jumper J4 and J6 are connected, the output of this DAC circuit is unipolar which is in the range of 0 to +5 Volt dc. For example, setting the digital input to DAC to 00000000b will produced 0 V at the output pin of the DAC, while setting the digital input to 11111111b will produce +5V. Other digital input value will produce different output voltage, which is proportional to the digital input.

Datasheet for Digital to Analogue, DAC0832 can be downloading from the internet.

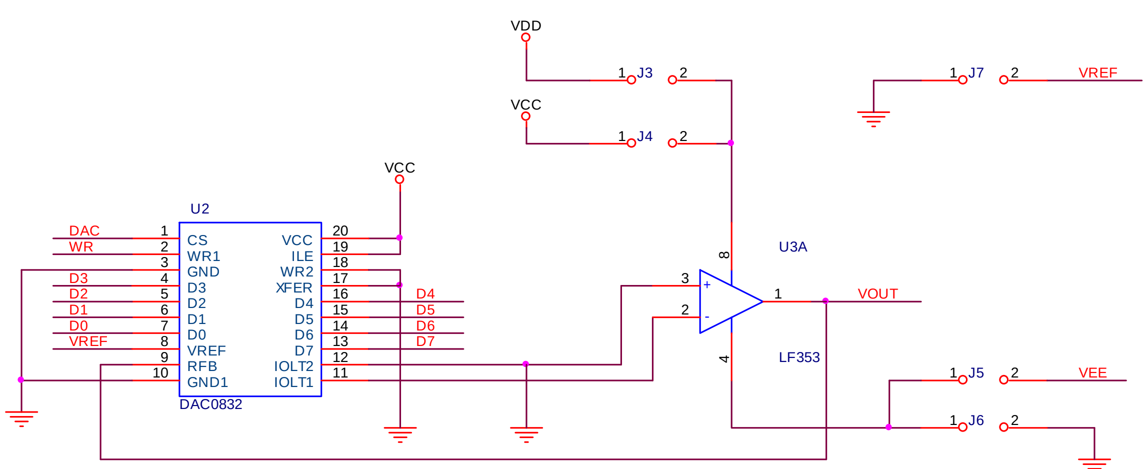


Figure 1 : DAC Circuit Diagram

Interface the DAC circuit shown above to the 8085 Microprocessor Development System. Connect DAC data line (D0-D7) to Port A and DAC and WR signal pull to GND. Write a program to output digital data to DAC and complete Table 1 below.

Table 1 : Digital vs Analog Values

Digital(Value (decimal)	Analog Value (Volt)
0	
10	
60	
80	
128	
180	
220	
255	

3. GENERATING WAVEFORM USING DAC

Based on the information given in Part A, a waveform can be generated by sending the digital data which represents each point of the waveform continuously. Write the following program to generate the waveform and check the waveform using oscilloscope. Sketch the waveform.

```
            CPU "8085.TBL"
PORTA:      EQU   80H
PORTB:      EQU   81H
PORTC:      EQU   82H
CTRLPORT:   EQU   83H

            ORG 2000H
            LXI SP, 3FF0H
            MVI A, 80H
            OUT CTRLPORT
START:      MVI B, 24
            LXI H, TABLE
REPEAT:     MOV A, M
            OUT PORTA
            INX H
            DCR B
            JNZ REPEAT
            JMP START
TABLE:      DFB   0,20,40,60,80,100,120,140,160,180,200,220,240,220,200,180
            DFB   160, 140, 120, 100, 80,60,40,20
```

4. DESIGN PROBLEM

Generate sine wave with 100 Hz frequency. (32 sample point)

LABORATORY 12
INTRODUCTION TO SERIAL I/O (SID AND SOD)

1. OBJECTIVE

1.1 To introduce Serial I/O (SID and SOD)

2. INTRODUCTION

The 8085 Microprocessor have two line specially designed for serial data transmission and reception. The two line are SID (Serial Input Data) and SOD (Serial Output Data). Data transfer is controlled using two instructions, SIM (Set Interrupt Mask) and RIM (Read Interrupt Mask). The RIM instruction, read the data from SID pin into Accumulator bit 7. Bit 0 to bit 6 Accumulator represent the interrupt status information in the 8085 system (Figure 1).

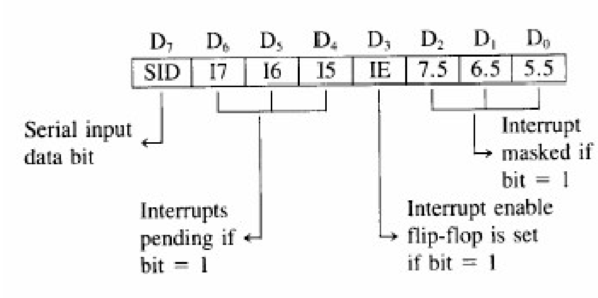


Figure 1

SIM instruction will transmit bit 7 in Accumulator through the SOD pin and bit 6 in Accumulator (SOE - Serial Output Enable) must be set to logic ‘1’ in order to enable the transmission (Figure 2).

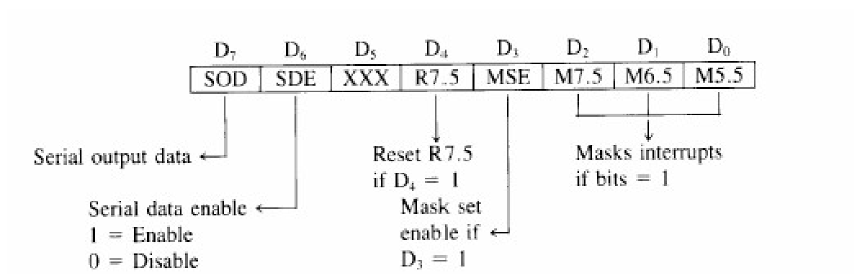


Figure 2

Serial data transmission through the SOD line is done by repeating the process of data transmission from the Accumulator to the SOD pin. For example, to transmit serial 8 bits data; the data will be transmit bit by bit through the SOD line using SIM instruction. The same concept is applied for receiving data process where the bit by bit of data is received through the SID pin. The method of transmission and reception of serial data in digital system is usually referred to the standard or specific format such as RS232. Baud rate, parity and stop bit must be considered as an important parameter for the process of transmission and reception serial data.

Example below shows how to write a program for transmitting serial data to a PC terminal. Data transmission is in asynchronous mode with 9600 baud rate, 8 bit data, none parity and 1 stop bit. Test the program using the 8085 system. Serial data transmission from the microprocessor to PC terminal is done by using the ***transmit*** subroutine. The data sent to the PC terminal should be in ASCII code format (Refer to Appendix 5).

```
baudtime:     equu        18
baudtime1:    equ         19

              org 2000h
              lxi sp,3ff0h
              lxi h, message
next_char:    mov a, m
              cpi 0
              jz exit
              call transmit
              inx h
              jmp next_char
exit:         rst 1

message:      dfb     0dh,0ah, “Welcome to Microprocessor LAB”,0

transmit:     push h
              push b
              mov c,a              ;4
              mvi b,10             ;7
              mvi a,01000000b      ;7
              sim                  ;4      ;send start bit = 0
tx_loop:      mvi h,baudtime       ;7
tx_loop1:     dcr h                ;4
              jnz tx_loop1         ;7/10
              mov a,c              ;4
              stc                  ;4
              rar                  ;4
              mov c,a              ;4
              rar                  ;4
              ani 80h              ;7
              ori 01000000b        ;7
              sim                  ;4
              dcr b                ;4
              jnz tx_loop                  ;7/10
              pop b
              pop h
              ret                  ;10
```

3. EXERCISE

Write a program for receiving 10 data from a terminal (keyboard) and store the data to memory location starting from 3000H address and then display the data to PC terminal. ***Received*** subroutine below is for receiving serial data from the terminal.

```
received:   push h
            push b
            mvi b,9             ;7
si1:        rim                 ;4
            ora a               ;4
            jm si1              ;7/10
            mvi h,baudtime1/2   ;7
si2:        dcr h               ;4
            jnz si2             ;7/10
si4:        mvi h,baudtime1     ;7
si3:        dcr h               ;4
            jnz si3             ;7/10
            rim                 ;4
            ral                 ;4
            dcr b               ;4
            jz rx_exit          ;7/10
            mov a,c             ;4
            rar                 ;4
            mov c,a             ;4
            jmp si4             ;10
rx_exit:    mov a,c             ;4
            pop b
            pop h
            ret
```

LABORATORY 13
INTERRUPT

1. OBJECTIVE

1.1 Understanding the interrupt structure in the 8085 microprocessor

2. INTRODUCTION

The 8085 microprocessor has five (5) interrupt source input. They are TRAP, RST7.5, RST6.5, RST5.5 and INTR. TRAP is non maskable and the others are maskable. During reset, all the maskable interrupt are disable, so the microprocessor only responds to TRAP. For maskable interrupt to be effective, its must be enabled under program control.

Only three (3) of this interrupt can be used using our 8085 development system: RST7.5, RST6.5 and RST5.5. The RST6.5 and RST5.5 are level sensitive while RST7.5 is rising edge sensitive.

When the microprocessor is interrupted by the external device, it will respond to the interruption by completing their current instruction, saving the program counter of the next instruction to be executed onto the stack and then jumping to the following addresses based on the type of interrupt request.

RST5.5	002CH
RST6.5	0034H
RST7.5	003CH

These addresses are located in the ROM space. For the 8085 development board that is used in the laboratory, the interrupt address of RST5.5, RST6.5 and RST7.5 are redirect to the new address location at RAM space using JMP instruction. Therefore, the users can easily enter their code of JMP instruction to the interrupt service routine. The new addresses are listed below.

RST5.5	3F2CH
RST6.5	3F34H
RST7.5	3F3CH

You can examine the contents of memory location 002CH, 0034H and 003CH using D command (Display memory). Its contain three byte of instruction for each interrupt address. The code is C3 which is representing the opcode of JMP instruction and followed by another two byte of data represented the destination address.

Two program steps are required to enable the RST7.5, RST6.5, RST5.5 interrupts:

- Clearing the interrupt masks using SIM instruction.
- Enabling all interrupt using EI instruction

The SIM instruction is used to implement the masking process of the 8085 interrupts RST7.5, RST6.5 and RST5.5. This instruction interprets the accumulator contents as follows:

SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5

SOD – Serial Output Data: Bit D7 of Accumulator is latched into the SOD output line and made available to serial peripheral if bit D6 = 1

SOE – Serial Output Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.

X – Don't care

R7.5 – Reset RST7.5: If this bit = 1, RST7.5 flip-flop is reset. This is an additional control to reset RST7.5.

MSE – Mask Set Enable: If this bit is high, it enables the function of bits D2, D1 and D0. This is a master control over all the interrupt masking bits. If this bit is low, bits D2, D1 and D0 do not have any effect on the masks.

M7.5 – D2 = 0, RST7.5 is enabled
D2=1, RST7.5 is masked or disabled

M6.5 – D1 = 0, RST6.5 is enabled
D1=1, RST6.5 is masked or disabled

M5.5 – D0 = 0, RST5.5 is enabled
D0=1, RST5.5 is masked or disabled

Note: SOD and SOE are irrelevant for interrupt setting.

For example, the following instruction sequence enables RST6.5 and disables RST7.5 and RST5.5:

```
MVI A, 00011101B
SIM
EI
```

3. EXERCISE

Connect the circuit as shown in Figure 1 and write the following program. Test the program using the 8085 development system by pressing the pushbutton switch connected to RST6.5.

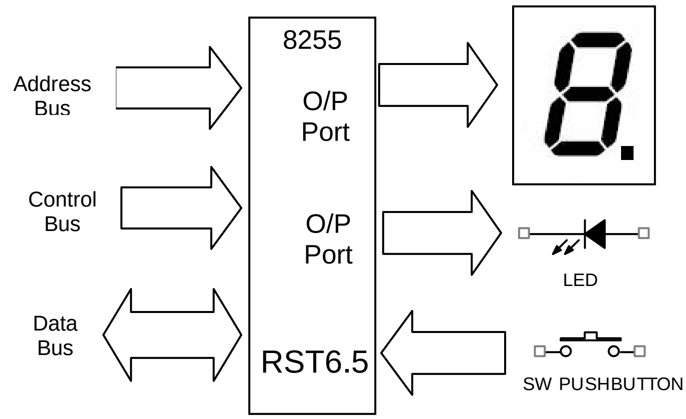


Figure 1

```
            ORG 2000H
            LXI SP, 3FF0H
            MVI A, 80H
            OUT 83H
            MVI A, 00011010B
            SIM
            EI

START:      LXI H, TABLE
            MVI B, 10
NEXT:       MOV A, M
            OUT 80H
            CALL DELAY
            INX H
            DCR B
            JNZ NEXT
            JMP START
TABLE:      DFB   __, __, __, __, __, __, __, __, __, __

DELAY:      MVI D, 255
LOOP1:      MVI E, 255
LOOP:       DCR E
            JNZ LOOP
            DCR D
            JNZ LOOP1
            RET
```

```
; Interrupt Service Routine RST6.5
ISR6_5:     PUSH PSW
            PUSH B
            PUSH D
            MVI A, 0
            MVI B, 15
REPEAT:     CMA
            OUT 82H
            CALL DELAY
            DCR B
            JNZ REPEAT
            POP D
            POP B
            POP PSW
            EI
            RET

            ORG 3F34H
            JMP ISR6_5
            END
```

Discuss your result. Connect another pushbutton switch to the RST5.5 pin. Modify your existing program to make the LED blink 20 times when the pushbutton switch is pressed.

REFERENCES

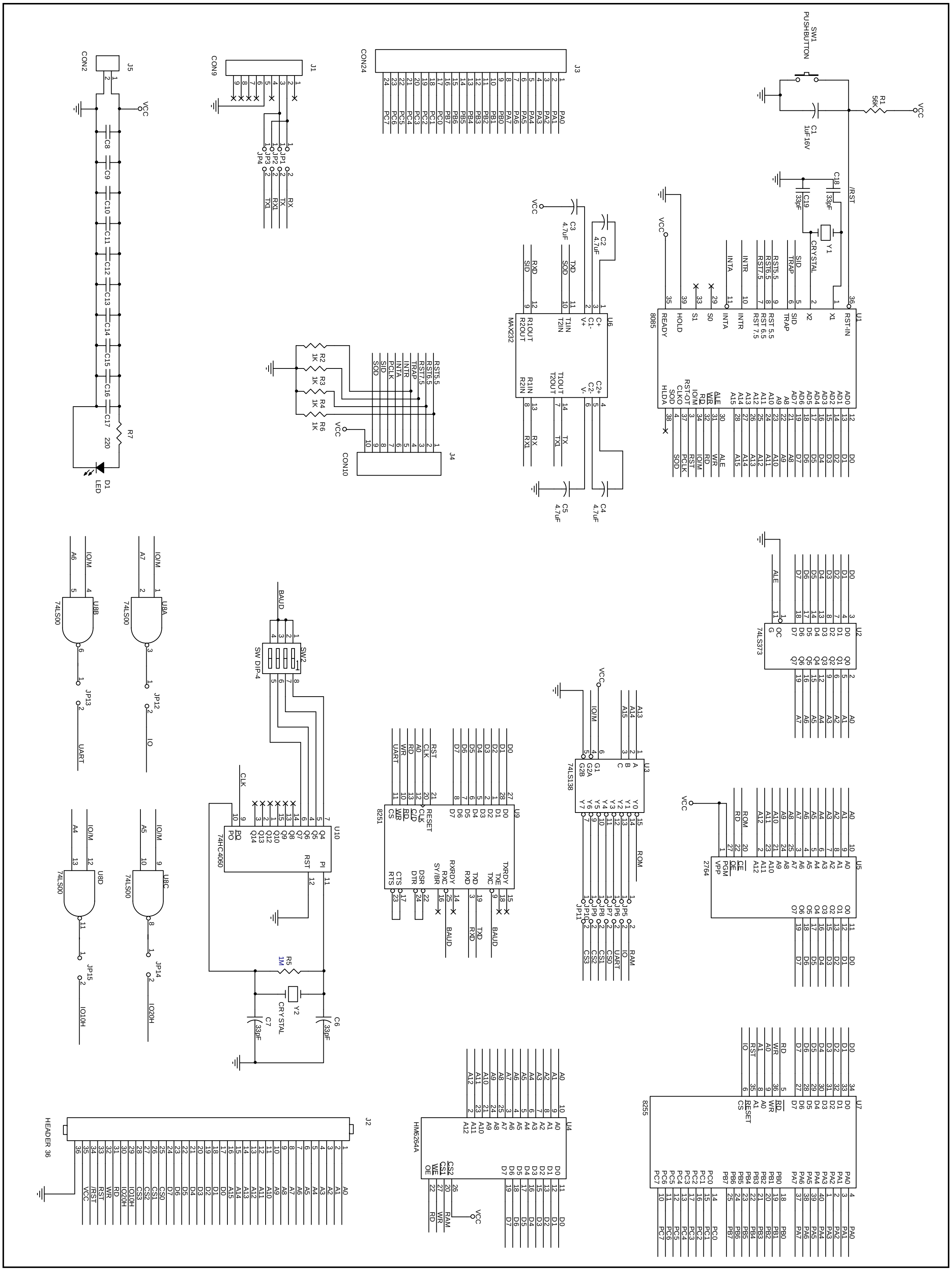
Gaonkar, Ramesh, *Microprocessor Architecture, Programming and Applications with the 8085,* Prentice-Hall.

William Kleitz (1998), *Microprocessor and Microcontroller Fundamentals : The 8085 and 8051 Hardware and Software,* Prentice-Hall.

Gilmore, C.M. (1996). *Microprocessors : Principles and Applications*. McGraw-Hill.

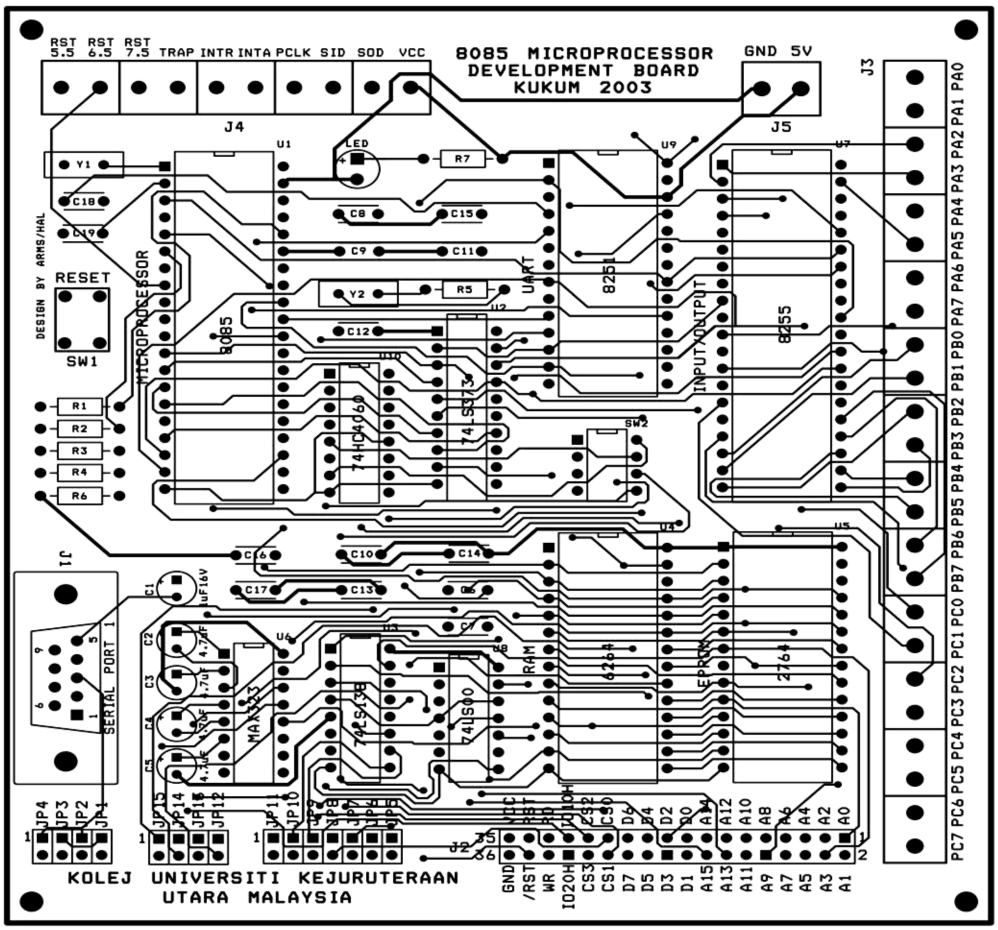
Short, K.L. (1998). *Embedded Microprocessor Systems Design*. Prentice-Hall.

APPENDIX 1 : SCHEMATIC DRAWING

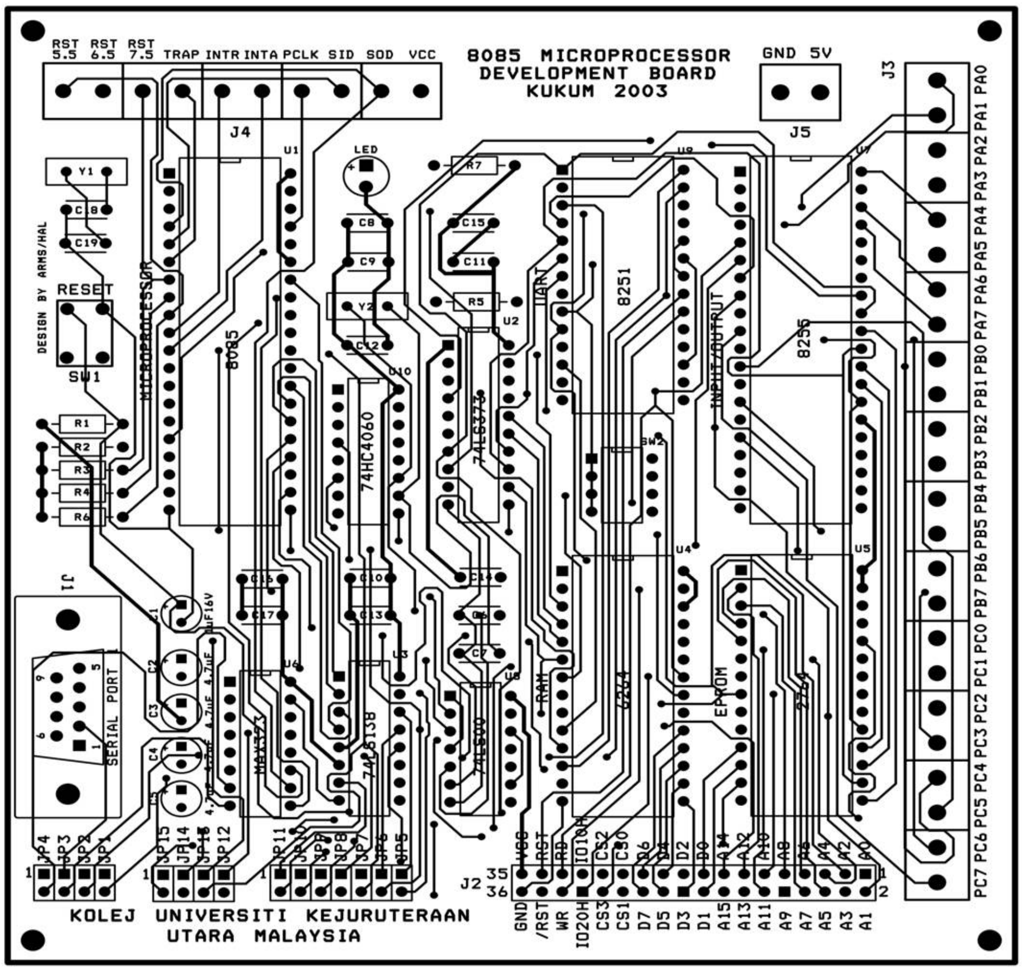


APPENDIX 2 : PCB LAYOUT

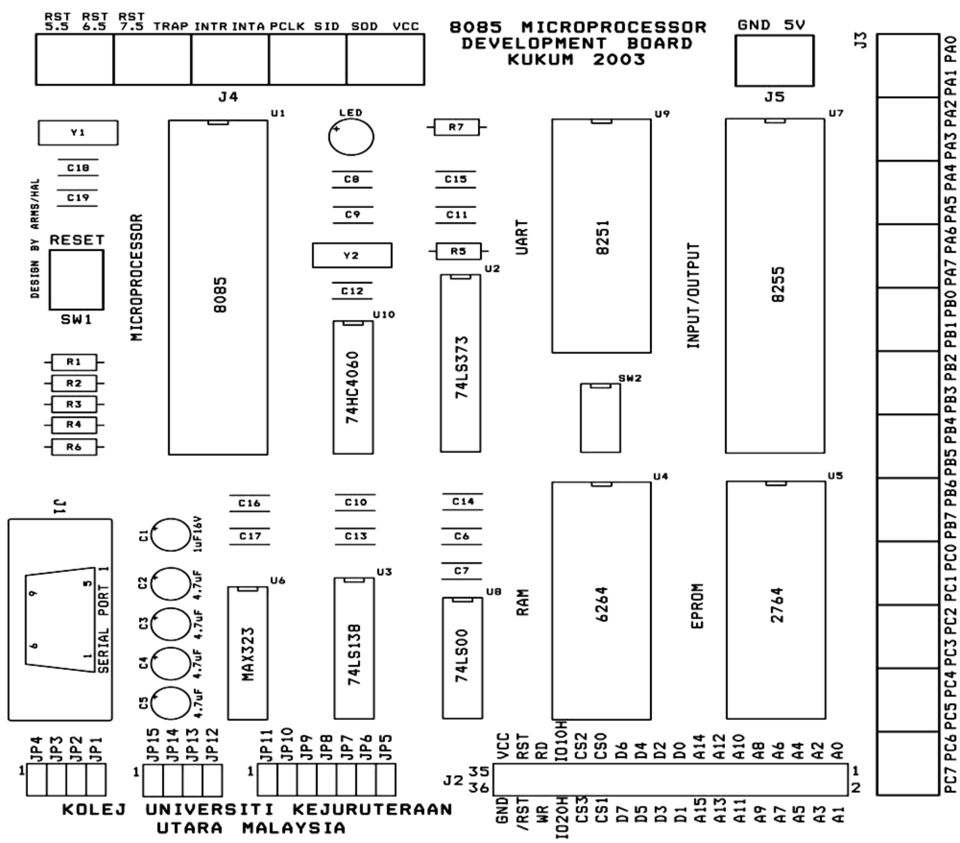
TOP LAYER LAYOUT



BOTTOM LAYER LAYOUT



COMPONENT OVERLAY



APPENDIX 3 : 8080A/8085A INSTRUCTION SET INDEX

Instruction	Code	Bytes	T States		Machine Cycles
			8085A	8080A	
ACI DATA	CE data	2	7	7	F R
ADC REG	1000 1SSS	1	4	4	F
ADC M	8E	1	7	7	F R
ADD REG	1000 0SSS	1	4	4	F
ADD M	86	1	7	7	F R
ADI DATA	C6 data	2	7	7	F R
ANA REG	1010 0SSS	1	4	4	F
ANA M	A6	1	7	7	F R
ANI DATA	E6 data	2	7	7	F R
CALL LABEL	CD addr	3	18	17	S R R W W*
CC LABEL	DC addr	3	9/18	11/17	S R./S R R W W*
CM LABEL	FC addr	3	9/18	11/17	S R./S R R W W*
CMA	2F	1	4	4	F
CMC	3F	1	4	4	F
CMP REG	1011 1SSS	1	4	4	F
CMP M	BE	1	7	7	F R
CNC LABEL	D4 addr	3	9/18	11/17	S R./S R R W W*
CNZ LABEL	C4 addr	3	9/18	11/17	S R./S R R W W*
CP LABEL	F4 addr	3	9/18	11/17	S R./S R R W W*
CPE LABEL	EC addr	3	9/18	11/17	S R./S R R W W*
CPI DATA	FE data	2	7	7	F R
CPO LABEL	E4 addr	3	9/18	11/17	S R./S R R W W*
CZ LABEL	CC addr	3	9/18	11/17	S R./S R R W W*
DAA	27	1	4	4	F
DAD RP	00RP 1001	1	10	10	F B B
DCR REG	00SS S101	1	4	5	F*
DCR M	35	1	10	10	F R W
DCX RP	00RP 1011	1	6	5	S*
DI	F3	1	4	4	F
EI	FB	1	4	4	F
HLT	76	1	5	7	F B
IN PORT	DB data	2	10	10	F R I
INR REG	00SS S100	1	4	5	F*
INR M	34	1	10	10	F R W
INX RP	00RP 0011	1	6	5	S*
JC LABEL	DA addr	3	7/10	10	F R/F R R †
JM LABEL	FA addr	3	7/10	10	F R/F R R †
JMP LABEL	C3 addr	3	10	10	F R R
JNC LABEL	D2 addr	3	7/10	10	F R/F R R †
JNZ LABEL	C2 addr	3	7/10	10	F R/F R R †
JP LABEL	F2 addr	3	7/10	10	F R/F R R †
JPE LABEL	EA addr	3	7/10	10	F R/F R R †
JPO LABEL	E2 addr	3	7/10	10	F R/F R R †
JZ LABEL	CA addr	3	7/10	10	F R/F R R †
LDA ADDR	3A addr	3	13	13	F R R R
LDAX RP	000X 1010	1	7	7	F R
LHLD ADDR	2A addr	3	16	16	F R R R R

Instruction	Code	Bytes	T States		Machine Cycles
			8085A	8080A	
LXI RP,DATA 16	00RP 0001 data16	3	10	10	F R R
MOV REG,REG	01DD DSSS	1	4	5	F*
MOV M,REG	0111 0SSS	1	7	7	F W
MOV REG, M	01DD D110	1	7	7	F R
MVI REG, DATA	00DD D110 data	2	7	7	F R
MVI M, DATA	36 data	2	10	10	F R W
NOP	00	1	4	4	F
ORA REG	1011 0SSS	1	4	4	F
ORA M	B6	1	7	7	F R
ORI DATA	F6 data	2	7	7	F R
OUT PORT	D3 data	2	10	10	F R O
PCHL	E9	1	6	5	S*
POP RP	11 RP 0001	1	10	10	F R R
PUSH RP	11 RP 0101	1	12	11	S W W*
RAL	17	1	4	4	F
RAR	1F	1	4	4	F
RC	D8	1	6/12	5/11	S/S R R*
RET	C9	1	10	10	F R R
RIM (8085A only)	20	1	4	-	F
RLC	07	1	4	4	F
RM	F8	1	6/12	5/11	S/S R R*
RNC	D0	1	6/12	5/11	S/S R R*
RNZ	C0	1	6/12	5/11	S/S R R*
RP	F0	1	6/12	5/11	S/S R R*
RPE	E8	1	6/12	5/11	S/S R R*
RPD	E0	1	6/12	5/11	S/S R R*
RRC	0F	1	4	4	F
RST N	11XX X111	1	12	11	S W W*
RZ	C8	1	6/12	5/11	S/S R R*
SBB REG	1001 1SSS	1	4	4	F
SBB M	9E	1	7	7	F R
SBI DATA	DE data	2	7	7	F R
SHLD ADDR	22 addr	3	16	16	F R R W W
SIM (8085A only)	30	1	4	-	F
SPHL	F9	1	6	5	S*
STA ADDR	32 addr	3	13	13	F R R W
STAX RP	000X 0010	1	7	7	F W
STC	37	1	4	4	F
SUB REG	1001 0SSS	1	4	4	F
SUB M	96	1	7	7	F R
SUI DATA	D6 data	2	7	7	F R
XCHG	EB	1	4	4	F
XRA REG	1010 1SSS	1	4	4	F
XRA M	AE	1	7	7	F R
XRI DATA	EE data	2	7	7	F R
XTHL	E3	1	16	18	F R R W W

Machine cycle types:

F	**Four clock period instr fetch**		
S	**Six clock period instr fetch**		
R	**Memory read**		
I	**I/O read**		
W	**Memory write**		
O	**I/O write**		
B	**Bus idle**		
X	**Variable or optional binary digit**		
DDD	**Binary digits identifying a destination register**		**B= 000, C=001, D=010, Memory=110**
SSS	**Binary digits identifying a source register**		**E= 011, H=100, L=101, A=111**
RP	**Register Pair DE=01.SP=11**	**BC=00, HL=10 DE=01, SP=11**	

*** Five clock period instruction fetch with 8080A**

† The longer machine cycle sequence applies regardless of condition evaluation with 8080A

. An extra READ cycle (R) will occur for this condition with 8080A

APPENDIX 4 : 8085A INSTRUCTION SET

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Move from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct

STA	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H

Store accumulator indirect

STAX	Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B

Store H and L registers direct

SHLD	16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD 2470H

Exchange H and L with D and E

XCHG	none	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG

Copy H and L registers to the stack pointer

SPHL	none	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL

Exchange H and L with top of stack

XTHL	none	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL

Push register pair onto stack

PUSH Reg. pair	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example: PUSH B or PUSH PSW

Pop off stack to register pair

POP Reg. pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP PSW

Output data from accumulator to a port with 8-bit address

OUT 8-bit port address	The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT F8H

Input data to accumulator from a port with 8-bit address

IN 8-bit port address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 8CH

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB R
M

The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB R
M

The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data

The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SUI 45H

Subtract immediate from accumulator with borrow

SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SBI 45H

Increment register or memory by 1

INR R
M

The contents of the designated register or memory are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: INR B or INR M

Increment register pair by 1

INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.
Example: INX H

Decrement register or memory by 1

DCR	R M	The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: DCR B or DCR M

Decrement register pair by 1

DCX	R	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H

Decimal adjust accumulator

DAA	none	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example: DAA

BRANCHING INSTRUCTIONS

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034H or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.

Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call

CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.

Example: CALL 2034H or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.

Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

RET none

The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

PCHL none

The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example: PCHL

Restart

RST 0-7

The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

Opcode	Operand	Description

Compare register or memory with accumulator

CMP R, M — The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:

if (A) < (reg/mem): carry flag is set
if (A) = (reg/mem): zero flag is set
if (A) > (reg/mem): carry and zero flags are reset

Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data — The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

if (A) < data: carry flag is set
if (A) = data: zero flag is set
if (A) > data: carry and zero flags are reset

Example: CPI 89H

Logical AND register or memory with accumulator

ANA R, M — The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.

Example: ANA B or ANA M

Logical AND immediate with accumulator

ANI 8-bit data — The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.

Example: ANI 86H

Exclusive OR register or memory with accumulator

XRA R, M — The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI 8-bit data

The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRI 86H

Logical OR register or memory with accumulator

ORA R
M

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI 8-bit data

The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORI 86H

Rotate accumulator left

RLC none

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7.S, Z, P, AC are not affected.

Example: RLC

Rotate accumulator right

RRC none

Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0.S, Z, P, AC are not affected.
Example: RRC

Rotate accumulator left through carry

RAL none

Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.CY is modified according to bit D7. S, Z, P, AC are not affected.

Example: RAL

Rotate accumulator right through carry

RAR	none	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RAR

Complement accumulator

CMA	none	The contents of the accumulator are complemented. No flags are affected. Example: CMA

Complement carry

CMC	none	The Carry flag is complemented. No other flags are affected. Example: CMC

Set Carry

STC	none	The Carry flag is set to 1. No other flags are affected. Example: STC

CONTROL INSTRUCTIONS

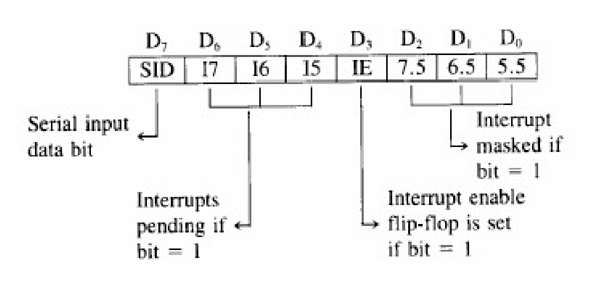
Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
HLT	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
EI	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts. (except TRAP). Example: EI

Read interrupt mask

RIM none

This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM

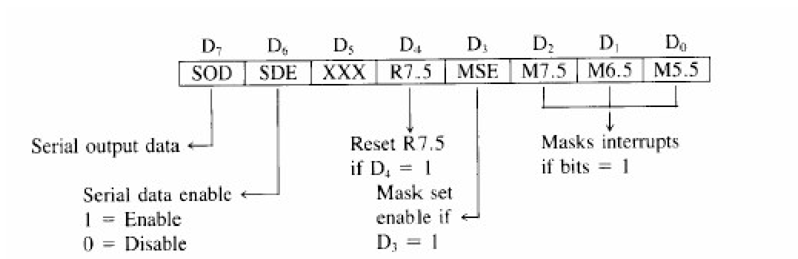


Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



SOD – Serial Output Data: Bit D7 of Accumulator is latched into the SOD output line and made available to serial peripheral if bit D6 = 1
SOE – Serial Output Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
XXX – Don't care
R7.5 – Reset RST7.5: If this bit = 1, RST7.5 flip-flop is reset. This is an additional control to reset RST7.5.
MSE – Mask Set Enable: If this bit is high, it enables the function of bits D2, D1 and D0. This is a master control over all the interrupt masking bits. If this bit is low, bits D2, D1 and D0 do not have any effect on the masks.
M7.5 – D2 = 0, RST7.5 is enabled
D2=1, RST7.5 is masked or disabled
M6.5 – D1 = 0, RST6.5 is enabled
D1=1, RST6.5 is masked or disabled
M5.5 – D0 = 0, RST5.5 is enabled
D0=1, RST5.5 is masked or disabled

APPENDIX 5 : ASCII Table

Dec	Hex	Symbol	Dec	Hex	Symbol
0	0	NUL	64	40	@
1	1	SOH	65	41	A
2	2	STX	66	42	B
3	3	ETX	67	43	C
4	4	EOT	68	44	D
5	5	ENQ	69	45	E
6	6	ACK	70	46	F
7	7	BEL	71	47	G
8	8	BS	72	48	H
9	9	TAB	73	49	I
10	A	LF	74	4A	J
11	B	VT	75	4B	K
12	C	FF	76	4C	L
13	D	CR	77	4D	M
14	E	SO	78	4E	N
15	F	SI	79	4F	O
16	10	DLE	80	50	P
17	11	DC1	81	51	Q
18	12	DC2	82	52	R
19	13	DC3	83	53	S
20	14	DC4	84	54	T
21	15	NAK	85	55	U
22	16	SYN	86	56	V
23	17	ETB	87	57	W
24	18	CAN	88	58	X
25	19	EM	89	59	Y
26	1A	SUB	90	5A	Z
27	1B	ESC	91	5B	[
28	1C	FS	92	5C	\
29	1D	GS	93	5D	]
30	1E	RS	94	5E	^
31	1F	US	95	5F	_
32	20	(space)	96	60	`
33	21	!	97	61	a
34	22	"	98	62	b
35	23	#	99	63	c
36	24	$	100	64	d
37	25	%	101	65	e
38	26	&	102	66	f
39	27	'	103	67	g
40	28	(	104	68	h
41	29	)	105	69	i
42	2A	*	106	6A	j
43	2B	+	107	6B	k
44	2C	,	108	6C	l
45	2D	-	109	6D	m
46	2E	.	110	6E	n
47	2F	/	111	6F	o
8	30	0	112	70	p
49	31	1	113	71	q
50	32	2	114	72	r
51	33	3	115	73	s
52	34	4	116	74	t
53	35	5	117	75	u
54	36	6	118	76	v
55	37	7	119	77	w
56	38	8	120	78	x
57	39	9	121	79	y
58	3A	:	122	7A	z
59	3B	;	123	7B	{
60	3C	<	124	7C	|
61	3D	=	125	7D	}
62	3E	>	126	7E	~
63	3F	?	127	7F	