CHAPTER 4 MICROPROCESSOR APPLICATION

- Interfacing of Keyboards
- Interfacing of seven segment LED Display
- Study of traffic light system
- Stepper Motor Controller

INTRODUCTION

A microprocessor is used as the central processing unit of a computer. They are used in industries for applications like process control, control of machines and equipment. They are also used in measurment, display and control of temperature, pressure, voltage, current, frequency, water level etc.

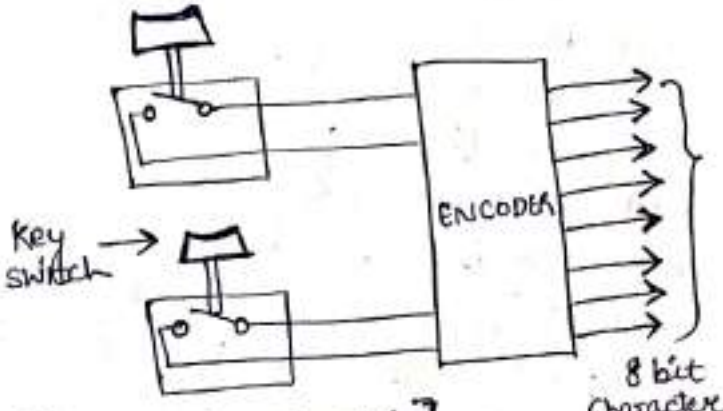
Microprocessor based systems are also used for communication purposes, traffic light controls, in military equipments, robotics, voltage control of generators etc.

KEYBOARD INTERFACING

It is a human oriented input peripheral. It is used to input data or program into the microcomputer. It consists of push button type switches. When a key is pressed, the microprocessor identifies key depression and then performs appropriate operation.

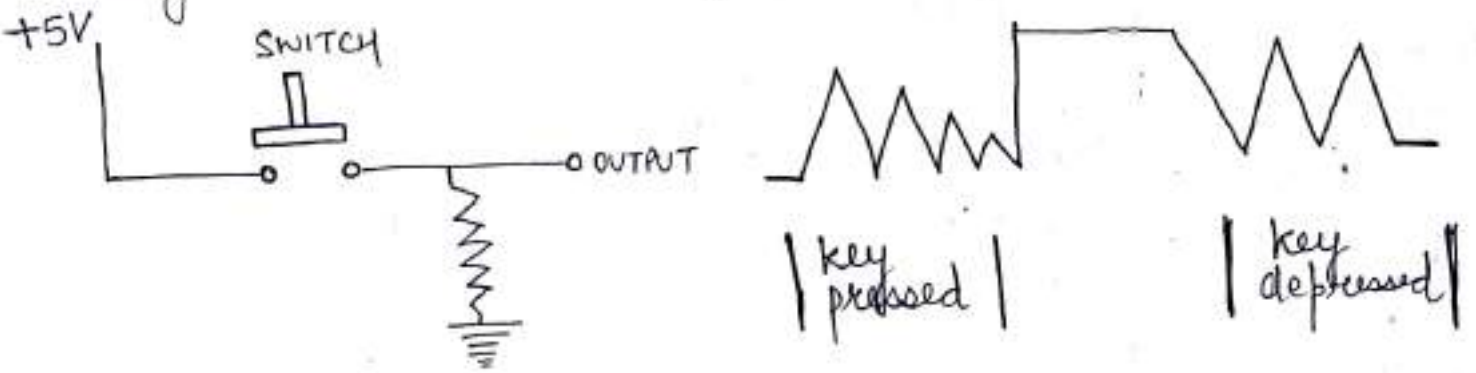
1. Key switch Mechanism

The aim of this mechanism is to generate and transmit a code each time a key is pressed. The mechanism should send one and only proper code, when the key is pressed.



[Generation operation of a keyboard]

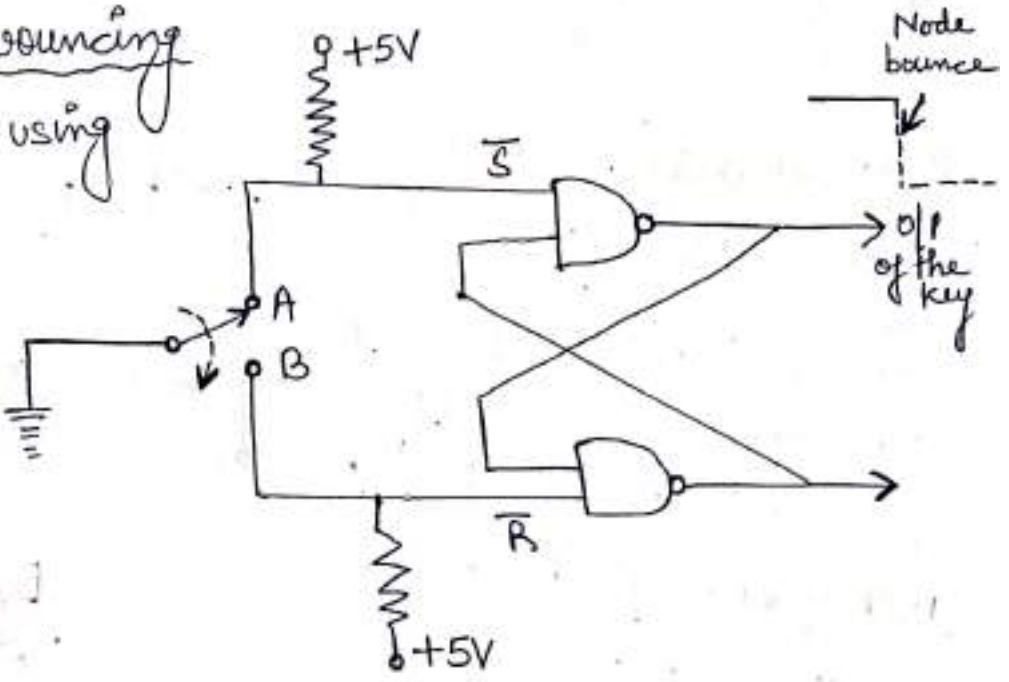
The input keyboard is composed of a set of labelled push button switches. Each switch makes electrical contact when pressed. The nature of the contact should be reliable, have long life and feel right. In case of a push button key, the metal contact bounces few times, hence the voltage across the switch fluctuates and generates spikes in the signal. Therefore, it is necessary to debounce the mechanical switches. The key debouncing is done through hardware and software.



(BOUNCING OF KEY SWITCH)

2. Hardware key Debouncing

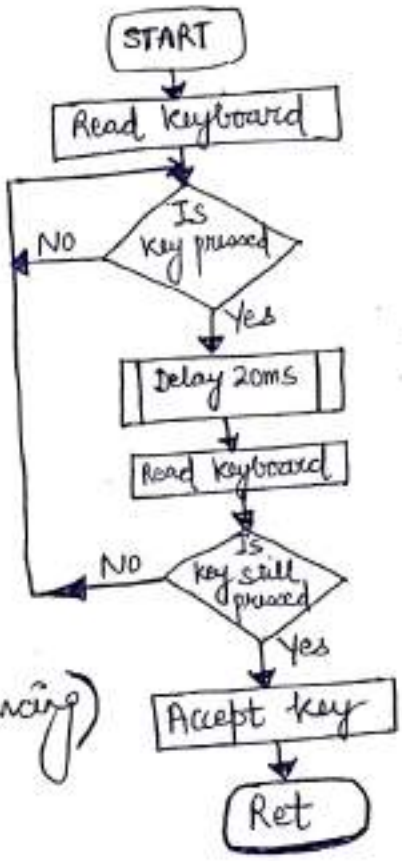
It is implemented by using flip flop or latch. When the wiper is connected to A, the output of the latch goes high. When the key makes contact with B, the output changes from 1 to logic 0. The wiper bounces many times on contact B, but the o/p doesnot fluctuate, b/w logic 1 and logic 0. When the wiper is not connected either to A or B, the output of the latch remains constant.



(Hardware key debouncing)

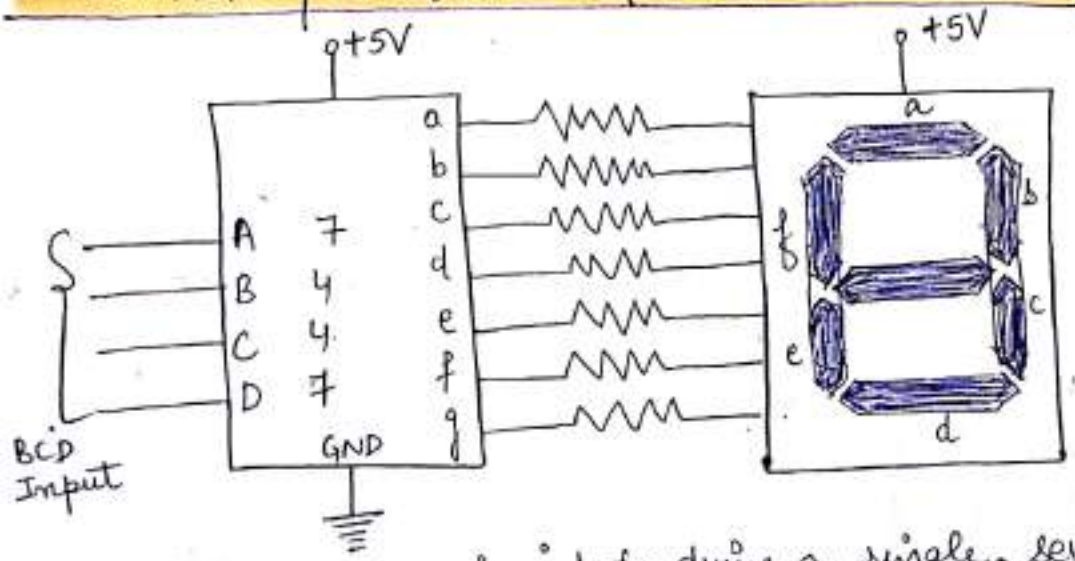
3. Software key Debouncing

In the software technique the microprocessor waits for 20ms before it accepts the key as an input. If after 20ms the key is pressed, the key is accepted by microprocessor.



(Software key debouncing)

INTERFACING OF SEVEN SEGMENT LED DISPLAY



(circuit for driving single seven segment LED display)

Figure shows, a circuit to drive a single, seven segment, common anode LED display. For common anode display, all the anodes are shorted and connected to positive supply (i.e +5V) and a low voltage is applied to the cathode to turn it on.

BCD to seven segment decoder, IC 7447 is used. The BCD to seven segment decoder is used to apply voltages at the cathodes according to the BCD inputs applied.

Inorder to restrict the current through the LED segments resistors are connected in series with the segments. This

Circuit connection is called as static display because the current is being passed through the display at all times.

The static display circuits work well just for driving one or two LED digits. These circuits are not capable of driving more number of LED digits say 8 digits.

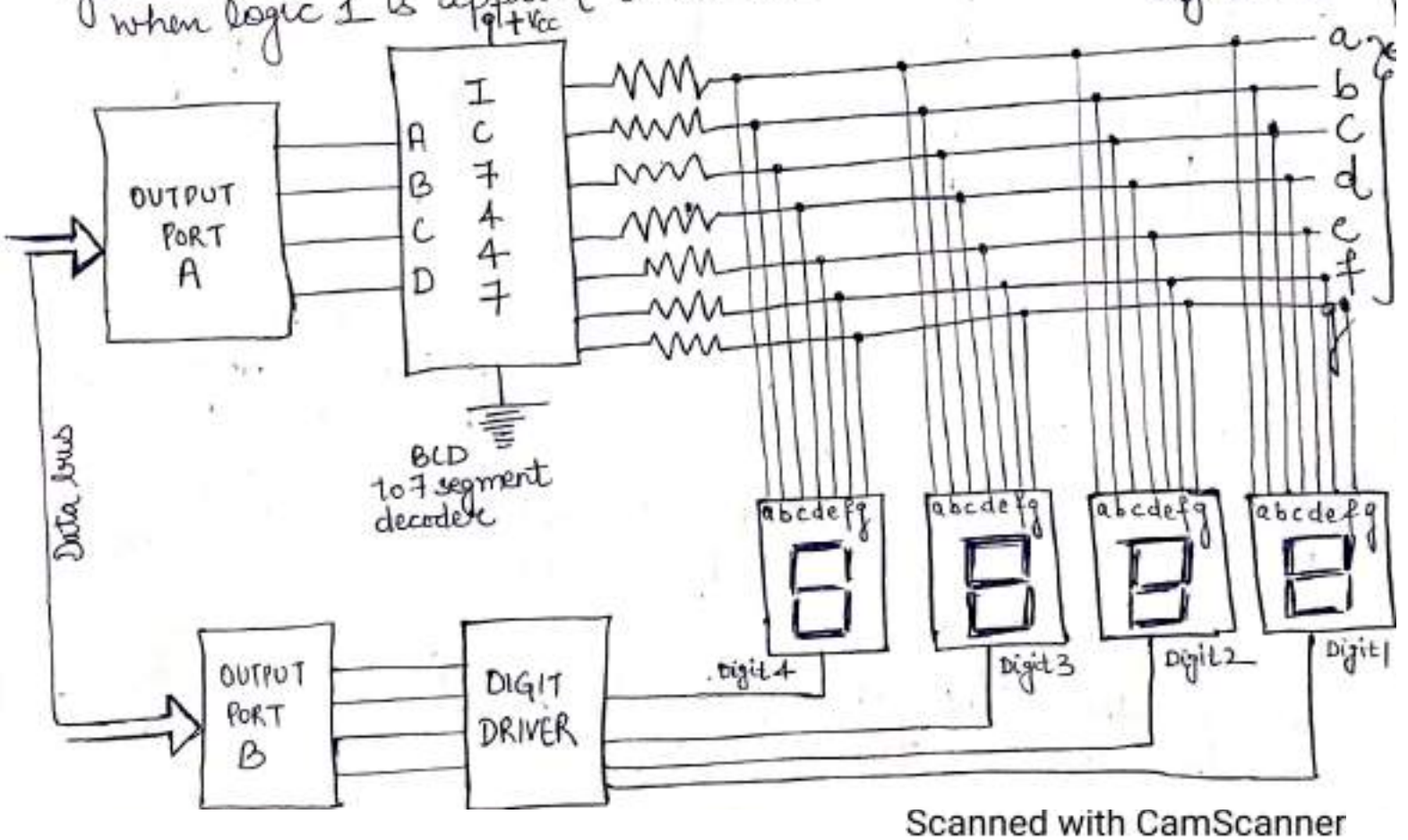
For common anode arrangement the current required to glow each segment is 15MA.

The problems that arise when number of digits increase are:

(1) Power consumption.

(2) Each display requires a separate BCD to 7 segment decoder.

In order to solve the above problems scanned multiplexed displays are used. figure shows interfacing of scanned multiplexed display: 4 seven segment displays are connected using multiplexing. Common anode arrangement is used ie all the anodes of LEDs are shorted and connected to positive supply. LED will glow when logic 0 is applied to the cathode. If common cathode arrangement is desired then all the cathodes of LEDs are shorted and connected to ground. LED will glow only when logic 1 is applied to the anode.

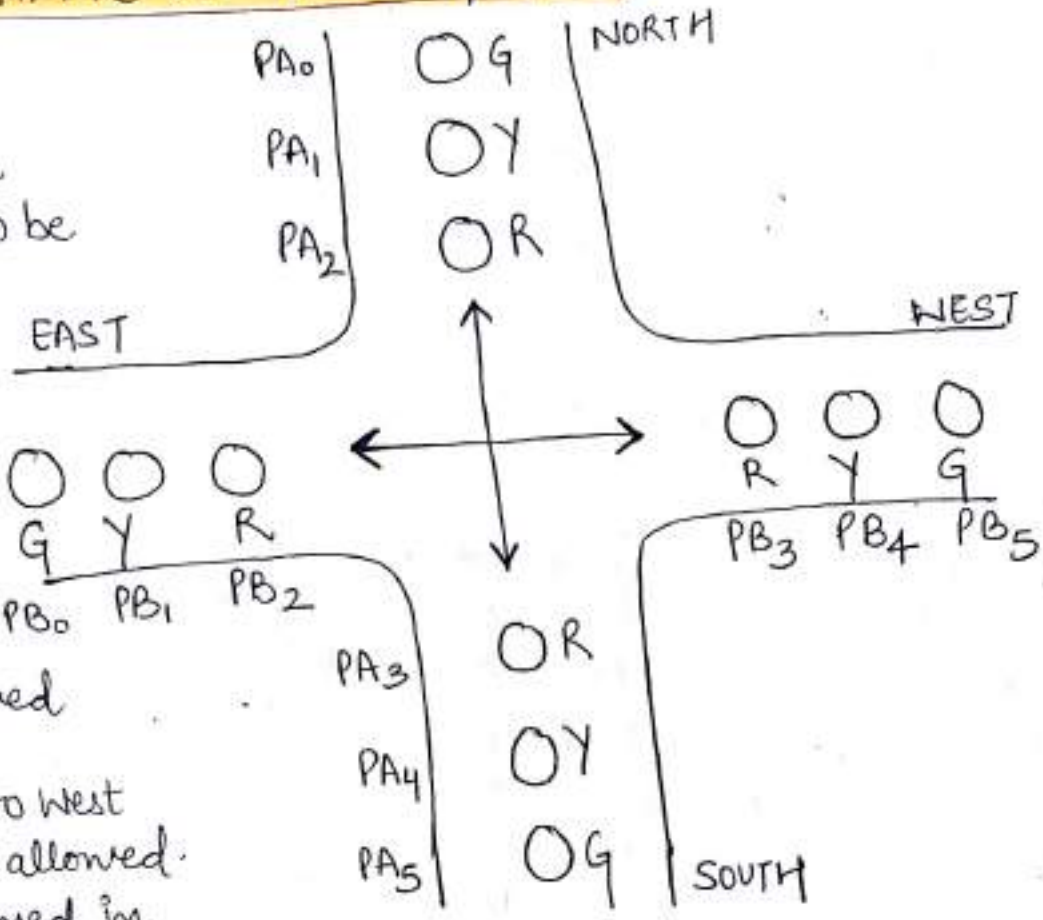


- As shown, the circuit has two output ports Port A and Port B. Port A is used to drive the LED segments and Port B is used to turn on the respective cathodes, so that the digits can be displayed.
- The output data lines of Port A are connected to seven segments of the LED and the output data line of Port B are connected to the cathodes of each LED.
- If all the digits are turned on at the same time then they will show the same number. But, in case of multiplexed display, the segment information is sent for all digits on common lines, but only one display digit is turned on at a time.
- The digit driver is composed of PNP transistors that are connected in series with the common anode of each digit and acts as the ON-OFF switch for that digit.
- The BCD code for digit 1 is first output from Port A to the 7447. The 7447, BCD to seven segment decoder outputs the seven segment code on the respective segment bus lines. The transistor connected to digit 1 is then turned on by outputting a low to that bit of Port B. All the rest bits of Port B are made high to ensure that digits 2, 3 and 4 remains off.
- After some delay say 1ms, digit 1 is turned off outputting all highs to port B. Now digit 2 will turned on by outputting the BCD code for digit 2 from port A to BCD to seven segment decoder and output of the bit pattern to turn on digit 2 on port B.
- After 1ms digit 2 is turned off and the cycle is repeated for digit 3 and digit 4. After the completion of turn of each digit, all the digits are lit again in turn.
- As each digit is sequentially turned on and off in multiplexed fashion, the display is called as <u>scanned multiplexed display</u>.
- With 4 digits and 1ms per digit, we get back to digit 1 every 4ms or about 250 times a second. This refresh

rate is fast enough, to our eye. Due to persistence of vision we will not be able to distinguish b/w two digits and all the digits appear to be ON at the same time.

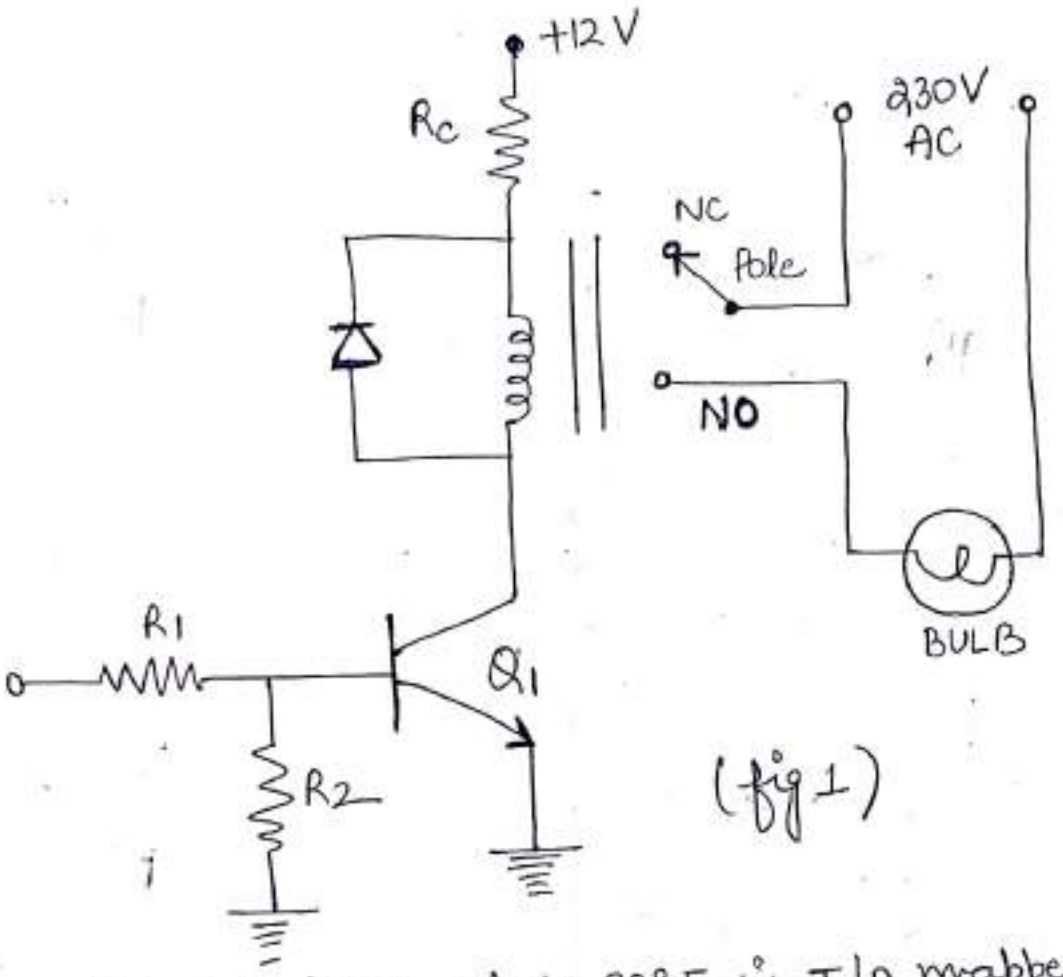
STUDY OF TRAFFIC CONTROL SYSTEM

Ex: A traffic controller is to be interfaced to 8085 through 8255. The traffic signals are located on cross roads. Traffic is allowed from North to South or East to West only. No turns allowed. Traffic is allowed in one direction for 55 seconds and yellow to red transitions time in 5 seconds.



Sol^n:-

Step1: Traffic signals use normal bulbs which require 230V AC as supply. The microcomputer works on 5V DC so we require a special interfacing arrangement as shown The interfacing uses a transistorised circuit to drive relay. The relay will make bulb ON or OFF which depends on input to transistorised circuit, when base of Q1 is logic high, the transistor will be ON and pass current through relay coil, which will make bulb ON. When base of Q1 is logic low, the transistor will be OFF and will not pass current through relay coil, which will make bulb OFF.



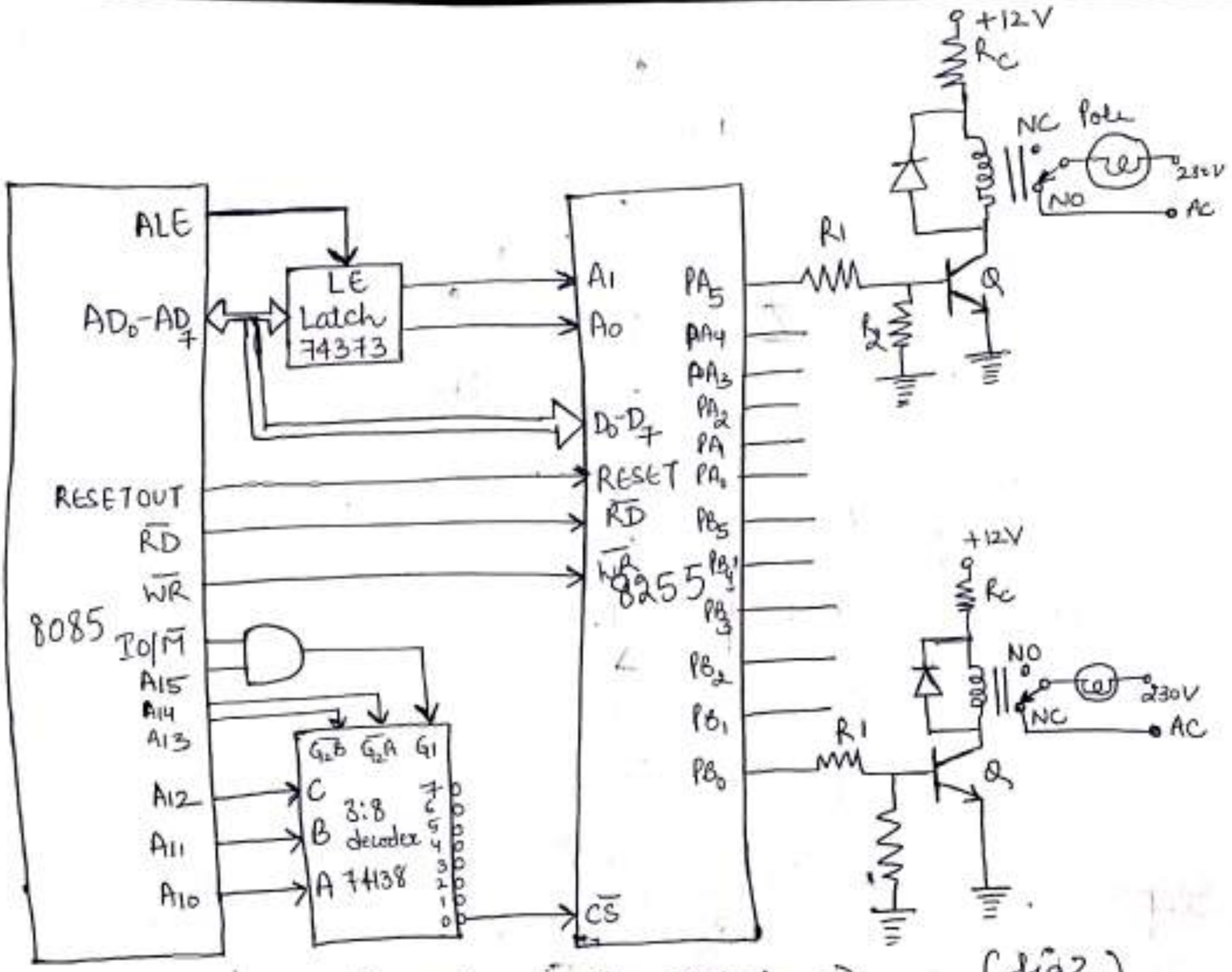
(fig 1)

Step2: The 8255 is interfaced to 8085 in I/O mapped I/O. In it, lines are directly connected and remaining lines to or to are used for decoder logic to generate chip select signal.

Step3: The transistorised circuit is interfaced to 8255 port bits and total interfacing schematic will be as shown in fig(2)

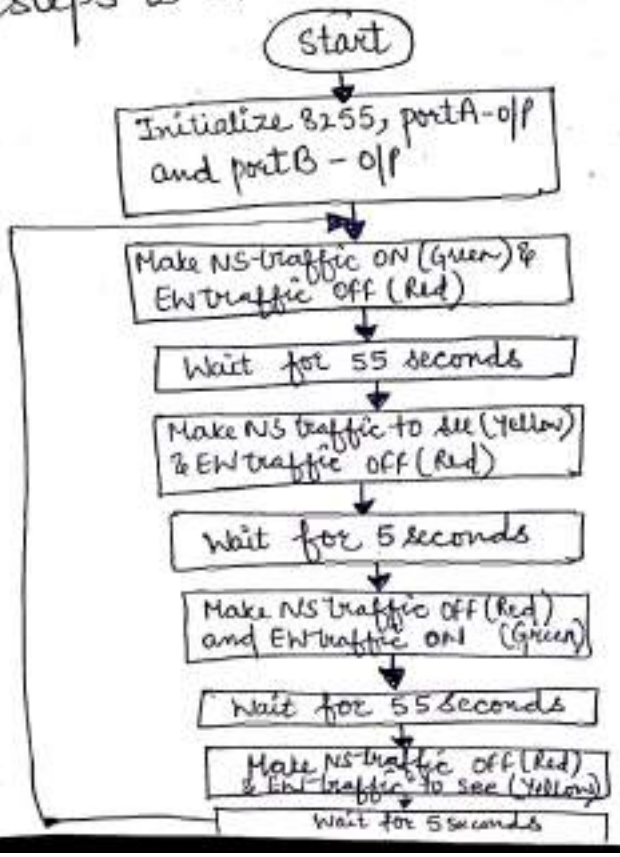
step4: The address of 8255 ports will be as follows:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_1	A_0	
0	1	1	0	0	0	0	0	= 60H PORT A
0	1	1	0	0	0	0	1	= 61H PORT B
0	1	1	0	0	0	1	0	= 62H PORT C
0	1	1	0	0	0	1	1	= 63H CWR



(Traffic Controller Interface) (fig2)

Step 5: The flowchart required to implement traffic controller steps is shown



Step6: The port A connected to NS traffic, port B connected to EW traffic. There are 4 possible combinations of time slots as follows:

TIME SLOT	TIME	NS	EW
T_1	55 sec	Green	Red
T_2	5 sec	Yellow	Red
T_3	55 sec	Red	Green
T_4	5 sec	Red	Yellow

The data required to achieve the above combination of time volts will be as follows:

(1) T_1 slot:

ON NS $PA_0=1$, $PA_5=1$, $PA_1=0$, $PA_2=0$, $PA_3=0$, $PA_4=0$

OFF EW $PB_0=0$, $PB_1=0$, $PB_2=1$, $PB_3=1$, $PB_4=0$, $PB_5=0$

NS	X	X	1	0	0	0	0	1	= 21H
EW	X	X	0	0	1	1	0	0	= 0CH

(2) T_2 Slot:

NS	X	X	0	1	0	0	1	0	= 12H
EW OFF	X	X	0	0	1	1	0	0	= 0CH

(3) T_3 Slot:

NS OFF	X	X	0	0	1	1	0	0	= 0CH
EW ON	X	X	1	0	0	0	0	1	= 21H

(4) T_4 Slot:

NS OFF	X	X	0	0	1	1	0	0	= 0CH
EW	X	X	0	1	0	0	1	0	= 12H

Step 7: PROGRAM:

Label	Instructions	Operation
	LXI SP, FFFFH	FFFF → SP
	MVI A, 80H	80 → A
	OUT 63H	A → CWR
UP:	MVI A, 21H	21 A
	OUT 60H	A → Port A
	MVI, 0CH	0C → A
	OUT 61H	A → Port B
	CALL DELAY1	Wait for 55 sec
	MVI A, 12H	12 → a
	OUT 60H	A → Port A
	MVI A, 0CH	0C → A
	OUT 61H	A → Port B
	CALL	Wait for 5 sec
	MVI A, 0CH	0C → A
	OUT 60H	A → Port B
	MVI A, 21H	21 → A
	OUT 61H	A → port B
	CALL DELAY1	Wait for 55 sec
	MVI A, 0CH	0C → A
	OUT 60H	A → Port A
	MVI A, 12H	12 → A
	OUT 61H	A → Port B
	CALL	Wait for 5 sec
	JMP UP	

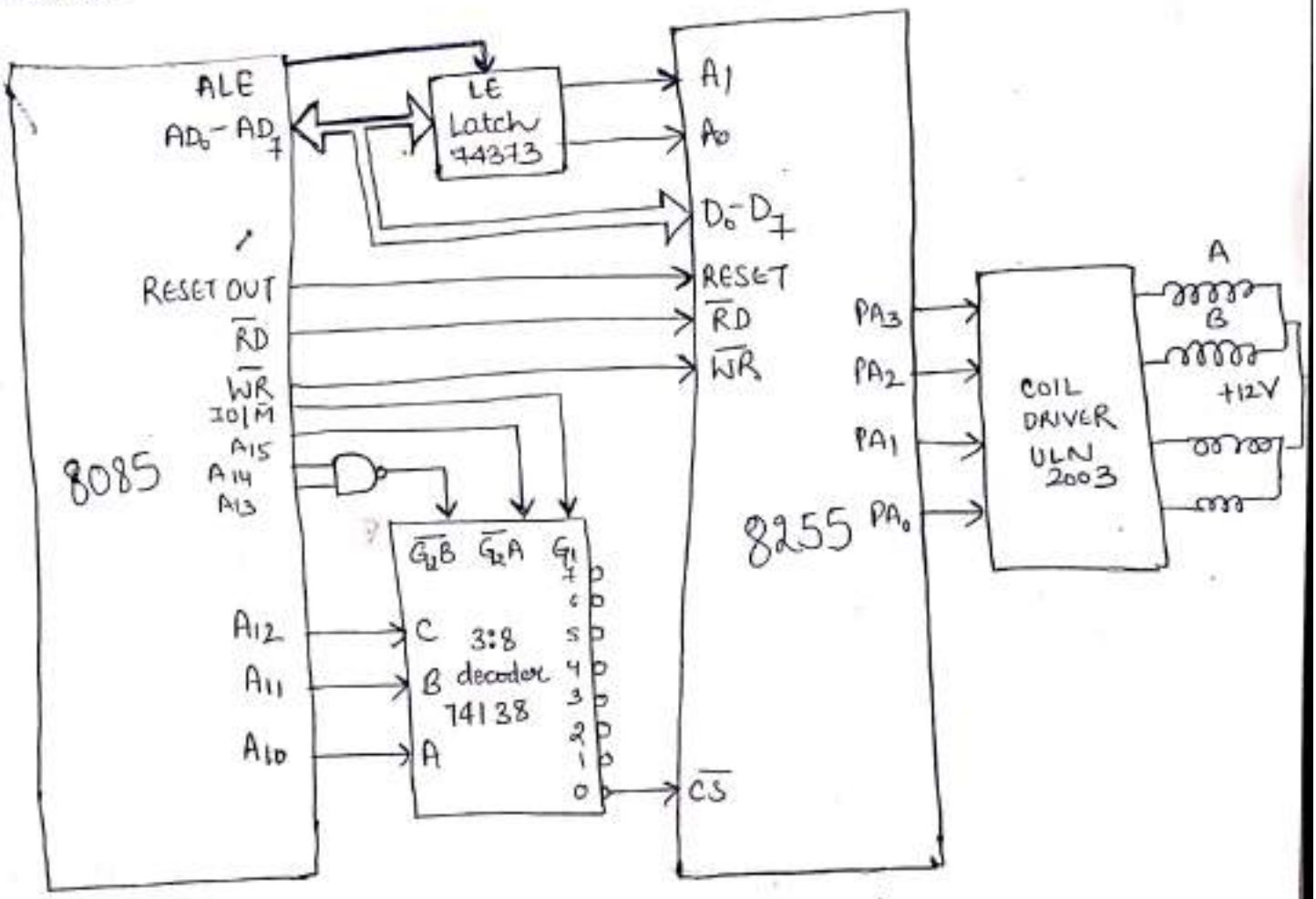
STEPPER MOTOR INTERFACING

Interface a stepper motor to 8085 using 8255. Interface 8255 in I/O mapped I/O.

Step1: The 8255 is interfaced to 8085 in I/O mapped I/O. The port is used to give steps to stepper motor. So function of port A will be output. No other signals except 4 step signals are required.

Step2: The 8255 provides very less current which will not be able to drive stepper motor coils so a coil driver ULN2003 is generally used or a transistorized driver can be used instead of ULN2003.

Step3: The interfacing diagram of stepper motor using 8255 will be

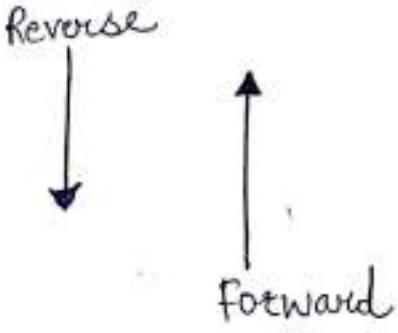


(Stepper Motor interface)

Step4: There are two possible step modes :

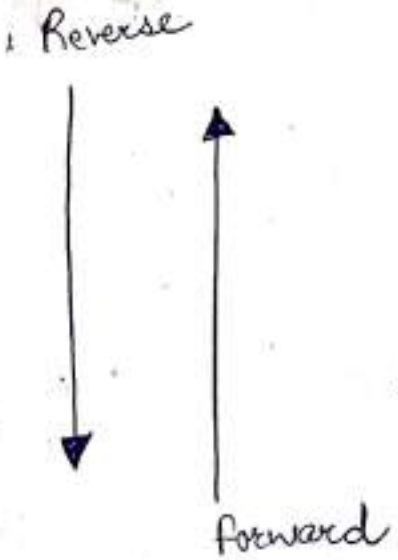
Mode 1 full stepping L:
In this case the motor moves by, to do this 2 bits are changed at a time, the bit patterns will be as follows:

A	B	C	D	
1	0	1	0	=0A
1	0	0	1	=09
0	1	0	1	=05
0	1	1	0	=06



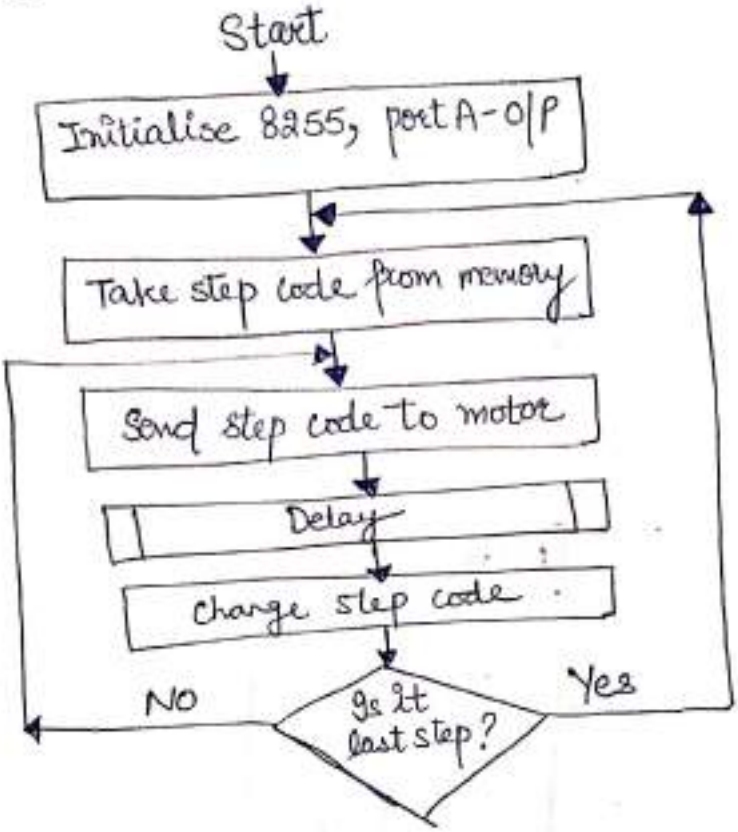
Mode 2 half stepping:
In this case the motor moves by, to do this 1 bit is changed at a time, the bit patterns will be as follows:

A	B	C	D	
1	0	1	0	=0A
1	0	0	0	=08
1	0	0	1	=09
0	0	0	1	=01
0	1	0	1	=05
0	1	0	0	=04
0	1	1	0	=06
0	0	1	0	=02
1	0	1	0	=0A



Step 5: The data is stored in memory and accessed using memory pointer. Depending on sequence in which data is stored, the motor will move in forward or reverse direction.

Step 6: To prepare a program for forward direction in full stepping mode, the flowchart required will be as



Step 7: Program for full stepping mode

LABEL	INSTRUCTIONS	OPERATION
	LXI SP, FFFFH	FFFF → SP
	MVI A, 80H	80 → A
	OUT CNR	A → CWR
BACK:	LXI H, C200H	C200 → HL
	MVI C, 04H	04 → C
UP:	MOV A, M	(HL) → A
	OUT PA	A → PA
	CALL DELAY	wait for delay
	INX H	HL+1 → HL
	DCR C	C1 → C
	JNZ UP	Is C=0, if no go to up
	JMP BACK	Go BACK

Data stored in memory for forward direction in full stepping mode will be as follows:

Address	Data
C200	0A
C201	09
C202	05
C203	06

The forward direction in full stepping mode can be easily changed to reverse direction by changing the sequence of data stored in memory as follows:

Address	Data
C200	06
C201	05
C202	09
C203	0A

If you are not interested in changing data stored in memory, just make few changes in program. The changes to be done are:

(i) LXI H, C203 instead of LXI H, C200

(ii) DCX H instead of INX H.

Step8: To prepare a program for forward direction in half stepping mode the flowchart required will be same as above flowchart.

Step9: Program for half stepping mode,

LABEL	INSTRUCTIONS	OPERATION
	LXI SP, FFFFH	FFFF → SP
	MVI A, 80H	80 → A
	OUT CWR	A → CWR
BACK:	LXI H, C200H	C200 → HL
	MVI C, 08H	08 → C
UP:	MOV A, M	(HL) → A
	OUT PA	A → PA
	CALL DELAY	Wait for delay
	INX H	HL+1 → HL
	DCR C	C-1 → C
	JNZ UP	Is C=0, is no go to UP
	JMP BACK	Go BACK

Data stored in memory for forward direction in half stepping mode will be as follows:

Address	Data
C200	0A
C201	08
C202	09
C203	01
C204	05
C205	04
C206	06