Need for DMA:

In microprocessor based systems data transfer can be controlled by either software & hardware.

Software controlled data transfer:

The following ~~steps~~ tasks are performed by MP to transfer data from I/O device to memory & from memory to I/O device.

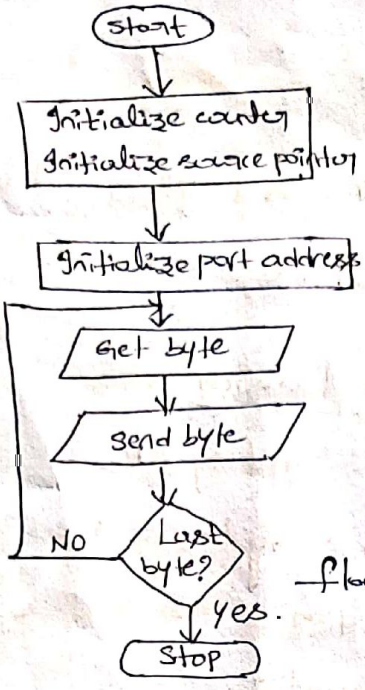
1) To fetch the instruction.
2) To decode the instruction.
3) To execute the instruction.

To carryout these tasks the MP requires considerable time, so this method of data transfer is not suitable for large data transfers such as data transfer from magnetic disk @) optical disk to memory. In such situations hardware controlled data transfer technique is used.

Ex:- Software controlled data transfer:

```
        MOV CX, count    ——> initialize count
        MOV DX, port-add ——> Load port address in DX
BACK:   MOV AL,[SI]      ——> get byte from memory
        OUT DX, AL       ——> send byte to o/p port
        INC DX           ——> Increment port address
        INC SI           ——> Increment memory pointer.
        LOOP BACK        ——> Repeat until CX=0.
        RET.
```

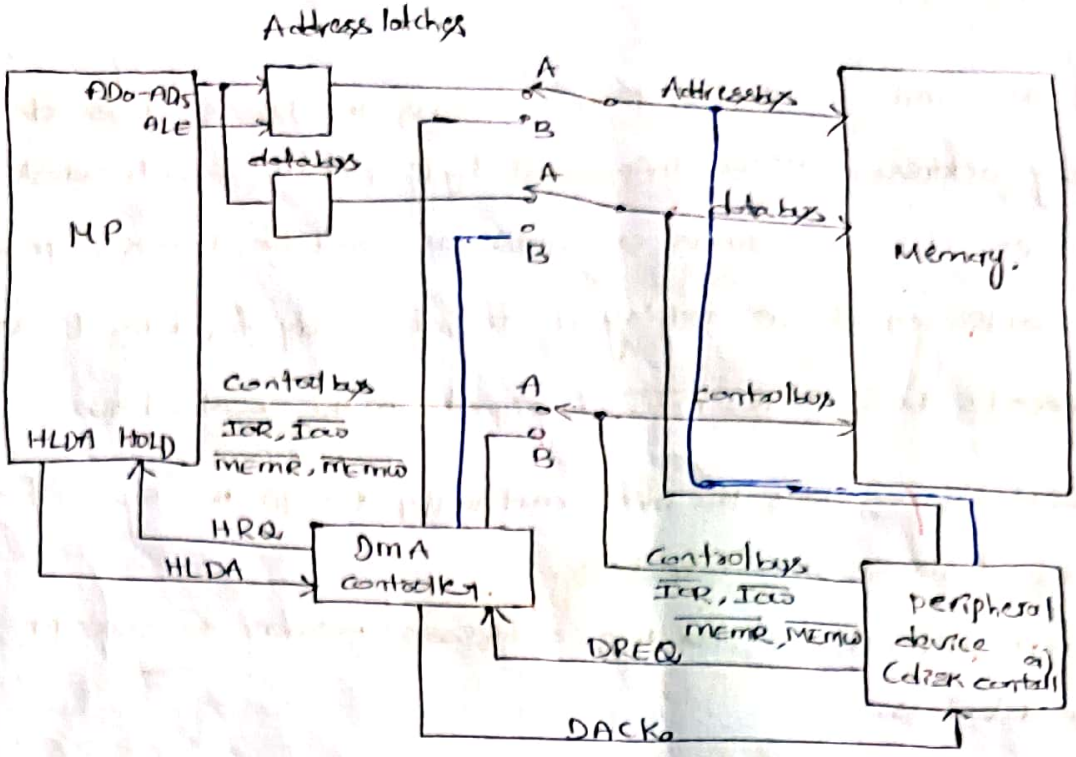
flow chart.



flow chart.

Hand wave controlled data transfer

External device is used to control data transfer. External device generates address & control signals required to control data transfer and allows peripheral device to directly access the memory. Hence this technique is referred to as Direct memory access (DMA) & External device which controls the data transfer is referred to as DMA controller.



DMA controller operating in a MP system.

DMA idle cycle:-

When the system is turned 'ON' the switches are at position 'A', the buses are connected from MP to the system memory & peripherals. To read a block of data from the disk MP sends a series of commands to the disk controller device telling it to search & read the desired block of data from the disk.

When the disk controller ready to transfer 1st byte of data, it sends DMA request DREQ signal to the DMA controller. Then DMA controller sends a hold request HRQ signal to the MP HOLD i/p. The MP responds to this HOLD signal & sends out HLDA (Hold acknowledge) signal to change switch position from A to B. This disconnects the MP from the buses & connects DMA controller to the bus.

DMA active cycle:-

Here the DMA controller gets control of the buses, it sends the memory address where the first byte of data from the disk is to be written. It also sends a DMA acknowledge, DACK signal to disk controller device telling it to get ready for data transfer, and it asserts both $\overline{IOR}$, $\overline{MEMW}$ signals on the control bus.

$\overline{IOR}$ → signal enables the disk controller to o/p the byte of data from the disk on the data bus.

$\overline{MEMW}$ → signal enables the addressed memory to accept data from the data bus.

In this techniques data is transferred directly from the disk controller to the memory location without passing through CPU ~~the DMA controller~~.

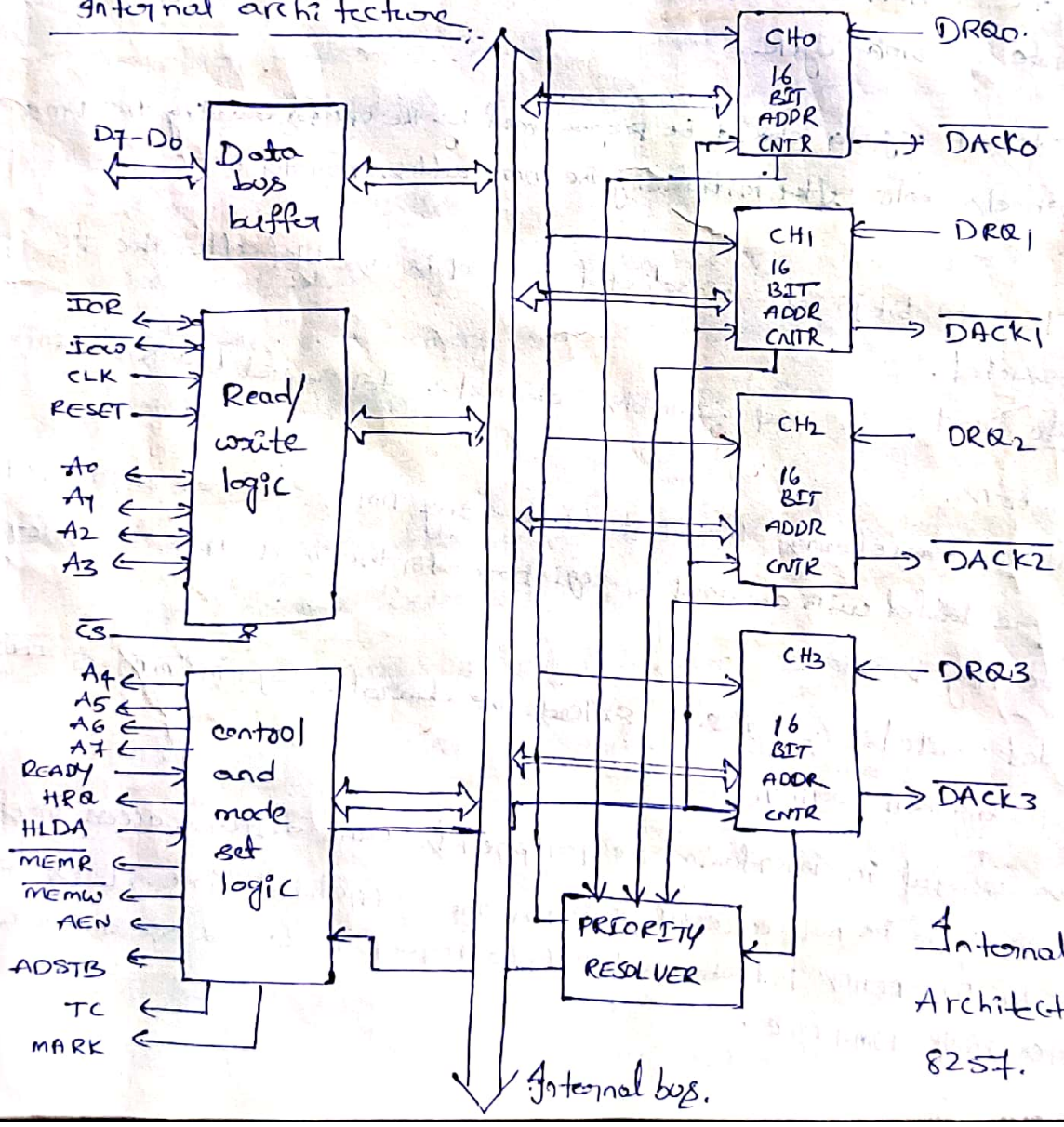
*

pg- 294, 295-297.

8257 DMA controller.

* A DMA controller is designed to complete the bulk data transfer task much faster than the CPU.
* The Direct memory access or DMA mode of data transfer is the fastest amongst all the modes of data transfer.
* In this mode the device may transfer data directly to/from memory without any interference from the CPU.
* The device requests the CPU, to hold the data, address & control bus, so that the device may transfer data directly to/from memory.
* 8257 is a four channel DMA controller designed to be interfaced with their family of MP's.

Internal architecture:



Internal Architecture of 8257.

The direct memory access & DMA.

TC :- The bits 14 & 15 of the TC register indicate the type of the DMA operation (Verify/W).

A15/RD	A14/WR	Type of DMA operation
0	0	verify DMA cycle
0	1	write DMA cycle
1	0	Read DMA cycle.
1	1	(Invalid)

Mode set Register :- The function of MR is to enable the DMA channels individually and also to set the various modes of operation.

A DMA channel should not be enabled till the DMA address register & the TC contain valid information, otherwise an unwanted DMA request may initiate a DMA cycle.

→ mode set register should be programmed by the CPU for enabling the DMA channels only after initializing the DMA address register & TC.

→ If TC Stop bit is set, the selected channel is disabled after the TC condition is reached.

→ Auto load If set, enables channel 2 for repeat block chaining operations.

→ After transferring the first block using DMA, channel 2 registers are re-loaded with channel 3 registers for the next block transfer, If update flag is set.

→ Extended write bit :- If set extends the duration of $\overline{MEMW}$ & $\overline{IOW}$ signals by activating them earlier.

This is useful in interfacing the peripherals with different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA cycle.

Block diagram of 8257 :- discription :-

Data bus buffer :-

* It is a bidirectional 8-bit buffer which interfaces the 8257 to the system data bus.
* In slave mode, it is used to transfer data b/w µp & internal registers of 8257.
* In master mode, it is used to send higher byte address (A8-A15) on the data bus.

Read/write logic :-

* when the CPU reading one of the internal registers of 8257 the Read/write logic accepts the I/O Read & I/p write signals, decodes the least significant four address bits (A0-A3) either writes the contents of the data bus into address register (a) place the contents of the address register on to the data bus.

* during DMA cycles the read/write logic generates the I/O read and memory write (or) I/o write and memory read signals, which control the data transfer b/w peripheral and memory device.

DMA channels :-

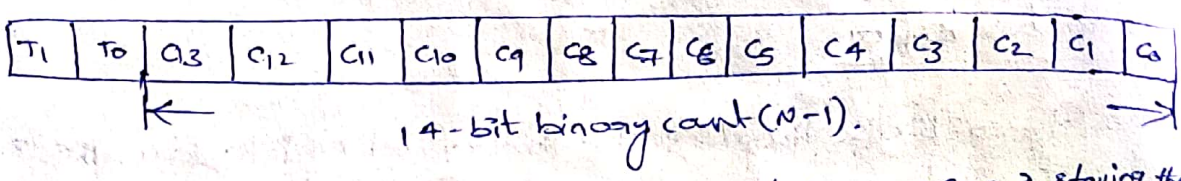
The 8257 provides four identical channels, labled CH0 to CH3. Each channel has two sixteen bit registers.
(i) DMA address register ii) terminal count register.

(i) DMA address register format :-

A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00

* DMA has 16-bit address register.
* It specifies the address of the first memory location to be accessed.
* It is necessary to load valid memory address in the DMA address register before channel is enabled.

Terminal count register.



T_1	T_0	Type of operation.
0	0	→ DMA verify cycle.
0	1	→ DMA write cycle
1	0	→ DMA Read cycle
1	1	→ Illegal.

Used for ceasing a stoping the data transfer through a DMA channel after the required no of DMA cycles.

* The 'TC' specifies the type of DMA operation to be performed.

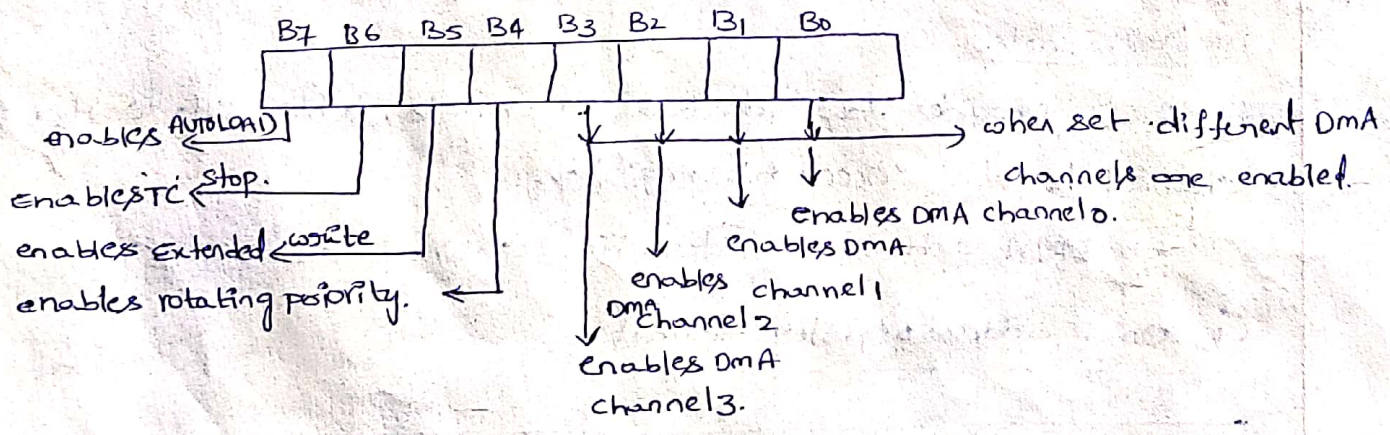
Control logic.

It controls the sequence of operations during all DMA cycles. (DMA read, DMA write, DMA verify) by generating the appropriate control signals.

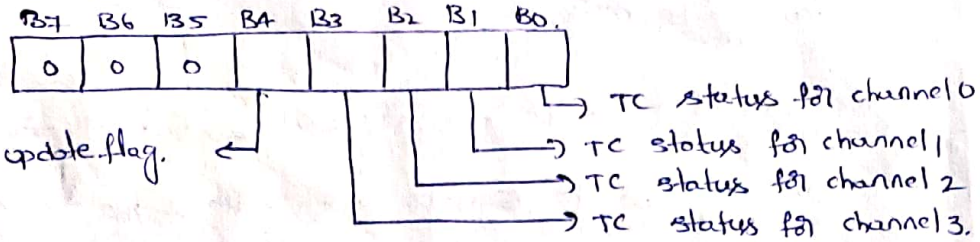
It consists of mode set register & status register.

* mode set register is programmed by the CPU to configure 8257
* Status register is read by CPU to check which channels have reached a terminal count condition.

Mode set register.



Status register:-



priority resolver:-

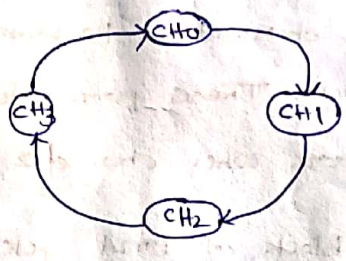
It resolves the peripherals requests. It can be programmed to work in two modes, either in fixed mode or rotating priority mode.

operating modes of 8257:-

The 8257 can be programmed to operate in following modes.

Rotating priority mode:-

In this, the priority of the channels has a circular sequence.



fixed priority mode:-

priorities for each channel are fixed.

	Priority	channel.
highest	1	0
	2	1
	3	2
Lowest.	4	3.

↓ Priority resolving table.

Extended write mode :-

The extended write option provides alternative timing for I/O and memory write signals which allows the devices to give an early READY and prevents the unnecessary occurrence of wait states in the 8257.

TC stop mode :-

If the TC stop bit is set, a channel is disabled after the TC o/p goes high, thus automatically preventing further DMA operation on that channel.

Auto load mode :-

In this mode, channel 2 parameters are initialized as usual for the first data block. These parameters are automatically duplicated in the channel 3 registers when channel 2 is initialized.

After the first block of DMA cycles is executed by channel 2, the parameters stored in the channel 3 registers are transferred to channel 2 during an 'update' cycle.

37

PIO 8255 [Programmable i/p-o/p port] PPI (1)

(2) PPI.

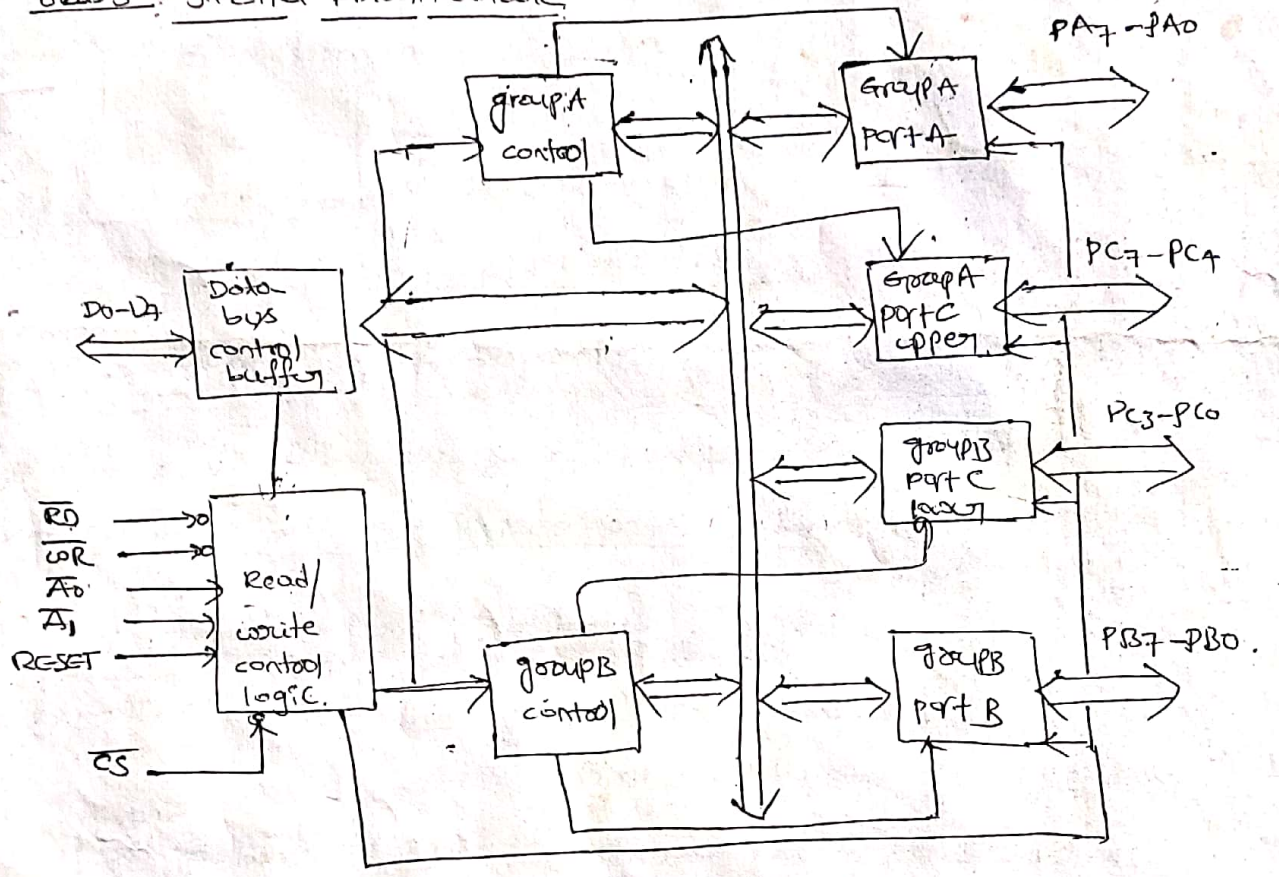
2019

The IInd i/p o/p port chip 8255 is also known as programmable peripheral i/p-o/p port.

* 8255 is designed by intel to the use with 8-bit, 16-bit and higher capability microprocessors.

*

8255: Internal Architecture



8.255 Internal Architecture.

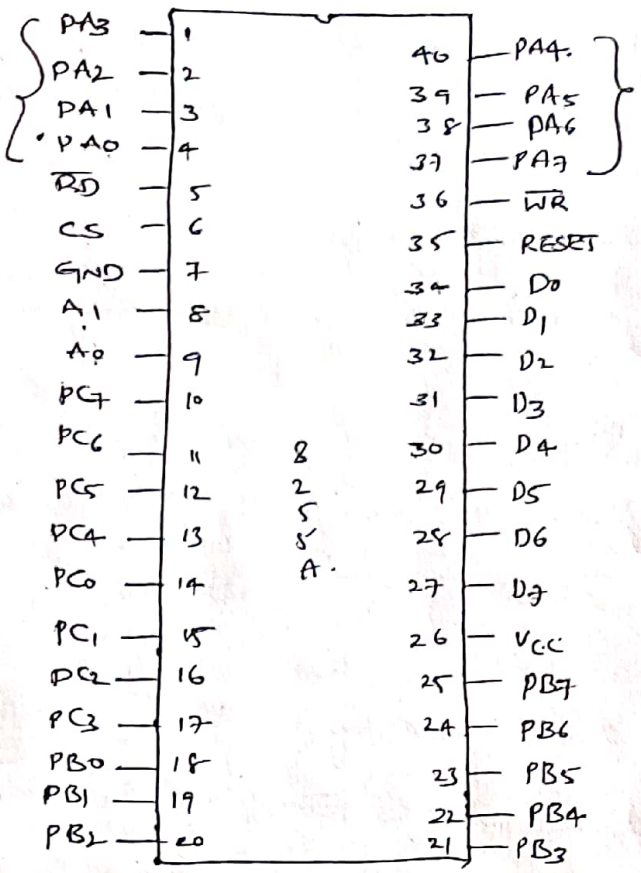
* It has 24 i/p & o/p lines.
* 24 i/p & o/p lines are individually programmed as two groups of 12 lines each & → two groups are named as group A & group B.
* 3 groups of eight lines each.
* A group contain a subgroup of eight I/O lines called as 8-bit port.

another sub group of four I/o lines & a 4-bit port.

* group A contains an 8-bit port A along with a 4-bit port C upper.
* port 'A' lines are identified by symbols PA0-PA7 while the port C lines are identified by symbols ~~PA0-PA~~ PC4-PC7.
* ~~III~~ group B contains an 8-bit port B, containing lines PB0-PB7 & a 4-bit port C with lower bits PC_0-PC_3.
* port 'c' upper & port 'c' lower can be used in combination as an 8-bit port C.
* Both port 'c' upper & port 'c' lower are assigned with the same address.
* So an 8255 PPI one may access as three 8-bit I/O ports or two 8-bit & two 4bit I/O ports.
* All these ports can function independently either as I/P or as O/P ports.
* Individual functioning of ports can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).
* 8-bit data bus buffer is controlled by the read/write control logic.
* The read/write control logic manages all of the internal & external transfers of both data & control words.
* $\overline{RD}$, $\overline{WR}$, A_0, A_1, RESET. are the I/P pins provided by the µp to the READ/WRITE control logic of 8255.
* 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.
* This buffer receives or transmits data upon the execution of I/P or O/P instructions by the micro processor.

The control words or status information is also transferred through the buffer

Pin configuration



PA7 - PA0 :- These are eight port A lines acts as either

latched o/p or
buffered i/p lines } depending on the CWR loaded the CWR Register.

PC7-PC4 :- upper nibble of port C lines. acts as either o/p latches or i/p buffer lines.

* also use for generation of handshake lines in mode 1 or mode 2.

PC3-PC0 :- lower port c lines —

* acts as either o/p latch or i/p buffer lines.
* also use for generation of handshake lines in mode 1 or mode 2.

PB0 - PB7 :- These are the eight port B lines which are used as o/p lines or buffered i/p lines as same as PORT A.

$\overline{RD}$:- i/p line driven by the MP.

* low state indicates the read operation to 8255.

$\overline{WR}$:- i/p line driven by the MP.

* low state indicates the write operation to 8255.

$\overline{CS}$:- chip select line.

$\overline{CS} = 0$ this state enables the 8255 to respond to $\overline{RD}$ & $\overline{WR}$ signals, otherwise $\overline{RD}$ & $\overline{WR}$ signals are neglected.

$A_1 - A_0$:- * Address i/p lines driven by the MP.

* $A_1 - A_0$ with $\overline{RD}$, $\overline{WR}$, $\overline{CS}$ forms the following operations for 8255. These address lines are used for addressing any one of the four registers.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	i/p (Read) cycle.
0	1	0	0	0	port A to data bus.
0	1	0	0	1	port B to data bus
0	1	0	1	0	port C to data bus.
0	1	0	1	1	CWR to data bus.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	o/p (write) cycle
1	0	0	0	0	data bus to port A
1	0	0	0	1	data bus to port B
1	0	0	1	0	data bus to port C
1	0	0	1	1	data bus to CWR.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	function.
x	x	1	x	x	data bus tristated.
1	1	0	x	x	data bus tristated.

$D_0 - D_7$: Data bus lines those carry data & control word to/from the MP.

RESET :- * logic high on this line clears the CWR of 8255.

* by default after reset all ports are set as i/p ports.

Modes of operation of 8255 PPI :-

Two basic modes of operation are → I/O operation
→ BSR (Bit set reset) mode of operation

I/O mode :- * In this mode, the 8255 ports work as programmable I/O ports.

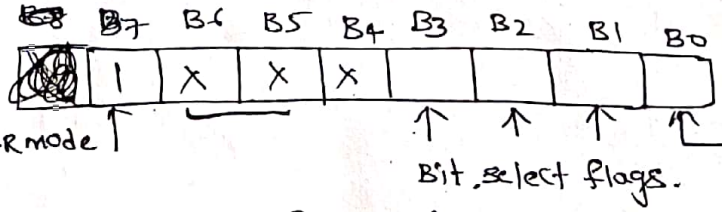
* I/O mode again can be carried out as mode 0, mode1, mode2 operations.

BSR mode :- * In this mode only port C (PC0 - PC7) can be used to set & reset its individual port bits.

* Any of the 8-bits of port C can be set & reset depending on B0 of the control word.

* The bit to be set & Reset is selected by bit select flag B3, B2 and B1 of the CWR.

The CWR format

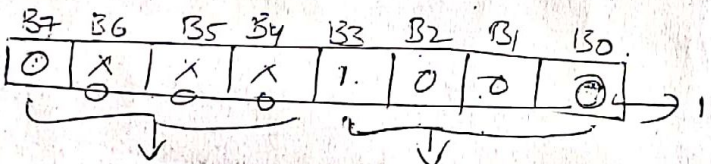


B3, B2, B1 are from 000 to 111 for bits PC0 to PC7.

B_3	B_2	B_1	selected bits of port C.
0	0	0	B_0 PC_0
0	0	1	B_1 PC_1
0	1	0	B_2 PC_2
0	1	1	B_3 PC_3
1	0	0	B_4 PC_4
1	0	1	B_5 PC_5
1	1	0	B_6 PC_6
1	1	1	B_7 PC_7

I/O modes:

Mode 0 : (Basic I/O mode).



CWR = 09h → Set

CWR = 0F → Reset.

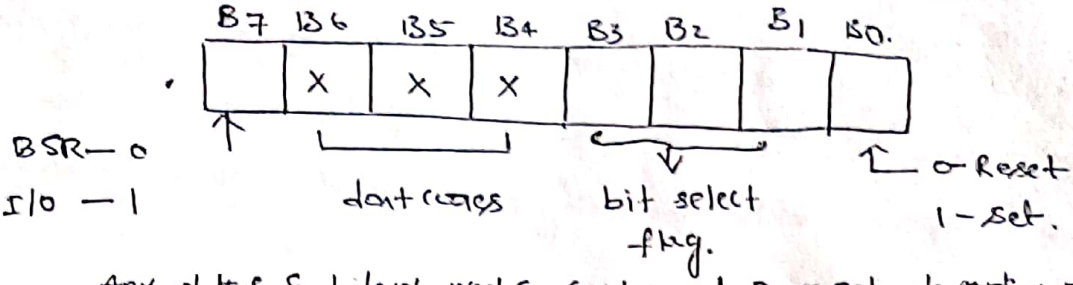
what is the function of handshaking signal in 8086 ?

Ans: I/O devices accept & release information at much slower rate than the µP.

Handshaking is the method that synchronize the i/o devices with the µP.

Control word formats:

In BSR mode of operation



CWR

(3) * Any of the 8-bits of port C can be set a reset depending on B0 of the CWR

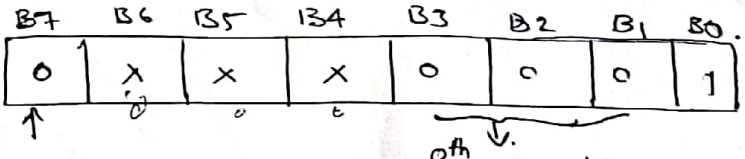
(2) * The bit to be ~~select~~ a Reset is selected by bit select flags B_3, B_2, B_1 of the CWR.

(1) * In this mode only port C can be used to set a reset its individual port bits. (PC_0-PC_7).

* The combination of B_3 B_2 B_1 selects the bits of port 'C' as follows:

B_3	B_2	B_1	selected bits of port C
0	0	0	B_0
0	0	1	B_1
0	1	0	B_2
0	1	1	B_3
1	0	0	B_4
1	0	1	B_5
1	1	0	B_6
1	1	1	B_7.

EX:-1) write the value of control word for setting (B_0 / PC_0) of port C.



CWR = 01h for setting 0th bit

CWR = 00h for resetting 0th bit bi

Interfacing :-

The working configuration of a general µp can be carried out by a key board, display system, memory system & I/O ports along with cpu. The µp may be seen as the heart of the system while all the peripheral ckts including memory system are built around the µp.

Most of the peripheral devices are designed and interfaced with a cpu either to enable it to communicate with the user or an external process & to ease the ckt operation so that the µp works more efficiently & effectively.

Ex2:- write value of CWR for setting the fourth bit of port 'C'.

B7	B6	B5	B4	B3	B2	B1	B0
0	X	X	X	1	0	1	1

⇒ 09h.

CWR = 09h.

4th bit of port C is selected.

for resetting the 4th bit the value is = 08h.

Ex-3:- write the value of CWR for setting the 6th bit of port 'C' & for Resetting that bit what is the value of CWR.

B7	B6	B5	B4	B3	B2	B1	B0
0	X	X	X	1	1	0	1

0Dh

6th bit is setted.

6th bit of port C is selected

Resetted value = 1100 ⇒ 0Ch.

(2). ~~using~~. using the ALP of program (1) design a delay of 10 minutes

(5)

Sol:- f = 10MHz

T = 0.1 μsec.

In BSR mode.

port c bits:	control word to set	control word to reset
B_0	01 h	00 h
B_1	03	02 h
B_2	05	04
B_3	07	06
B_4	09	08
B_5	0B	0A.
B_6	0D	0C.
B_7	0F	0E.

→ PC → 8.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0.	
0	x	x	x	-	-	-	-	→ 1 – Set, 0 – Reset.
	0	0	0					

0 0 0 → PC_0

0 0 1 → PC_1

0 1 0 → PC_2

0 1 1 → PC_3 → CWR to set 07. to Reset 06.

1 0 0 → PC_4

1 0 1 → PC_5

1 1 0 → PC_6

1 1 1 → PC_7.

~~Stack & Stack structure~~

→ states for execution.

MOV CX, count → 4

DEC CX → 2

NOP → 3

JNZ label → 16

RET → 8

Ex:- write a program to generate a delay of 100 ms using an 8086 system that runs on 10 MHz f.

Sol:- $T_d = 100 msec$.

No-of clock cycles required for execution of loop once.= 2 + 3 + 16 = 21
dec nop

mov cx, count is not in the delay generation loop.

Time required for execution of the loop once = $nT = 21 \times \frac{1}{10 MHz} = 21 \times 0.1 \mu s$
$= 2.1 \mu sec.$

Sol:- MOV CX, BA03h. load count register.

WAIT : DEC CX
dec

NOP
wait till

JNZ WAIT
count register.

$$\text{count } N = \frac{\text{Required delay } (T_d)}{n * T}$$

$$\text{Required count} = \frac{T_d}{n*T} = \frac{100 \times 10^{-3}}{2.1 \times 10^{-6}} = 47.619 * 10^3.$$

$$= 47619 = BA03H.$$

I/O modes of 8255 PPI

Mode 0: Simple i/p & o/p mode.

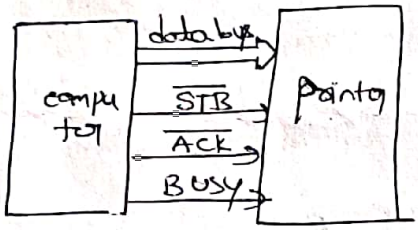
In this mode

* Port A & port B are used as two simple 8-bit I/O ports
* Port C as two 4-bit ports.
* Each port can be programmed to function as simply an i/p port & an o/p port.

The i/p/o/p features of in mode 0 are as follows

1) o/p's are latched
2) i/p are buffered, not latched.
3) ports do not have handshake & interrupt capability.

Mode-1: I/P/O/P with handshake.



* In this mode i/p & o/p data transfer is controlled by handshaking signals
* handshaking signals are used to transfer data b/w devices whose data transfer speeds are not same.

Eg:-

* handshaking signals are used to tell computer whether printer is ready to accept the data or not. If printer is ready to accept the data then after sending data on data bus, computer uses another handshaking signal ($\overline{STB}$) to tell printer that valid data is available on the data bus.

8255 modes → BSR (bit set Reset)
→ I/O → mode 0
→ mode 1
→ mode 2.

mode 0 features: also called as simple I/O mode
1) we can program port A & B as I/O ports
2) port C (upper) & lower 4 bits also programmed
3) Handshaking feature is not implemented

mode 1: port A & port B can be programmed as I/O mode
port C upper & lower 4 bits also programmed
Handshaking feature is implemented

mode 2: also called as Bidirectional I/O mode
only port 'A' can be programmed as I/O.
port C bits are used for the generation of handshaking signals.

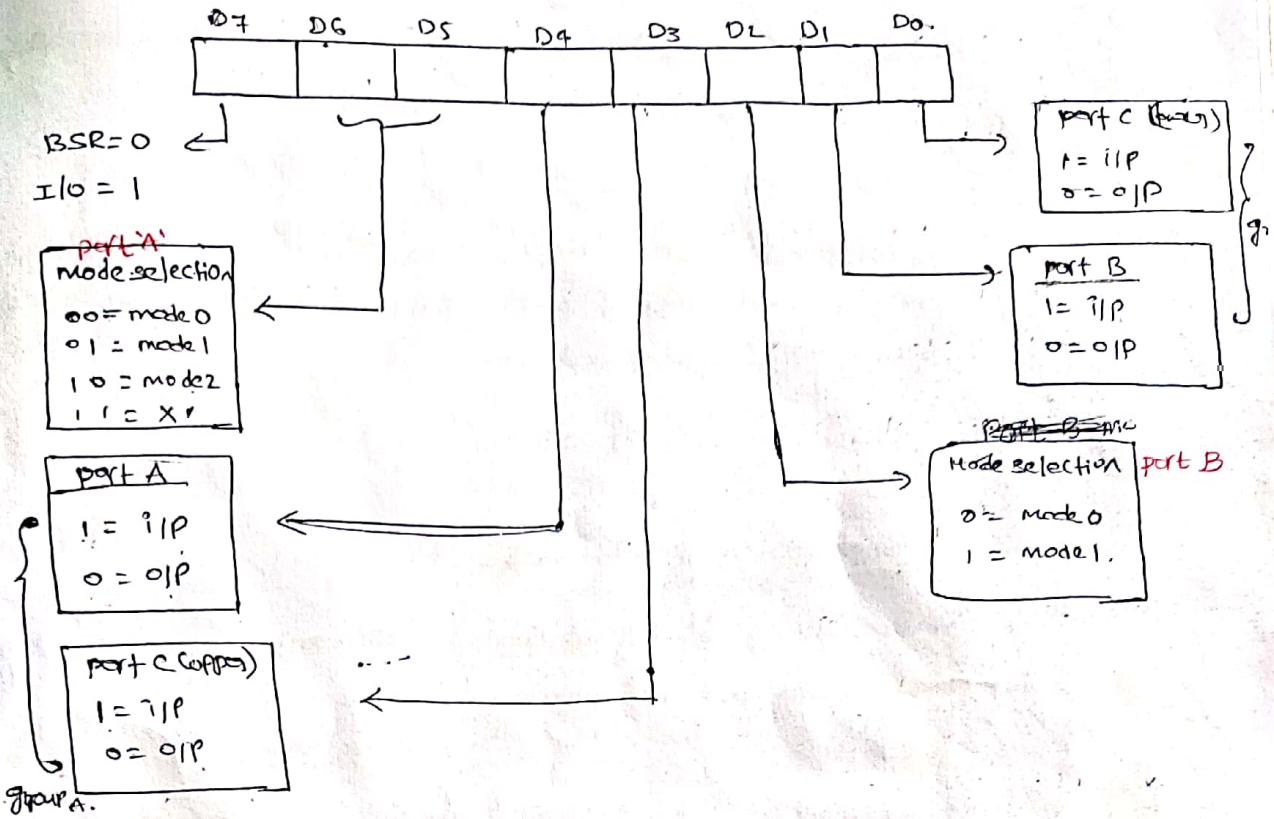
CWR: 8bit reg. there by programming the individual fields we can assign the req. function on 8255 PPI by the selection of required bits.

simple means: mode '0'

31.

Control word format in I/O mode.

(7)



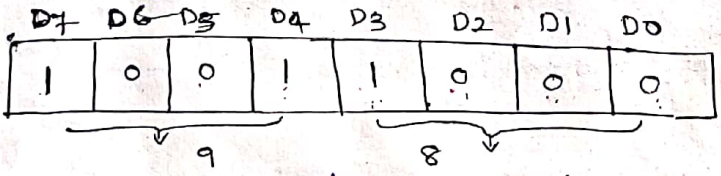
Ex: write a program to initialize 8255 in the configuration given below

AH AL
AX (04 | 00)
0400

1. port A : Simple i/p → mode '0'
2. port B : simple o/p
3. port C_L : o/p
4. port C_U : i/p.

Assume address of the control word register of 8255 is 83H.

Sol:-



Simple i/p & o/p port operation is considered in mode '0'.

MOV AL, 98 h → Load control word

OUT 83h, AL → send control word to 83h.

Read IN
write OUT

Mode 1 features:-

* Two ports port A & port B function as 8-bit I/O ports. they can be configured either as i/p or o/p port.
* Each port uses three lines from port C as handshake signals. the remaining two lines of port C can be used for simple I/O functions.
* I/P and O/P data are latched.
* Interrupt logic is supported.

Mode-2 :- (Bi-directional I/O data transfer).

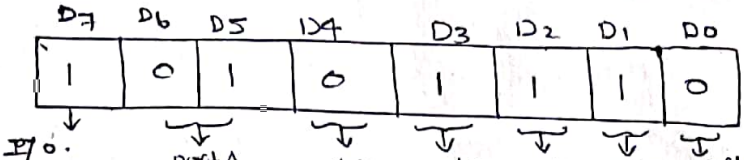
* This mode allows bi-directional data transfer over a single 8-bit data bus using handshaking signals.
* This feature is available only in group A with port A as the 8-bit bi-directional data bus. PC3- PC7 are used for handshaking purpose.
* both i/p & o/p are latched.
* The remaining lines of port C PC0-PC2 can be used for simple I/O functions.
* Port B can be programmed in mode 0 & in mode 1.
* when port B is programmed in mode 1, PC0- PC2 lines of port C are used as handshaking signals.

(2) write a program to initialize 8255 in the configuration given below. (3)

1. port A : o/p with hand shake.
2. port B : i/p with handshake.
3. port CL : o/p
4. port CU : i/p.

Assume address of control word register of 8255 is 23h.

Sol:



⇒ CWR = AE h

hand shake operation carried out by initializing port in mode

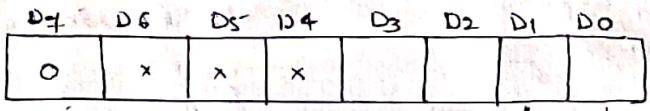
```
MOV  AL, 0AEh  → load control word
OUT  23h, AL.  → send control word.
```

Ex3 blink port 'C' bit of 8255.

write a program for the above requirement. assume the address of control word register is 83h .. use BSR mode.

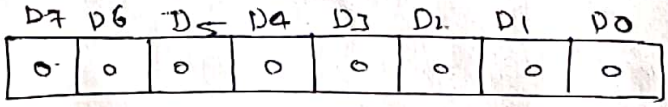
Sol:

SET ⇒



0 0 0 0 0 0 0 1. ⇒ CWR = 01h

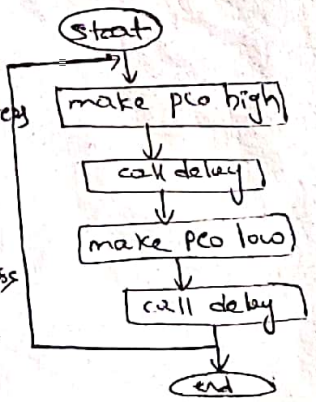
RESET ⇒



⇒ CWR = 00h.

program

```
Back: MOV  AL, 01h  → load CWR to AL
      OUT  83h, AL  → load CWR value to 83h address
      CALL delay    → call delay subroutine
      mov  AL, 00h  → load CWR to AL
      OUT  83h, AL  → CWR value to 83 address
      CALL delay    → call delay subroutine
      JMP  back     → Repeat.
```



9 0 1 0 10

delay delay

IN :- i/p a byte or word from port.

IN → instruction will copy data from a port to the accumulator.

8-bit port —data→ AL

16-bit port —data→ AX.

Ex:- IN AL, 0F8h : copy a byte from port 0F8h to AL.

IN AX, 93h : copy a word from port 93h to AX.

OUT:-

MOV DX, 30F8h. → Load 16-bit address of port in DX.

IN AL, DX → copy a byte from 8-bit port 30F8h to AL

IN AX, DX → copy a word from 16-bit port 30F8h to AX.

OUT:- send a byte or word to a port.

The out instruction copies a byte from AL or a word from AX to the specified port.

Ex:- OUT 0F8h, AL → copy contents of AL to 8 bit port 0F8h.

OUT 0F8h, AX → copy contents of AX to 8 bit port 0F8h.

(or) MOV DX, 30F8h → load 16-bit address of the port in DX

OUT DX, AL → copy the contents of AL to port 30F8h

OUT DX, AX → copy the contents of AX to port 30F8h.

IN → read data from port

OUT → write data to port.

Interface an 8255 with 8086 to work as an I/O port.

Initialise port A as o/p port.
port B as i/p port.
port C as o/p port.

port A address should be 0740h.

Write a program to sense switch positions SW0-SW7 connected at port B the sensed pattern is to be displayed on port A; to which 8 LEDs are connected, while the port C lower displays number of on switches out of the total eight switches.

Sol:- CWR =

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

= 82h.

I/O — port A mode — port A o/p — port C o/p — port B mode — port B i/p — port C o/p.

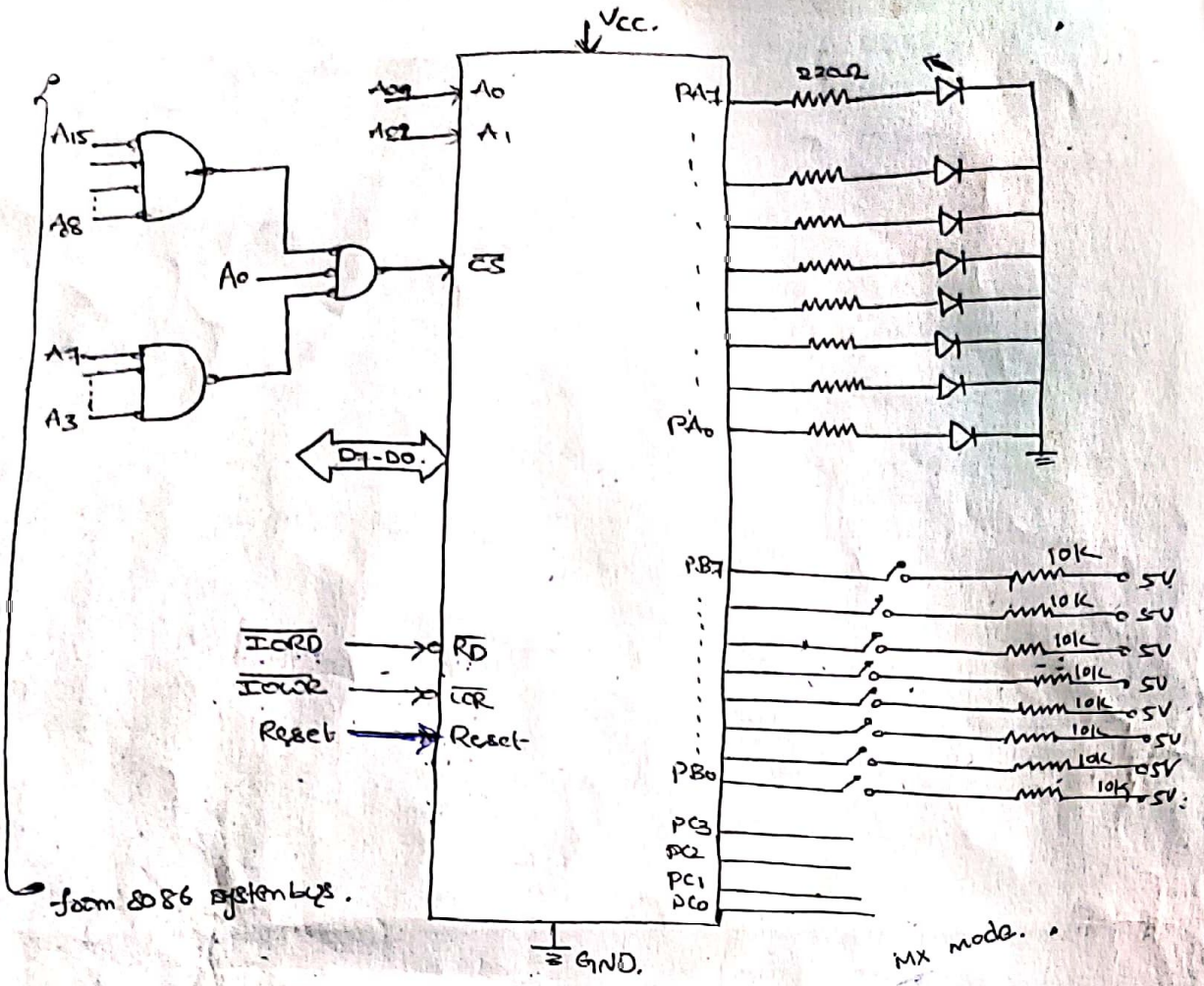
Address of port A = 0740h.

port address calculation of 8255 PPI.

8255 A0 A1 — A1 A2 — 8086 processor C

0 0 → A
0 1 → B
1 0 → C
1 1 → CWR

8255 ports	I/O lines. A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A0
port A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0
port B	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1
port C	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0
CWR	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0



MX mode.

Program:-

```
        MOV   DX, 0746h   ;  initialise CWR with
        MOV   AL, 82h     ;  control word 82h.
        OUT   DX, AL      ;  write
        SUB   DX, 04      ;  0742  get address of port B in DX.
        IN    AL, DX      ;  0740  Read port B for switch.
        SUB   DX, 02      ;  Positions in to AL and get port A address in DX.
        OUT   DX, AL      ;  Display switch positions on port A.
        MOV   BL, 00h     ;  initialise BL for switch count.
        MOV   CH, 08h     ;  initialise CH for total switch number.
YY :    RCL   AL          ;  Rotate AL through carry to check.
        JNC   XX          ;  wether the switches are on &
        INC   BL          ;  off, i.e either 1 & 0
XX :    DEC   CH          ;  07 00  check for next switch. If
        JNZ   YY          ;  all switches are checked, the
        MOV   AL, BL      ;  0744  number of on switches are
        ADD   DX, 04      ;  in BL. Display it on port C
        OUT   DX, AL      ;  lower.
        HLT               ;  Stop.
```

* Initialise port A as i/p port in mode 0.

Sol:-

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1	0	0	0	0
I/O	Port A mode		Port A I/P	Port C I/O	Port B mode	Port B I/O	Port C I/O.

CCOR = 90h.

Program:-

```
MOV SI, 1500h.
MOV AL, 90h.   → Load control word in to AL.
OUT C6, AL     → write AL content to port C6
IN  AL, C6     → Read data from port C6 to AL.
MOV [SI], AL   → Load the AL content in to address pointed by SI [1500h].
INT 3.
```

MOV AL, 90h
OUT C6, AL.
IN AL, C6.
MOV [SI], AL.

Ex:2 * Initialise port A as i/p port & port B as o/p port in mode '0'.

D7	D6	D5	D4	D3	D2	D1	D0.
1	0	0	1	0	0	0	0

CCOR = 90h.

program:-

```
MOV AL, 90h   → initialise port A as i/p port.
OUT C6, AL.   → write data to port C6
IN  AL, C6    → Read data from port.
OUT C2, AL    → write data to port C2.
INT 3.
```

Ex:3 * Initialise port C as o/p port in mode '0'.

31.

```
MOV AL, 90
OUT C6, AL
MOV AL, 80
OUT C4, AL
INT 3.
```

Interface 16-bit 8255 ports with 8086. The address of port A is F0h.

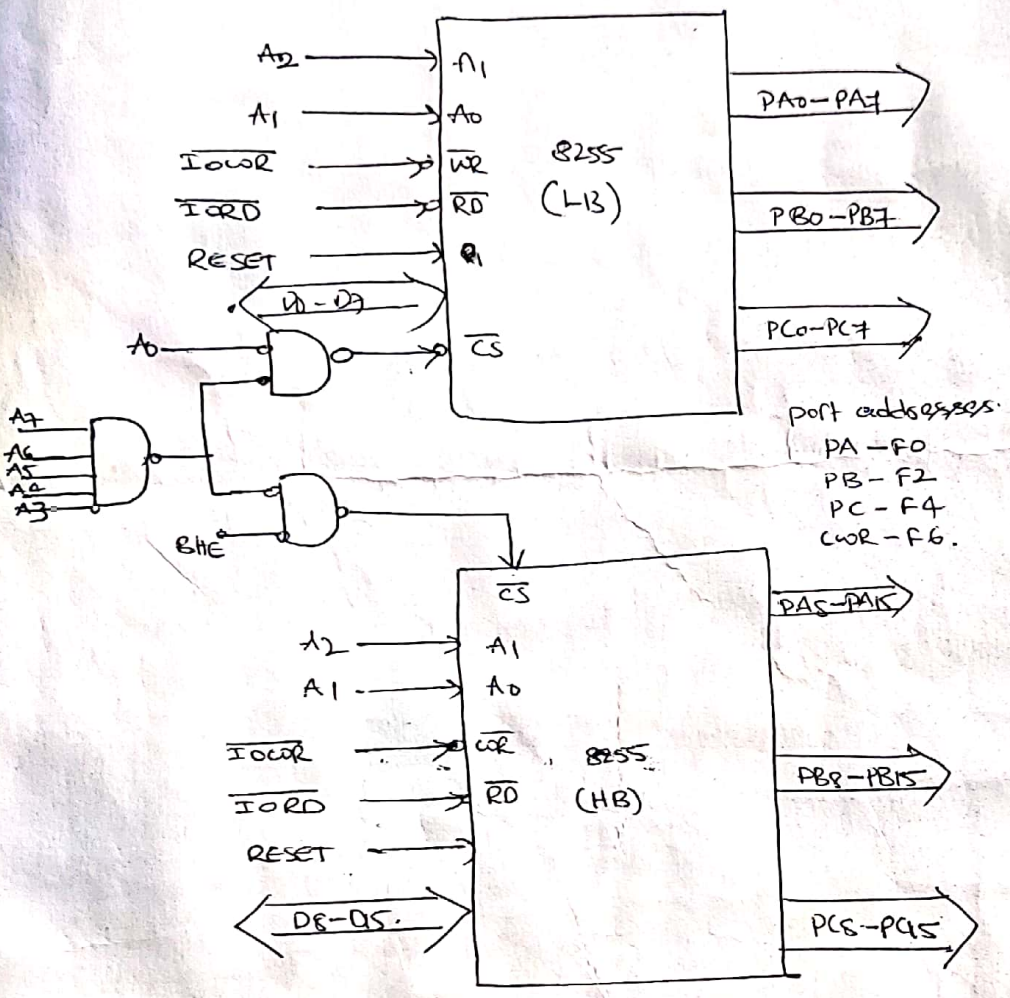
Sol:- To implement 16-bit port two 8255 are required.

one will act as lower 8-bit port D0-D7.

upper 8-bit port D8-D15.

while initialising AL & AH both should be loaded with a suitable control

In this system, port A, port B & port C all may work as 8-bit ports



Interfacing 16-bit 8255 ports with 8086.

→ 16-bit devices are interfaced directly with the 16-bit data bus using A0 and $\overline{BHE}$ pins of 8086.

→ 8 bit I/P or O/P devices are interfaced with lower order data bus of

Interface ADC 0808 with 8086 using 8255 ports. use port A of 8255 for transferring digital data o/p of ADC to the CPU and port-C for control signals. Assume that an analog i/p is passed at I/P$_2$ of the ADC and a clock i/p of suitable f is available for ADC. Draw the schematic and write required ALP.

Sol:

CWR = 98h. => A → i/p
C$_U$ = i/p
C$_L$ = o/p.

Analog i/p I P$_2$

Address Pins
ABC
0 1 0 → to select IP$_2$

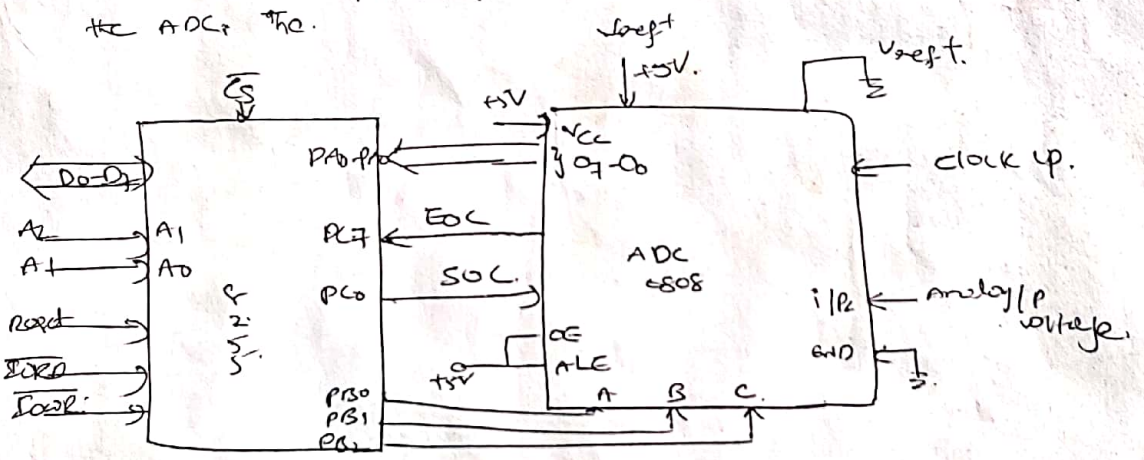
ALE, OE → get connected with +5V. to select ADC & enable the o/p's.

C$_U$ = acts as an i/p port to receive EOC signal.

C$_L$ = o/p port to send SOC to the ADC.

port A acts as a 8-bit i/p data port to receive the digital data o/p from the ADC.

port B → o/p port to write the address




```
MOV AL, 98h  } initialise 8255 CWR
OUT CWR, AL  }
MOV AL, 01.  } select I/P2 of analog i/p.
OUT port B, AL
MOV AL, 00h  } give start of conversion.
OUT port C, AL  } pulse to the ADC.
```

```
OUT port C, AL
MOV AL, 00h
OUT port C, AL.
Wait: IN AL, port C  } check for EOC
RCL                  } reading port C
JNC Wait.            } and rotating
IN AL, port A        } & carry,
HLT.                 } if EOC, read
```

Interface 4*4 keyboard with 8086 using 8255. (5½) (15) Write an ALP for detecting a key closure and return the key code in AL. The debouncing period for a key is 10ms. use software key debouncing technique. DEBOUNCE is an available 10ms delay routine.

Sol:

4 * 4 ⇒ 4 rows & 4 columns.

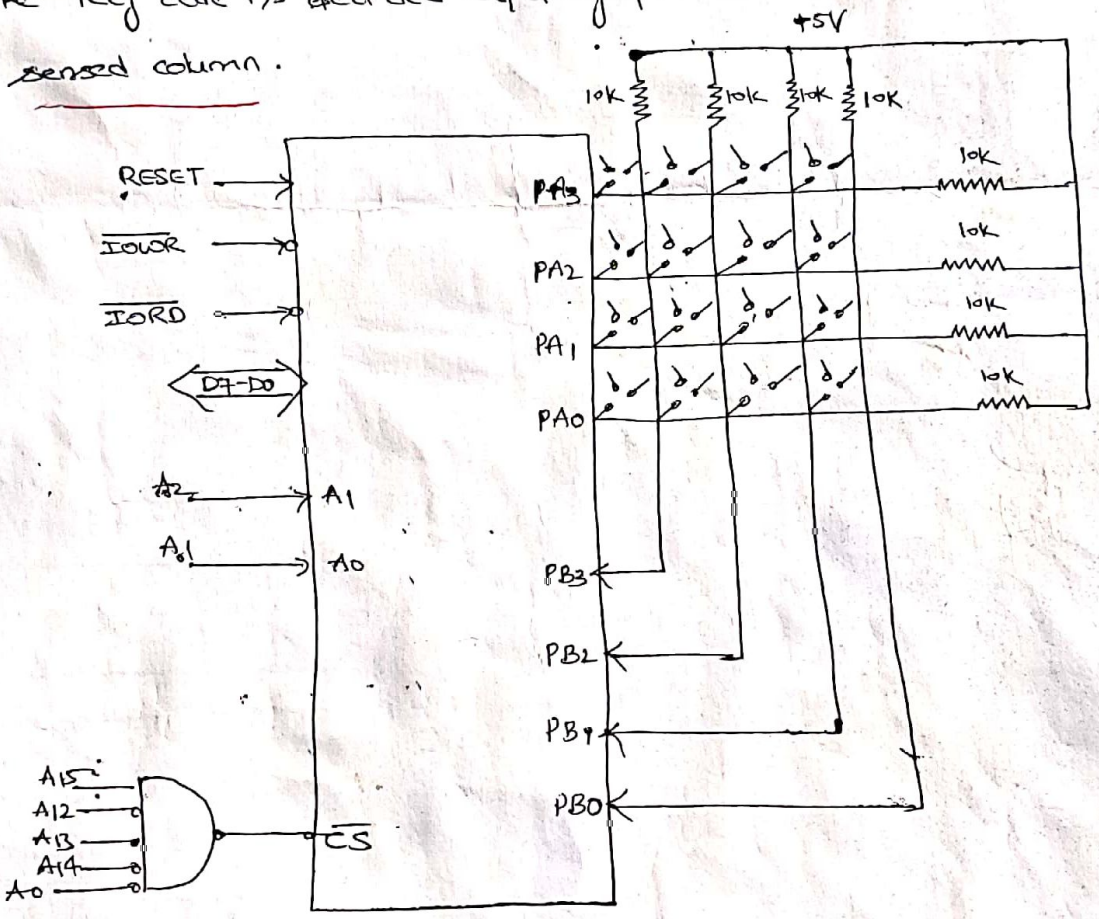
port A used as o/p port for selecting a row of key.

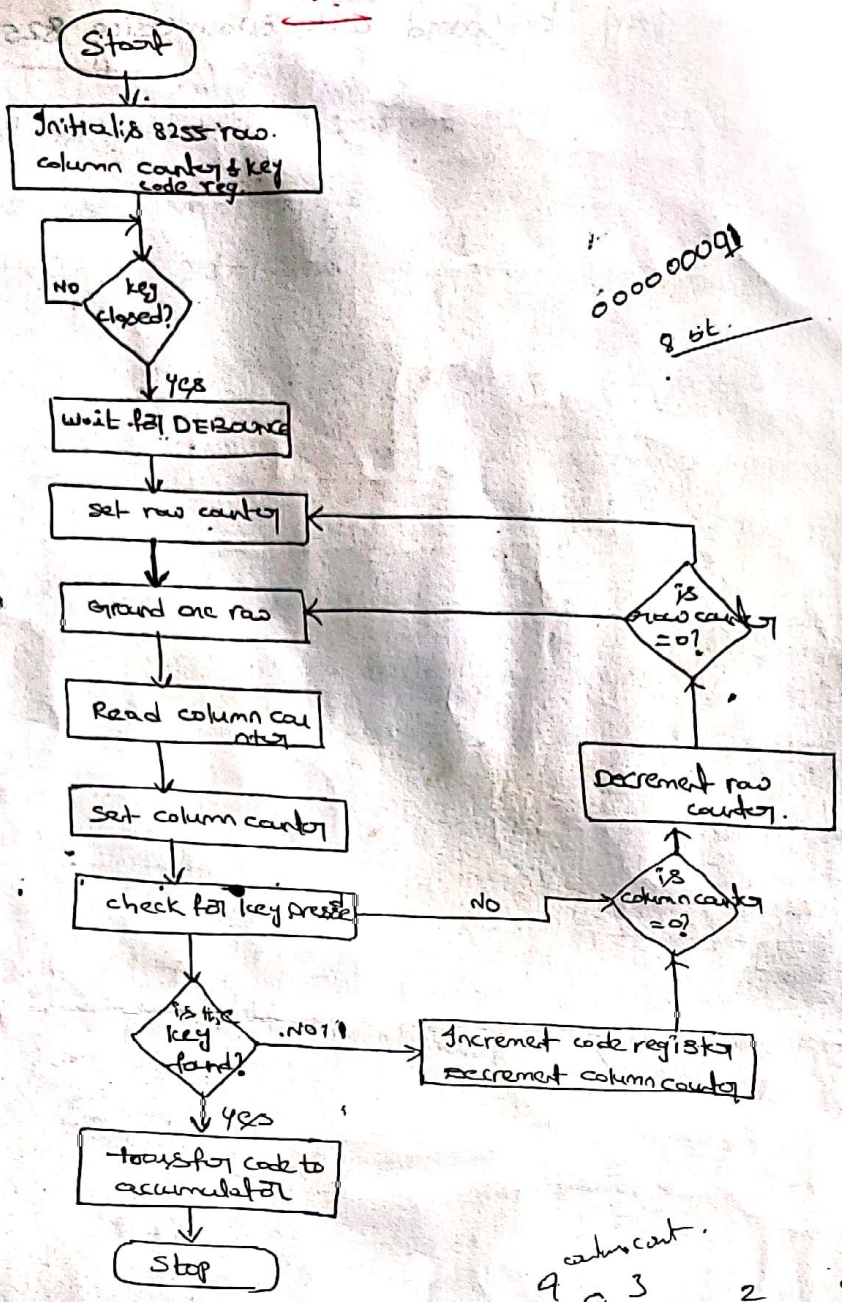
port B " " i/p " " sensing a closed key.

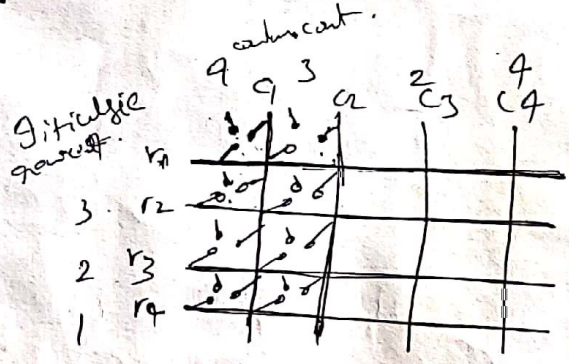
∴ key board lines are selected one by one through port A

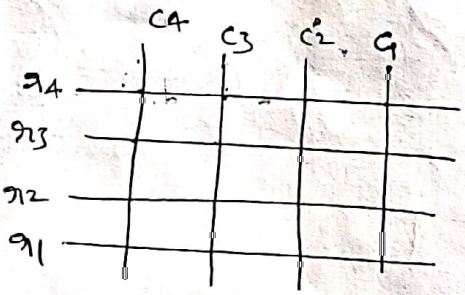
port B lines are polled continuously, till a key closure is sensed, then routine DEBOUNCE is called for key debouncing.

The key code is decided depending upon the selected row and a low sensed column.









ADC (Analog to digital converter interfacing).

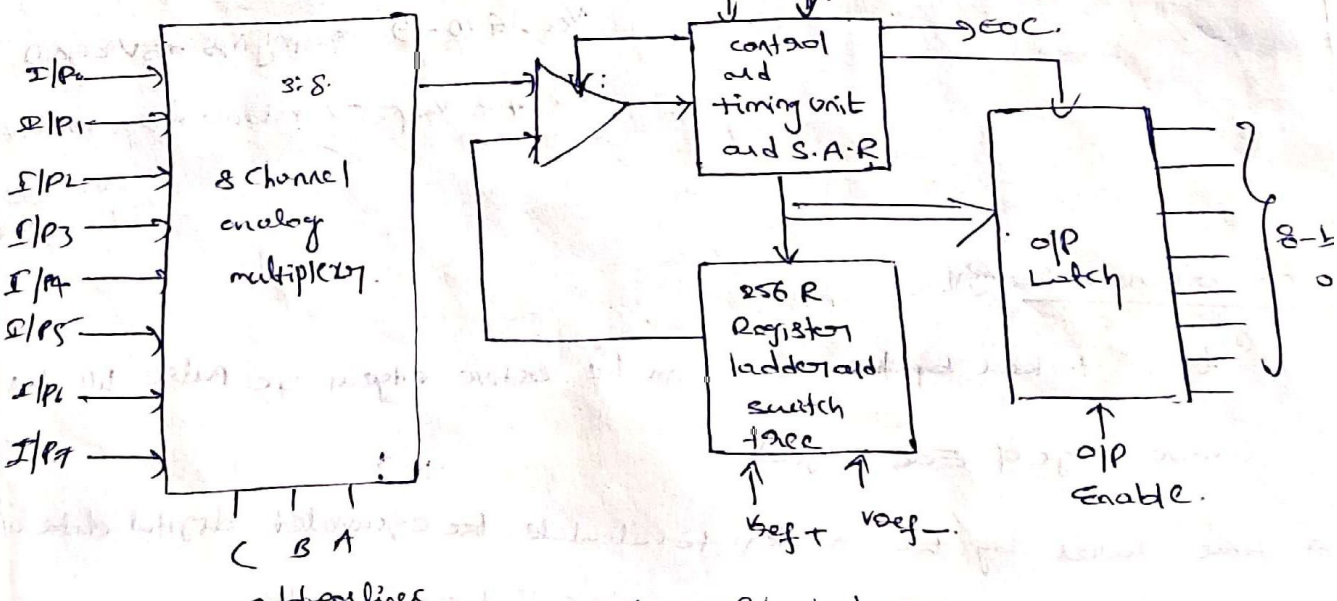
(1) DAC & ADC Interfacing.

General Algorithm to ADC interfacing.

1) Ensure analog i/p signal, applied to the ADC.
2) Issue SOC pulse to ADC.
3) Read EOC signal to mark the end of conversion process.
4) Read digital data o/p of the ADC or equivalent digital o/p.

ADC 0808/0809. features:

* 0808 ~~0809~~ ADC is an 8-bit.
* ADC uses CMOS, successive approximation converters.
* conversion delay is 100μs.
* clock f = 640 kHz.
* the converter internally has 3:8 along multiplexers so at a time eight different analog i/p's can be connected to the chips.
* out of these eight i/ps only one can be selected for conversion by using address lines ADD A, ADD B & ADD C.
* using these address i/p, multichannel data acquisition systems can be designed using a single ADC.
* The CPU may drive these lines using o/p port lines in case of multichannel applications. SOC CLOCK



ADC Block diagram.

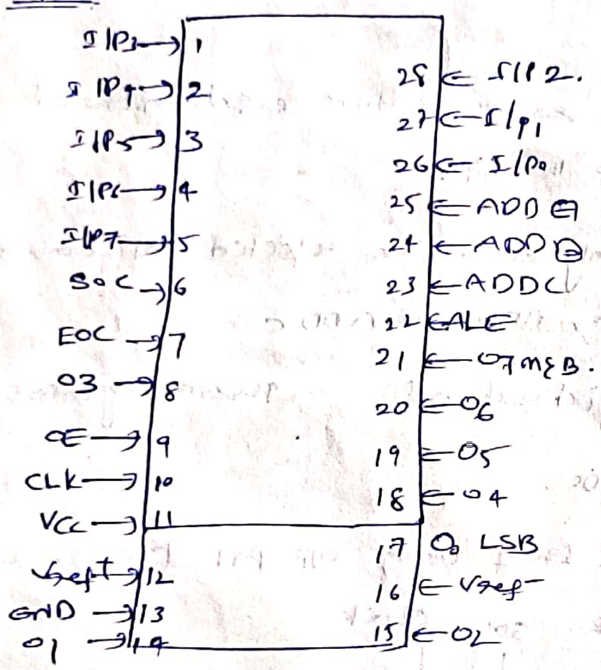
Analog i/p selected	C	B	A
I/P0	0	0	0
I/P1	0	0	1
I/P2	0	1	0
I/P3	0	1	1
I/P4	1	0	0
I/P5	1	0	1
I/P6	1	1	0
I/P7	1	1	1
~~P7~~	1	1	1

(Address lines: C, B, A)

~~ADC → 8~~

ADC 8 bit 0808/0809
12 bit 7109
dual slope.

Pin:



I/P0 – I/P7 → Analog i/p s.

ADD A, B, C → address lines for selecting analog i/ps

O7 – O0 → Digital 8-bit o/p O7 MSB & O0 LSB.

SOC → start of conversion signal pin

EOC → end of conversion signal pin.

OE → o/p latch enable pin.

OE = 1 enable o/ps

= 0 disable o/p.

CLK → clock i/p for ADC.

Vcc, GND → supply pins +5V & GND.

Vref+ & Vref− → reference voltage +ve (+5V max)
−ve (0V min)

conversion delay (T)

* time taken by the ADC from the active edge of soc pulse till the active edge of EOC signal.

* time taken by the converter to calculate the equivalent digital data o/p from the moment of the start of conversion is called conversion delay.

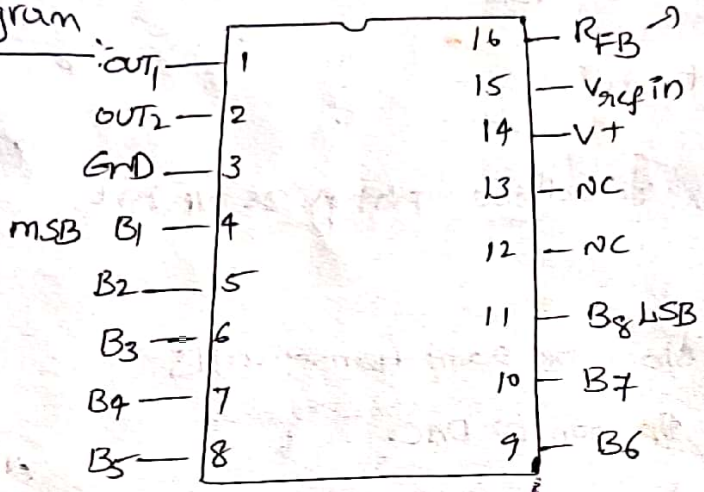
AD 7523 - 8 bit multiplying DAC :-

DAC → convert binary nos into their analog equivalent voltages
DAC → Applications: Digitally controlled gains, motor speed controls, programmable gain Amplifiers etc

Features:-

* 16 pin DIP package.
* contains R-2R ladder (R = 10KΩ) for digital to analog conversion
* along with Single pole double throw Nmos switches are used to to connect the digital i/p's to the ladder.
* supply range from (+5V to +15 V.)
* Vref range = −10V to +10V.
*

Pin diagram :-

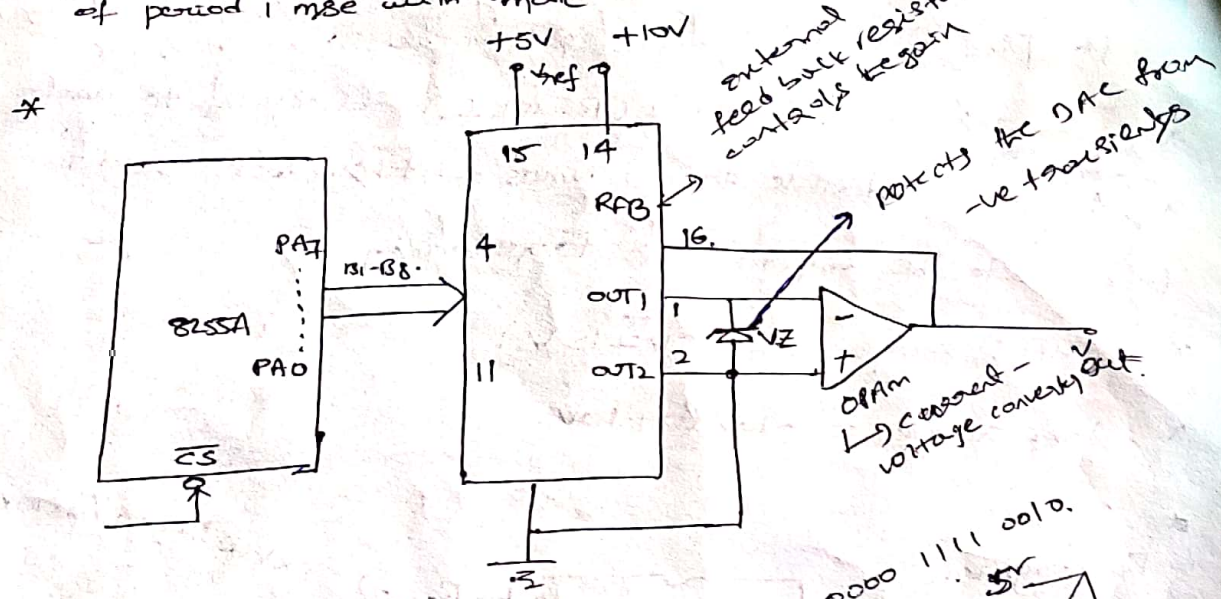


* Zener diode is connected b/w OUT1 & OUT2 to save the DAC from -ve transients.
* An op-amp is used as a current to voltage converter at the dp of AD 7523 to convert the current dp of AD7523 to a proportional o/p voltage.

V+ supply range = Extended supply range is +5V to +15V.

Vref in range = −10V to +10V.

Ex) Interface DAC AD7523 with an 8086 cpu running at 8 MHz write an assembly language program to generate a saw tooth of period 1 msec with V_{max} 5V.



* port A o/p port.

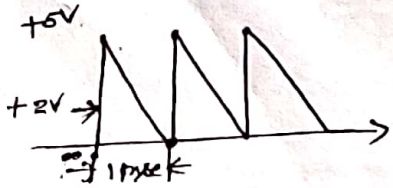
```
        mov   AL, 80h     Initialise port A as o/p port.
        OUT   CWR, AL.
Again : mov   AL, 00h     start the ramp from 0 volts.
back  : OUT   PortA, AL   o/p 00h to DAC.
        INC   AL          Increment AL to get ramp o/p.
        cmp   AL, FFh     ; is upper limit reached?
                  ↑ required.
        JB    back        If not, then increment the ramp
        JMP   Again.      else start again from 00h & continuing sequence.
```

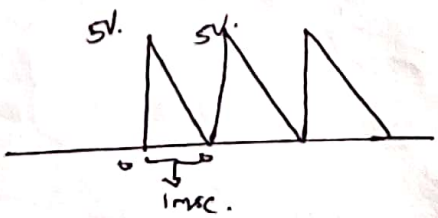
ENDS.

$T = 1/8 \; 0.125\mu s.$

Time required to the executing Loop once $T_L = 21 \times 0.125\mu s = 2.625\mu s.$

$$\text{Count} = \frac{T_d}{nT} \Rightarrow \frac{1ms}{2.625\mu s}.$$

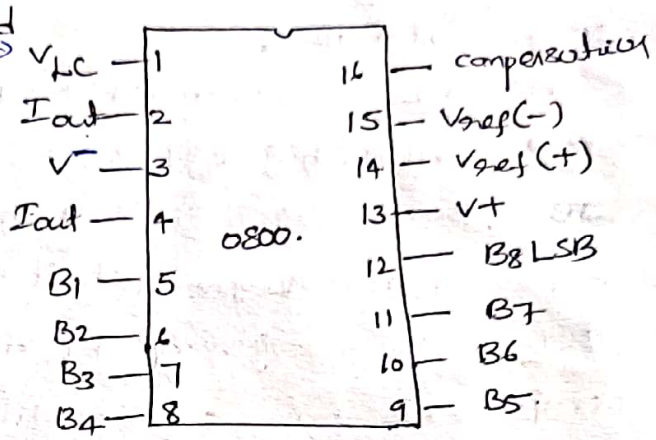




DAC 0800: 8-bit Digital to analog convertor.

features: 0800 is a monolithic 8-bit DAC manufactured by National semiconductor.

* settling time around 100ns.
* operates at supply voltages i.e from 4.5V to +18V.
* Usually V+ is at +12V & V− kept at −12V.
* Pin diagram

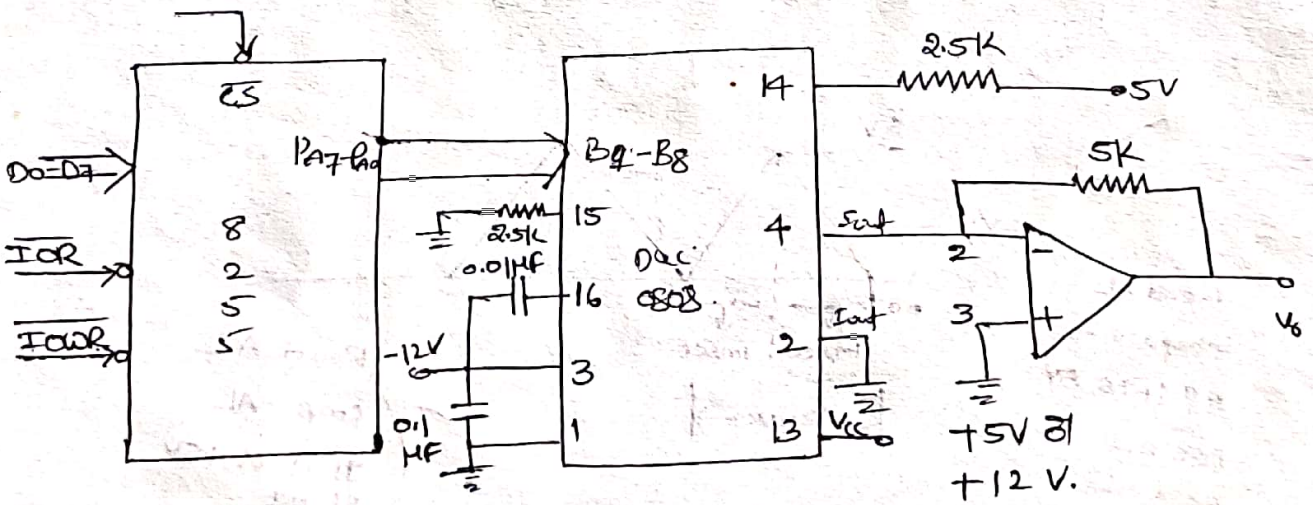


V+ → range is from 4.5V to 18V.
usually V+ is +5V @ +12V.

V⁻ → kept at minimum −12V.

* write an assembly language program to generate a triangular wave of f 500 HZ using the interfacing ckt given in fig. The 8086 system operates at 8 mHz. The amplitude of the triangular wave should be +5V.

Sol:-

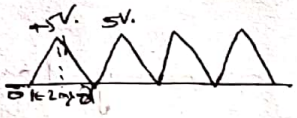


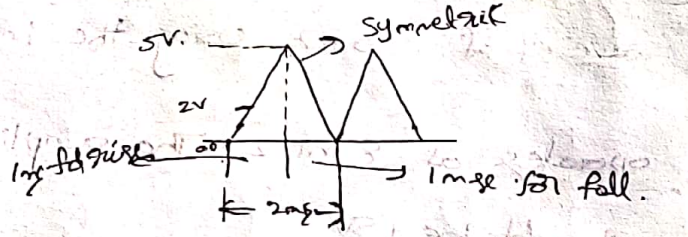
V_{ref} = +5V. to generate a wave of +5V amplitude.

Eg:- f = 500Hz. T = 2msec.

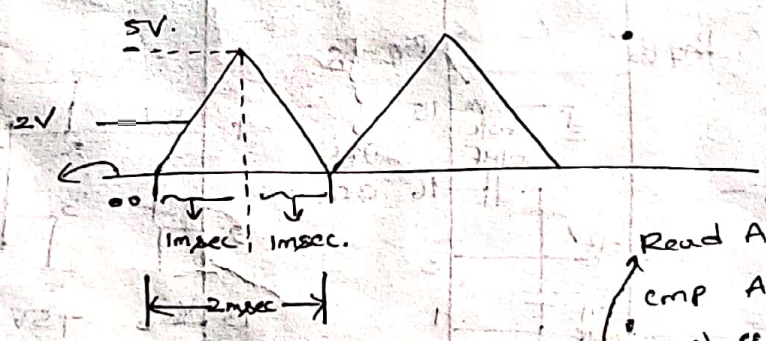
Assume the wave to be generated is symmetric, the wave fall rise for 1ms and fall for 1ms. This will be repeated continuously.

```
mov AL, 80h   ; initialization of port A.
OUT COR, AL

mov AL, 00h   ; start rising ramp from 0V by sending 00h to DAC.

BACK: OUT portA, AL .
      INC AL      ; Increment ramp till 5V.
      cmp AL, FFh ; i.e FFh --> if it is not equal
      JB Back     ; if it is FFh then,
BACK1: OUT portA, AL ; output it and start the falling
      DEC AL      ; ramp by decrementing the
      cmp AL, 00  ; counter till it reaches zero.
      JA Back1    ; then start again for the next cycle.
      JMP Back
ENDS
```





this will be repeated continuously.



Read AL
compare AL
if it is 5V
DEC AL
Read AL

Read AL
cmp AL
if it is +5V
if not
INC AL Loop Back
DEC AL
cmp AL, 00
if not 00 Loop Rep ; jmp Back.

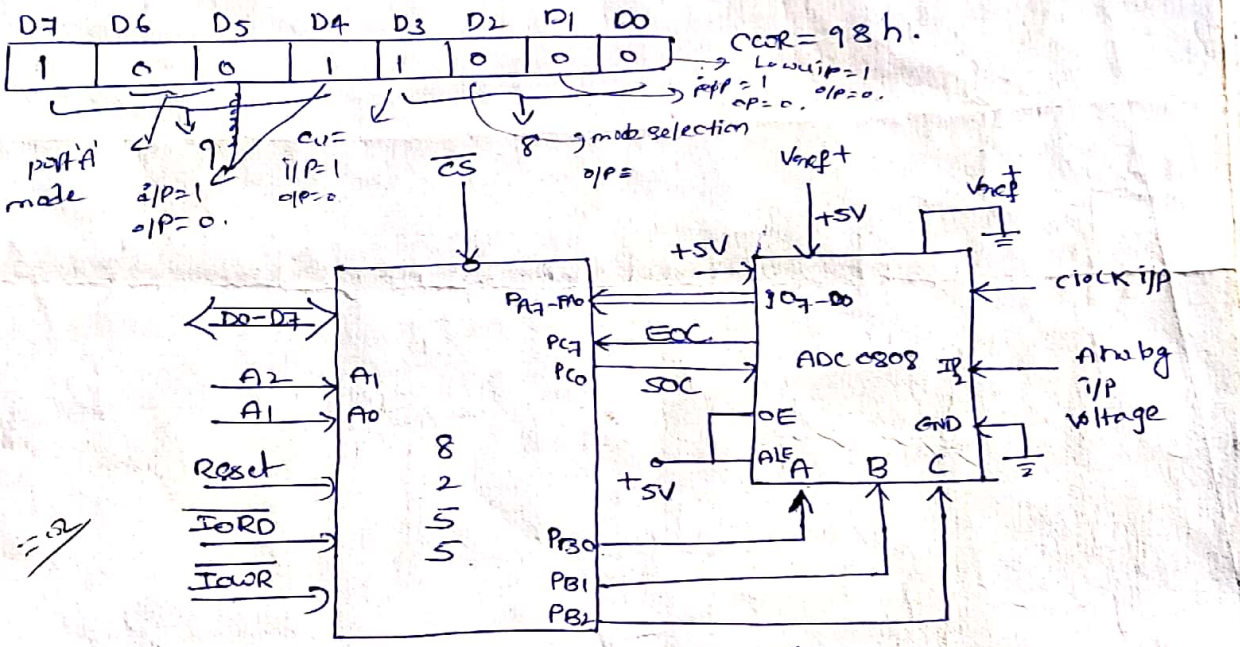
Interface ADC 0808 with 8086 using 8255 ports. Use port A of 8255 for transferring digital data o/p of ADC to the CPU & port C for control signals. Assume that an analog i/p is present at I/P2 of the ADC & a clock i/p of suitable f is available for ADC. Draw the schematic & write required A.L.P.

port 'A' acts as 8-bit i/p port to receive digital data o/p from ADC.

port 'C' upper (P_{C7}) used as i/p port to receive the EOC.

port 'C' lower (P_{C0}) used as o/p port to send the SOC.

the CCOR



Interfacing 0808 with 8086.

the address for selection of analog i/p IP2 is 010 = 02h.

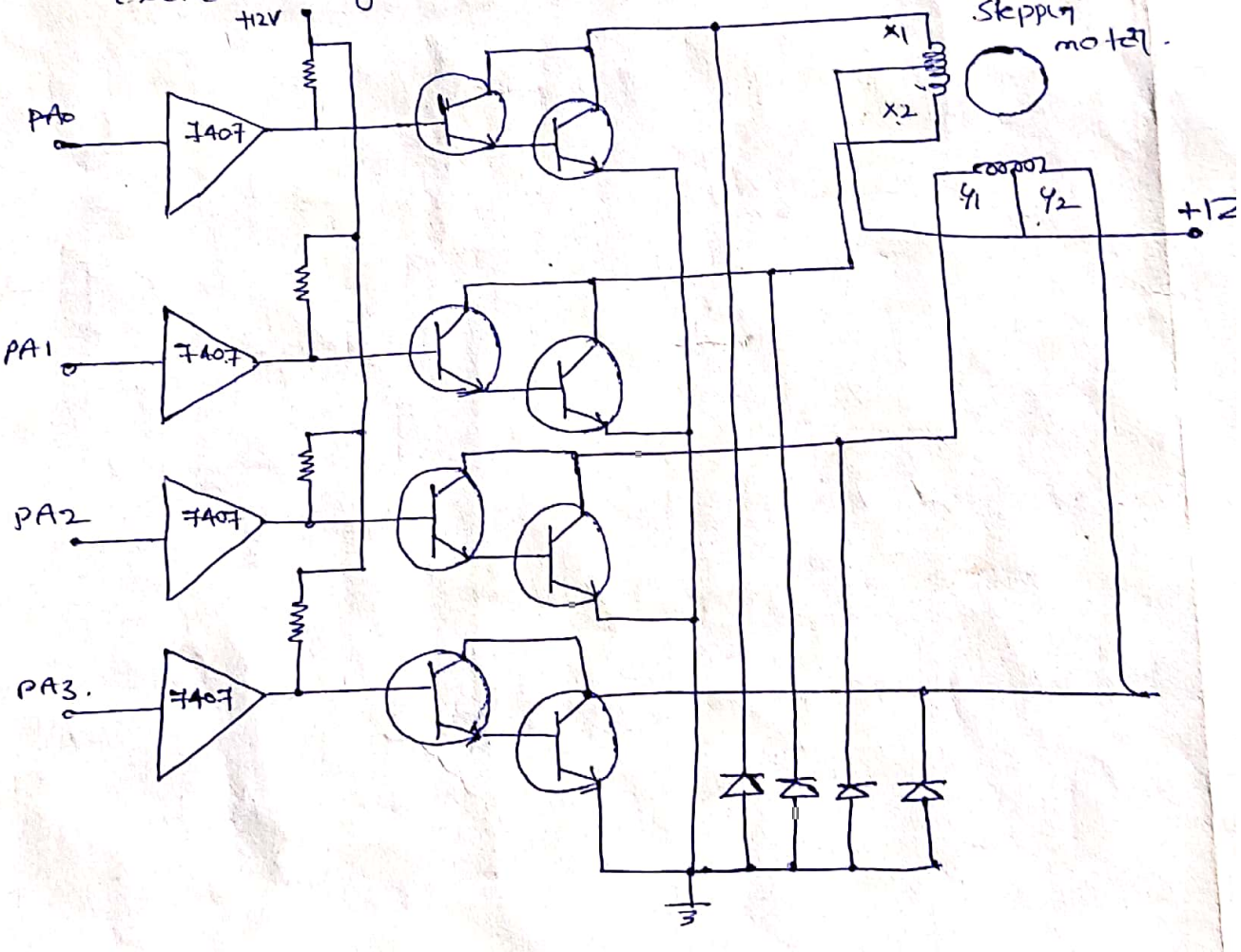
program:

```
MOV  AL, 98h   ; initialise 8255 as
OUT  CCOR, AL  ; discussed above.
MOV  AL, 02h   ; select I/P2 as analog i/p
OUT  port B, AL ;
MOV  AL, 00h   ; give start of conversion
OUT  port C, AL  pulse to the ADC.
```

```
        MOV   AL, 09h
        OUT   port C, AL  ->  mov  AL, 00h
                              OUT  portC, AL
wait:   IN    AL, port C      check for EOC by
        RCL                   Reading port C upper and
        JNC   wait            rotating through carry.
        IN    AL, port A      If EOC, read digital equivalent in AL
        HLT                   stop.
```

Stepper motor interfacing:-

* A stepper motor is a digital motor.
* fig shows the typical 2-phase motor interfaced using 8255.
* Motor has two ~~windings~~ phases, with center-tap winding.
* The center taps ~~are~~ taps of these windings are connected to the 12V supply.
* motor can be excited by grounding four terminals of the two windings.
* Motor can be rotated in steps by giving proper excitation sequence to these windings.
* The lower nibble of port A of the 8255 is used to generate excitation signals in the proper sequence.

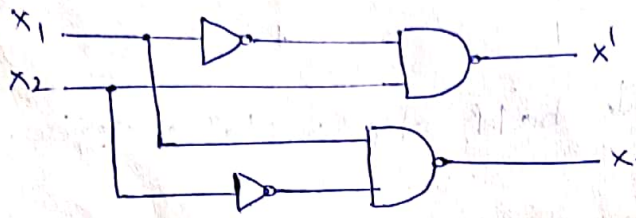


Stepper motor interfacing.

The excitation sequence is shown in table. In this, motor rotates in clock wise direction.

To rotate motor in anti clock wise direction, we have to excite motor in reverse sequence.

* Simultaneous excitation for both ends of winding leads to damage of motor windings.
* To avoid this, digital locking system must be designed.



digital locking system.

from the above shown digital locking ckt only one o/p ~~made~~ is activated (made low) when properly excited, otherwise o/p is disabled (made high).

Step	X_1	X_2	Y_1	Y_2
1	0	1	0	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0
	0	1	0	1

sequence :-

Y_2 X_2
Y_2 X_1
X_1 Y_1
X_2 Y_1
X_2 Y_2.

full step excitation sequence.

The above excitation sequence is called full step ~~sequence~~ in which excitation ends of the phase are changed in one step.

The phase winding excitation may be in half ~~step~~ step sequence the excitation sequence table is as follows.

step	x_1	x_2	y_1	y_2
1	0	1	0	1
2	0	0	0	1
3	1	0	0	1
4	1	0	0	0
5	1	0	1	0
6	0	0	1	0
7	0	1	1	0
8	0	1	0	0
1	0	1	0	1

$\bar{y}_2 \; x_2$

$\bar{y}_2 \; x_1$

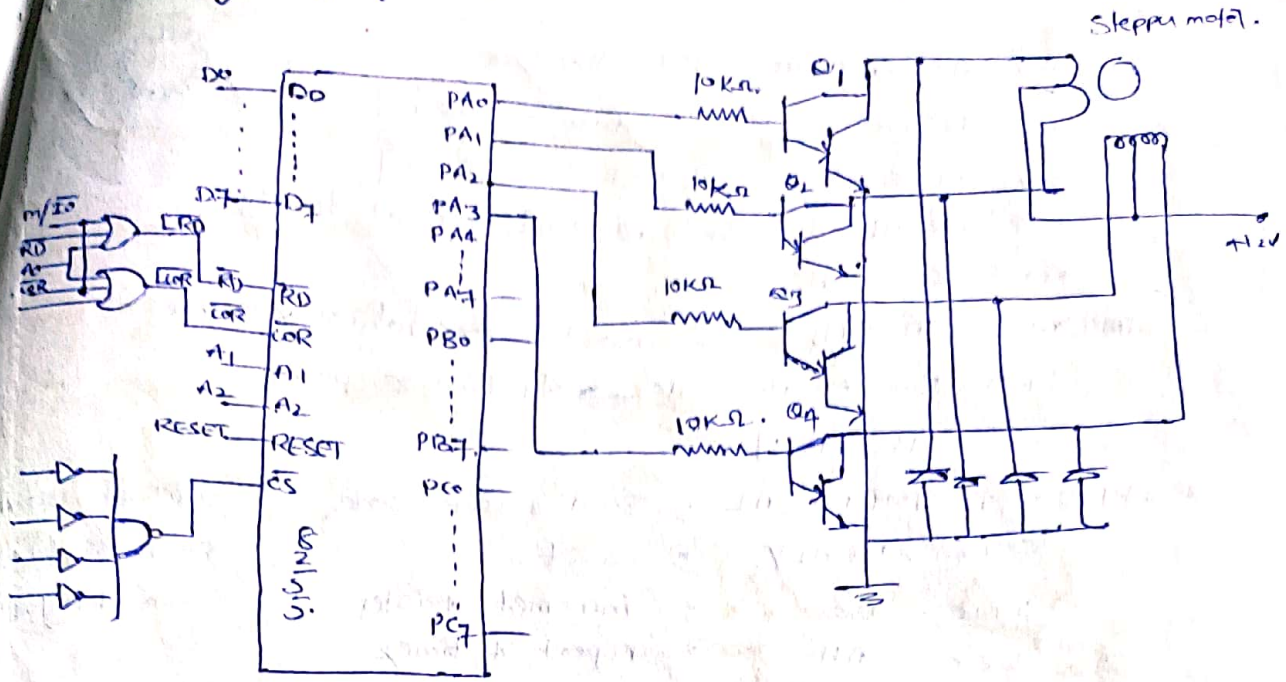
$\bar{x}_1 \; y_1$

$\bar{x}_2 \; y_1$

$\bar{x}_2 \; y_2$

↓ half step excitation sequence.

Interface stepper motor to the 8086 µP system and write an 8086 assembly language program to control the stepper motor.



The above fig shows the typical 2-phase motor rated 12V/0.67A/Ph interfaced with the 8086 µP system using 8255.

* Motor shown in fig has two phases, with center-tap winding. The center taps of these windings are connected to the 12V supply. Due to this, motor can be excited by grounding four terminals of the two windings. Motor can be rotated in steps by giving proper excitation sequence to these windings. The lower nibble of port 'A' of the 8255 is used to generate excitation signals in the proper sequence. These excitation signals are buffered using driver transistors. The transistors are selected such that they can source rated current for the windings. motor is rotated by 1.8° per excitation.

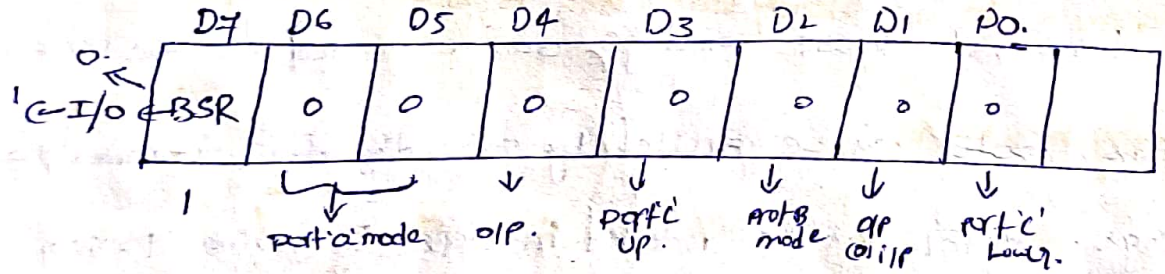
Procedure to rotate a stepper motor clock wise by 90°.

Let Excitation code . DB , 03h, 06h, 09h, 0ch.

```
MOV   AX, @DATA.     [Initialise
MOV   DS, AX                data segment]
MOV   AL, 80h.       [Initialise 8255]
OUT   CR, AL.        ; set repetition count to 50₁₀
MOV   CL, 32H.
START: MOV  AH, 04h.  ——→ count excitation sequence
       LEA  BX, Excite-code. ——→ Initialise pointer.
       mov  AL, [BX].
BACK1: OUT  PORTA, AL.  ; Send Excite-code.
       CALL DEALY       ; count
       INC  BX          ; increment Pointer
       DEC  AH.         ; repeat '4' times.
       JNZ  BACK1
       DEC  CL          ; Repeat 50 times.
       JNZ  START
       RET
       ENDP
```

requirement 50 rotation. Each rotation with delay. (1 msec)

LEA
Load EA to the register.
Load external address to the register.



Read → I/P
Write → O/P

As 80H is used as ctrl wrd for 8255 it 80H

Interfacing of switches :-

$\pi^c = 180$

$1^c = \frac{180^\circ}{\pi}$

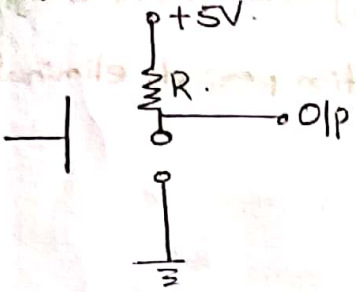
$180 \rightarrow \pi$

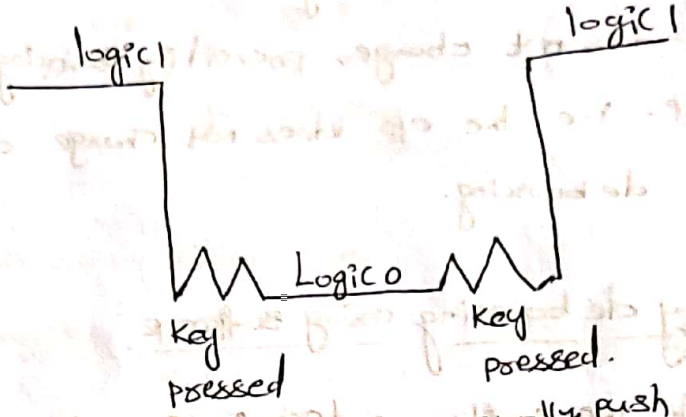
$1 \rightarrow ?$

$= \frac{\pi \times 1}{180^\circ}$

* Interfacing of switches is carried out in two ways.
  (i) by software
  (ii) by hardware.

1) software interfacing.

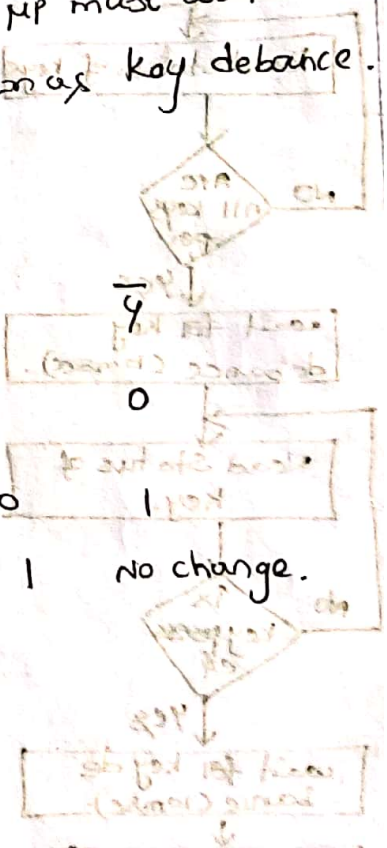


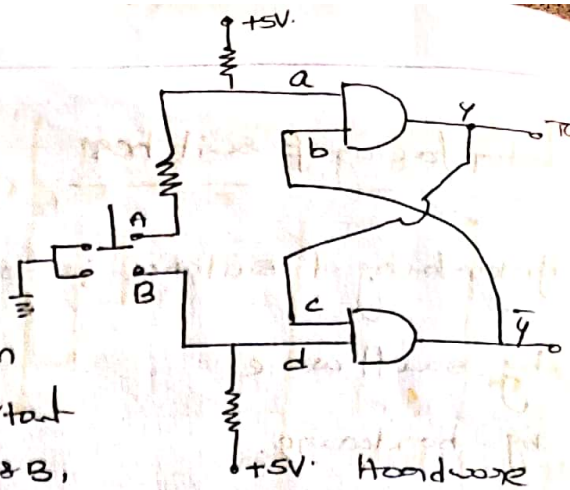


for interfacing switches to the MP based systems, usually push buttons keys are used. These push button keys when pressed, bounces a few times, closing & opening the contacts before providing a steady state reading as shown in fig. Reading taken during bouncing period may be faulty. Therefore MP must wait until the key reach to a steady state, this is known as key debance.

Key debounce using hardware.

key position	a	b	y	c	d	$\bar{y}$
A	0	0	1	1	1	0
B	1	1	0	0	0	1
B/w A & B	1	$\bar{y}$	No change	y	1	No change.





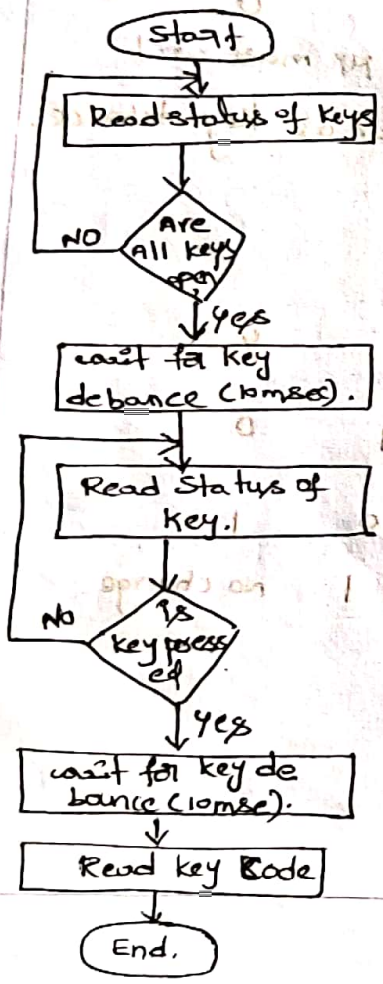
Hardware key debounce ckt diagram.

The ckt consists of flip-flop. The o/p of flop-flop shown is logic 1 when key is at position 'A' (unpressed) and it is logic '0' when key is at position B. It is important to note that, when key is in b/w A & B, o/p does not change, preventing bancing of key o/p. i.e the o/p does not change during transition period, eliminating key debancing.

Key debouncing using Software

In the software technique, when a key press is found, the MP waits for at least 10 ms before it accepts the key as an I/P. This 10 ms period is sufficient to settle key at steady state.

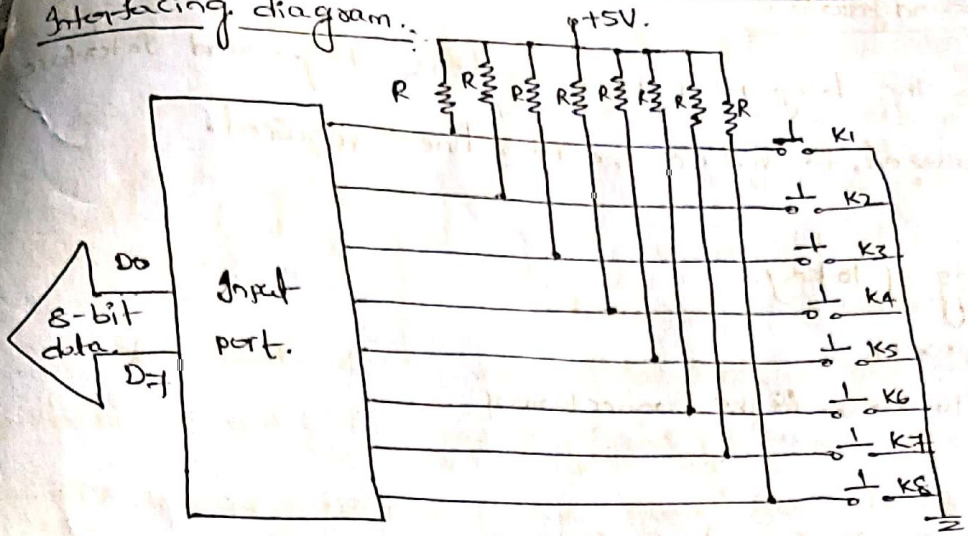
Flowchart



Software programe:

```
START : IN   AL, IN-PORT  → Read key status,
        CMP  AL, FFh      → check if key's are open.
        JNZ  START        → If no, go to start otherwise continue
        CALL DEBOUNCE-DELAY → call debance delay
AGAIN : IN   AL, IN-PORT  → Read key status.
        CMP  AL, FFh      → check, if any key is pressed.
        JZ   AGAIN.       If no, goto AGAIN, otherwise continue.
        CALL DEBOUNCE-DELAY → call debance delay.
        IN   AL, IN_PORT  → Get key code.
        RET.              → Return from subroutine.
```

Interfacing diagram:-



Key	Key code							
	D7	D6	D5	D4	D3	D2	D1	D0
K1	1	1	1	1	1	1	1	0
K2	1	1	1	1	1	1	0	1
K3	1	1	1	1	1	0	1	1
K4	1	1	1	1	0	1	1	1
K5	1	1	1	0	1	1	1	1
K6	1	1	0	1	1	1	1	1
K7	1	0	1	1	1	1	1	1
K8	0	1	1	1	1	1	1	1

Matrix Key board Interface

To interface the large no of keys matrix key board interface technique is used, to reduce the no of lines required.

Ex:- Interfacing of 16 key.

* Normal approach
  It requires 16 lines for the connection of key connection.
* If we arrange it in matrix form the no of lines required are less to '8'.
  4 – lines for row connection
  4 – lines for column connection.

This type of arrangement is shown in fig.

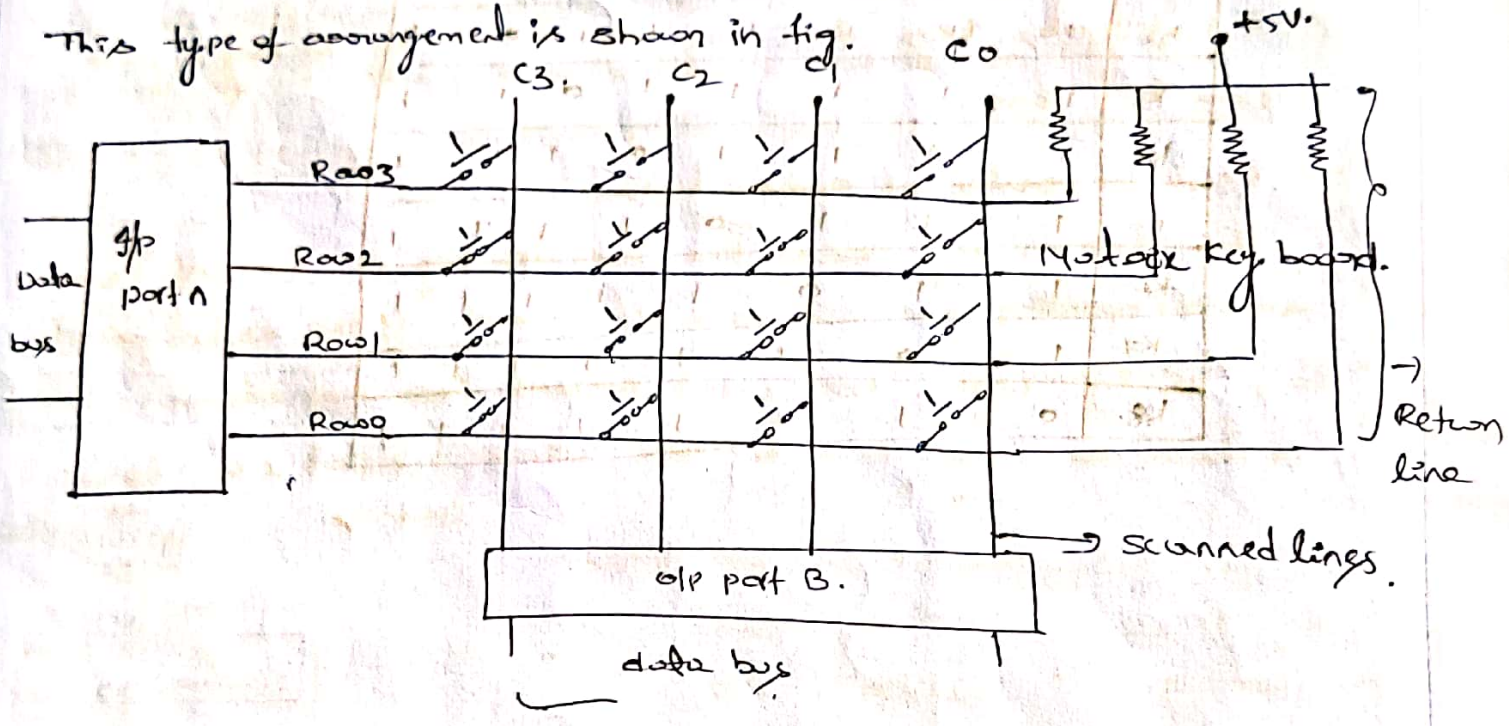


fig shows the interfacing of matrix key board. It requires two ports. an i/p port & an o/p port. Rows are connected to the i/p port referred to as retuned lines, columns are connected to the o/p port referred to as scan lines.

* when all keys are open, row and column do not have any connection.