# Microprocessors

## Unit -3

Remember: RAM on top ROM on the bottom
RAM start at 00000H
ROM ends at FFFFFH

## * Questions on Memory Interfacing

(1) Interface two 4K x 8 EPROMs and two 4K x 8 RAM chips with the 8086 microprocessor.

Step 1: Find the no. of address lines needed for the ROM & RAM.

RAM - no. of chips = 2

Total size = 4K + 4K = 8K

$= 2^3 \times 2^{10}$

$= 2^{13}$

⇒ 13 address lines

Ques For ROM : no. of chips = 2

Total size = 8K

$= 2^{13}$

= 13 address lines

Step 2: Find out starting and ending address for each

ROM addresses: end = FFFFFH

ROM size = 8K $= 2^3 \times 2^{10} = 2^{13}$

1 1111 1111 1111
1 F F F

start = FFFFF
− 1FFF
FE000H

Start = FE 000 H
end = FFFFF H

You can put the RAM anywhere, but for the sake continuity:

end
~~start~~ = FDFFFH

size = 8K

= 1FFFH

beginning: FDFFFH
1FFFH
FC000H

start = FC000H
end = FDFFFH

Step 3: Draw the memory map

| | A19 | A18 | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EPROM | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RAM | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

no. of address lines = 13

=> $A_0 - A_{12}$

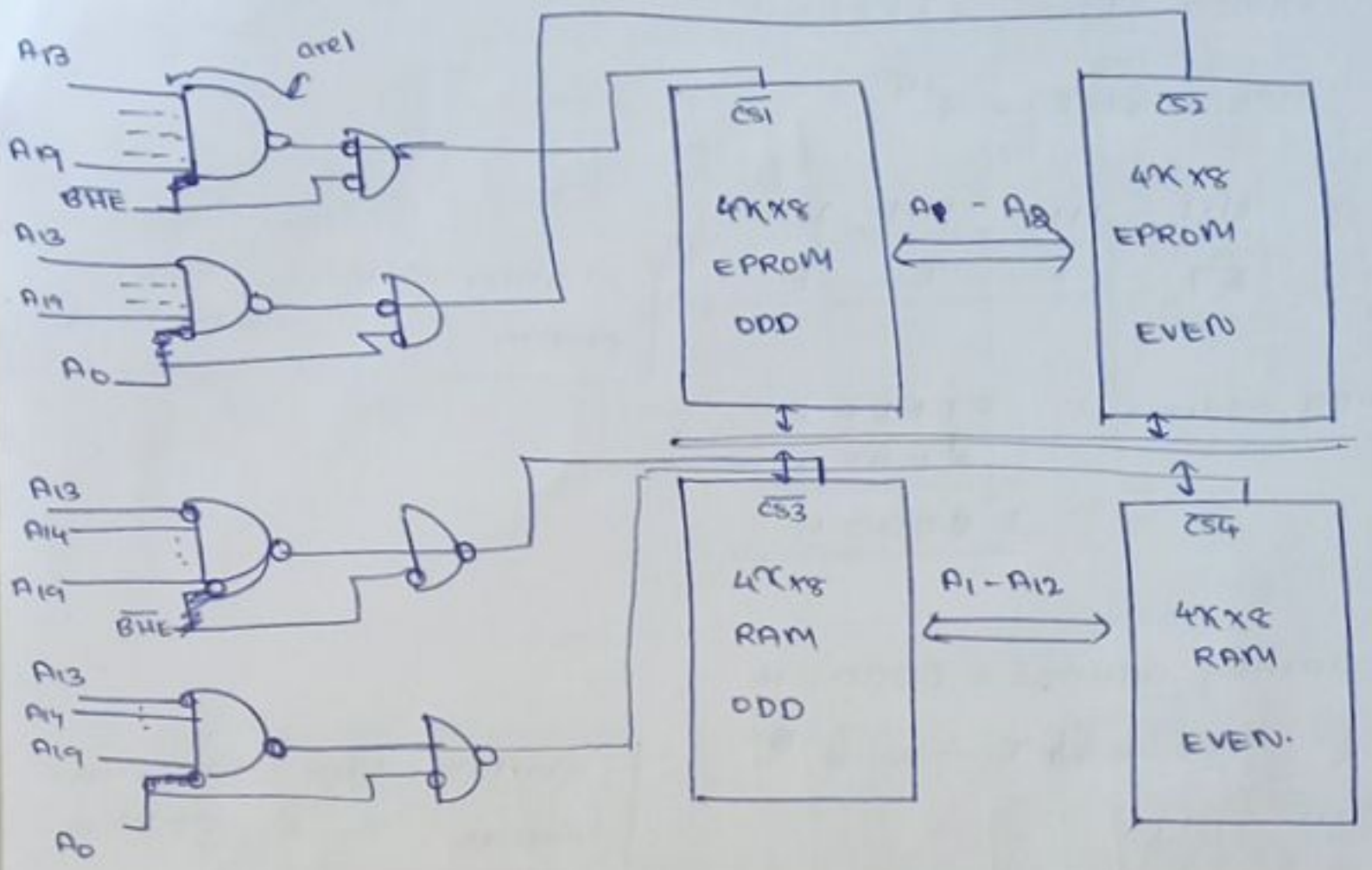
Step4: Figure out the chip select lines

$A_{13} - A_{19}$ to be used for CS

Differentiate between EROM and RAM with A13

Select even bank with $A_0$

odd bank $\overline{BHE}$




(2) Interface 2 chips of 16K x8 EPROM and 2 chips of 32K x8 RAM with 8086

Step 1: Find the no of address lines needed for ROM & RAM

ROM : no. of chips = 2

16K x 8 ROM

Total size = 32K

$= 2^5 \times 2^{10} = 2^{15}$

$\Rightarrow$ 15 address lines

RAM : no. of chips = 2

32K x 8

Total size = 64K

$= 2^6 \times 2^{10} \Rightarrow 2^{16}$ = 16 address lines

Step 2: Find the starting & ending address for each

ROM: ending address = FFFFFH

size = 32K = $2^{15}$

⇒ 1111 1111 1111 1111

F F F F

starting address : FFFFFH
− 7FFFH
F8000H

starting address = F8000H
ending = FFFFFH

RAM: starting address = 00000H

size = 64K = $2^{16}$

1111 1111 1111 1111

F F F F

ending address = 00000H
− FFFFH
0FFFFH.

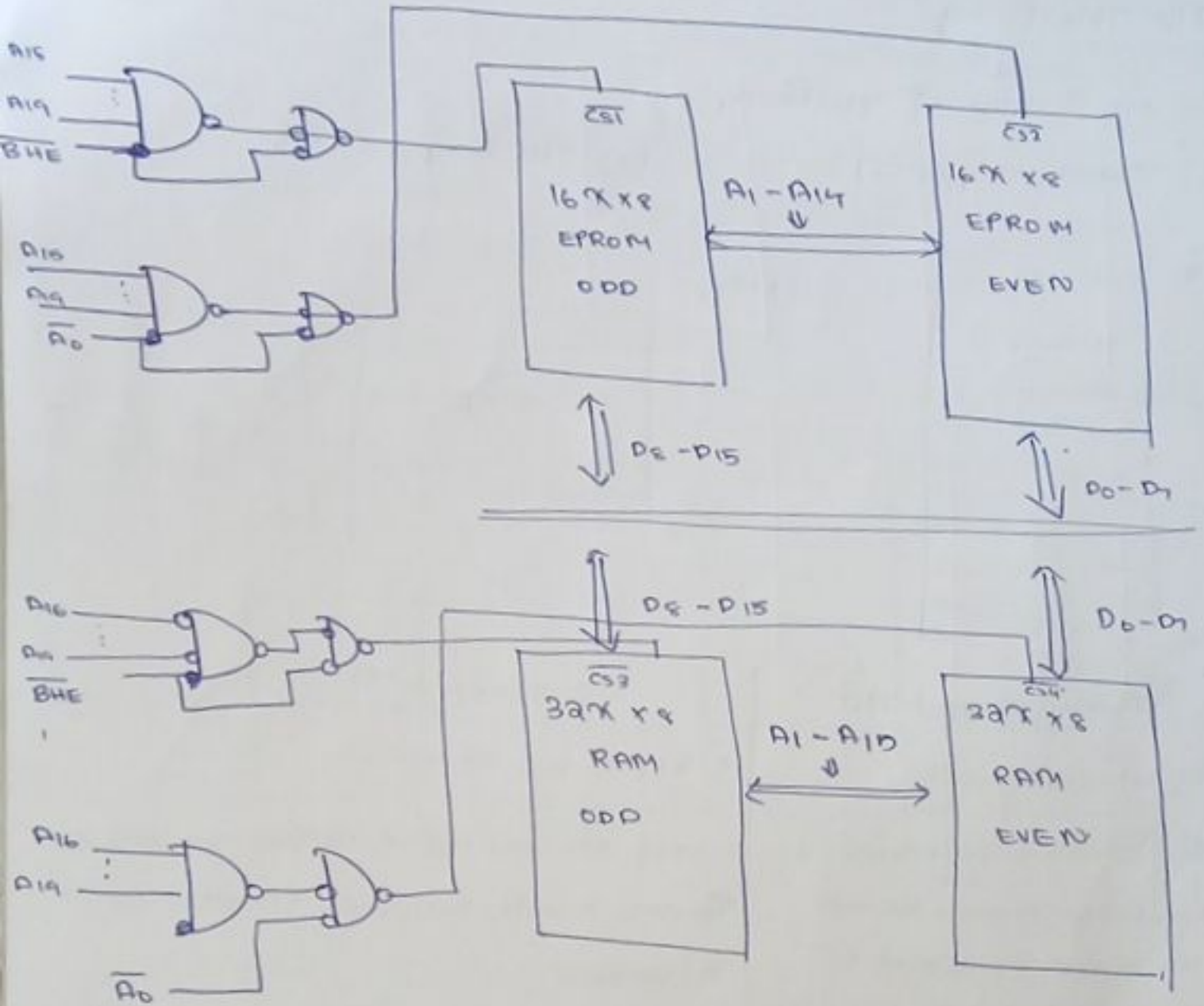
Step 3: Draw the memory map

| | $A_{19}$ | $A_{18}$ | $A_{17}$ | $A_{16}$ | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROM | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RAM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Step 4: Figure out the chip select lines

ROM – 15 address lines ⇒ $A_{15}$ – $A_{19}$ used for chip select

RAM – 16 address lines ⇒ $A_{16}$ – $A_{19}$ used for chip select




(3) A circuit containing 32 KB of RAM is to be interfaced to an 8086 so that the first address of the RAM is at 48000H.

What is the entire range of the RAM address?

Ans

starting address = 48000H

size = 32 KB = $2^{15}$

= 111 1111 1111 1111

= 7 F F F

range = 48000H to 4FFFF

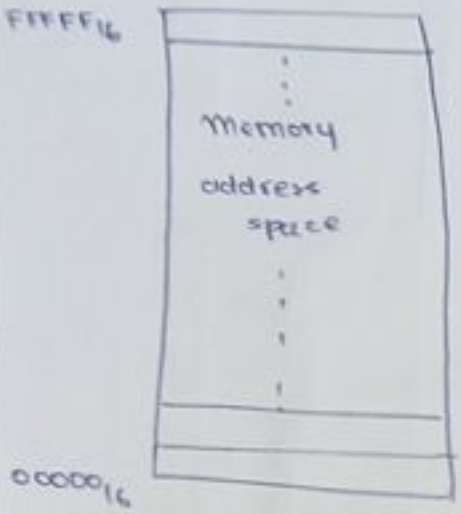
ending addr = 48000H
+ 7FFFH
4FFFF

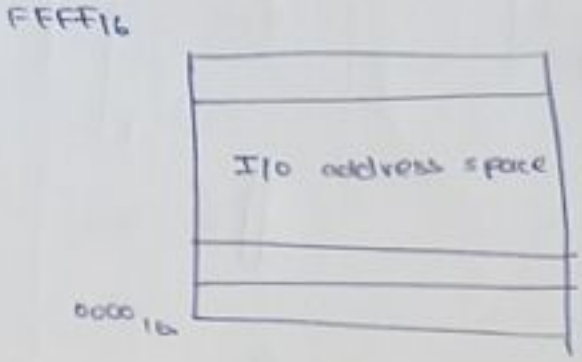
check pg 5 on mpt pdf (has a longer q)

## * I/O Interfacing

There are 2 ways of interfacing:

(i) Memory mapped I/O (ii) I/O mapped I/O

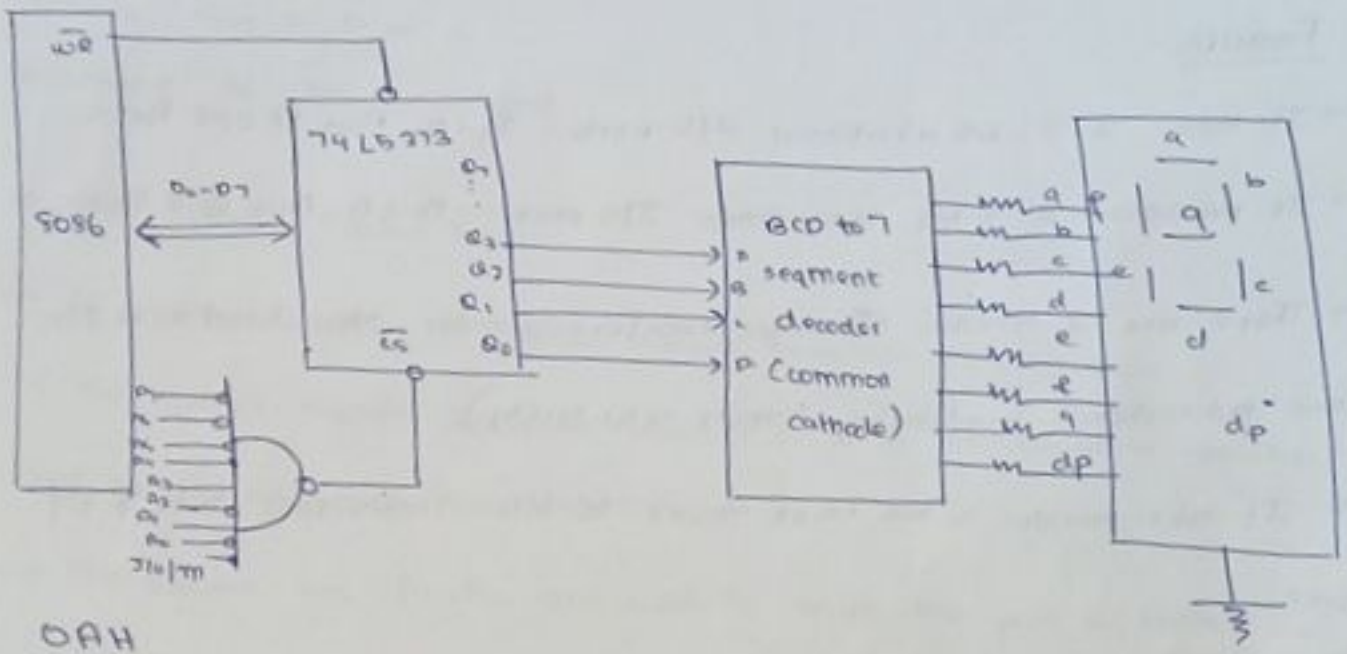



| Memory mapped I/O | I/O mapped I/O |
|---|---|
| 1. 20 bit addresses | 1. 8 or 16-bit addresses |
| 2. The IO ports or peripheral can be treated like memory locations so all instructions related to memory can be used for data transfer. | 2. Only IN and OUT instructions can be used for data transfer between I/O devices & the processor. |
| 3. data can moved from any register to port & vice-versa. | 3. data transfer can happen only between the accumulator & the ports |
| 4. When memory mapping is used for I/O devices, the full memory address space cannot be used for addressing memory. | 4. When IO mapping is used for I/O devices, the full address space can be used for addressing memory. |
| 5. 'M/IO' is high | 5. 'M/IO' is low. |
| 6. I/O device data is also given to ALU | 6. ALU operations not directly applicable to input-output data. |

Example - Design an 8086 interface & write ALP to displaying the count from 0 to 9 continuously in a 7 segment LED display. Select the port address suitably. Use I/O mapped I/O for interfacing.

Ans. If the port is 0AH




0AH

00001010

ALP

```
L1 :  MOV AL,00
L2 :  OUT 0AH,AL
      CALL DELAY
      INC AL
      CMP AL,0A
      JZ L1
      JMP L2
```

# * Parallel Communication Interface

→ Programmable Peripheral Interface - 8255

→ It is an I/O Port chip used for interfacing I/O devices with a microprocessor.

## Features

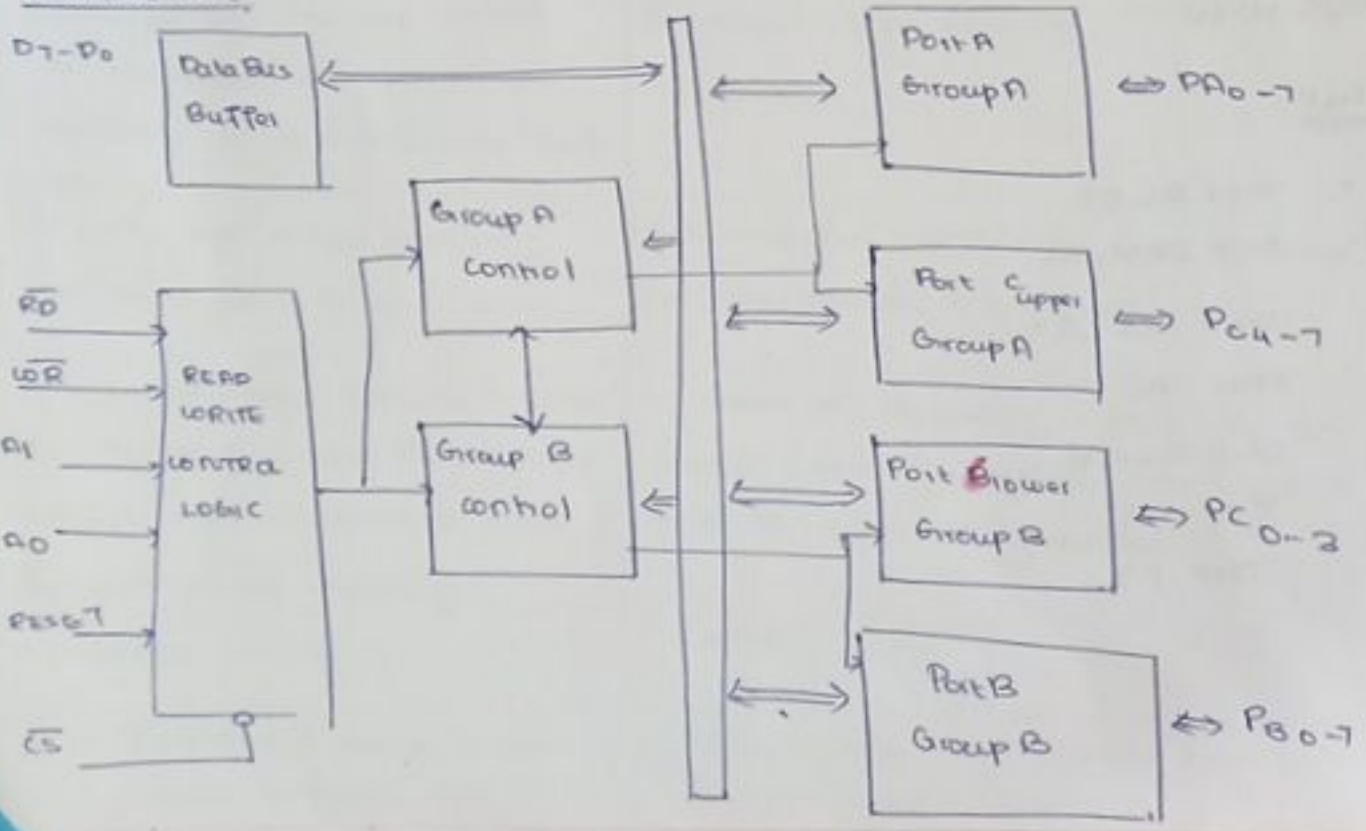
→ It has 3 8-bit directional I/O ports: Port A, Port B and Port C.

→ It provides a 8-bit bidirectional I/O ports - Port A, Port B & Port C.

→ There are 3 modes of data transfer - simple I/O, handshake I/O and bidirectional handshake (more reliability)

→ It also provides a bit reset mode to alter individual bits of Port C.

## Architecture



## Explanation of Architecture.

### 1. Data bus buffer

→ An 8-bit bidirectional buffer used to interface the internal data bus of 8255 with the external system data bus

→ It can send to 4 places :
data to Port A, B or C
command to the control word

### 2. Read / Write Control logic

→ It accepts addresses & control signals from the microprocessor.

→ The control signals determine whether it is a read or write operation and also selects / resets the 8255 chip

→ The address bits $A_1$, $A_0$ are used to select the port or control word register

| For 8255 | For 8086 | Selection |
|---|---|---|
| 0 0 | 0 0 | Port A |
| 0 1 | 0 1 | Port B |
| 1 0 | 1 0 | Port C |
| 1 1 | 1 1 | Control word |

### 3. Group A control
- controls Port A and Port C upper.

### 4. Group B control
- accepts controls Port B and Port C lower.

## 5. Ports and Mode of operation

| | Mode 0 | Mode 1 | Mode 2 |
|---|---|---|---|
| | Simple I/O | I/O with handshake | Bidirectional data transfer mode. |
| $P_A$ | ✓ | ✓ | ✓ |
| $P_B$ | ✓ | ✓ | $M_1/M_0$ |
| $P_C$ | ✓ | X sacrificed for handshaking | X handshaking |

handshaking needs

4 additional lines

⇒ give 3 lines of port C to Port A (take upper)

& 3 lines of port C to Port A (take lower)

## * 8255 Control words

Commands come to the same address (86)

msB identifies command type : → 1 ⇒ I/O

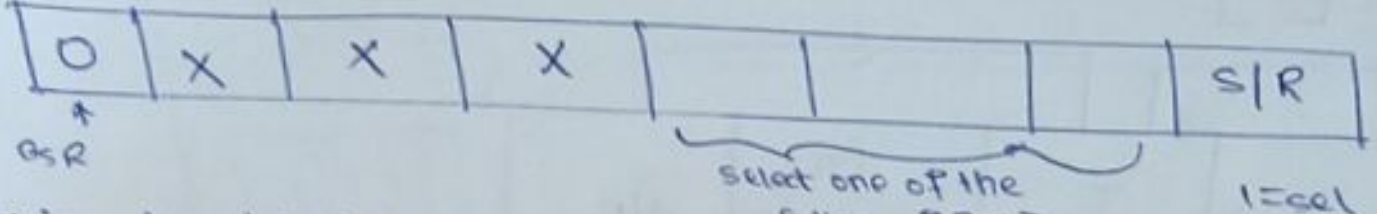
→ 0 ⇒ BSR

### A. I/O Command

For every port, you need to identify mode & direction of i/o

| 1 | mode of PA | $P_A$ dir | $P_C$ u dir | mode of port B | $P_B$ dir | $P_C$ L dir |
|---|---|---|---|---|---|---|
| ↑ | | | | | | |
| operating in I/O mode | 00 - $m_0$<br>01 - $m_1$<br>1X - $m_2$ | 1 → i/p<br>0 → o/p | 1 → i/p<br>0 → o/p | 0 → $m_0$<br>1 → $m_1$ | 1 → i/p<br>0 → o/p | 1 → i/p<br>0 → o/p |

## B. BSR Command




during handshaking

only 3 lines are taken

from $Pc_{upper}$ or $Pc_{lower}$

⇒ 2 lines are free - they can be controlled using the BSR mode, bit wise.

## Data Transfer using the 3 modes

① Mode 0 - Port A & B are used a simple 8-bit bidirectional input ports.

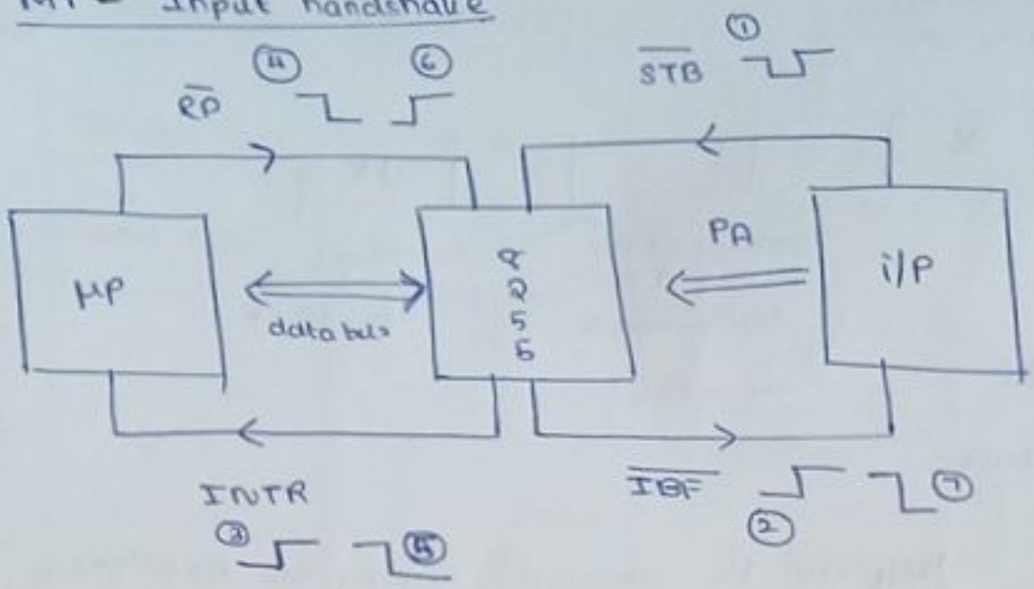
→ Each port can be programmed as input or output ~~new~~ individually

② Mode 1 (Handshake IO)

→ In mode1, handshake signals are exchanged between the devices before the data transfer take place.

→ Port A & B are used as 2 8-bit IO ports, each port uses 3 lines from Port C for hand shake

## M1 - Input handshake




Working

→ The peripheral device places data on the port bus & informs by making the STB low

→ The input port accepts the data and informs the peripheral to wait by making IBF high. This prevents the peripheral from sending more data to the 8255 (Data loss is prevented)

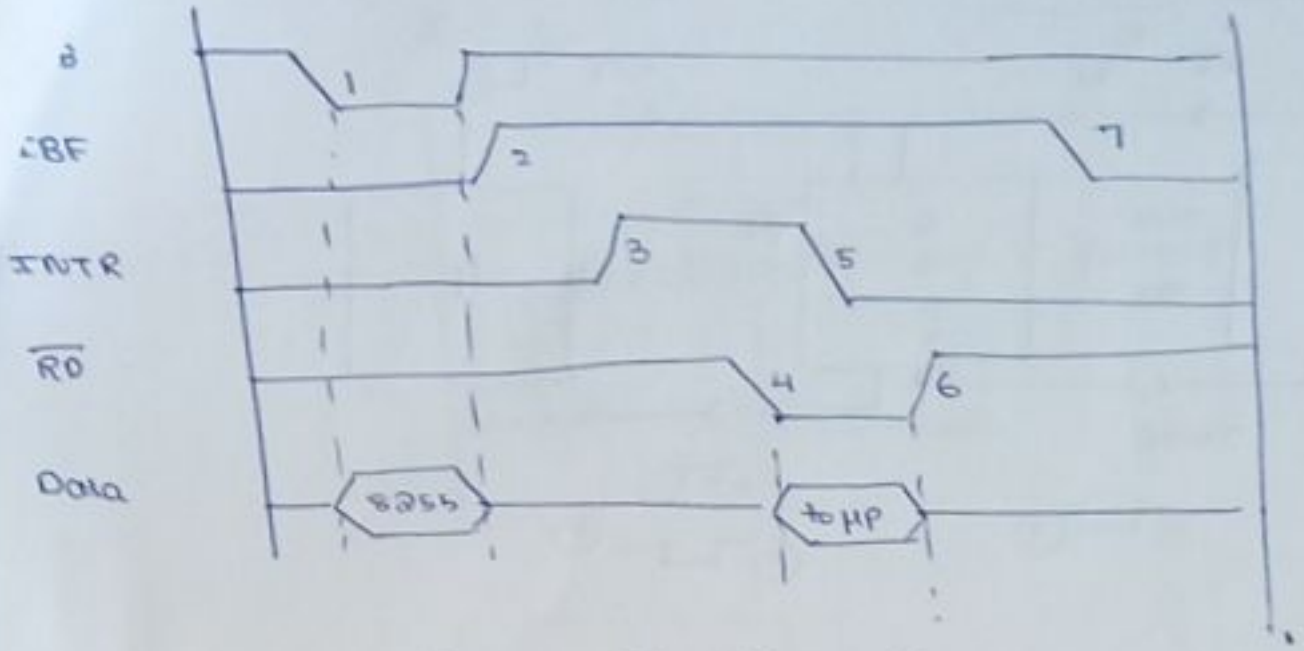
→ 8255 interrupts the μP through the INTR line by setting it (when it is free) to high.

→ The μP begins to read data by setting the $\overline{RD}$ to low.

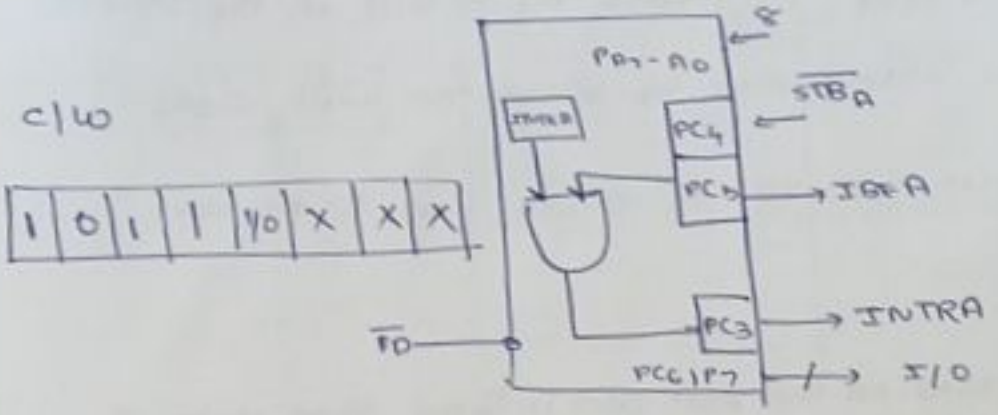
→ As soon as it begins to read, the $\overline{INTR}$ is set back to high, so that there is no duplication of data

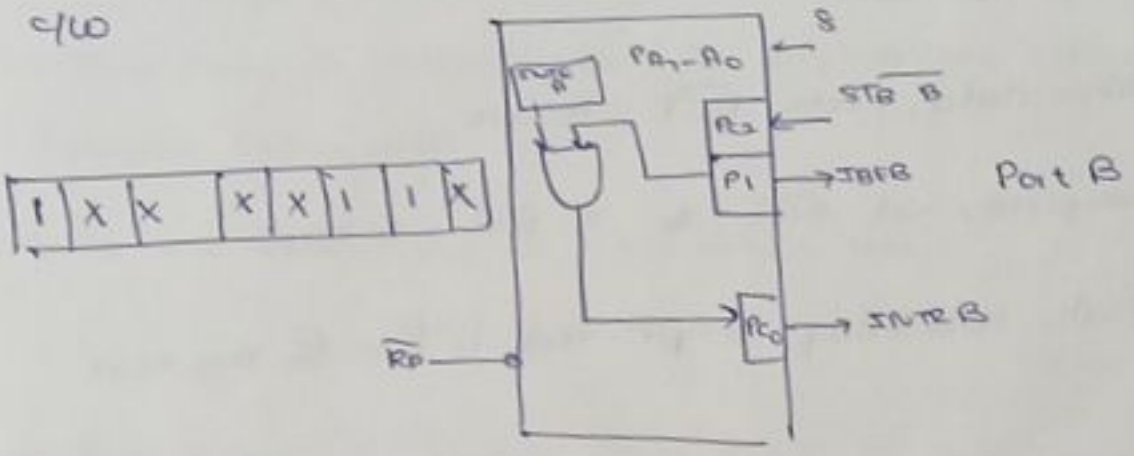
→ Once all the data is read, $\overline{RD}$ goes back to high

→ $\overline{IBF}$ goes to low, to indicate that the peripheral can send more data.

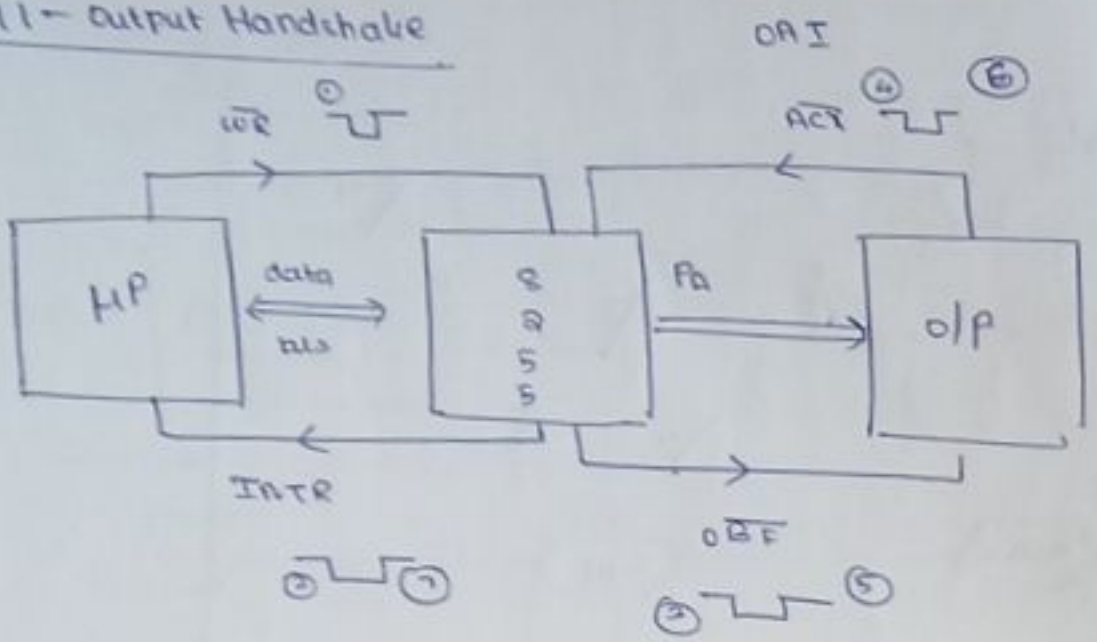



Pin Representation of Model Input Handshaking



## M1 – Output Handshake




Working:

→ When the output port is empty, (indicated by a high on the INTR line) the μP writes data on the output port by giving the $\overline{WR}$ signal.

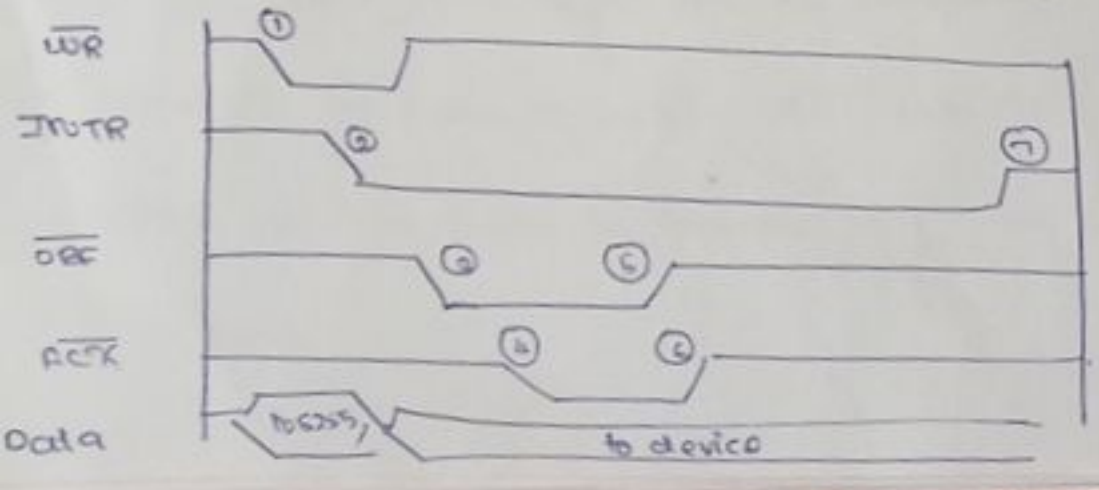
→ As soon as $\overline{WR}$ is given, INTR is made low to prevent overwriting

→ 8255 puts the data on the port and informs that data is available by making $\overline{OBF}$ low.

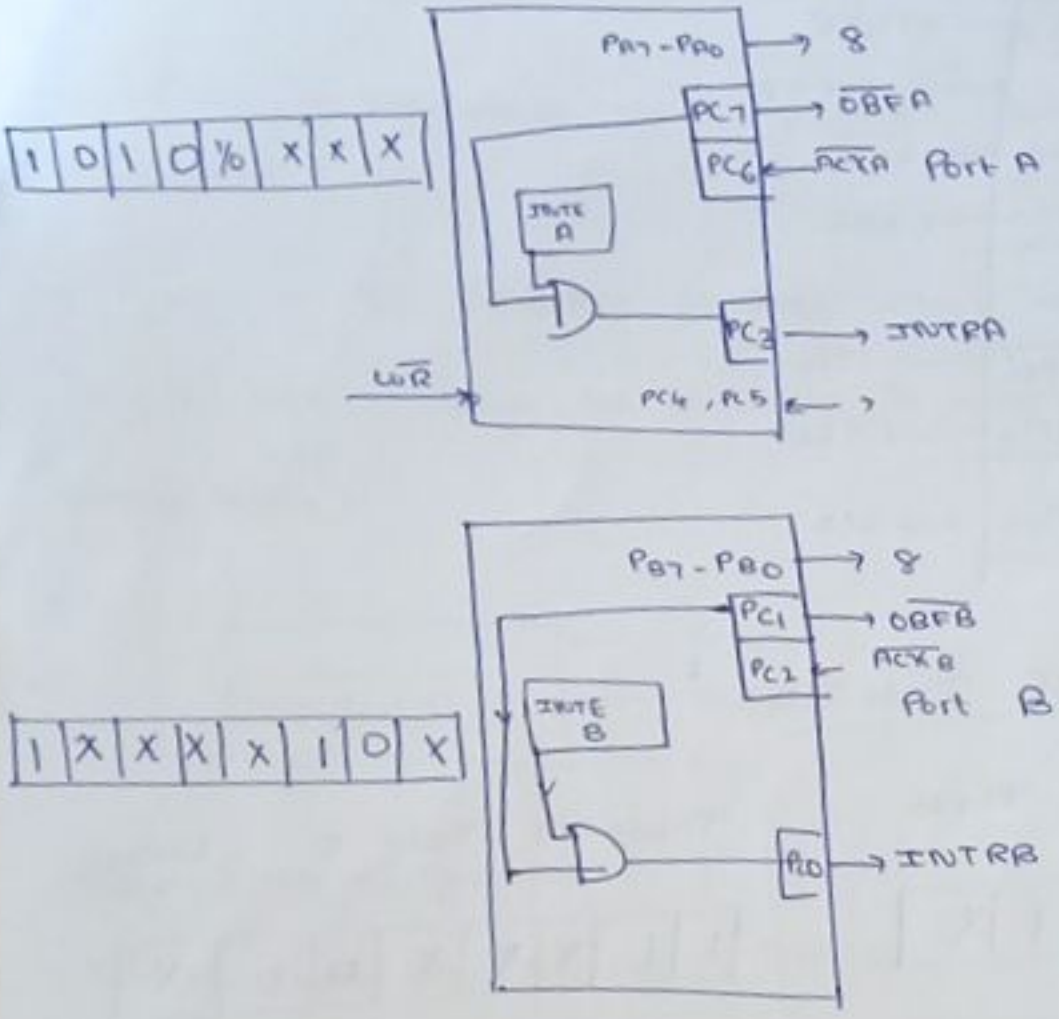
→ O/P device takes data, sets $\overline{ACK}$ to low

→ once o/p is complete, set $\overline{ACK}$ & $\overline{OBF}$ to high

→ Set INTR to high, indicating to μP that is free for the next transfer.



→ In Representation




## Mode 2 - Bidirectional Handshake I/O

→ This mode is primarily used in applications such as data transfer between 2 computers.

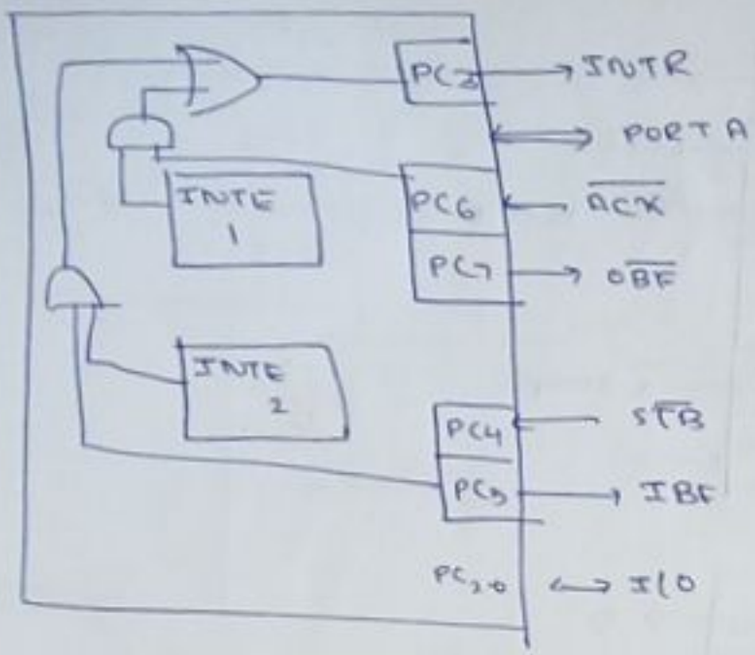
→ In this mode, Port A = bidirectional port

Port B = mode 0 or mode 1

→ Port A uses 5 signals from Port C as handshake signals for data transfer.

→ The remaining 3 signals from Port C can be used either as simple I/O or as handshake for port B.

## Pin Representation




## Different Combinations of Mode I/O

Mode 2 & Mode 0 - Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | X | X | X | 0 | 1 | X |

mode0

Mode 2 & Mode 0 - Output

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | X | X | X | 0 | 0 | X |

mode0

Mode 2 & Mode 1 - output

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | X | X | 1 | 1 | 0 | X |

Mode 2 & Mode 1 - Input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | X | X | X | 1 | 1 | X |

## * Timer Interface - 8253/8254

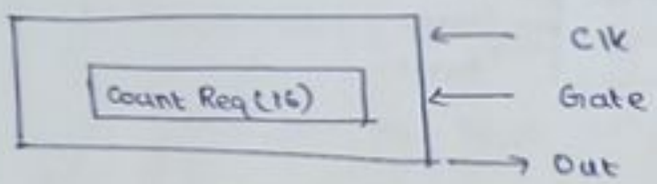
→ 8253/8254 = PIT = Programmable Interval Timer - used to produce delays by countings, a kind of hardware delay

→ The benefit is that the processor is still free to carry out other activities while 8253 does the delay/counting

→ 8253 has 3 16-bit down counters, 3 different delays can be produced simultaneously.

→ Since it has 16-bit counters, it can count to FFFF (65,535)

→ Note - the data bus through which the count is to be passed is 8 bits - must be given in 2 cycles (lower bytes then higher bytes)




Steps of operation

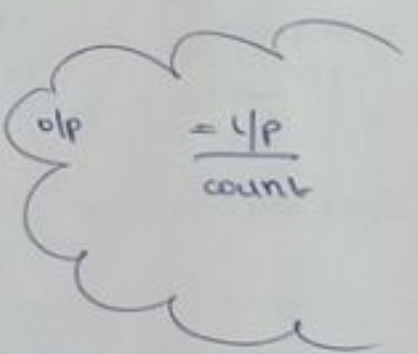
→ Load count into the reg in 2 cycles.

→ Apply a clock pulse (CLK) - decides the frequency at which the timer decrements - used as a trigger to cause the next change

→ To enable counting, set Gate to 1.

→ Once the timer / counter is done - the 8253 notifies the µP by setting the Out

Architecture



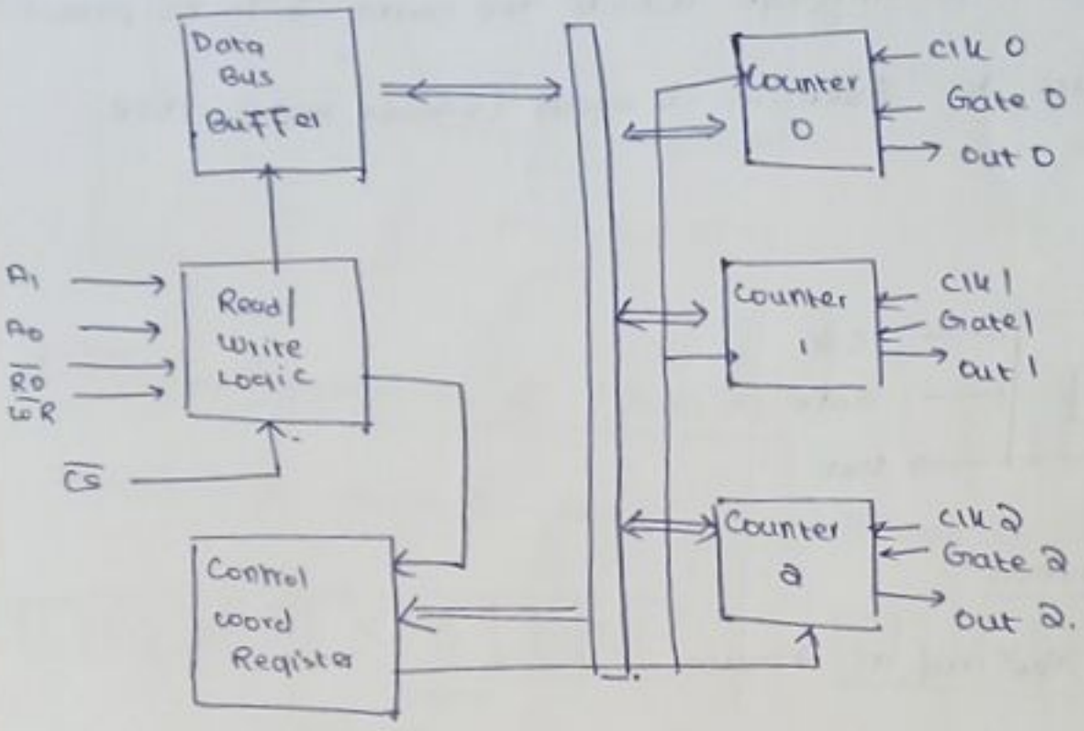

→ The data bus buffer takes in the count value

→ There are 3 counters, each with their own clk, gate, out.

→ Both the count & the command (relating to the mode of operation) can be given through the data bus.

→ Four address lines are needed to control the counters + control word register.

→ The addressing is done with the read/write logic with the $A_1$ & $A_0$ pins.




## Control Word Format

SC = select count

can give higher, lower or both bytes

mode selection (1 of 6)

| SC0 | SC1 | RW1 | RW0 | M2 | M1 | M0 | BCD |
|---|---|---|---|---|---|---|---|

00 → C0
01 → C1
10 → C2

01 → LB
10 → HB
11 → Both

000 - m0
001 - m1
010 - m2
011 - m3
100 - m4
101 - m5

1 = BCD (dec)
0 = bin count (hex)

write a pram to produce a sq. wave of 1KHz from an input freq of 16.525 MHz

$$op = \frac{i/p}{count} \qquad count = \frac{16.525 \times 10^6}{1 \times 10^3} \text{ (convert to hex)}$$

## * 8253 Timer Modes

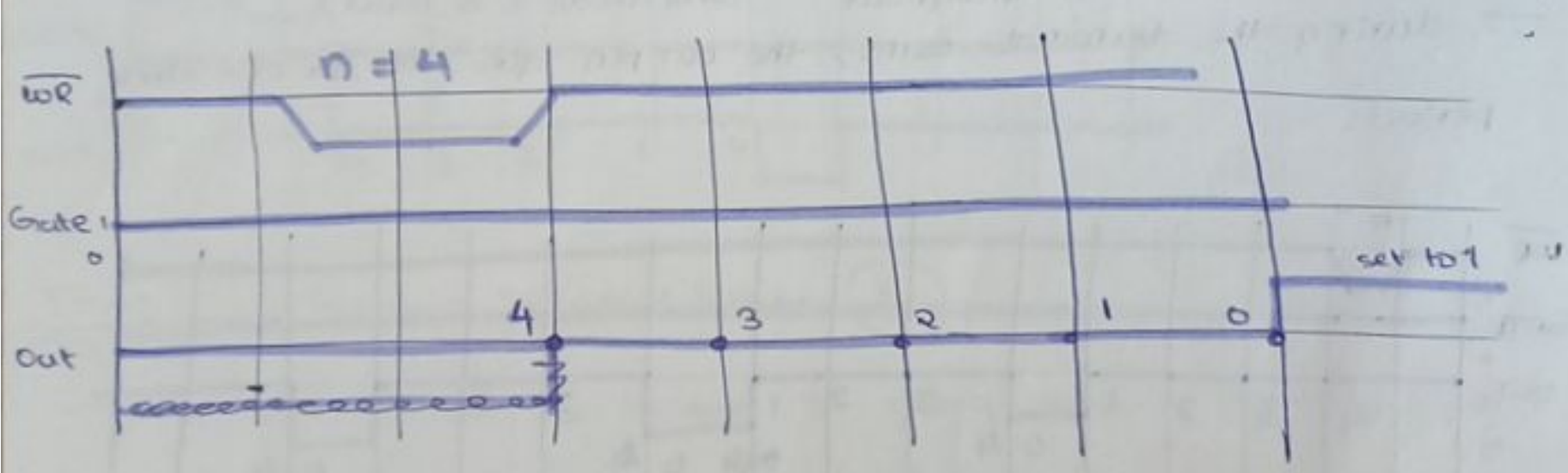
Mode 0 - Interrupt on Terminal Count (5)

→ Program counter to an initial value.

→ The counter counts down, reaches 0 & then stops.

Working :

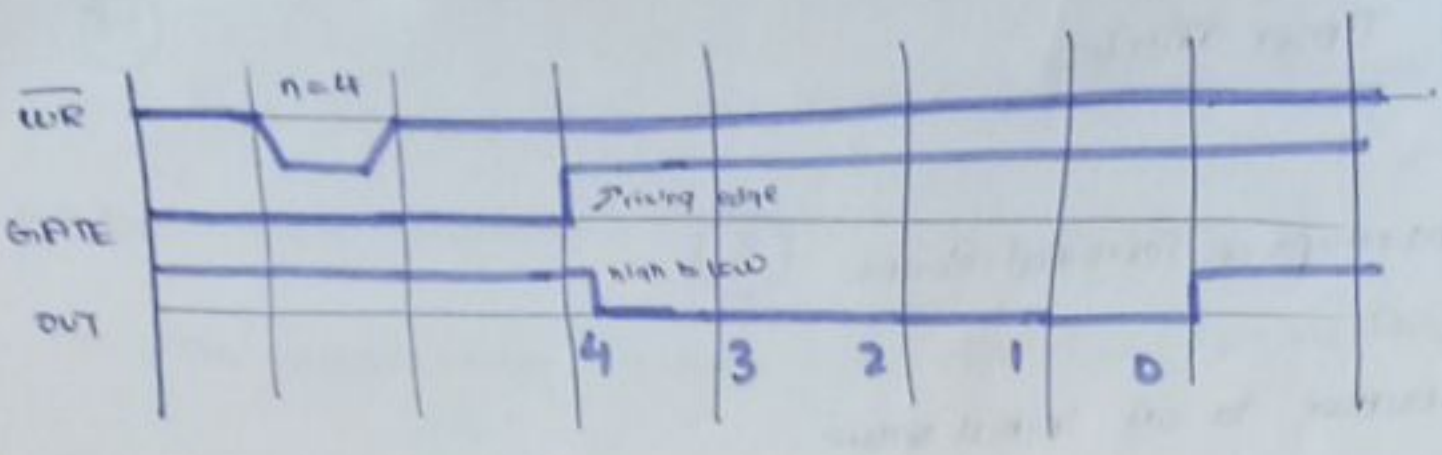
- Load count into 8253 w/ $\overline{WR}$
- Set gate to high
- Begin count after count is loaded
- Set out to high (1), at terminal count

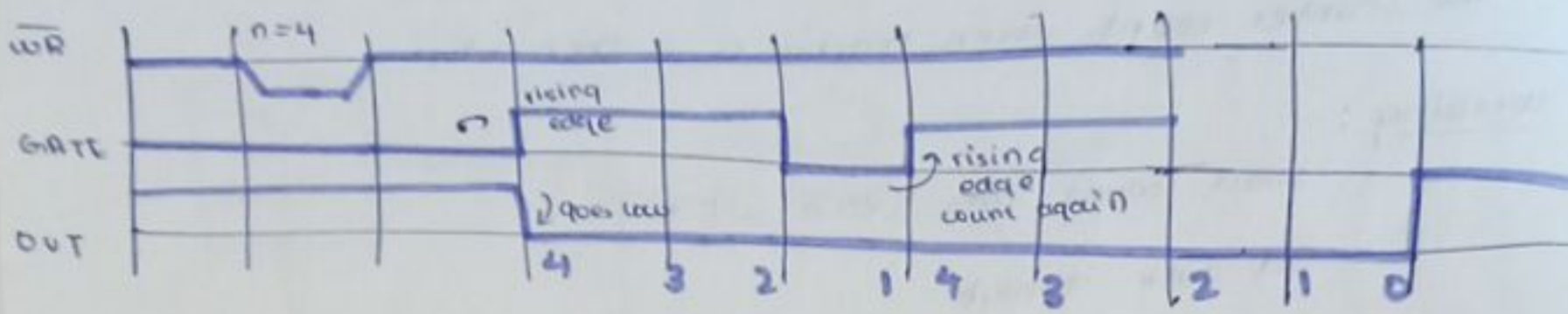



Mode 1 - Programmable One Shot (5)

→ Counting can be retriggered again and again with the help of the rising edge of the gate input

→ Note that in this case, when counting happens OUT moves from high to low & back to high again, once counting is done.



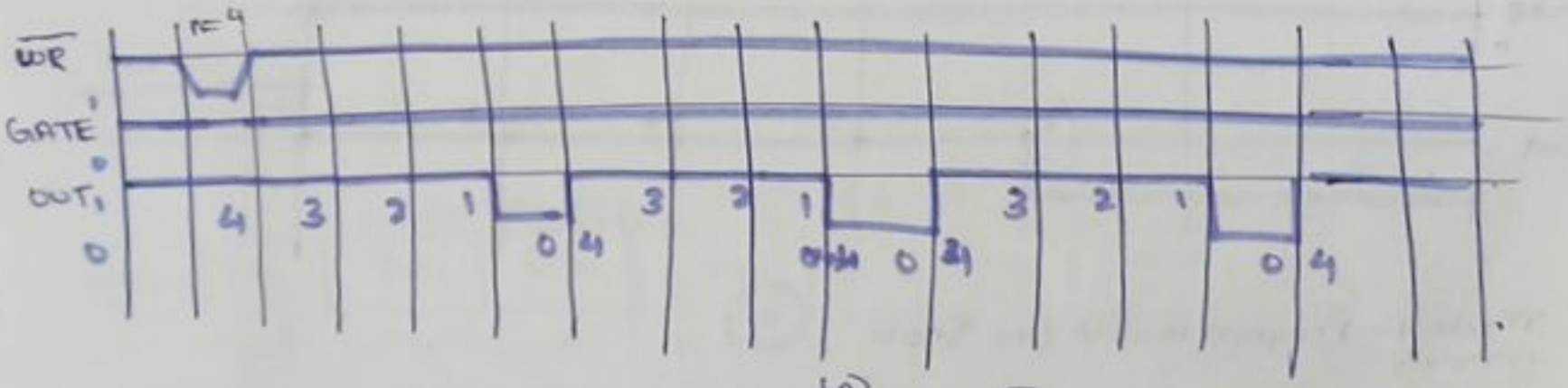




## Mode 2 - Rate Generator ©

→ becomes a 'divide by n' counter.

→ Gate remains high throughout (only mode 2 is weird)

→ during the terminal count, the OUT pin goes low for one clock period.



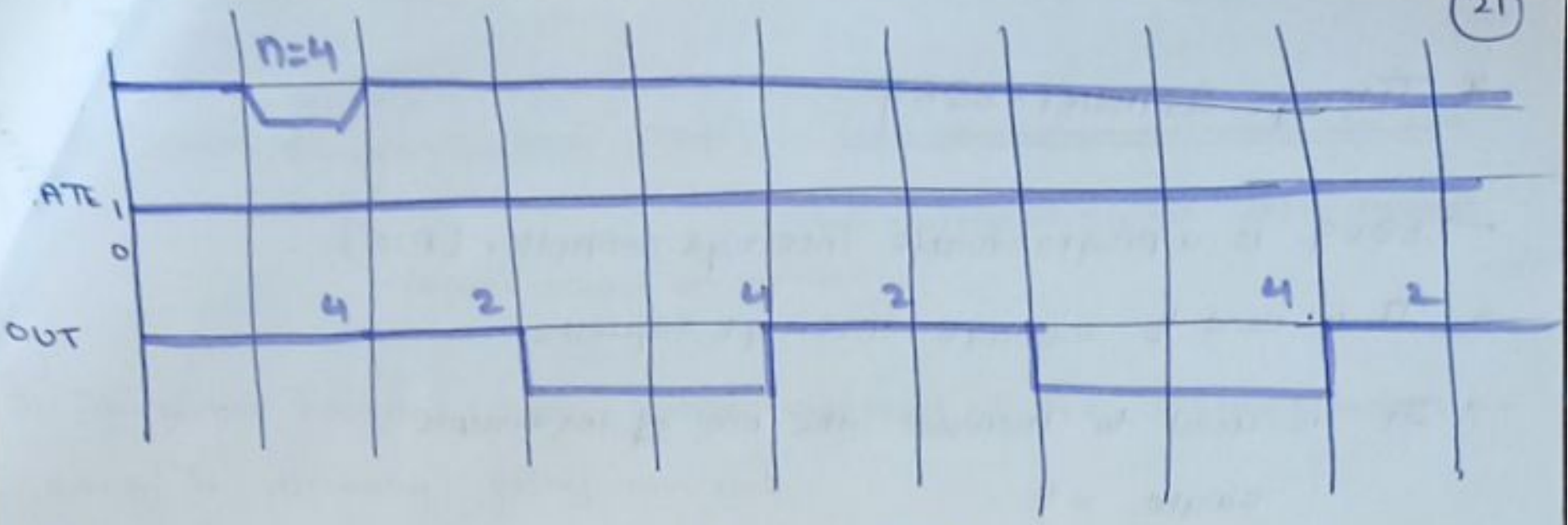

remember count $= \frac{f_i}{f_{out}}$ (n) n = value to be loaded

## Mode 3 - Square wave generator ©

→ Similar to mode 2 except that the output is high for half the period and low for another half

→ If the count is odd - the o/p will be high for $(n+1)/2$
low for $(n-1)/2$

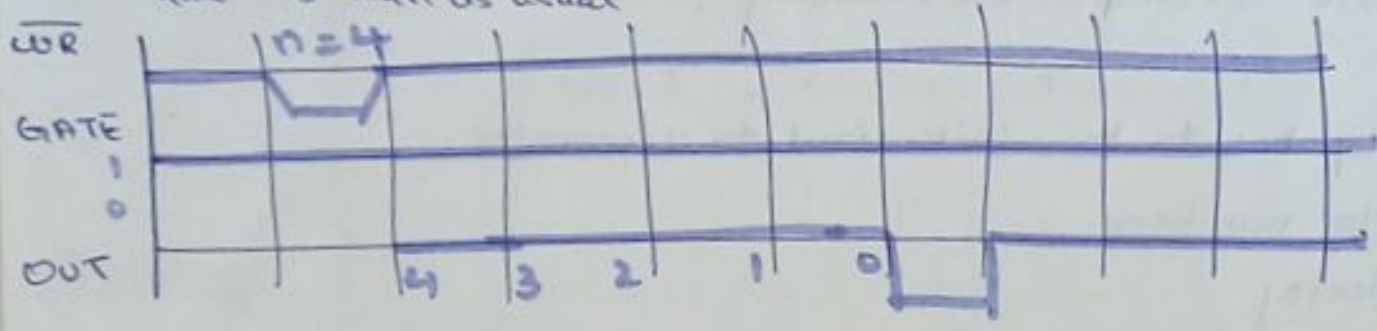
→ Gate is at high throughout




Mode 4 - Software Triggered Strobe (S)

→ Counts down to terminal count and then gives a strobe output, i.e the output goes to logical 0 for one clock pulse, and then goes back to logical 1.

→ gate is high as usual



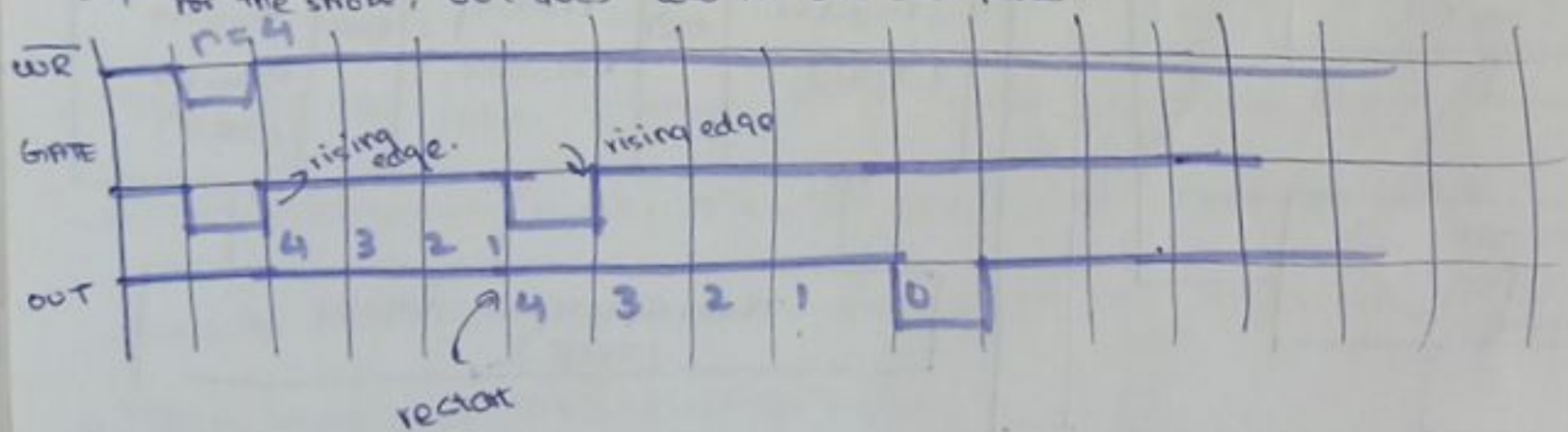

Mode 5 - Hardware Triggered Strobe (S)

→ Similar to mode 1, because the rising edge of the trigger input starts the counting of the counter.

→ called h/w triggered because the gate pin triggers the count

→ It is retriggerable, and count will start again if there is another rising edge.

→ for the strobe, OUT goes low for one clock pulse



# * Interrupt Controller - 8259

→ 8259 is a programmable interrupt controller (PIC).

→ It is used to manage interrupt requests

→ It is used to increase the no. of interrupts

single = 8

cascaded = up to 64

→ It can → handle edge and level triggered interrupts

→ has a flexible priority structure

→ mask interrupts individually.

→ It compulsorily has to be initialized to determine:

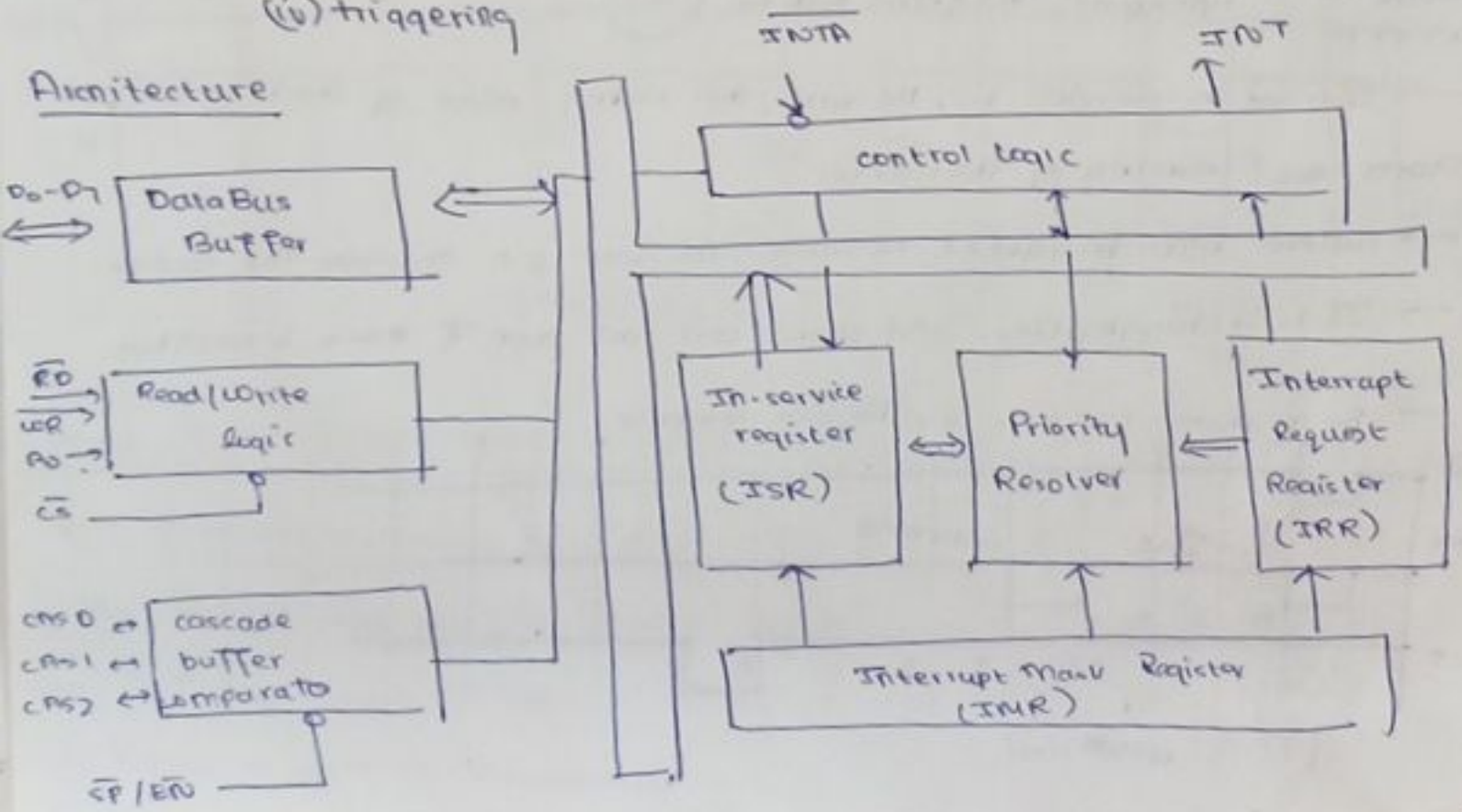
(i) vector numbers

(ii) priority

(iii) masking

(iv) triggering

## Architecture



## Components

1) Interrupt Request Register (IRR) – has 8 interrupt lines $IR_0$-$IR_7$
– set corresponding bit when interrupt request occurs on a line.

In-service register (ISR) – stores the level of the interrupt request which is currently being serviced

3) Interrupt mask register (IMR) – Stores the masking pattern

4) Priority resolver – examines the IRR, ISR and IMR to determine which interrupt is of the highest priority

5) Control logic – INT – send interrupt to µP
$\overline{INTA}$ – interrupt acknowledge

## Working

1. One or more of the interrupt request lines ($IR_0$-$IR_7$) is set
2. The 8259 evaluates these requests and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an $\overline{INTA}$ pulse
4. Upon receiving the $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset
5. The 8086 sends another $\overline{INTA}$ pulse. On this second interrupt acknowledgement cycle, 8259 sends the interrupt vector byte of data to the CPU, which is a pointer of the interrupt to be processed
6. This completes the interrupt cycle.
7. The ISR bit is reset after the third INTA pulse

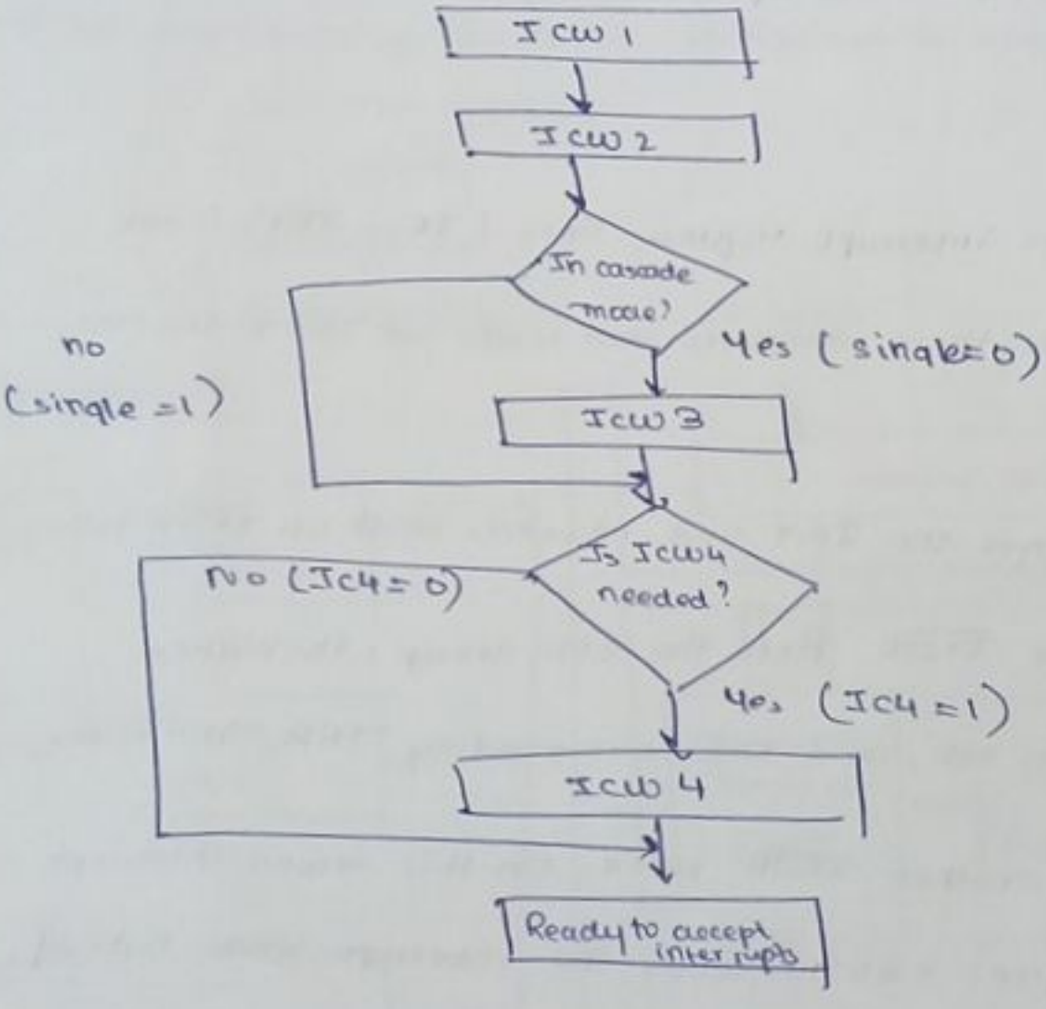
## * 8259 Command Words

There are 2 command words in 8259

① Initialization command words (ICWs) : Before normal operation begin, each 8259A must be brought to a starting point. There are 4 ICWs

② Operation Command Words (OCWs). They command the 8259A to operate in various interrupt modes. There are 3 OCWs (they are optional).

## * 8259 Initialization Sequence



# ICWs

ICW1 = compulsory

| A7 | A6 | A5 | LTIM | 1 | ADI | single | IC4 |
|---|---|---|---|---|---|---|---|
| X | X | X | | | X | | |

1 ⇒ because of 8086

used in 8085

8085

IC4 → is IC4 necessary?

1 = 1

0 = no

must be one

single or cascaded

level triggered

S = 1 - single

C = 0 - cascaded

interrupt mode

1 = L

0 = E

ICW2 = compulsory = vector no. of $IR_0$

| $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|

→ during initialization - it is sufficient to give the vector no. of $IR_0$, the rest are taken in sequence

→ it can be a no. ~~starting~~ ending with 0 or 8, if any no. in between is given - automatically round to 0 or 8

for eg. if $IR_2 = 42$, then $IR_0 = 40$

ICW3 - optional - used when cascaded mode is used.

ICW-3M - | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

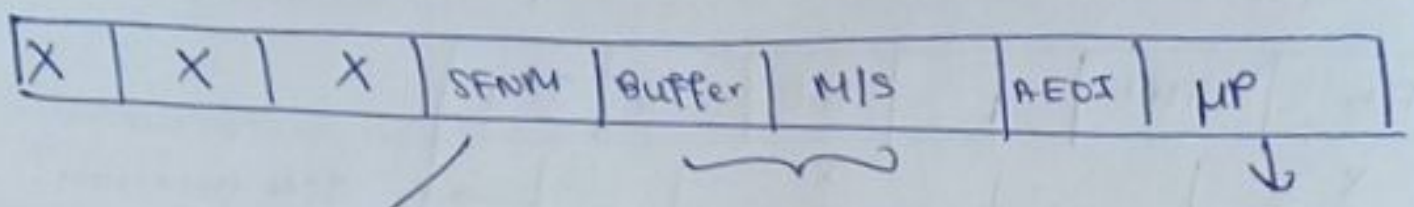
if set to 1 ⇒ slave

0 ⇒ not a slave

ICW-3S - if slave - identify yourself 000 - 007

| X | X | X | X | X | ID | ID | ID |
|---|---|---|---|---|---|---|---|

ICW4 - optional (but must give if 8086)




1 = special fully nested mode

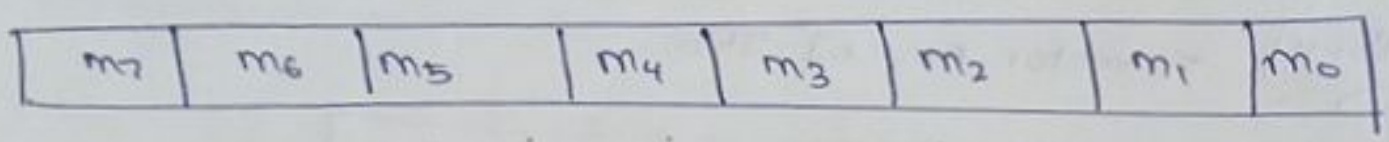
0 = not special fully nested mode

0X - non buffered
10 - buf / slave
11 - buf / master

1 = 8086
0 = 8085

OCWs

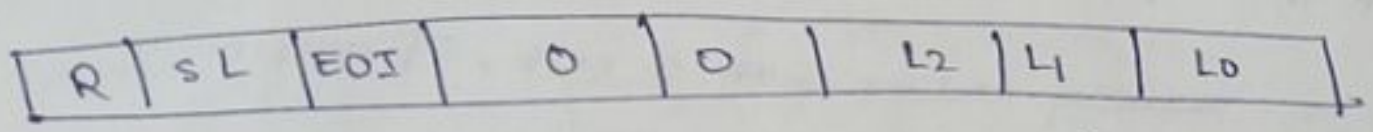
OCW1




1 = mask

0 = no mask

OCW2

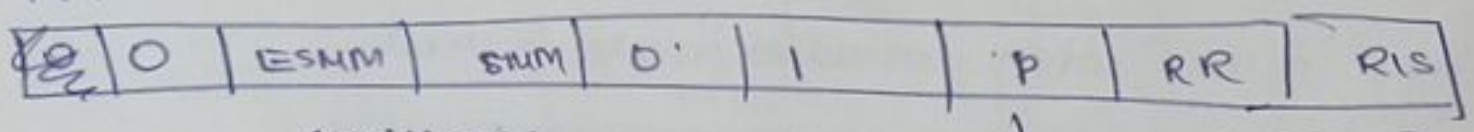



rotation

end of interrupt

000 - 007 - $IR_0$ - $IR_7$

OCW3




special mask mode

00 - X
01 - X
10 - exit SMM
11 - enter SMM

poll

00 - X
01 - X
10 - read IRR
11 - read ISR

| | A0 |
|---|---|
| ICW1 | 0 |
| ICW 2 | 1 |
| ICW 3 | 1 |
| ICW 4 | 1 |
| OCW1 | 1 |
| OCW2 | 0 |
| OCW3 | 0 |

## * 8259 Working Modes

(1) Fully Nested Mode

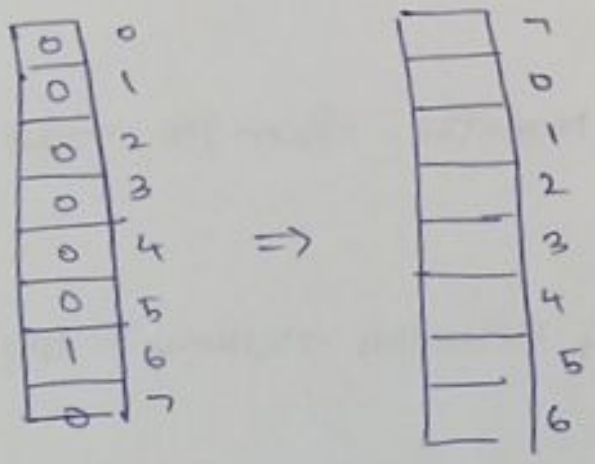
→ default mode - entered after initialization

→ order interrupts from $IR_0$ - $IR_7$

→ write sequence of ops from architecture

(2) Rotating Priority mode

→ After an IR on a particular line is serviced, move it to the bottom, by rotation



③ Special mask mode

→ to dynamically alter system priority structure

→ for eg. may want to inhibit lower priority requests

→ In special mask mode, when a mask bit is set at OCW1, it inhibits further interrupts at that level, and allows interrupts at all other levels

④ Polled mode

→ µP asks 8259. if there any interrupts

→ In response, the 8259 gives back a poll word

→ can be achieved by setting P=1 in OCW3

## * 8279 - Keyboard Display Controller

## * 8237/8257 - DMA Controller

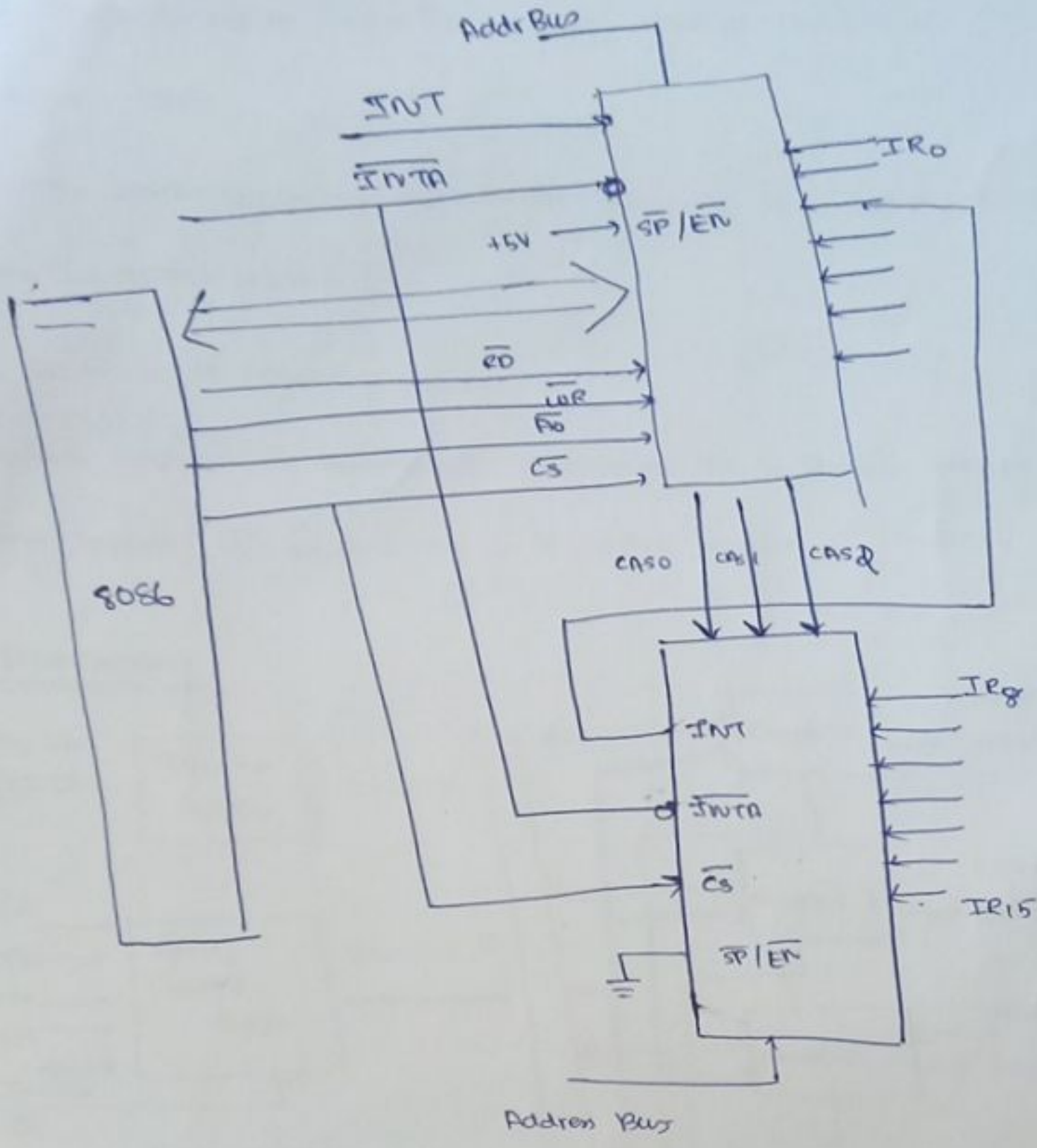
→ DMA is a process by which an external device takes over control of the system bus from the CPU.

→ DMA is used for high-speed data transfer from/to mass storage peripherals.

→ Blocks of data is transferred directly between memory & the peripherals.

→ The data doesn't go through the microprocessor but the data bus is occupied.

# 8259 in Cascaded Mode



## * DMA Controller

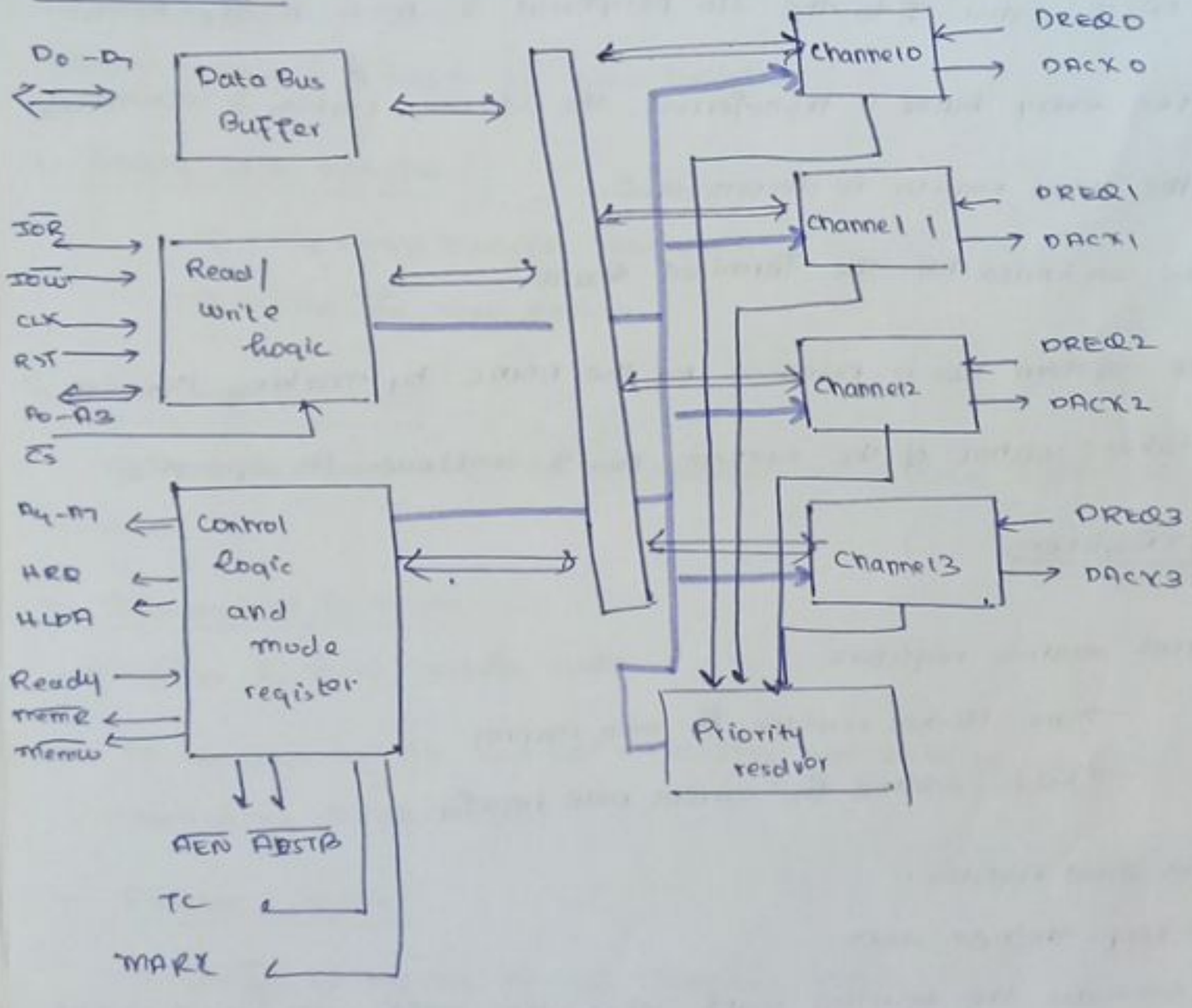
A DMA controller interfaces with several peripherals that may request DMA.

→ The DMA controller used with 8086 is the 8257/8237 programmable device.

→ It is a 4-channel device.

→ Each channel is ~~capable of~~ dedicated to a specific peripheral device and capable of addressing 64K bytes section of memory.

## * Architecture



Working:

1. An I/O device requests the DMAC, to perform DMA transfer, by activating the DREQ line
2. The DMAC in turn sends a request signal to the µP through the HOLD line
3. The µP finishes the current machine cycle & releases the system bus.
4. It acknowledges receiving the HOLD signal through the HLDA line
5. The DMA acquires control of the system bus. The DMAC sends the DACK signal to the I/O peripheral & DMA transfer begins
6. After every byte is transferred, the address register is incremented, and the count register is decremented.
7. This continues till the Terminal Count
8. The system bus is released by the DMAC by making HOLD=0 µP takes control of the system bus & continues its operation

* 8237 Registers:

1. Current address registers
   - → one 16-bit register for each channel
   - → holds address for current DMA transfer
2. Current word register
   - → keeps the byte count
   - → generate the terminal count when the count goes from 0-FFFF signal

3. Command register - used to program 8257

1. Mode register

→ program each channel to:

(i) read or write

(ii) autoincrement / autodecrement the address

(iii) autoinitialize the channel

Request register - for software initiated DMA

Mask register - disable a specific channel

. status register

. Temporary register - used for memory to memory transfer

## Types of Data Transfer

8237 supports 4 types of data transfer

1. Single cycle transfer

→ only single transfer takesplace

→ useful for slow devices

2. Block transfer mode

→ transfers data until TC is generated or external $\overline{EOP}$ signal is received

3. Demand transfer mode

→ similar to block transfer mode

→ In addition to TC and EOP, transfer can be terminated by deactivating DREQ signal

4. Cascade mode

→ useful to expand no. of channels beyond 4.

# Serial Communication Interface - 8251

## * Serial vs. Parallel Data Transfer

| Serial | Parallel |
|---|---|
| → uses a single line data | → uses an n-bit data line |

## * Synchronous vs. Asynchronous

→ Asynchronous transfer does not require a clock signal. However it transfers extra bits (start & stop bit) during data communication

→ Synchronous does not transfer extra bits. However, it requires clock signal.

## * 8251 USART

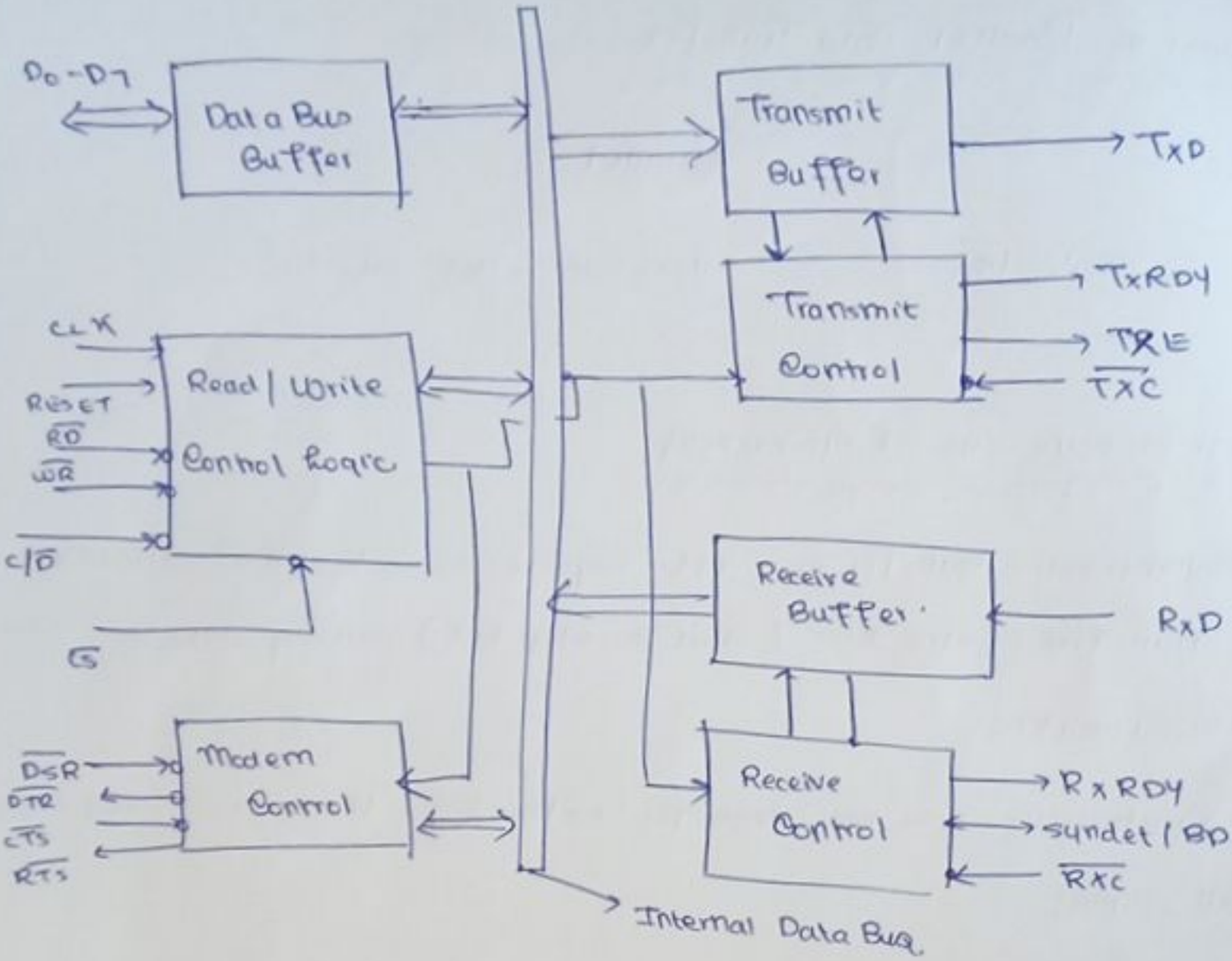
→ The 8251 USART - Universal Synchronous Asynchronous Receiver Transmitter is capable of implementing either an asynchronous or synchronous serial data communication

→ The 8251 can:

(i) receive parallel data from CPU & transmit serial data after conversion

(ii) receives serial data from the outside & transmits parallel data to the CPU after conversion.

# *8251 Architecture




**Data Bus Buffer** - an 8 bit bidirectional data bus, that

(i) gets control words & transmits data from the CPU

(ii) sends status words & receives data from the CPU

**CLK** - internal device timing

**$\overline{WR}$ / $\overline{RD}$** - active low signals for reading & writing

**$\overline{C/D}$** - select whether to send data or command & status words

**$\overline{CS}$** - chip select

## Transmission

TXD - o/p terminal for sending data

TXRDY - indicates that the 8251 is ready to accept a transmitted character

TXEMPTY - indicates all characters have been transmitted

TXC - determines transfer speed of transmitted data.

## Reception

$\overline{RXD}$ - terminal that receives serial data

$\overline{RXRDY}$ - indicates that 8251 has a character that is ready to READ

$\overline{RXC}$ - determines the transfer speed of received data.

(ix) SYNDET/BD - function changes according to mode

(a) In internal synchronous mode

→ set at high if sync characters are received

→ if a status word is read, the terminal will be reset

(b) In external synchronous mode

→ high ⇒ 8251 starts receiving data characters

(c) In Asynchronous mode

→ high → when a break character is detected.

## Modem Control

DSR - i/p port for MODEM interface

DTR - output port for MODEM interface

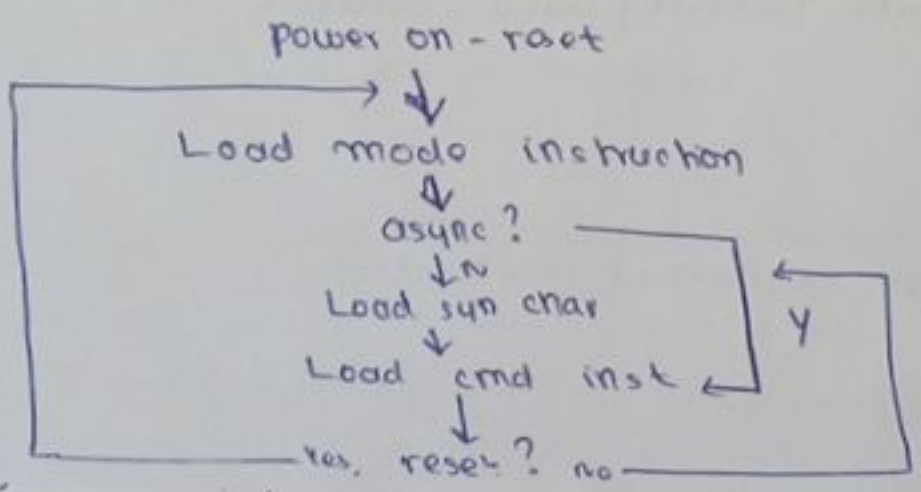
CTS - control a transmit circuit - input terminal

RTS - control reception - output terminal

## * Configuration Modes of 8251

| $\overline{CS}$ | C/$\overline{D}$ | $\overline{RD}$ | $\overline{WR}$ | operation |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | command read<br>⇒ status → CPU |
| 0 | 1 | 1 | 0 | command write<br>⇒ control word ← CPU |
| 0 | 0 | 0 | 1 | data read<br>⇒ CPU ← data. |
| 0 | 0 | 1 | 0 | data write<br>⇒ CPU → data. |

## * Initialization of 8251

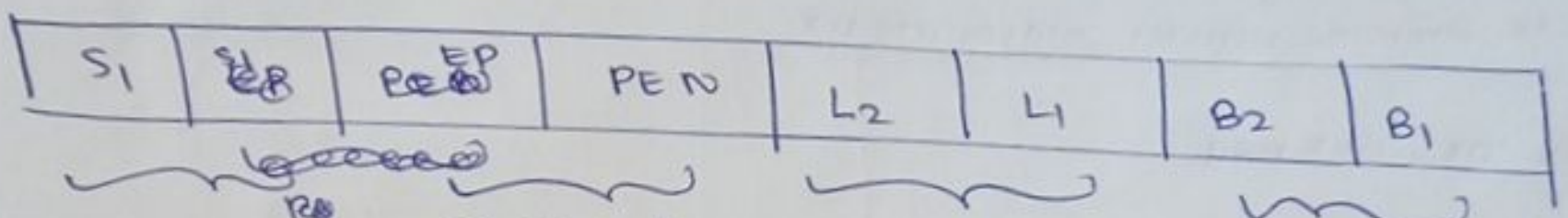


# * Modes of Operation of 8251

1. Mode Instruction - setting a function
2. Command - setting of operation
3. Status word - to see internal states

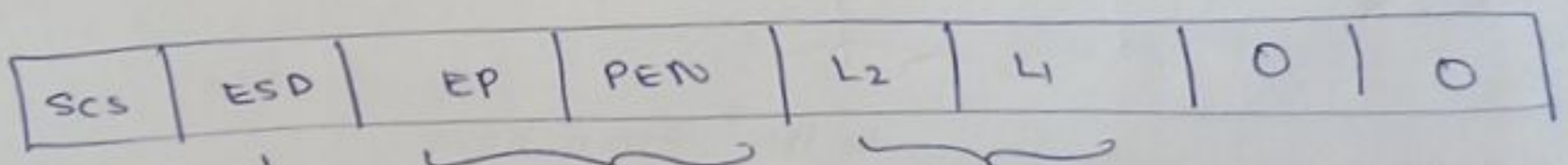
## (1) Mode Instruction

### Asynchronous

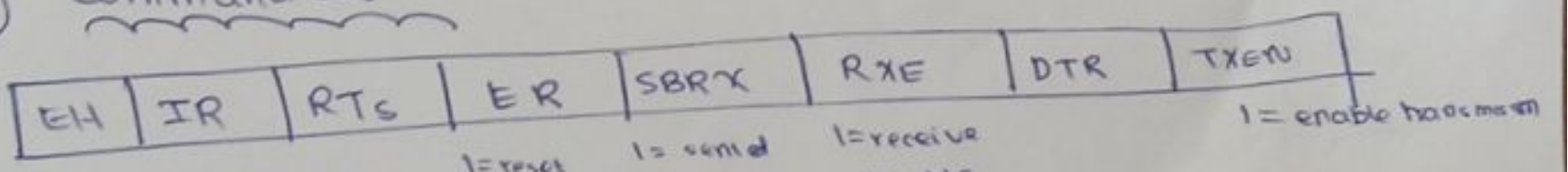



01 - 1 bit
10 - 1.5 bits
11 - 2 bits

### Synchronous




## (2) Command word



③ Status Word

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| DSR | SYNDET | FE | OE | PE | TXEMPTY | RXRDY | TXRDY |

## * 8251 Programming

### A. Initialization of 8251

// move command register address into DX

```
mov DX, 0FFF2H
```

// dummy command to ensure that is used to ensure that the device is in the command instruction format

```
mov AL, 00H
OUT DX, AL
mov CX, 2
```

```
D0: LOOP D0
    OUT DX, AL
    mov CX, 2
D1: LOOP D1
    OUT DX, AL
    mov CX, 2
```

do this 2 more times (delay)

```
// do internal reset

L: LOOP D2
   MOV AL, 40H
   OUT DX, AL
   MOV CX, 2

   //set command word

D3: LOOP D3
   MOV AL, C7        ' (see command word)
   OUT DX, AL
```

B. Transmitting Data using Polling Mode

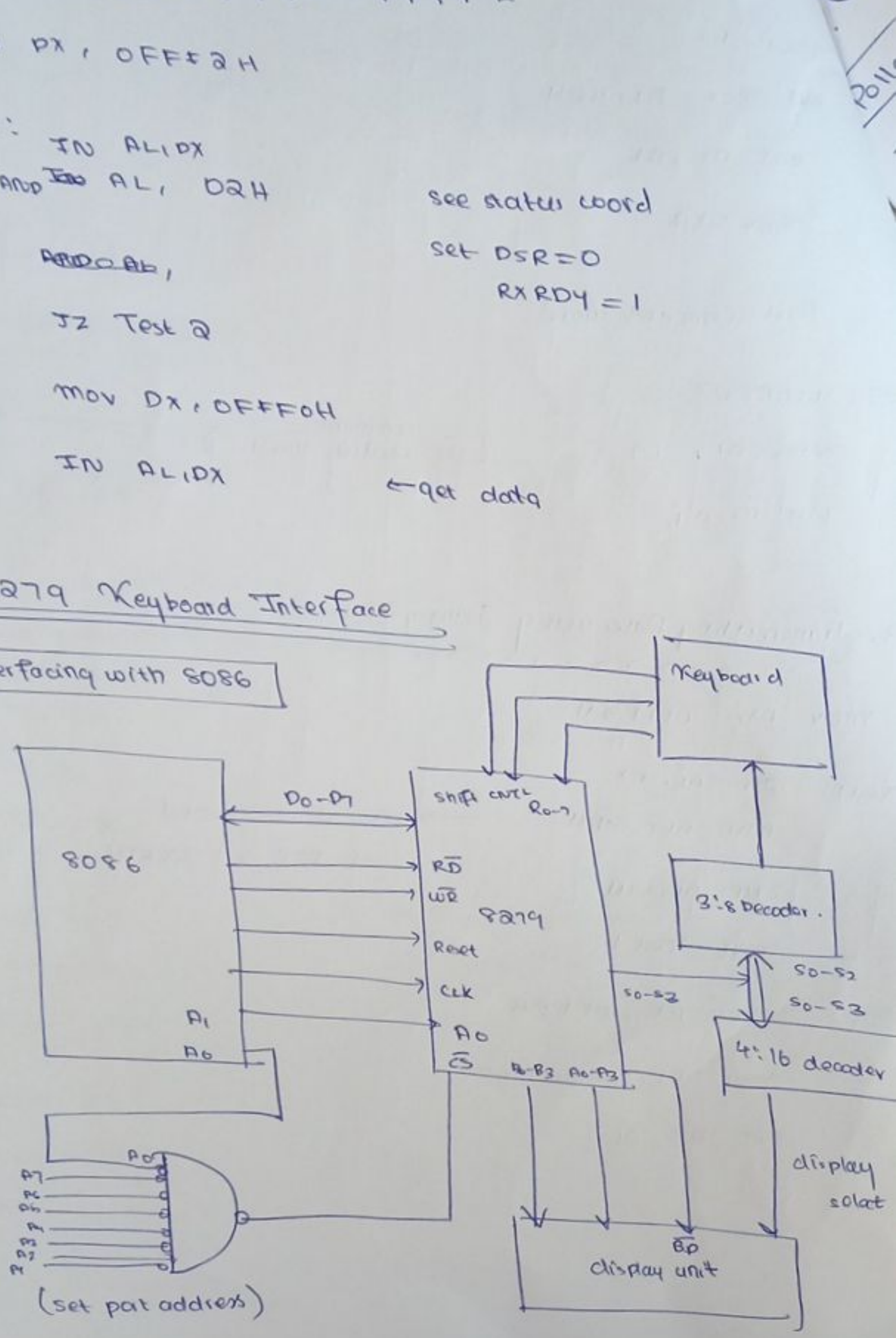
```
   MOV DX, 0FFF3H
Test1: IN AL, DX
   AND AL, 81H      ---> see status word
                         set DSR & TXRDY
   CMP AL, 81H
   JNE Test1:
   MOV DX, 0FF0H
   MOV AL, data to send
   OUT DX, AL
```

c. Receiving Data using Polling mode

```
mov  DX , 0FFF2H

Test2:
     IN   AL,DX
     AND  AL,  02H        see status word

     ~~AND AL,~~          set DSR=0
                          RXRDY = 1
     JZ   Test2

     mov  DX, 0FFF0H

     IN   AL,DX           ← get data
```

## * 8279 Keyboard Interface

Interfacing with 8086



## (07) Polling Mode & Interrupt Mode to Read a Key Press

### Polled Mode

C2 = control register
C0 = data register.

```
Test :  mov BX, 1100   ← reg to store input
        IN AL, C2      ←┐
        TEST AL, 07     │  keep checking for a key press
        JZ Test        ──┘

        mov AL, 40     ← read FIFO RAM
        OUT C2, AL

        IN AL, C0      ← take input & store   C0 = data register
        mov [BX], AL   ← move to register
        HLT
```

D5, D6, D7 used to set mode

| D7 | D6 | D5 | |
|---|---|---|---|
| 0 | 0 | 1 | - clock |
| 0 | 1 | 0 | - read FIFO |
| 0 | 1 | 1 | - read display |
| 1 | 1 | 0 | - clear FIFO |

### Interrupt Mode

```
// Initialization mode.

mov AL, 00
mov [1901], AL

// Interrupt Routine

PUSH SI
PUSH AX
PUSHF

    mov SI, 1901
    mov [1901], 40   → read FIFO
    DEC SI

    mov AL, [SI]
    AND AL, 3F
    mov [2500], AL
```

# Question Paper Solutions

## Memory Interfacing

CAT-2 – Q8

8. Construct an interface with 2 chips of 32K x 8 ROM and 4 chips of 32K x8 RAM with the following address mapping

| | |
|---|---|
| ROM 1 & 2 | F0000 – FFFFF |
| RAM 1 & 2 | D0000 – DFFFF |
| RAM 3 & 4 | E0000 – EFFFF |

**Ans** **Step1** – find no. of address lines

ROM – size $= 2 \times 32 = 64K = 2^6 \times 2^{10} = 2^{16}$

$= 16$ address lines

RAM – size $= 2 \times 32 = 64K = 2^6 \times 2^{10} = 2^{16}$

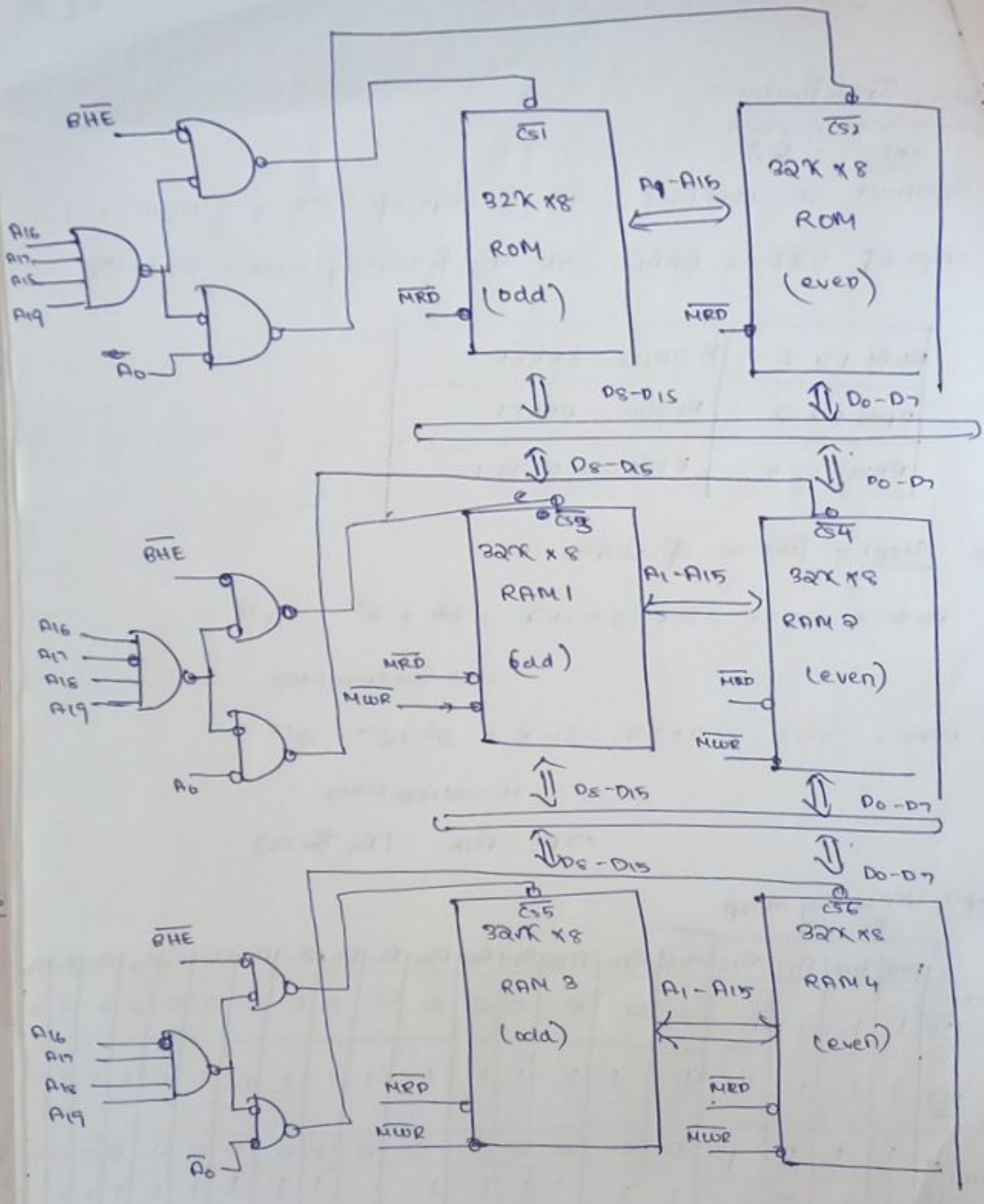
$= 16$ address lines

$\Rightarrow A_0 - A_{15}$ ($A_0$ for CS)

**Step2** Memory map

| | A19 | A18 | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROM ① F0000 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ② | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RAM ① st | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ② end | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ③ st | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ④ end | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

for CS


BHE
A16
A17
A18
A19
A0
CS1
32K x 8
ROM
(odd)
MRD
A1-A15
CS2
32K x 8
ROM
(even)
MRD
D8-D15
D0-D7
D8-D15
D0-D7
CS3
32K x 8
RAM 1
(odd)
MRD
MWR
A1-A15
CS4
32K x 8
RAM 2
(even)
MRD
MWR
D8-D15
D0-D7
D8-D15
D0-D7
CS5
32K x 8
RAM 3
(odd)
MRD
MWR
A1-A15
CS6
32K x 8
RAM 4
(even)
MRD
MWR

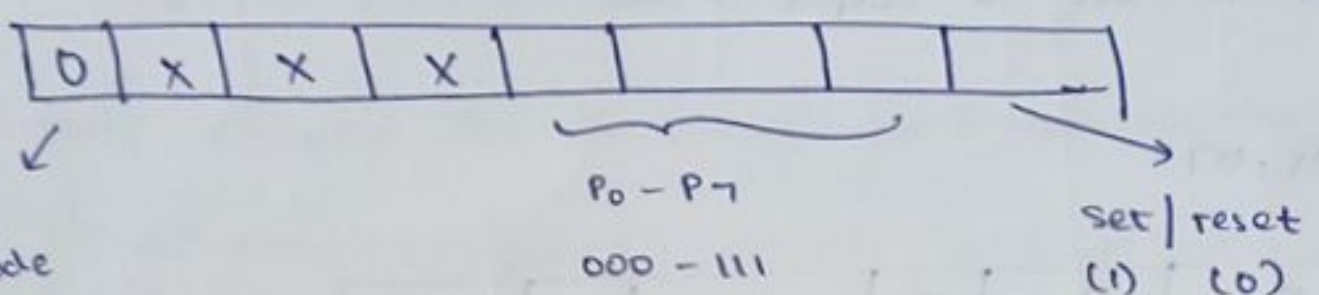
# 8255 Questions

## CAT-2 Q2

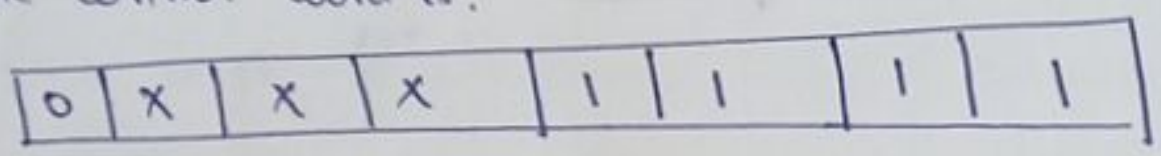
Identify the 8255 BSR mode control word for setting PC7.

Ans.

Format




The control word is:

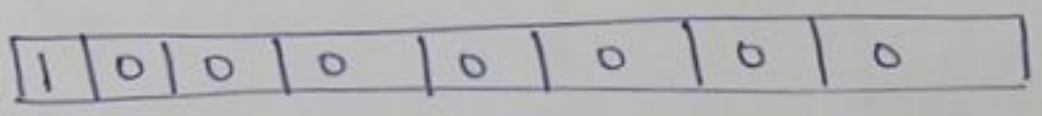



## ② CAT-2 Q5

Identify the status of PORT C in 8255 after executing the following 8086 code snippet.

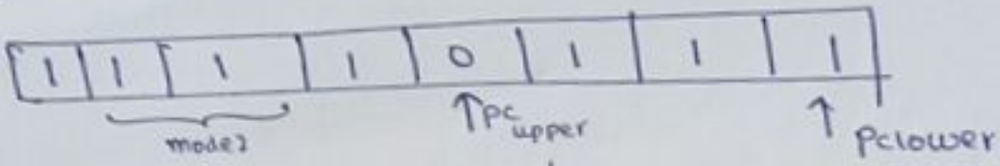
```
mov AL, 80
OUT control-req-addr-of-8255, AL
mov AL, 0F7
OUT port-C-addr-of-8255, AL
mov AL, 07
OUT control-req-addr-of-8255, AL
```

Ans Line1 : mov AL, 80




Initialize, add I/O control word

Line 2: mov AL, 0F7



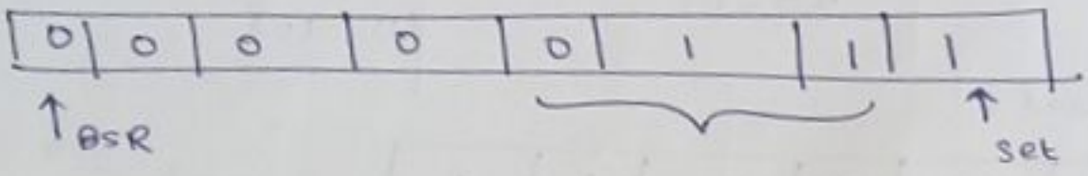

Initialize ~~port~~ I/O control word

set to mode 2

port C upper is set to ~~input~~ output mode

port C lower is set to output mode.

Line 3: mov AL, 07




set to BSR mode

set PC3

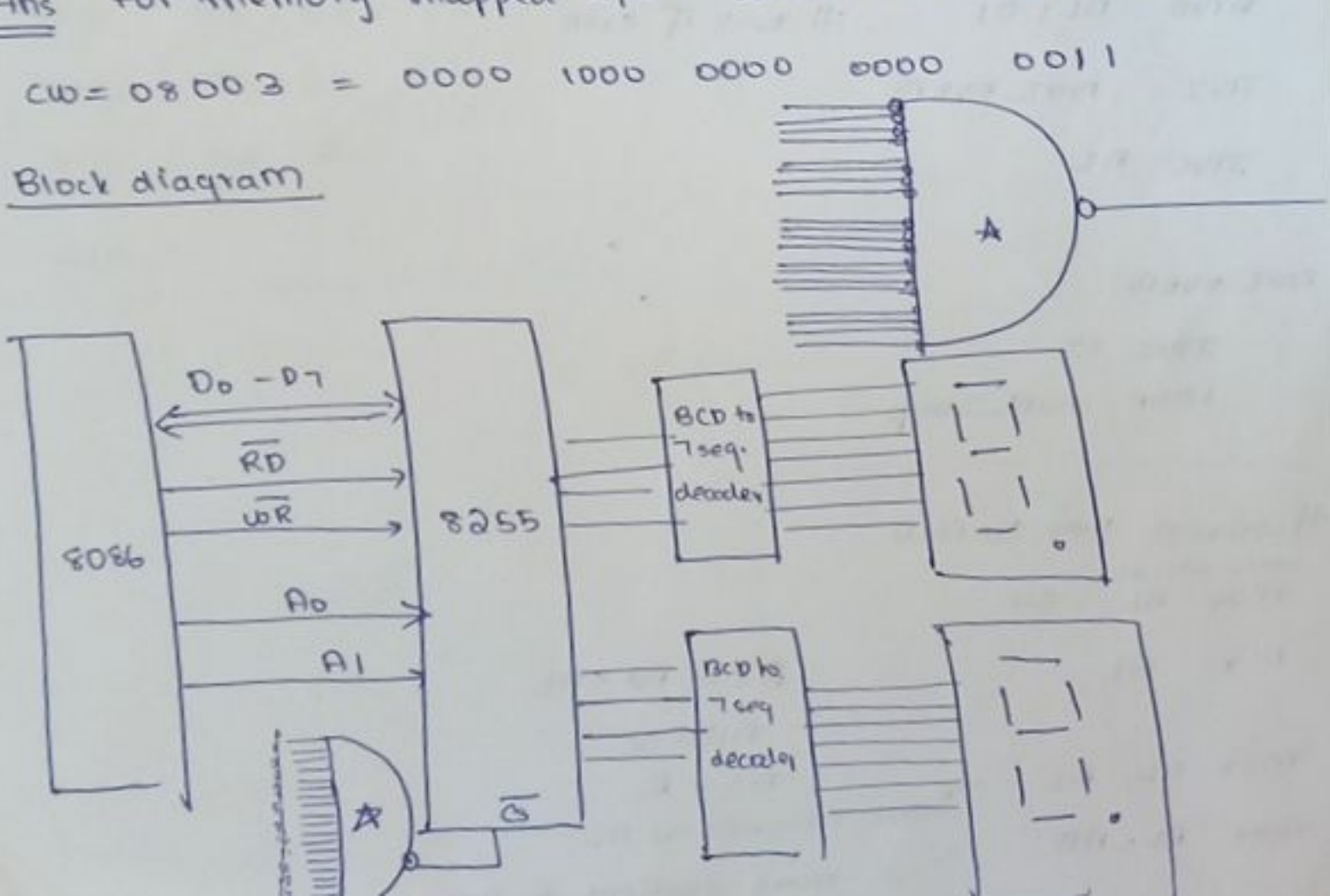
# 8255 Interfacing Questions

Construct a system that interfaces 8255 with 8086 to display the count of even numbers in BCD format after examining an array of 16 eight bit numbers available in memory to seven segment displays connected to port A and port B. Use memory mapped I/O for interfacing & assume the port addresses as PortA = 08000, portB = 08001, port C = 08002, Control reg = 08003, PS = 0000. Explain the system design along with 8086 ALP and explain the changes required if the solution demands I/O mapped I/O

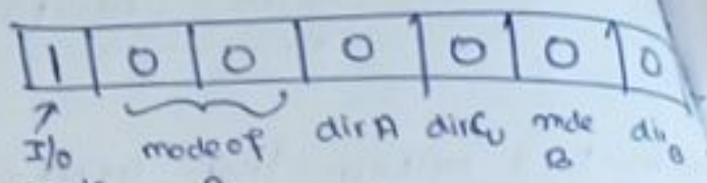
Ans For memory mapped I/O - control word = 20 bits

CW = 08003 = 0000 1000 0000 0000 0011

Block diagram



## Assembly Program




1 = i/p
0 = o/p

= 80H

# Initialize the control register

```
mov DX, 0803H
mov AL, 80H
mov
OUT DX, AL
```

# Initialize an array of 8 bit numbers

```
mov SI, 1000H
mov CX, 16
mov AL, 0        #set counter

count_loop:
   mov DL, [SI]
   AND DL, 01    #check if even
   JNZ NOT_EVEN
   INC AL

NOT_EVEN:
   INC SI
   LOOP count_loop
```

---

# convert Hex to BCD

```
mov AH, 00.
MOV BL, 64
DIV BL          AX ← AX ÷ BL
                AH = Q
                AL = R
mov DL, AL  → move remainder to DL
mov AL, AH  → move quotient to AH
```

```
MOV AH, 00
MOV BL, 0A
DIV BL
```

AH , AL
0000 XXXX

~~AH~~ ~~AL~~
remainder quotient

```
MOV CL, 04
ROR AL, CL
```

swap move around by 4

~~BH~~ ~~AH~~

# moving results to output

```
MOV DX, 08000H
MOV BH, AH
OUT DX, BH

MOV DX, 08001H
MOV BL, AL
OUT DX, BL

HLT
```

Q2. Construct a system that interfaces 8255 w/ 8086 to display the counting sequence in BCD using 7 segment LED displays connected to port A and B. The system should act as an up counter when the switch connected to PC7 is 0 & as a down counter when it is logic1. The counter counts from $(00)_{10}$ to $(99)_{10}$. Use I/O mapped I/O.

Port addresses: A = 8000
B = 8001
C = 8002

Control reg = 8003.

Explain the design w/ 8086 ALP & explain changes if mem-mapped I/O

Ans

```
mov DX, 8003H
IN  AL, 80H
OUT DX, AL

mov SI, 0000H      } Initialize count
mov CH, 0099H
mov DL, 0000H

REP
Count_loop
   mov DX, 8002H
   IN  AL, DX
   AND AL, 40H
   JNZ down_counter:
```

up-counter: call output

```
INC SI

CMP SI, DH

JAE reset
```

down-counter:

call output

```
DEC SI
CMP SI, DL
JB reset
```

reset:

```
MOVE SI, PL
JMP count_loop
```

output:

```
mov AH, 00
mov AL, SI

mov BL, 64
DIV BL

mov DL, AL
mov AL, AH

mov AH, 00
mov BL, 0A
DIV BL

mov CL, 04
ROR AL, CL
```

#output

```
mov  DX, 8000H
mov  BH, AH
OUT  DX, BH

mov  DX, 8001H
mov  BL, AL
OUT  DX, BL

HLT
```