Module 3

Stacks in 8086

Interrupts

Interface

Stacks in 8086

Out of 16 segment 1 segment is treated as stack segment

Base address is in SS. offset in SP

SS : SP 5000 : 2500

This address points to the top of the stack

SS : SP 5000 : 2050

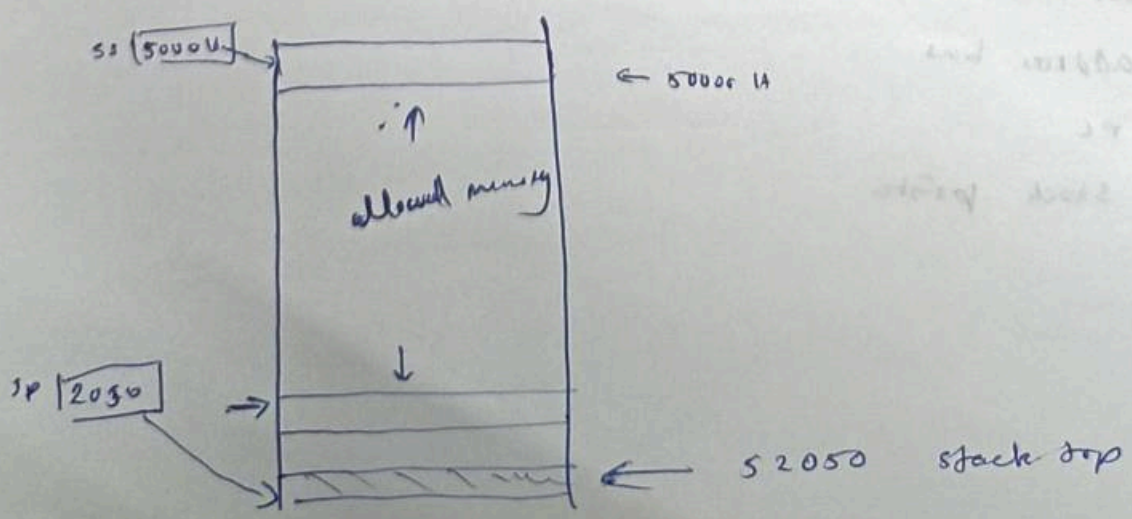
52050 — top

while pushing eg : PUSH AX

It decrement the top by 1 (2 in case of 16 bit) and push the value

So push the

i) 1st decrement SP by 2 ie 2048



ASSUME SS: STACK

STACK SEGMENT

ST DB 100H dup(?) ; initializes sp value

STACK ENDS

CODE SEGMENT

MOV AX, DATA // base address is given by DATA

MOV DS, AX

MOV AX, STACK

MOV SS, AX

MOV SP, OFFSET ST

CALL ADDA ; processor itself move content of IP is stack and ; after executing procedure IP value is restored from the stack

Far and near procedure

function is written inside CODE segment then it is known as NEAR

CS:IP pair point to each instruction

Here only IP is incremented

So only IP is pushed into stack

CODE SEGM
:
PROC NAME
:
ENDP
ENDS

PROCEDURE SEGMENT ; new segment is initialized

For Procedure

```
CODE SEGMENT
    ASSUME CS: CODE
    CALL ADD
      :
      :
CODE ENDS

PROCEDURE2 SEGMENT          ; here we are initializing new
    ASSUME CS PROCEDURE1      segment. Here the fun definition
      :                       is written so calling this
PROCEDURE2 ENDS               requires pushing CS and IP to
                              stack
```

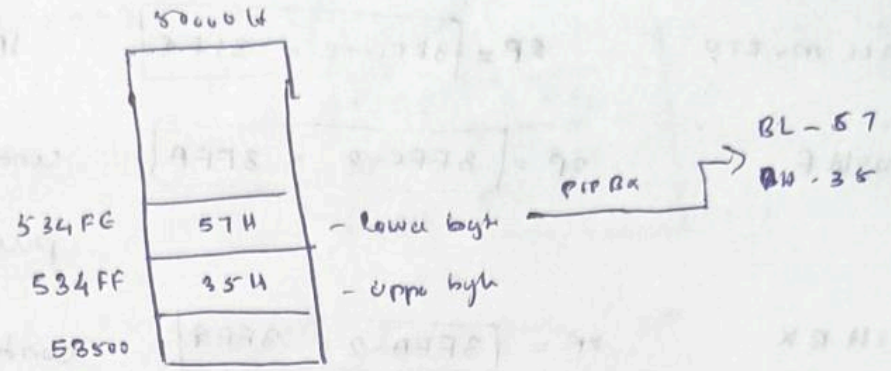
While using function we have to define stack segment

- Top down data structures whose elements are accessed using a pointer that is implemented using SS and SP
- while pushing decremented by 2
  poping incremented by 2
- Stack is required in case of CALL instruction

SS - 5000 H

SP - 8500 H

AX - 8557 H



1st decrement by 2 then value only value is added

- Use concept of stack in 8086 and show the effect of each of the following instructions on the stack pointer and on the contents of the stack

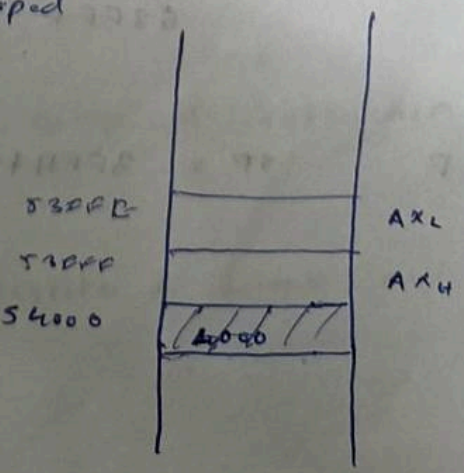
```
MOV  SP, 4000 H
i   PUSH AX
ii  CALL MULTO          ; IP is pushed to stack
    POP AX
MULTO  PROC  NEAR
       PUSHF
       PUSH BX
       :
       POP BX
       POPF
       RET              ; top of stack is poped
MULTO  ENDP
```

ans

a. SS : 5000

3FFE

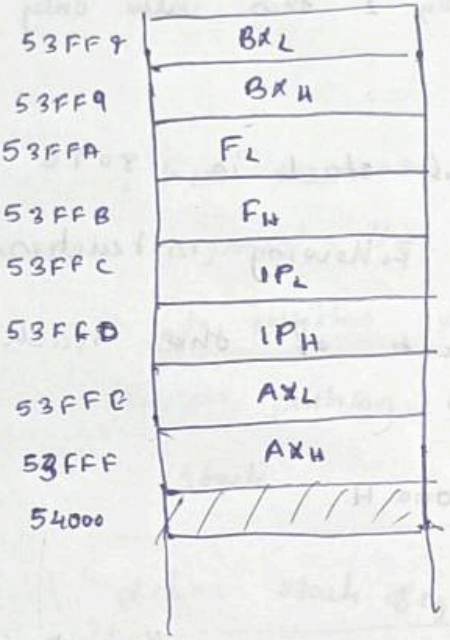
PUSH AX SP = 4000 - 2 = 3FFE



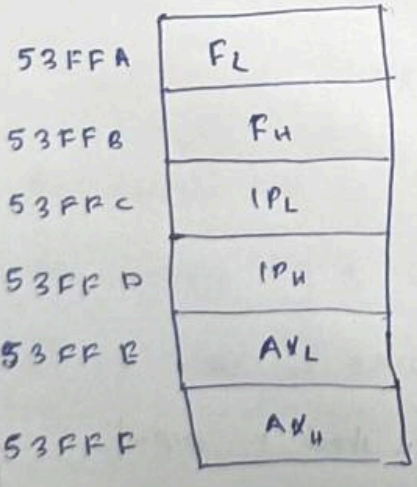
CALL MUL TO SP = 3FFE-2 = 3FFC IP is pushed to stack

PUSH F SP = 3FFC-2 = 3FFA contents of flag register is pushed to stack

PUSH BX SP = 3FFA-2 = 3FF8 contents of BX is pushed to stack

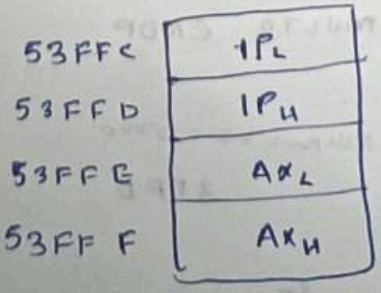


POP BX SP = 3FF8+2 = 3FFA contents of BX is popped to stack and

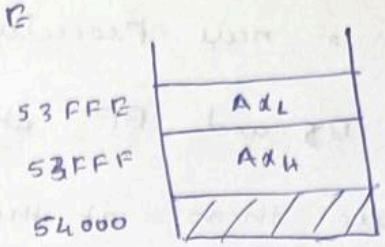


F_L - flag register lower bit

POP F SP = 3FFA+2 = 3FFC



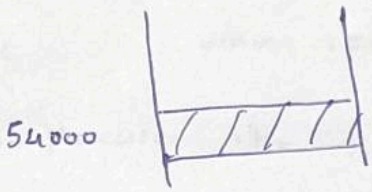
RET SP = 3FFE + 2 = 3FFF



now the content of stack is puped into IP

now ~~go back to~~ return back main from procedure to main program

POP AX SP = 3FFE + 2 = 4000



b) What effect would it have on the execution of the program if the POPF instruction in the procedure was accidently left out

ans During the RET instruction instead of poping the actual address of the instruction, it will pop the flag contents. These flag contents will be puped to the IP. So it will not get the actual address after the call instructions. It will execute from that address

c) Show the 8086 instruction or group of instruction which will

i) initialize stack segment register to 4000 H and SP to 8000 H

ii) CALL a near procedure named FIXIT

iii) save BX and BP at the start of a procedure and restore them at the end of procedure

Answers

i)
```
MOV AX, 4000
MOV SS, AX
MOV SP, 8000
```

ii)
```
CALL FIXIT
```

iii)
```
Sample PROC NEAR
        PUSH BX
        PUSH BP
        :
        POP BP
        POP BX
        RET
Sample ENDP
```

Interrupts in 8086

Anything that alters the normal flow of pgm

Types

i External / hardware interrupts

INTR - Interrupt request
NMI
} pins for external interrupts

INTR - maskable (Interrupt flag is used for this purpose when IF=1 processor accept interrupt o neglects)

NMI - non maskable

when NMI gets the signal processor stop the executing after completing that instruction and servica the interrupt

- There are 256 external interrupts
- Programmable interrupt controller (PIC) act as b/w micro processor and external devices. If multiple interrupts comes at same time PIC assign the interrupt with high priority

$\overline{INTA}$ - interrupt acknowledgement

ii) Software interrupts

It occurs through PIN

eg: INT 21H

This are raised to do particular task

iii) Interrupt by Exceptions

Interrupt due to error in the pgm

eg: division by zero etc

Steps taken by processor when an interrupt occurs

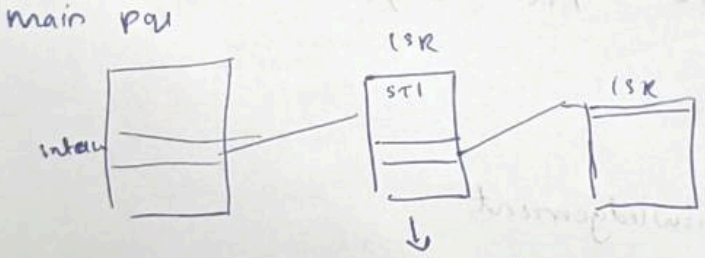
when an interrupt occurs IP and CS is pushed to stack

It is FAR Jump.

- Flag register is also pushed, because ISR may affect the flag
- Then CS, IP is pushed
- Reset IF, IF=0 (interrupt are disabled)

If the ISR allows other interrupt to occur then

1st in ISR STI set IF



content of flag, CS IP are again pushed

- Reset Trap flag (done by microprocessor)

when TF=1 (debugger will enters single step execution)

we can see content register after single step)

when TF=1 we need to push content of flag to stack each time

when a line is executed

so trap flag is reset

~~we cannot ch~~

- there is no direct instruction for setting trap flag

  push flag register, then pop into another register

```
PUSHF          00000000 00000000      0000 0001 0000 0000
                                      0   R   D   0
POP AX         0001 0001
MOV AX, 0010
PUSH AX
POP F
```

- Calculate address of ISR

  we need to find base address and offset

There are 256 interrupt (0-255)

All the interrupts base address and offset

(16 bit 2byte) (16bit 2byte)

256 × 4 = 1024

Lowest 1024 bytes are reserved for storing ~~table~~ interrupt vector table

How 8086 response to interrupt

When an interrupt occurs

- 1st ~~the~~ flag contents of flag register is pushed to stack
- then ~~contents~~ contents of CS and IP is pushed to stack
- ~~IF~~ IF is reset
- TF is reset

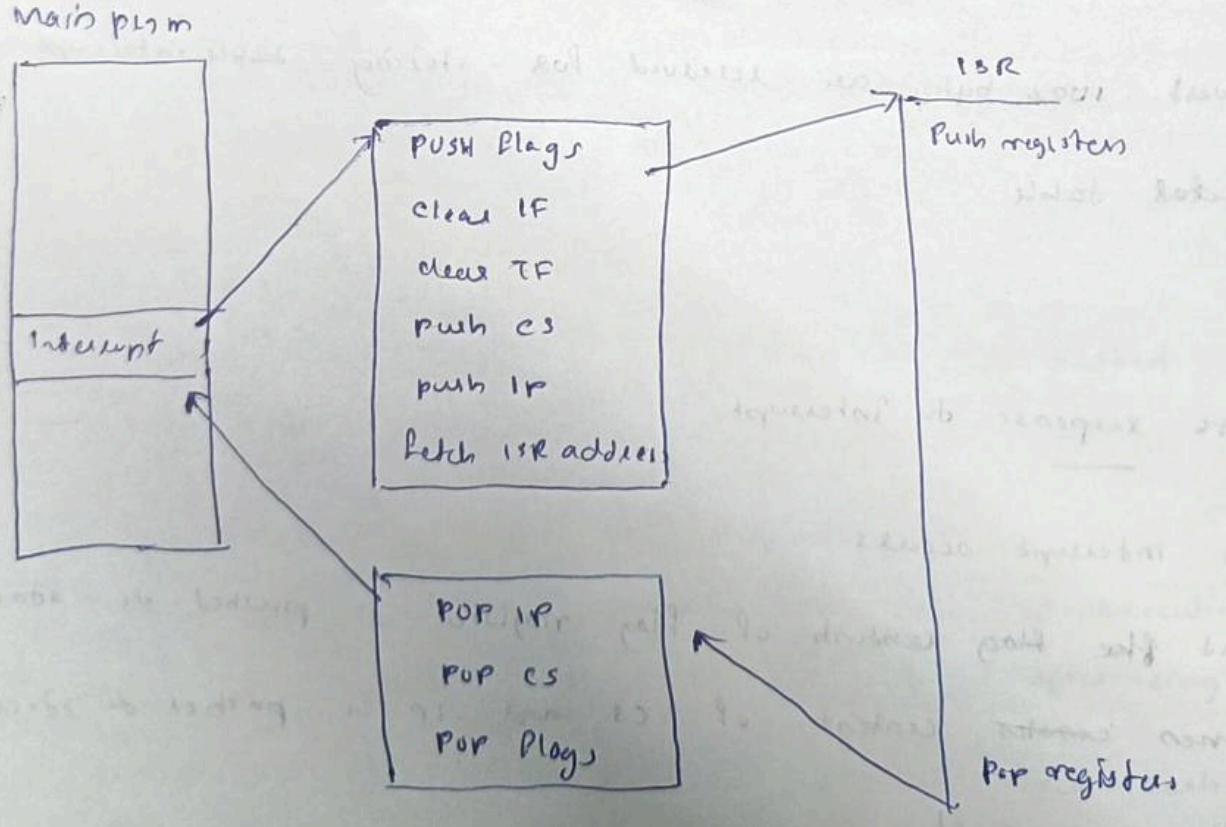
- If we need to allow other interrupts to occur we can set the IF
- If we need to set TF to allow single step execution, If have to set TF manually by ~~pushing~~ changing contents of Flag register
- Then we have to calculate the address of ISR

For this we need base address and offset

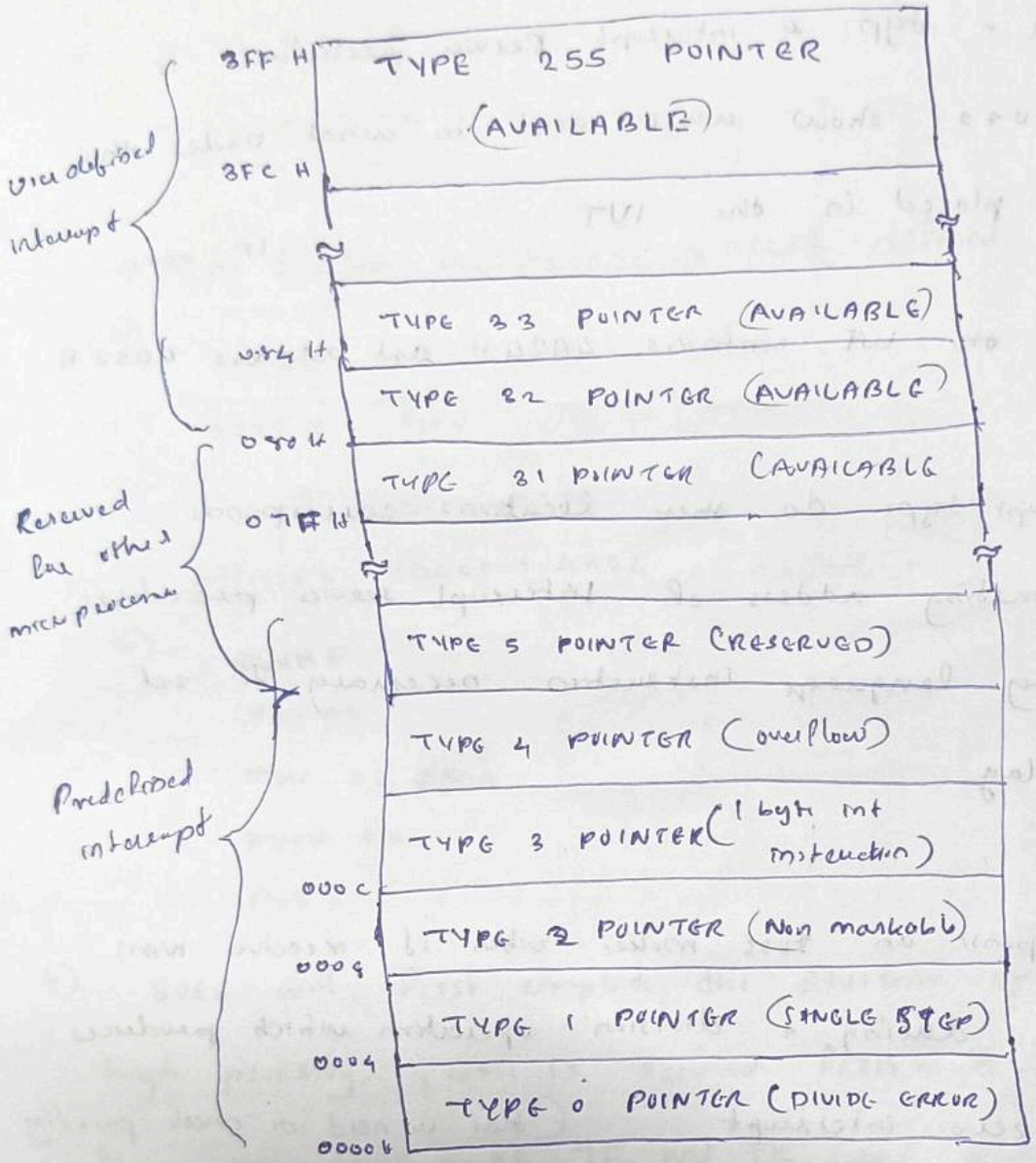
All the interrupt base address and offset is stored in interrupt vector table.

256 × 4 = 1024

the lowest 1024 bytes are reserved for interrupt vector table



Format of interrupt vector table



INT 0 → instruction to jmp to overflow interrupt

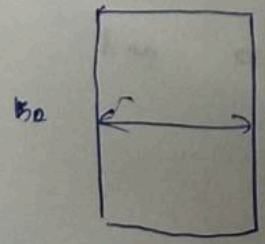
— breakpoint interrupt

00, 01 } FF

02, 03 } E5

When an interrupt occurs, its type is known by microprocessor

type * 4 to get the address in interrupt vector table



} to execute as a whole

after this line line by line execution is possible

used for debugging

1) What addresses in the interrupt vector table (IVT) are used for a type 2 interrupt

2) Starting address for a type 4 interrupt service procedure is 0010 : 0042. Show where and in what order this address should be placed in the IVT

IP : 0042

3) Address 0080H in the IVT contains 4A24H and address 0082H contains 0040H

i) to what interrupt type do these locations corresponds

ii) what is the starting address of interrupt service procedure

iii) Show assembly language instruction necessary to set 8086 Trap flag

iv)

4) Describe the response an 8086 makes when it receives NMI interrupt signal during a division operation which produces a divide by zero interrupt (Here we need to check priority of interrupt)

Answers

1) $2 \times 4 = 0008h$

0008h - IP

000Ah - CS

2) lower byte address is used ~~not~~ for storing IP and upper byte for storing segment address

$4 \times 4 = 16$

0010 H : 42

0011 H : 00

0012 H : 10

0013 H : 00

3)

i) these location corresponds to user defined interrupts

ii)

0040H - 4A24 it is offset

0042 H - 0040 it is segment

0040 : 4A24

address = 00400 + 4A24 = 04E24 H

00400 +
4A24
4E24

iii)

```
PUSH R
POP AX
MOV AX, 0100
PUSH AX
POPF
```

4) 8086 will first complete the division operation. As NMI has high priority, NMI is serviced first. The 8086 will push the flags on stack, reset TF and IF and push CS and IP value to the stack. It get CS value for type 2 from 0008 and IP from 0006

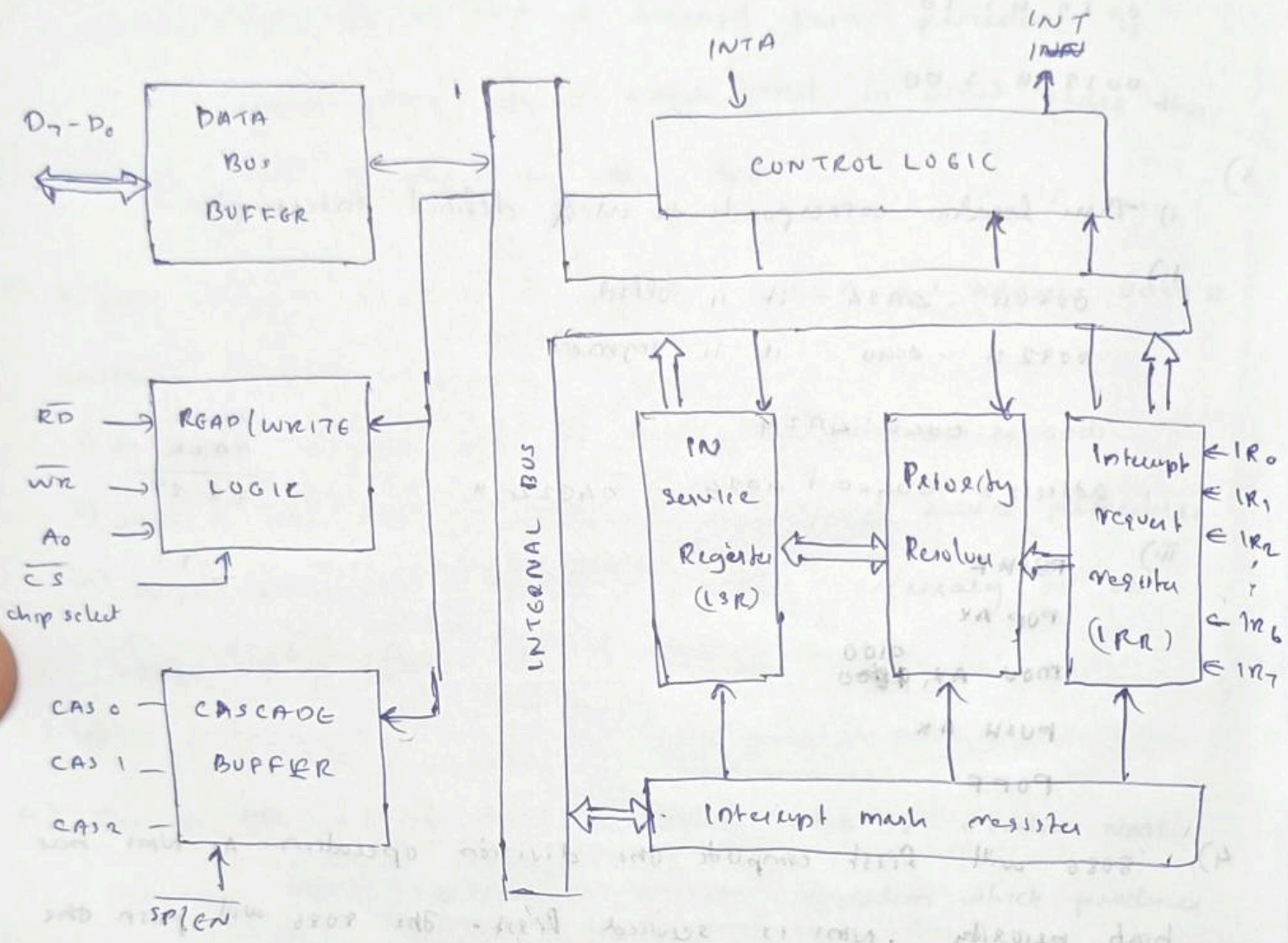
After completing the NMI ISR, the processor will return and handle divide by zero interrupt

IMP

⇒ group of data

→ single data

8259A Programmable Interrupt Controller

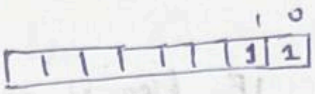


Sometimes 8086 can be connected to many external devices But there is only 2 pins in 8086 for interrupt.

8259A acts as interface b/w 8086 and external devices Main fn of this is to select one of the interrupt from available devices

One 8259A PIC can handle 8 interrupt.

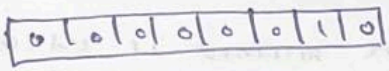
They are connected to IRR of size 8 bit.



if IR0 and IR1 are set bit 0 and 1 are set

If many request comes at a time, priority resolver look on to IRR to check which are requesting service and select one of these request

- There many priority modes, one is fixed priority mode
  In this IR0 has highest priority and IR7 has lowest
- There is rotating priority mode, here in each cycle the highest priority changes
- Priority resolver look on to interrupt mask register (size 8 bit)

ex:-  It mean IR1 cannot be processed at this time

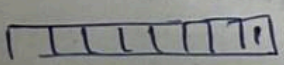
If all are unmask (full 0) then Priority resolver look s on ISR (8 bit)

INT - interrupt request signal. It is connected to INTR of 8086

~~When~~ 1st clk cycle of $\overline{INTA}$:- type of interrupt is requested (8 bit data)

~~2nd cycle~~ :-

The type of interrupt is passed through Data bus buffer Then goes push Flag register

IR0 is in service register  at this time

IRR(0) is cleared. when IR1 comes, now high priority interrupt is in service. So it have to wait

When IR0 service is finished, 8086 sends control signal, then ISR is cleared.

Then process IR1

Now when IR0 comes, it look for IF flag is set, else it performs IR0, now in ISR bit 1 and 0 are set. after servicing IR0, it return to IR1 and service that

A0 :- 8259 has 2 internal address FFF0, FFF2 (A1: 0000, 0010)

8086 A1 is connected to A0 of 8259

Cascade buffer

It is used to connect multiple 8259A to 8086 system.

If many 8259A are present master slave method is used

If slaves request pin gets request it is forwarded to master

CAS0, CAS1, CAS2 are used for selecting the master or slave. There can be total 8 ~~devices~~ PIC on master and 7 slave

$\overline{SP/EN}$:- slave program enable

Sample question:

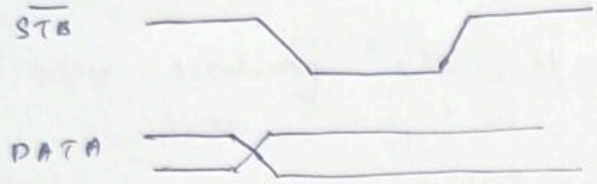
- Describe the sequence of action that an 8259 and 8086 will take when 8259 receive an interrupt signal on its IR_2 input. Assume IR_2 is unmasked in 8259 and 8086 INTR input has been enabled by STI instruction

. Describe the response that an 8259 will make if it receives an interrupt signal on IR_3 and IR_5 inputs at same time. Assume fixed priority for IR inputs. What response will be 8259 make if it is servicing an IR_5 input interrupt and an IR_3 interrupt occurs

Module 4

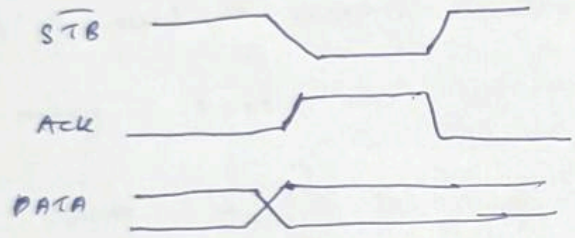
External device interfacing

Strobe signal ($\overline{STB}$) for transferring data.

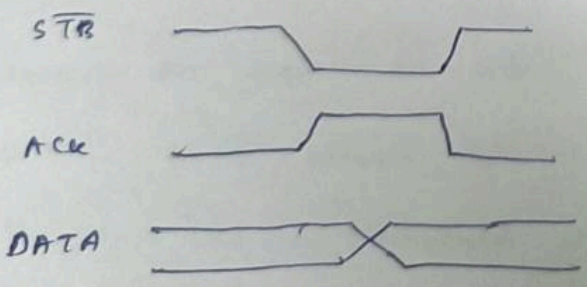


When $\overline{STB}$ is at low, at that time DATA is also send

Single handshake I/O

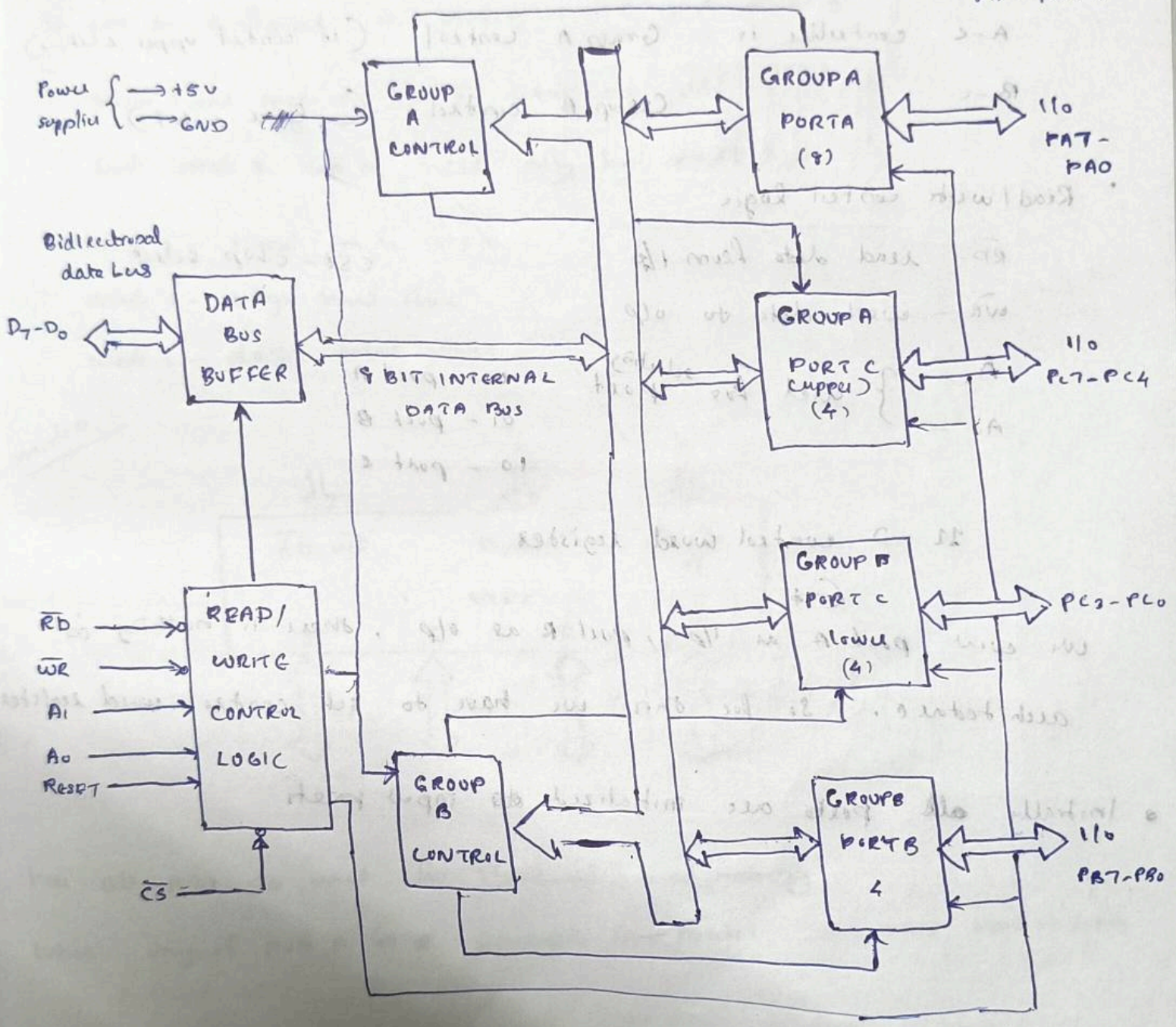


Here there is an Acknowledgement. But when $\overline{STB}$ is set low Data is send at the same. If the ~~only~~ device is ready to accept Data Ack is set



8255 PPI (Programmable Peripheral Interface)

It is an interface for connecting 8086 with I/O ports



When there are 3 ports

8255 has 3 ports. Each are 8 bit

there are Port A, Port B and port C.

They can be used as I/p as well o/p port

- 4 bit of port c can be used with A and B for 12 bit data transfer
- There exist 2 controller for A-c and Bc

  A-c controller is Group A control (it control upper 4 bits)

  B-c → Group B control (lower 4 bit)

- Read / write control logic

  $\overline{RD}$ - read data from i/p

  $\overline{WR}$ - write data to o/p

  A_1, A_0 } used for selecting port

  00 - port A

  01 - port B

  10 - port c

  11 → control word register

  $\overline{CS}$ - chip select

We want port A as i/p, port B as o/p, there is nothing in architecture. So for this we have to set control word register

- Initially all ports are initialized as input ports

- Address decoder select the which device has to select. (like 8255, 8255A ...)

- Data bus buffer :- to transfer data

Modes of operations in 8255

It can be configured in mode 0, mode 1 and mode 2

mode 0 and mode 1 can be used by all ports

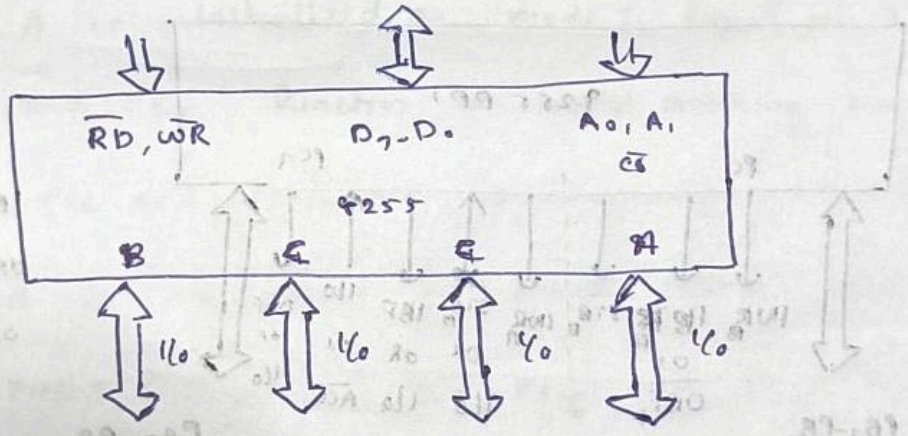
but mode 2 can be used only by port A.

mode 0 - simple / single strobe

mode 1 - single hand shake

mode 2 - double hand shake

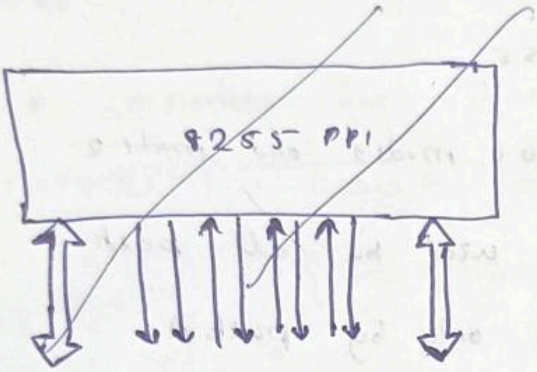
mode 0



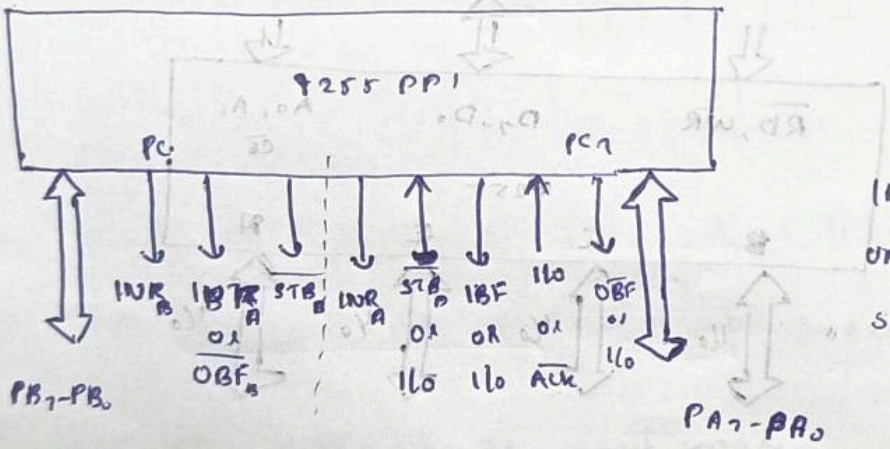
here all pins are used for I/O no handshaking

when any of port A or B do work in mode 1, we need handshaking

mode1



mode 1



port A and B can be configured in mode 1.

PORT C pins act as handshaking signal.

3 pins of PORT C are used as handshaking for PORT B

Rest pins for PORT A.

When 8255 has to send data, 8255 send a strobe signal to o/p device. ~~If data is send to buffer~~ OBF is send mean your data is in buffer (o/p is in latches) then send Interrupt

when a input is given, IBF signal is set. @ (data is send to 8255)

Then 8255 signal, acknowledgment.

Incase of port A

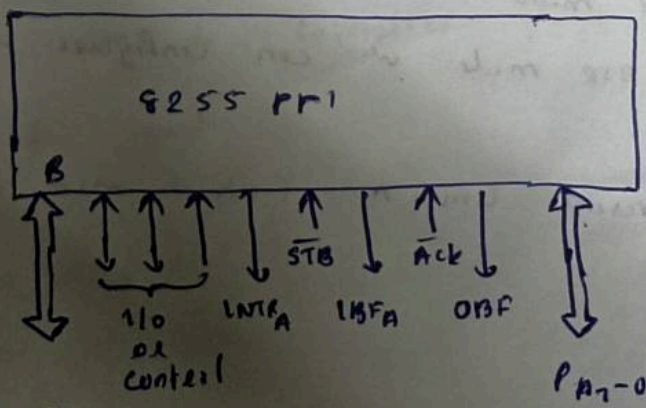
Here some of pins can be used for I/O transfer.

If A is in mode 0, port C last 4 pins are used as I/O

In mode 1 some of the pins of port C function as hand shake lines. The pins PC_0, PC_1 and PC_2 function as hand shake lines for port B if it is initialized in mode 1.

- If port A is initialized as mode 1 input port the pins PC_3, PC_4, and PC_5 function as hand shaking signal. The pins PC_6 and PC_7 can function as I/O lines
- If port A is initialized as hand shake output port then PORT C pins PC_3, PC_6 and PC_7 will function as hand shake signal. PC_4 and PC_5 can be used as I/O lines

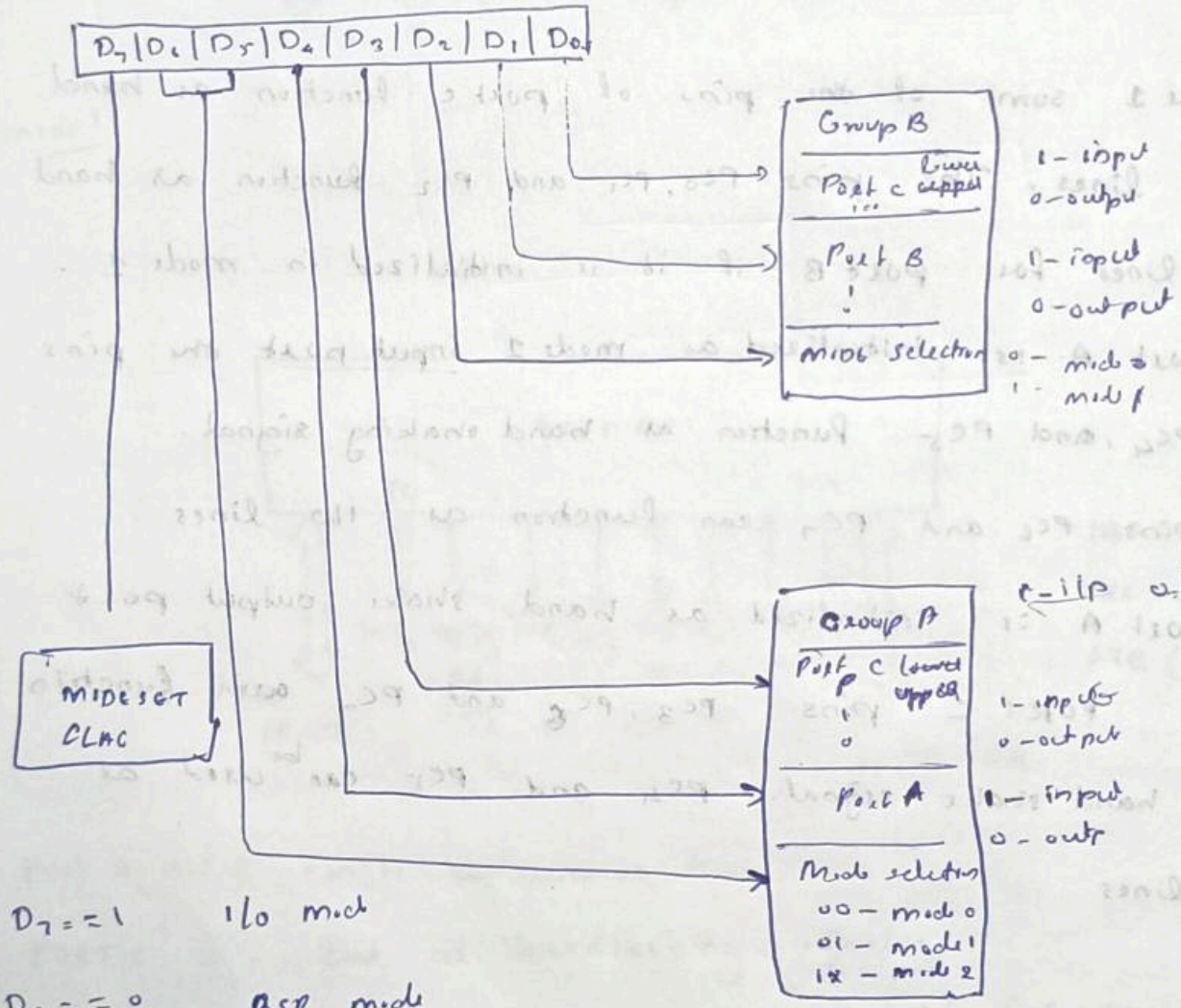
mode 2



When B is mode 0 : $PC_0 - PC_3$ act as I/O

when B is mode 1 : $PC_0 - PC_3$ act as control.

I/O mode control word



- $D_7 == 1$ I/O mode

$D_7 == 0$ BSR mode

In BSR mode we can configure port C bit by bit

- D_5 and D_6 represent Group A (which mode to work)

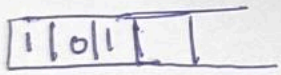
$D_7 - D_3$ are used for controlling group A

$D_2 - D_0$ - for group B

D_2 is for mode select. as port B and C can be used in 2 mode only 1 bit is used to select

---

a) port A in mode 1 as output, port B in mode 0 I/P and port C → mode 0



Mov address of 8255 to DX

CW to AL

Content of AL should be mov to DX

OUT DX, AL

b) suppose that you want to initialize 8255 as follows

PORT B as mode 1 input

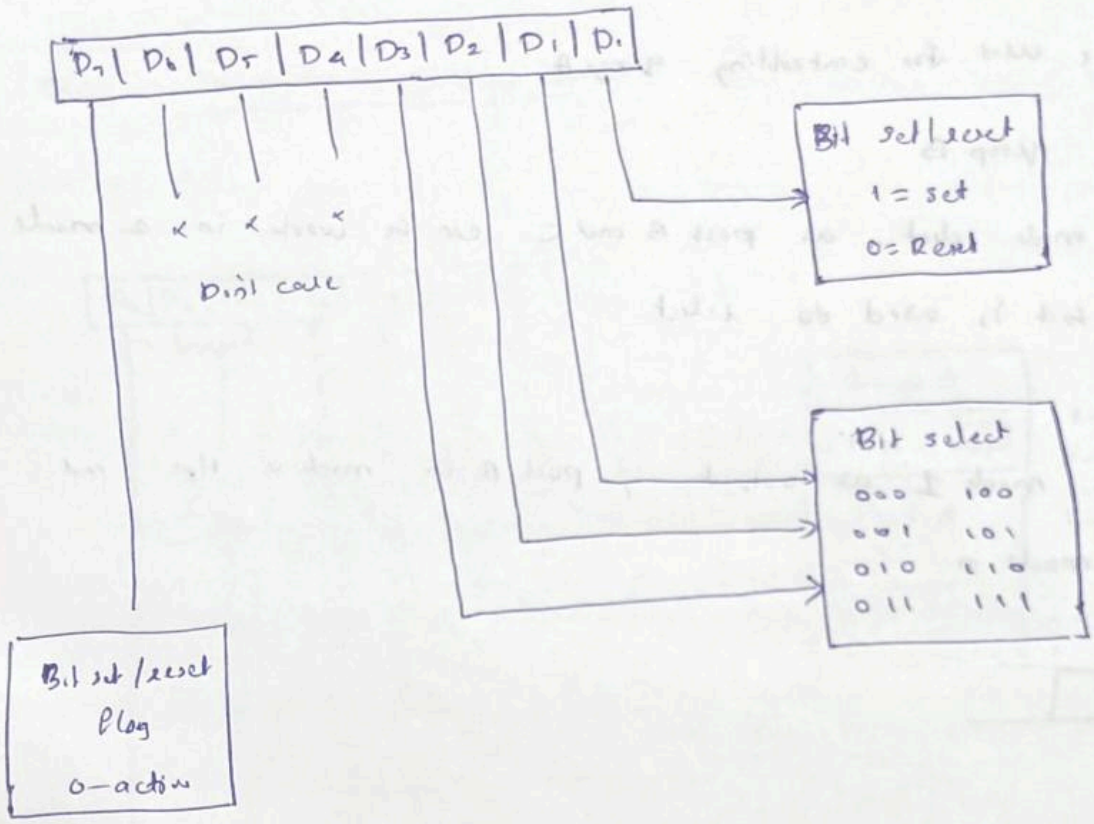
PORT A as mode 0 output

PORT C upper as input

PORT C bit 3 as output

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	1	1	0

BSR mode (Bit set reset mode)



In previous question if the portc bit is specified, we have to again reload AL with register value

From this CW we can't know if portc is in i/p or o/p mode It is based on the value of previous control word.

If bit in portc is set, and we don't want to change a bit in portc we have to again set a CW.

Using a control word in BSR mode only 1 bit in portc can be set at a time.

Modes of operation of 8255

2 modes of operation i) I/O mode (working as programmable I/O ports

ii) Bit set Reset mode (BSR) :- only port c can be used to set or reset the individual port bits

There are 3 modes in I/O

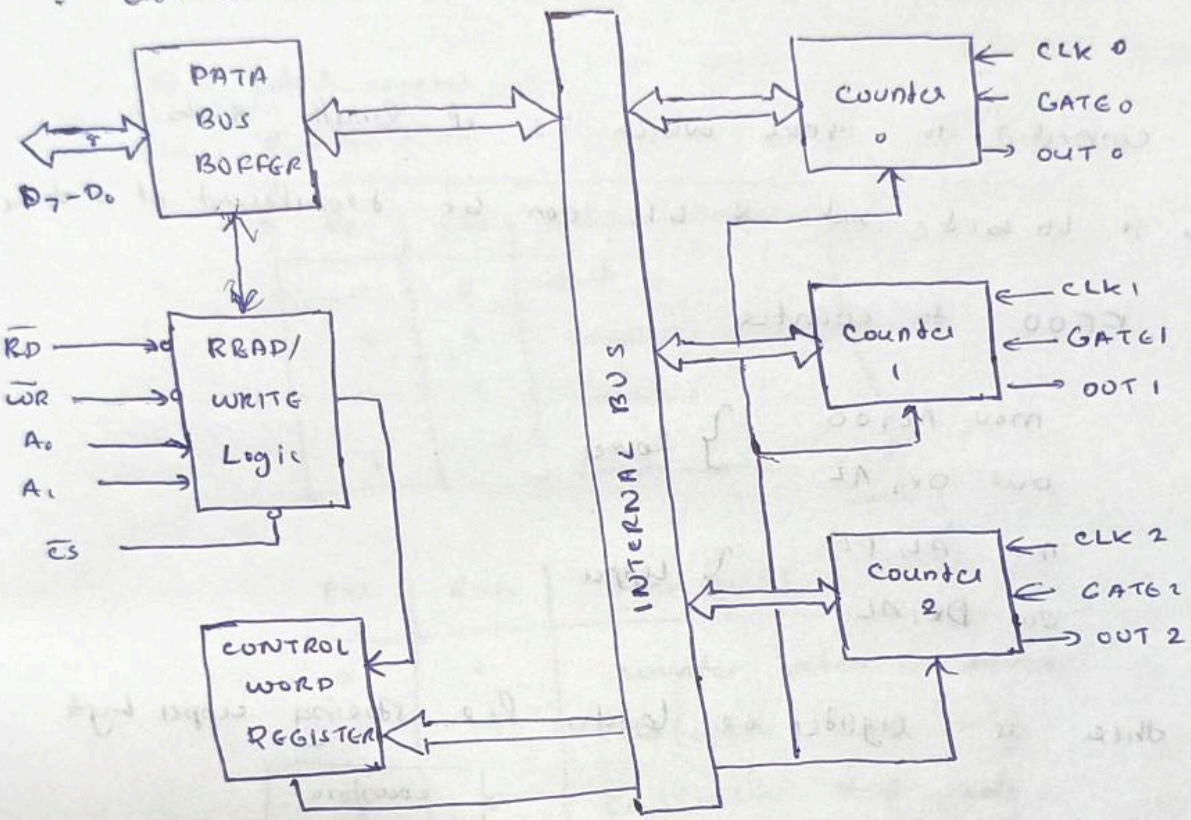
Mode 0 :- Basic I/O mode (

mode 1 :- Strobed I/O mode

mode 2 :- strobed bidirectional I/O (only port A can be worked in mode)

8254 Timer

- Counter is the basic function circuit in timer.
- 8254 down counter
- each is 16 bit counter



CW can only be written and cannot be read

GATE :- is 1 then only counter will become active

It determine whether the counter is active or not

OUT :- decrimenting is done through OUT

A_0 A_1 :- determine from which counter the data is to read or write

00 - counter 0
01 - 1
10 - 2
11 - control word

to load a particular value to counter or read value from counter

To specify which counter to be used, we need to put some value to control word register. According to value in CW the timer works.

Data bus is connected to 8086 which is of length 8 bit.

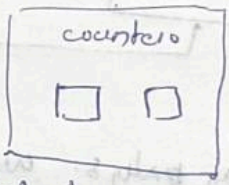
The counter is 16 bit, but 8 bit can be transferred at a time

eg: To write FF00 to counter

```
mov AL,00   } lower
out DX, AL

mov AL, FF  } upper
out DX, AL
```

In counter there is register or latches for storing upper byte and lower byte.



If it is 8 bit to load, it should be loaded to LSB of the counter.

```
mov AL, 04
OUT DX, AL
```

04

upper byte is automatically set to zero

Both the instruction for LSB and MSB are same.

We need to tell timer in what (LSB/MSB) is ~~set~~, to ~~timer~~

~~what~~ to set

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

SC - select counter M - mode BCD :- BCD or binary

RW - Read or write

SC_1	SC_0	select counter
0	0	counter 0
0	1	counter 1
1	0	counter 2
1	1	select control word

RW_1	RW_0	Read/write
0	0	counter latch command
0	1	Read/write LSB only
1	0	Read/write MSB only
1	1	Read/write LSB 1st then MSB

M_2, M_1, M_0

Timer can work on six mode (mode 0 to mode 5)

M_2, M_1, M_0 are used to selects the mode

BCD

If it is 0 it each as BCD counter otherwise as binary/hexadecimal counter

8 bit

*) To write a value to counter 1

CW = 0 1 01 0000

```
MOV AL, 50H
OUT DX, AL
```

1) - 1st we need to write into CW register then only we can write into counter

CW = 1 1 01 0000 which counter to write is will specified first

Modes of 8254

Mode 0 (Interrupt on terminal count)

CW = 10 LSB=4 CW = 0000 1000

CW to counter 0 is selected. LSB is only set

out is the corresponding signal that is going to generate

(Out becomes high when count reaches 0)

after LSB=4

after writing data to counter 1st it is in latch, from there

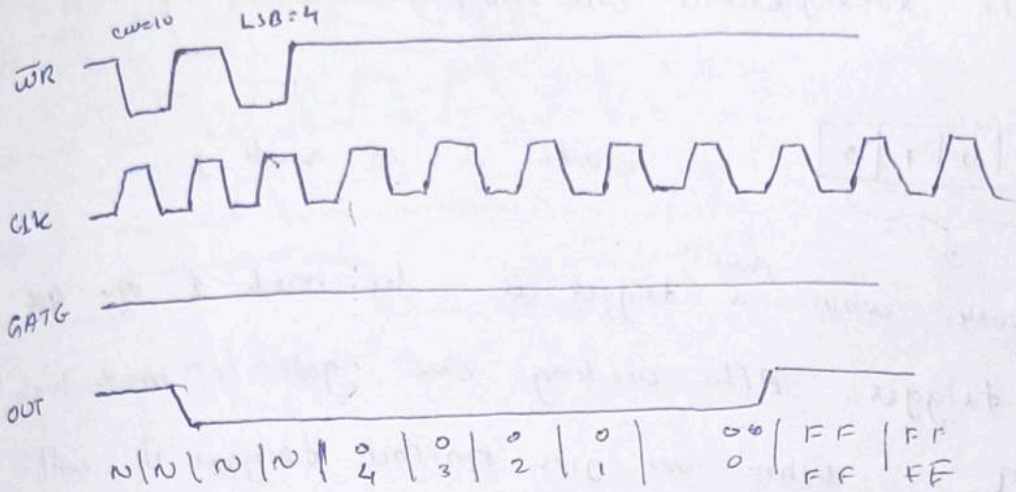
it will be loaded to 4 in next clock cycles

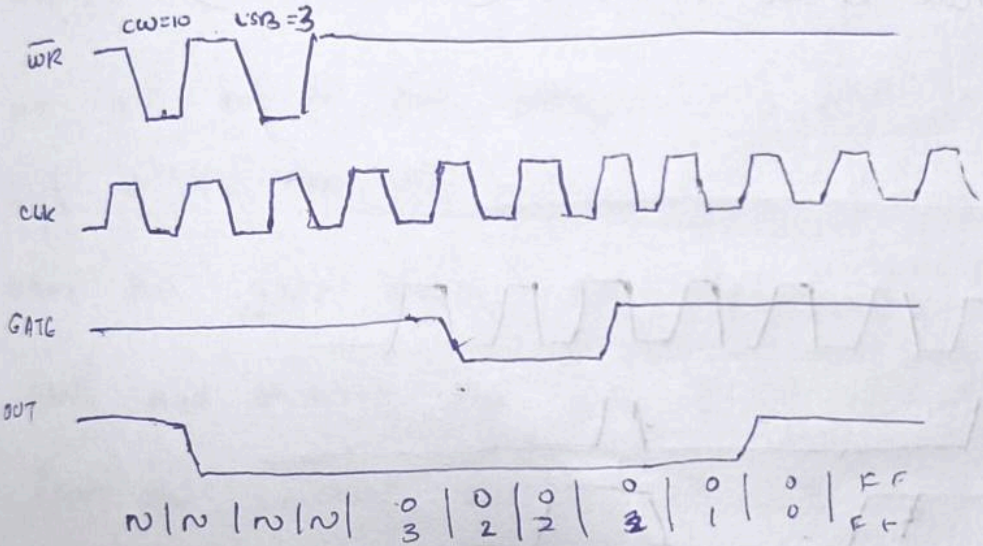
When count is 0 then OUT signal is set to high.

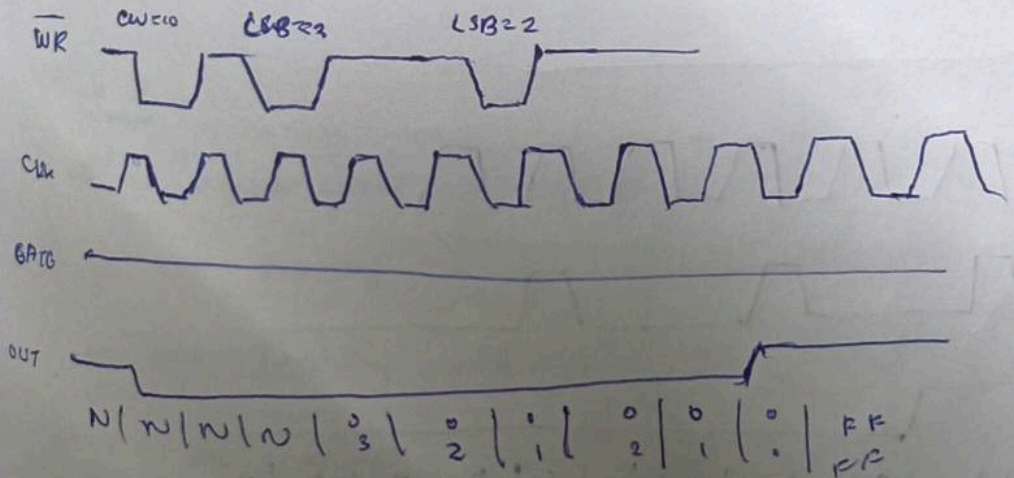
At the same time an interrupt is raised

When we want a interrupt to be generated and handled by

microprocessor then mode 0 is used


CW=10
LSB=4
WR
CLK
GATE
OUT
N N N N 4 3 2 1 0 FF FF
0 FF FF


CW=10
LSB=3
WR
CLK
GATE
OUT
N N N N 3 2 2 3 1 0 FF
FF


CW=10
LSB=2
LSB=2
WR
CLK
GATE
OUT
N N N N 3 2 1 2 1 0 FF
FF

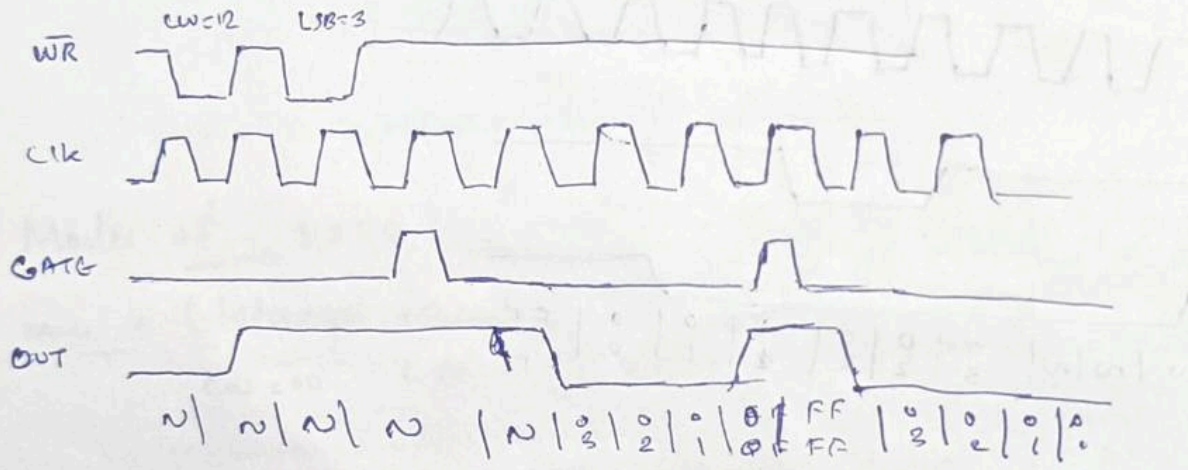
Mode 1 (Hardware retriggerable one shot)

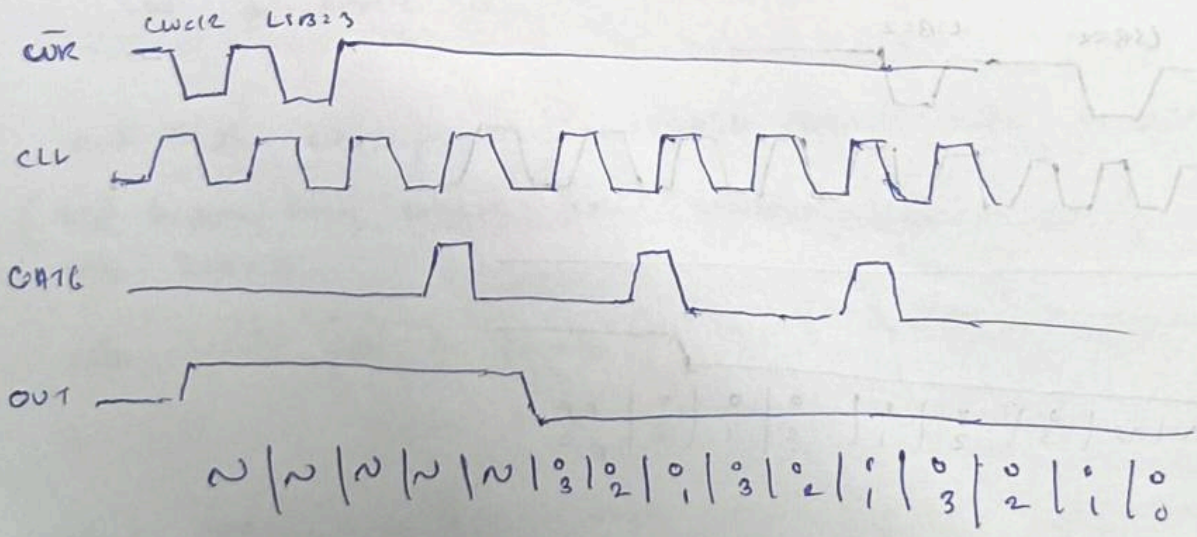
0	0	0	1	0	0	1	0

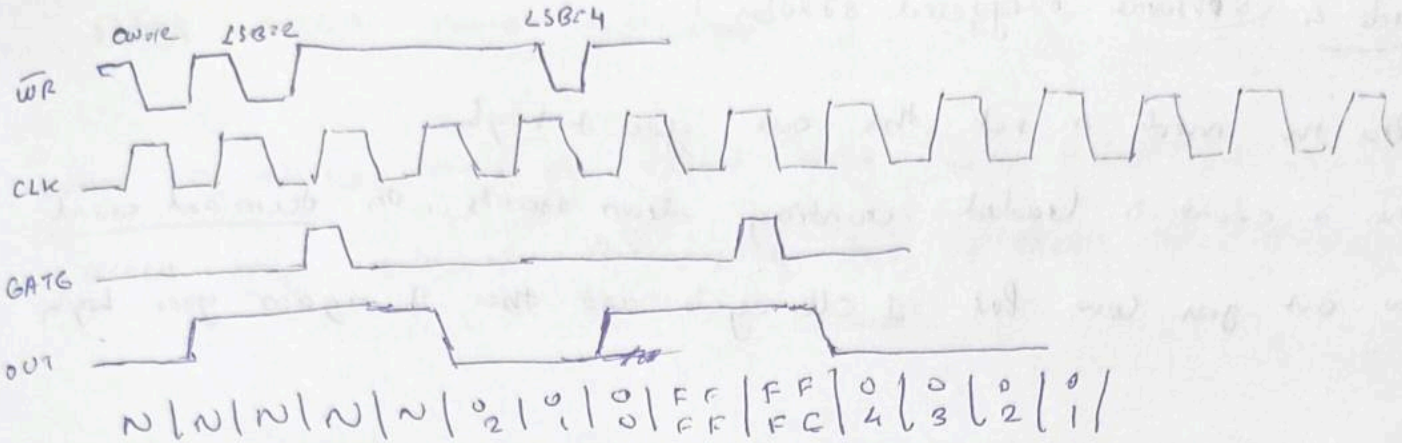
counter 0 in mode 1

1 2

down counter only works when a trigger is given. In mode 1 the gate signal act as trigger. After writing CW gate is made high and then low (⎍). When we give another trigger it will count from the value loaded to LSB (counter latch)







MODE 2

OUT will set to low when counts from 2 to 1. Then it becomes high at the next clk cycle.

When the gate become low counter remains in the same state and when the gate becomes high instead of counting from the current counter state, it reset to the value loaded into the latch (LSB)

MODE 3

It acts as square wave generator

If we are loading even no, square wave is generate

If we are loading odd no, high wave will be a 1 greater than the lower signal

When count N is loaded is even, then for half of the count the output remains high and for remaining half it remains low

. If the count loaded is odd, 1st clock pulse decrement by 1 resulting in an even count value & and counts

Mode 4 (Software triggered strobe)

After the mode is set the out goes high
when a count is loaded, counting down starts. On terminal count
the out goes low for 1 clk cycle and then it again goes high

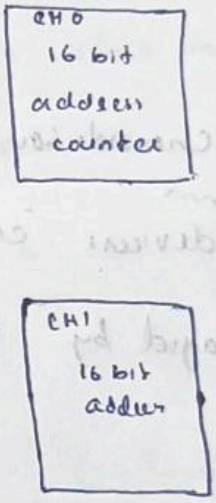
Mode 5 (Hardware triggered strobe)

- This mode of operation also generate a strobe in response to the rising edge at the trigger input.
- This mode may be used to generate a delayed strobe in response to an externally generated signal
- Once this mode is programmed and the counter is loaded, the out goes high.
  The counter start counting after the rising edge of trigger input (GATE)
- The out goes low for 1 clk cycle, when terminal count is reached
- The out will not go low until the counter content becomes zero after the rising edge of any trigger

~~DMA~~ 8257 DMA controller

When IO devices need to access mly, with DMA we can access mly without interrupting the process of 8086.

8257 architecture:



In each channel there are 2 16 bit registers. DMA memory address

There is terminal count register in it. 14 bit are used for current count. bit 14 and 15 are used to represent type of DMA operation

DRQ - for connecting i/o device. It sends request to DMA

DACK -

- All the channels have different priority. 0 > 4.

If 2^{n} devices come with request at same time. Priority is managed by priority resolver

In DMA there are 2 modes

i) slave mode ii) master mode

DMA transfer mode :- master mode

In slave mode DMA only interacts with CPU. (for initializing registers)

Read write logic-

IOR - used to read internal registers of DMA (in slave mode)

in master mode it is used to read data from connected

devices

Initially DMA is in slave mode. After initialization (how many cycles are used for DMA) (if 5 cycle are need for DMA is is loaded in to terminal count register)

$\overline{IOW}$:- for writing data

- A_0, A_1, A_2, A_3 are used for choosing channels in slave mode
  In master mode then are used for address
- $\overline{CS}$ - used for selecting

Control logic and mode set register

- A_4, A_5, A_6, A_7 are used along with A_0, A_1, A_2, A_3 for sending 8 bit address
- Ready :-
- HRQ :- DMA request used to transfer the control of bus
  It is send when DRQ of any channel is set.
- HLDA :- If 8086 is ready to give control then ALDA is set. (It is connected to HLDA of 8086)
- $\overline{MEMR}$:- to read data from mly
  $\overline{MEMW}$: for writing into memory.

ADSTB1-

TC: Terminal count

It is set when there is no DMA cycles. (means DMA transfer is over)

Memory interfacing with 8086

8086 has to communicate with memory for

Total address space : 1MB

This 1MB can be for different device

if 4K RAM chip (4kb is used we can access it with 12 address lines

• Interface two 4k X 8 EPROM and two 4kx4 RAM chips with 8086. Select suitable address maps.

ans 4k x 8 : 4k memory locations each of capable of storing 8 bits

. 4x1024 for EPROM . and 4x1024 for RAM

Now total memory is 8k 8k < 1MB it is possible

8k

For 8k memory 13 bits are need

8×1024

$2^3 \times 2^{10} = 2^{13}$

13 address lines are needed to access 8k memory

• Total 8k is taken as a single unit and it is divided to 2 for RAM and EPROM.

We have to specify the address to RAM and ROM

A_{13} A_{12} A_{11} A_{10} . . . A_2 A_1 A_0

0 0 0 0 - - 0 0 0

EPROM

RAM

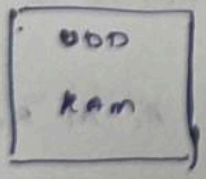
13 bits are used for address and other unused bit (some remaining line) are used as chip select lines.

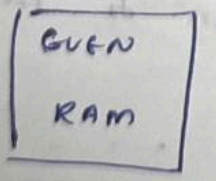
All other bits are set at either '0' or '1'

	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
PROM																				
FE000	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
FFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM																				
FC000	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
FDFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

A_0 BHE is used for selecting odd or even bank

A_{12} is used for selecting ROM and RAM





A_{19}	A_0	$\overline{BHE}$		
0	0	0	both even/odd address	RAM
0	0	1	even	"
0	1	0	odd	"
0	1	1		
1	0	0	both even/odd	ROM
1	0	1	even	"
1	1	0	odd	"
1	1	1		

There are $\overline{CS}$ for select each chip to select that particular chip

3x8 decoder is used for selecting particular chip