Register organization:

1. GPR (Gen-purpose)
2. IR (Index)
3. SR (segment)
4. PR (pointer reg)
5. flag reg.

GPR:

AH	AL	Accumulator
BH	BL	base
CH	CL	count
DH	DL	Data

→ AX → 16 bit, AL, AH → higher order 8 bit, lower order 8 bit
→ BX → BL, BH → used in ALU data transfer operation, result is placed in AX or AL.

→ Bx is used as address reg to form phy add. in case of indexed reg (indexed) addressing mode.

→ CX → used as default counter in string and loop inst and shift/rotate inst.

→ DX → used as an implicit operand or destination in few inst.

Segment register → used as port no. in I/O operations.

1MB is divided into 16 logical segments and each segment holds 64KB memory.

CS :- 16-bit reg holds the starting address of CS (64K CS) base

SS :- where stack data is stored. SS can be changed directly using POP inst.

DS : points the data seg. of the memory where prog data is stored. (POP, LDS inst)

ES :- ES register points to the base address of data segment of memory where prog. data is stored. ES reg. can be changed directly using POP & LES inst.

Index reg :- particularly useful for string operation (indexed, base index and relative base index reg addressing mode).

SI :- used to store the offset of source data in data reg.

DI :- string inst: use DI to access the M.L's in data or extra segment.

Pointer reg: contain offset of data (variable, labels and inst. from their base segments (default segments).

→ SP store the base add of SS. (hold & top of the stack)

→ BP use BP to access data in other segments

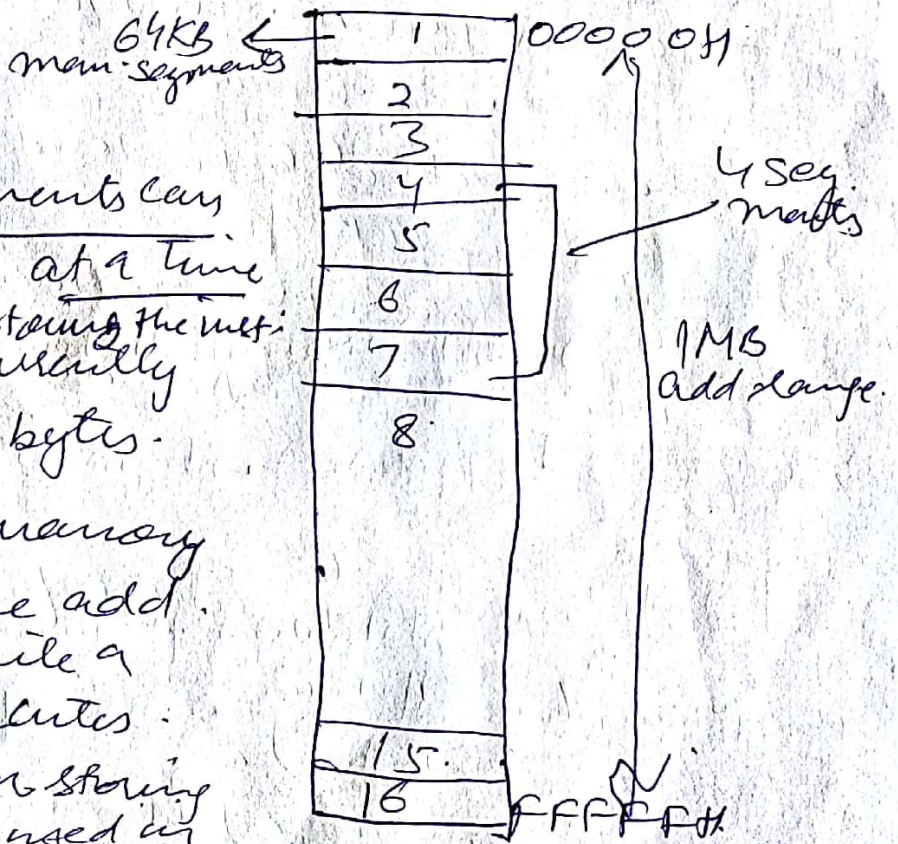
→ IP: fetching next inst to be executed.

* Flag register:

Memory Segmentation: 8086 has a 20-bit add bus. It can access 1MB memory.

Phy. addl mem. is divided into logical segments (64K) = 1MB

→ At a time 8086 work with 64KB of memory.



* Only 4 segments can be addressed at a time

Code seg is used for storing the inst. BIU internally fetching inst code bytes.

SS A section of memory set aside to store add. and data while a subprogram executes.

DS and ES - used for storing data values to be used in the program.

[There can be any no of CS, DS, SS, ES depending upon user program]

Adv. of Segmentation:

(1) 8086 - 16-bit

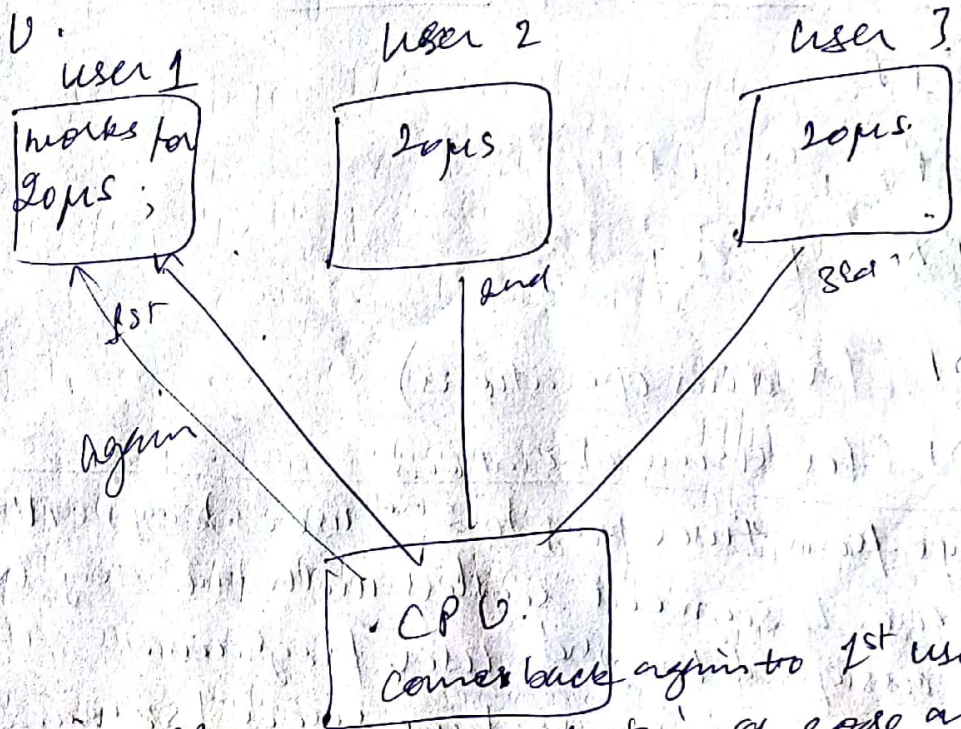
(1) To access memory, using seg: offset approach instead of 20-bit address.

(2) Segment: offset scheme requires only 16-bit no. to represent base address and offset to access any location in a segment.

→ Means 8086, manipulate and store only 16-bit quantities

→ This makes for an easier interface with 8-bit and 16-bit wide memory boards and with 16 bit registers in the 8086.

(2) In a time sharing mode, several users share a CPU.



* each time CPU access new section of code and data.
* Segmentation makes this switching quite easy.
* Each user's program can be assigned a separate set of logical segments for its code and data.
* The user's program will contain offsets or displacements from these segment bases.

* Adv of segmented memory scheme:-

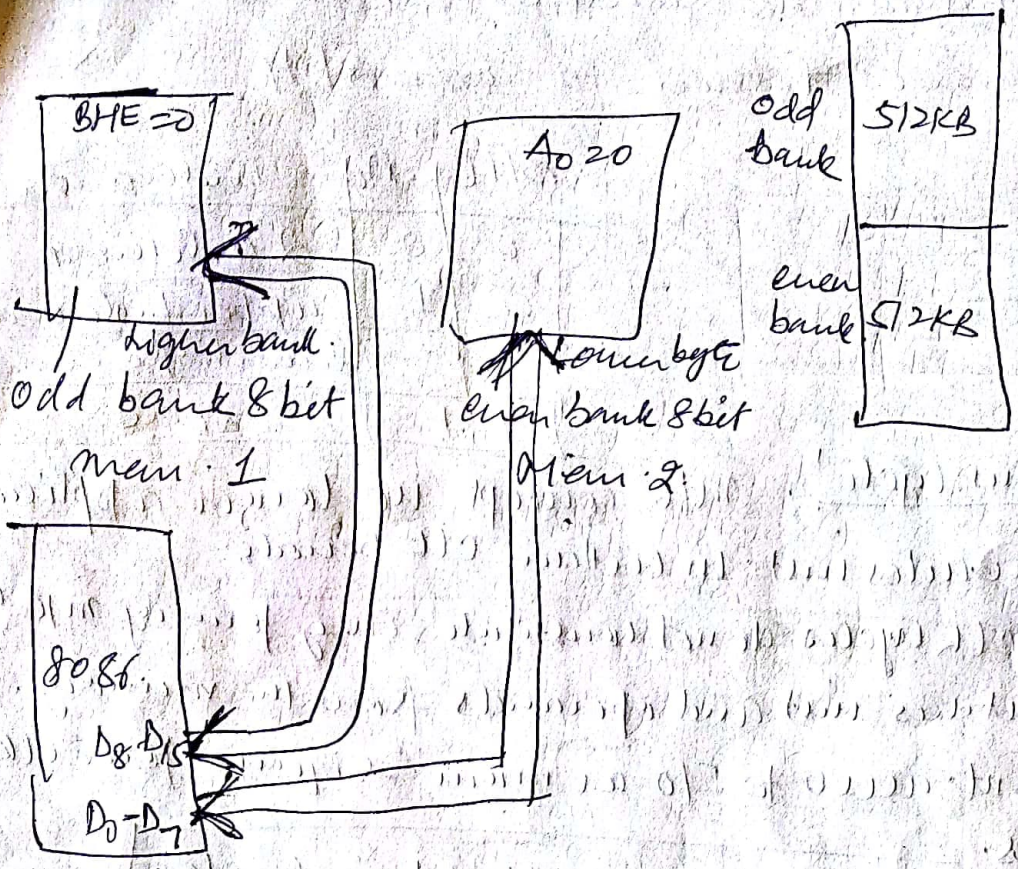
1. Segments may be overlapped or non-overlapped.
2. Allows the placing of code, data and stack portions of the same program in diff. parts / segments of the mem. for data and code operation.
3. The seg. reg. are used to allow the instruction data or stack portion of a program to be more than 64KB long. It can be achieved by using more than one code, data & stack segments.

* Segment and add reg. comb.

1. CS : IP
2. SS : SP , SS : BP
3. DS : BX DS : SI
4. DS : DI (for other operations)
5. ES : DI (for string operations)

To change from 1st user prog to 2nd user's prog, CPU has to reload the 4 segment registers with the seg. base address assigned to 2nd user program.

⇒ So segmentation, helps to keep users programs, data from one another, makes switching easier.



If processor fetch any word (2 byte) diff cond apply :-

→ both byte maybe data operands.

→ both byte contain opcode bits.

→ One byte may be opcode other maybe data.

* When word data come BIU required 2 M/C.

i.e. that depends on starting byte i.e even or odd.

If word data is on even location, it require only 1R+1W cycle.

If odd add :- 1st even cycle + 2nd odd cycle

D0-D7 D8-D15

if 16 bit data transfer then, first it goes to 00000H, then to 00001H.

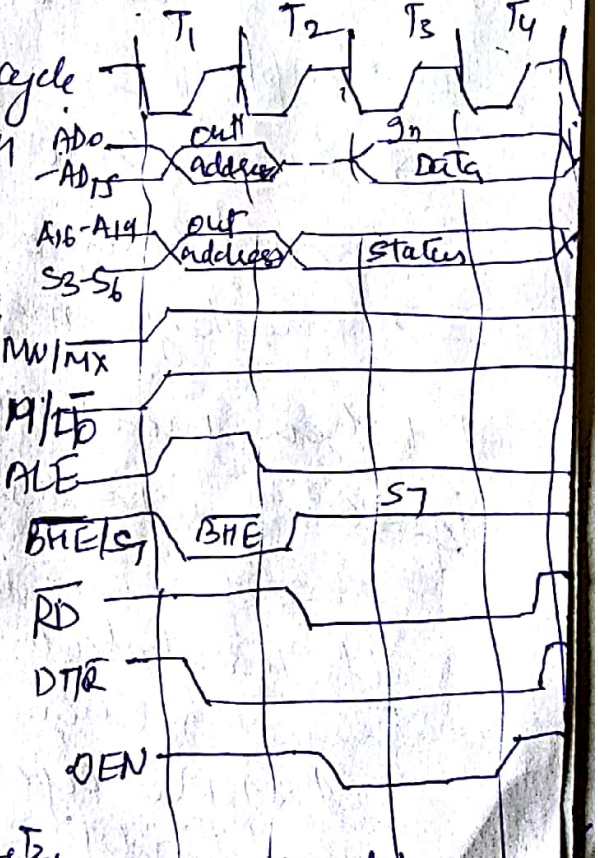
$\overline{BHE}$	A_0	indication
0	0	2 byte word
0	1	upper byte odd add
1	0	lower byte even add
1	1	None

8086 Bus cycle: M/C concept no longer applicable.

* EU executes inst in certain clk periods.
→ These clk cycles do not constitute any form of m/c cycle
→ BIU fetches inst and operands from the memory.
→ Any ext. access to I/O or mem. require four clk cycles.

4 clk cyc is called bus cycle

→ There are M/Mor I/O read bus cycle
4 4 Write " "

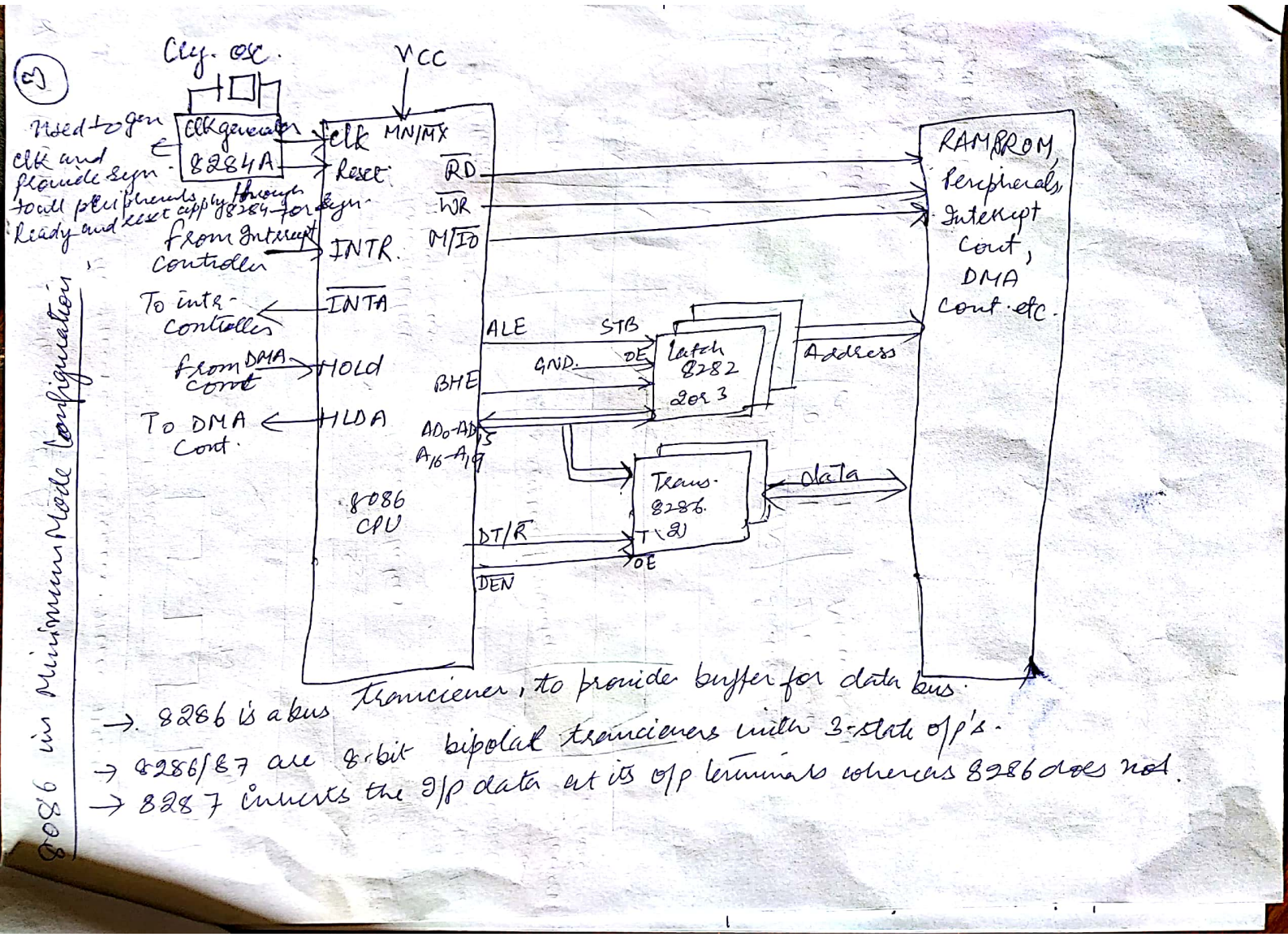


→ 1 Bus cycle = 4 clk periods
→ 20 bit address sent on AD_0-A_{15}, A_{16}-A_{19}
→ during T_1 carries address.
→ during T_2, AD_0-AD_{15} remain in high impedance state
→ During T_3 and T_4, data are transmitted on AD_0-AD_{15}.
→ T_2, T_3, T_4 → status signal carried by A_{16}-A_{19} lines.
→ M/IO is 1, indicates address sent by CPU is for memory. RD goes low during T_2, when low data can be read by CPU.
→ DT/R is used by 8286,87, DEN used by 8286,87 are used to control bidirectional (data) buffers.

Memory read for minimum mode of operation

Scanned by CamScanner

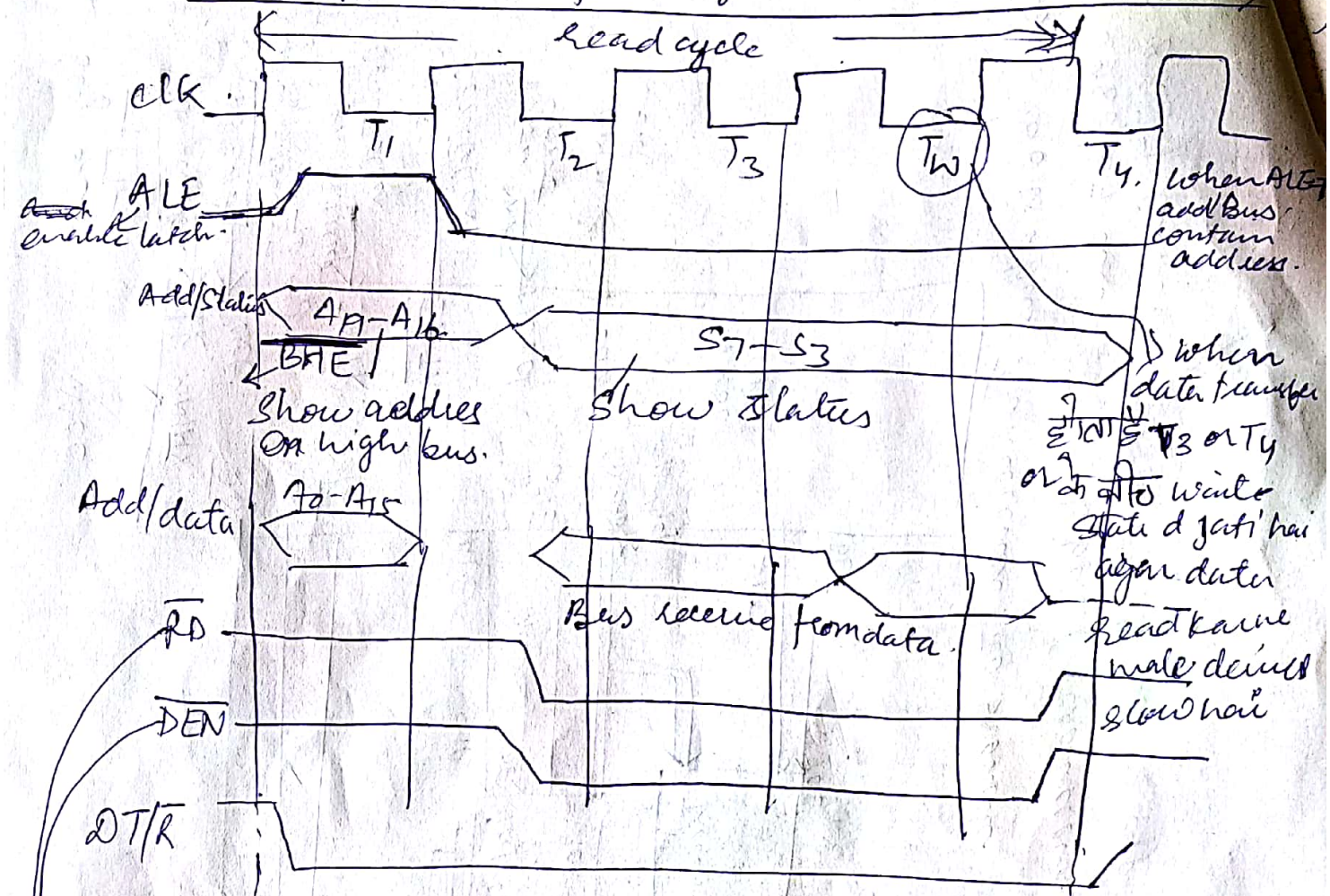
8086 in Minimum Mode Configuration



→ 8286 is a bus transciever, to provide buffer for data bus.
→ 8286/87 are 8-bit bipolar transcievers with 3-state o/p's.
→ 8287 inverts the i/p data at its o/p terminals whereas 8286 does not.

* Minimum Mode of 8086

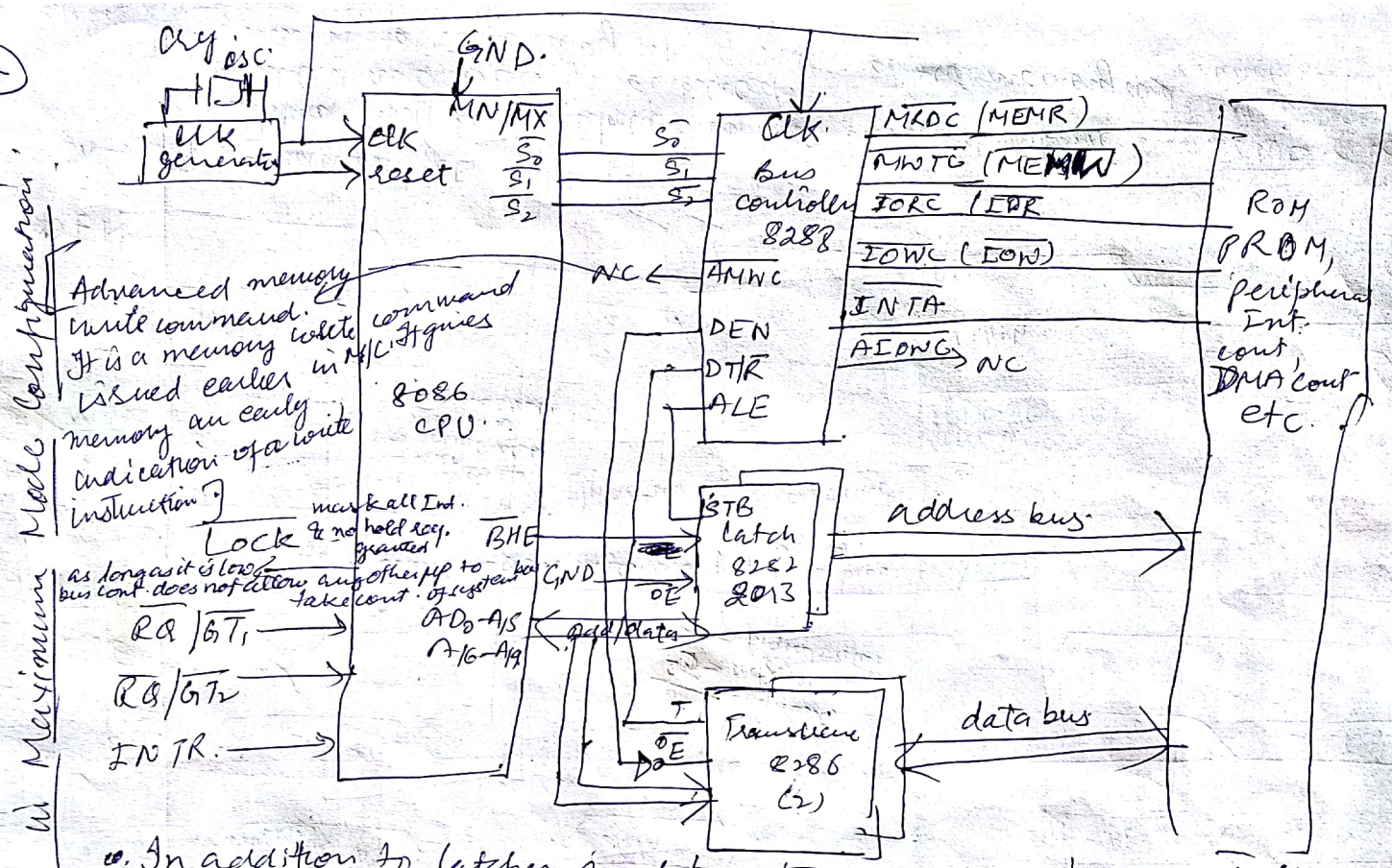
* Read cycle timing diagram - min mode 8086



M/IO	RD	DEN	Indication
0	0	1	I/O read
1	0	1	Mem read

(4)

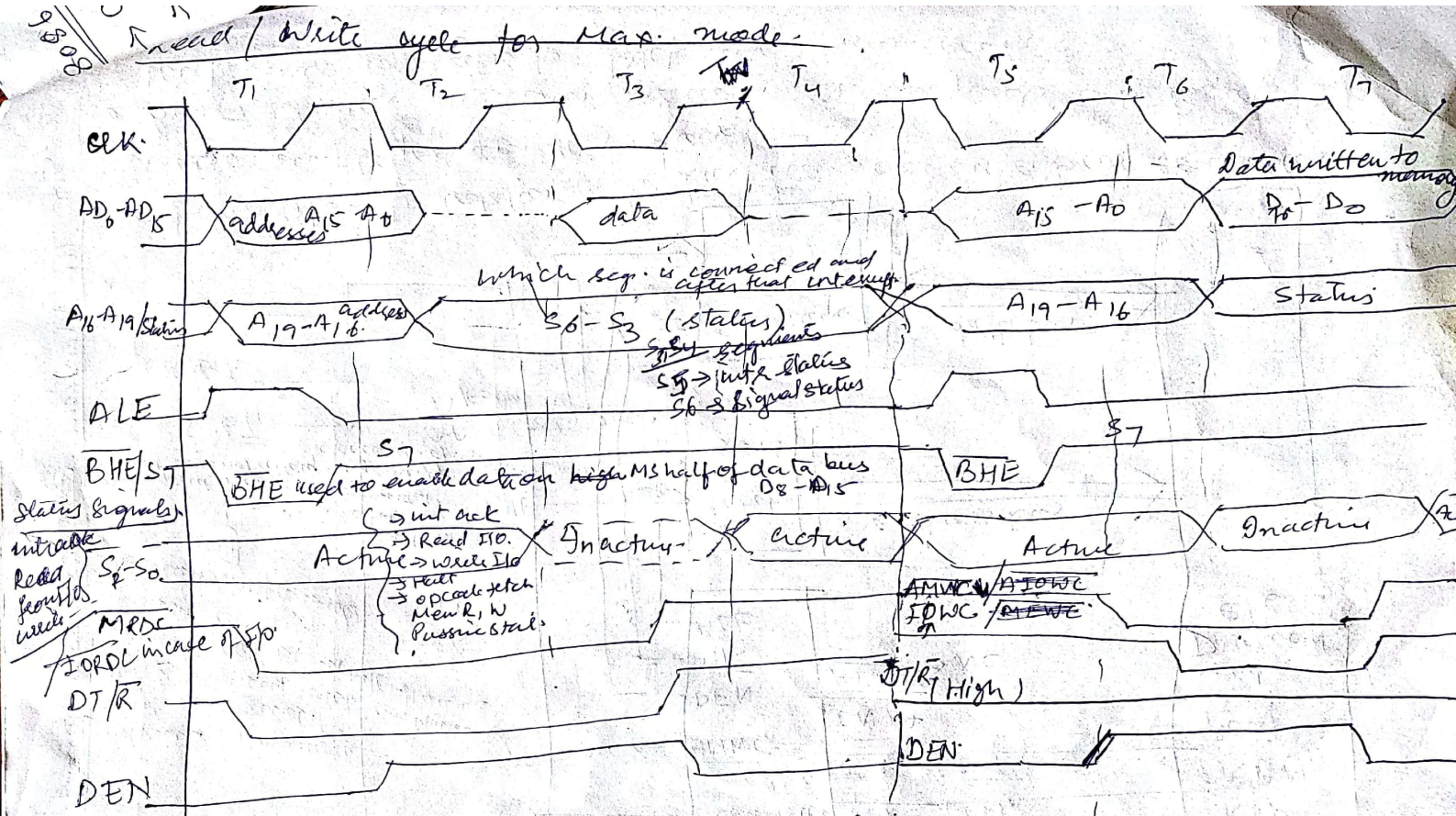
8086 in Maximum Mode configuration:



- In addition to latches and bus transceivers, a bus controller is also employed for this map.
- Bus controller provides various cont. signals to Memories / I/O, peripherals and also to latch and Transceiver.

8086

Read / Write cycle for Max. mode.



I_n → T_1 :- address gets latch on 8282

T_2 :- data will be available on data bus and $\overline{MRDC}$ and $\overline{IOWC}$ are enabled and DEN is enabled.

T_3: data is forwarded by µp i.e written on the memory and all other signals are deactivated ($\overline{AMWC}$ / $\overline{AIOWC}$)

T_4: all the signals get deactivated.

8086 Interrupts cycle.

Int. means break the seq operation.

→ CPU int. ko handle karta hai.

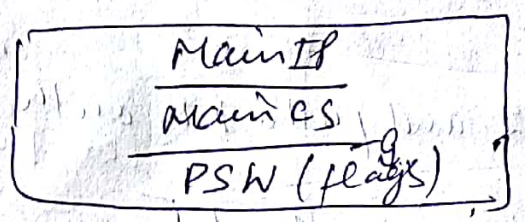
→ 8086 has two int. pins (NMI, INTR)

Interrupts

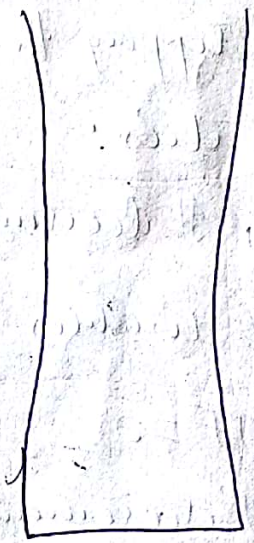
ext. → int

↳ internally generated.

I/O



Memory bank.



Interrupt vector table

Trap
NMI
Type } independent of IF

Interrupts.

Hardware

→ An interrupt caused by an ext-device is called hardware

→ NMI } Non maskable.
INTR } applied to I/P pins of μp.
maskable.

Software.

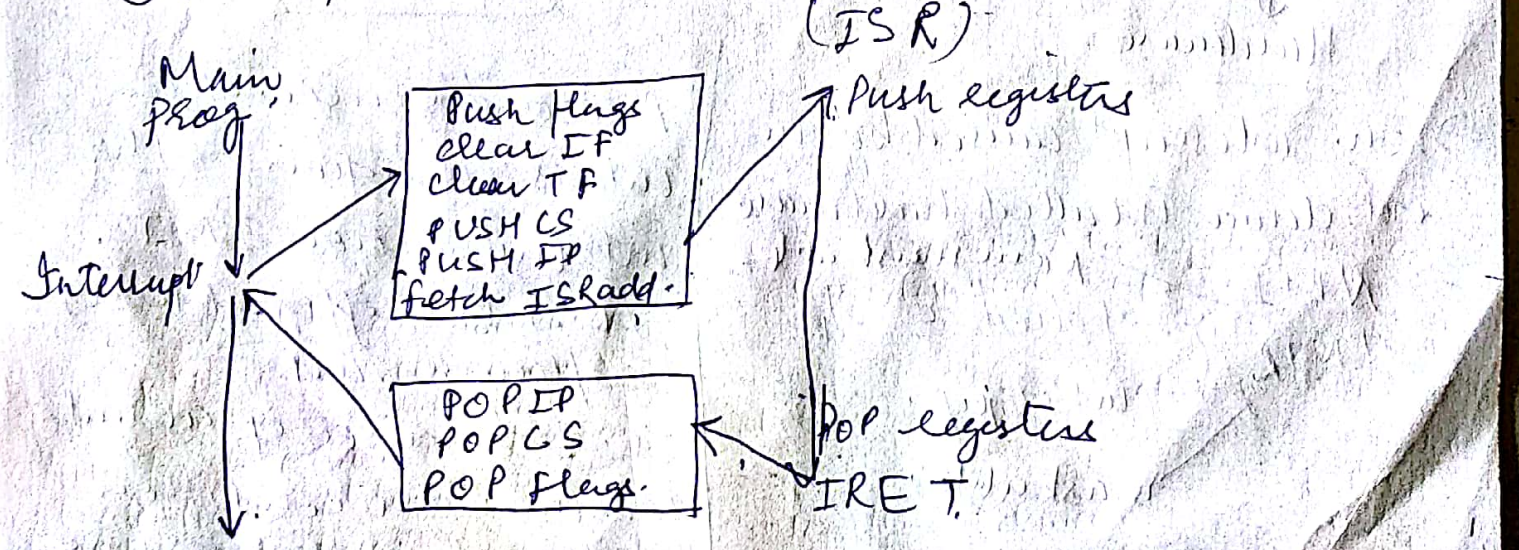
A second source of an int. is execution of the int instruction INT. This is referred to as software int.

The 3rd source of an int. is some error condition produced in the 8086 by the execution of an inst.

for eg :- divide by zero. if ~~the~~ attempt to divide 8086 an operand by zero, the 8086 will automatically interrupt the currently executing prog.

→ At end of each inst-cycle, 8086 checks to see if any interrupt have been requested. If an int. req, the 8086 respond to the interrupt by stepping through the following series of major activities.

(1) It decrements SP by 2 and push FR on the stack
(2) It disables INTR by clearing the IF.
(3) TF=0
(4) It decrements SP by 2 and pushes current CS contents on stack.
(5) It dec. stack pointer again by 2 and pushes current IP contents on stack.
(6) It Jump to start of ISR.

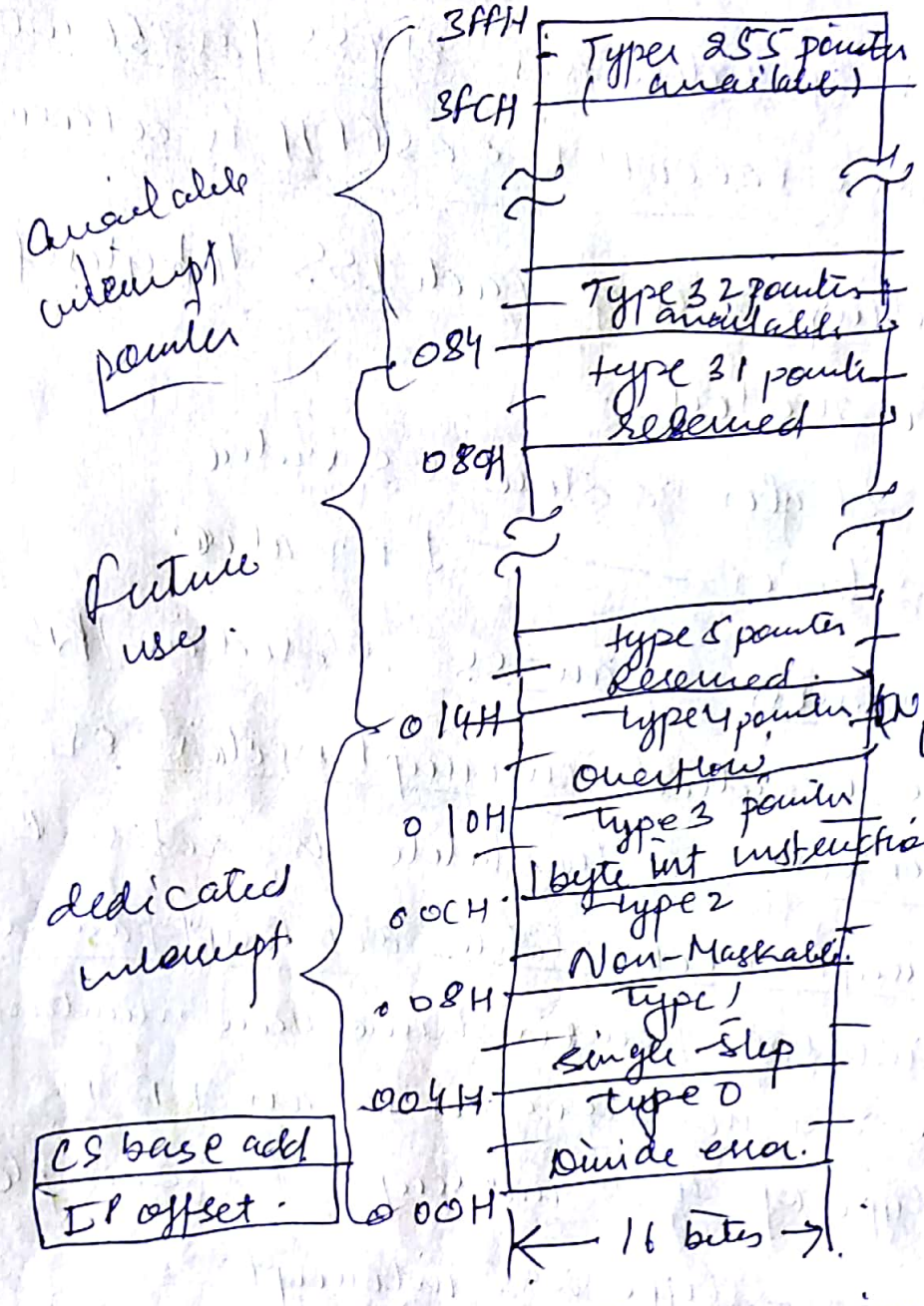


8086 can handle upto 256 hardware + software int's.
To store the starting addresses of ISS for 256 int's.
1KB memory from 00000H to 003FFH is set aside.
To store the starting add of each ISS 4 bytes of memory space is needed :-

2 bytes to store CS value
2 bytes ——— IP value.

The starting address of an ISS stored in 1KB memory space is called interrupt vector or interrupt pointer, so the table is referred to as the interrupt vector table or interrupt pointer table. 256 interrupt pointer have been numbered from 0 to 255. The no. assigned to an IP is known as type of that interrupt. for example type 0, type 1, — type 255 interrupt.

→ The lowest five types are dedicated to specific interrupts
→ Interrupts 5 to 31 are reserved by Intel for use in more complex μp.
→ The upper 224 interrupt type from 32 to 255 are available to user for hardware and software interrupts.



the Lock signal goes low from T_2 of last ack cycle to T_2 of 2nd Ack. cycle, Therefore, HOLD request is not honoured till the end of 2nd bus cycle. *

The vector for each int-type requires 4 Mem locations. when 8086 responds to a particular Type interrupt, it automatically multiplies the type by 4 to produce the desired address in the vector table. It then goes to that address in the Table to get the starting address of the ISP. (INTR) Maskable, can be enabled/disabled using IF. → 8086 has INTR & INTA send

NMI can't be disabled.

* used by MP to handle emergency cond. used in case of power failure to store data and program. → Ckt - electronic ckts needed to detect power failure and to send an int. signal to 8086 through NMI.

It encounters two consecutive interrupt ack bus cycle.

1st Issue INTA signal to inform the ext device to get ready.

2nd ext device sends Int-Type to 8086 through Low order data bus

this no × 4 = to get Starting address from int table

How
* 8086 respond to interrupt : example (type 0)

As type 0 is divide by zero interrupt

→ for this, 8086 DIV and IDIV inst

DIV : allow us to divide a 16-bit unsigned binary no. in AX by an 8-bit unsigned no. from a specified memory location or register. The 8-bit result quotient will be left in AL register " remainder " " AH reg.

Also — 32bit unsigned no in DX and AX by a 16-bit no. in a specified register, M.L.

16-bit quotient → AX

16-bit remainder → DX.

Illy IDIV : used to divide 16/32 bit signed binary no in AX by an 8-bit signed no. in specified reg or ML. Or 32bit signed no in AX and DX by 16-bit signed no ~~from~~ specified M.L or register.

Now if the result of 16-bit no is too large to fit in AL or quotient from dividing 32 bit difficult to fit in AX, the result of division is meaningless —

Special case : divide by-zero gives infinity. So whenever the quotient from DIV or IDIV operation is too large to fit in the result register. the 8086 auto matically do a type 0 interrupt

→ 1st → 8086 → decrement SP by 2 and copies flag register to the stack.

→ Then clear IF and TF.

→ It saves return address on the Stack.

→ by decrementing SP by 2,

It pushes CS values of return address on the

again decrement SP by 2 and pushes IP values on Stack.

→ 8086 then gets the starting address of ISP from the type of locations in the interrupt vector table.

It gets CS values from 00002H and 00003H and IP from 00000H and 00001H.

→ After the starting address of CS and IP, 8086 then fetches and executes the first inst. of the procedure.

- at end of procedure, IRET inst is used to return execution to the interrupted program.

IRET → pops off IP and increment SP, then pop the values of CS again inc by 2 → restores the flags by popping off the stack value and SP by 2. After that again enable TF and IF flags.

Type 0 Cannot be disabled in any way, so care should be taken while using DIV or IDIV inst that no. divide by zero cond occurs or if so then simply write an ISP which takes the desired action when an invalid division occurs.

Type 1: Single-step interrupt used in monitor programs and debuggers programs.

⇒ When we tell a system to single-step, it will execute one inst and stop and wait for further direction from user.

The 8086 (TF) and type 1 int response make it quite easy to implement a single step feature in an 8086 based system.

→ If 8086 TF = 1, 8086 will automatically do a type 1 intr after each inst executes.

→ When 8086 does a type 1 intr, it pushes flag reg on the stack, reset TF and IF and pushes CS and IP values for the next inst. on the stack.

→ It then gets CS value for the start of the type 1 ISP from addr 00006H and IP from 00004H.

⇒ To set or reset TF, first pushing the flag reg to stack then changing the TF bit to what you want it to be, then popping the flag reg. back off the stack.

```
PUSHF                          Push flags on stack
MOV BP, SP                     copy SP to BP for using index mode
OR WORD PTR [BP+0], 0100H      set TF bit
POPF                           Restore flag reg.
```

TO reset TF, simply replace OR inst by AND, WORD PTR [BP+0], 0FEFFH.

Non maskable interrupt - Type 2 :-

→ 8086 automatically do a type 2 int response when it receives a 0 to 1 transition on its NMI I/p pin.

→ 8086 → push flags on stack
Reset T & and IF,
Push CS and IP for next inst on stack,
get new CS and IP from address 00008H and 00008H.

→ It cannot be disabled

for eg.
→ Steam boiler
→ dc power failure device has to consider entire system.

I/P → 8086 NMI means 8086 has to consider entire device.

breakpoint int - Type (3) :- produced by execution of the INT 3 inst. used to implement a breakpoint func in a system.

Program
- inst
3
breakpoint

inst execution upto breakpoint then goes to breakpoint procedure.

* overflow interrupt - type 4, of will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in destination reg or mem. location

for eg. 8 bit signed no. 01101100 (108 decimal)
01010001 (81)
10111101 (189 decimal)
↳ this would be correct if result we are adding unsigned binary no, but not correct in signed result,

For signed operation,
1 in MSB indicates -ve and in 2's complement form.

The result, 10111101, represent -67 decimal which is not correct value for 189)

Two ways to detect and respond to an overflow error.

(1) Put the JO, instruction after the arithmetic instruction if the OF is set as a result of arithmetic operation execution will jump to the address specified in JO inst. At this address you can put an error routine which responds to the overflow in the way you want.

(2) Put an INTO, interrupt on overflow, put INTO after arith. operation, if OF is not set then 8086 executes INTO it will function as an NOP.

Hamming, if over flag is set, 8086 will do a type 4 intr.

⇒ When the 8086 do it, it pushes flag reg to stack, reset TF and IF, pushes CS and IP to stack, get new CS and IP from 00012H and 00010H. Just in ISP then perform the desired response to error condition.

Software interrupts - type 0 through 255.

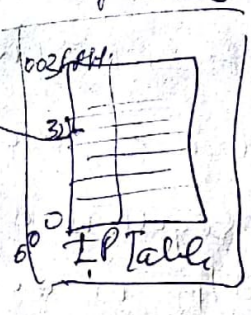
→ INT inst. cause 256 possible interrupt types in 8086.

→ for eg. INT 32, will cause the 8086 to do a type 32 interrupt response -

8086 push FR on stack
Reset TF and IF
Put CS and IP

Get new CS and IP from value for the start of an ISP from the Int. point Table in memory.

for IP = 4 × type 32
mem IP value = 128 decimal (80H).



and CS value is two added location's higher → 82H.

* Software interrupts produced by INT inst have many uses.

for eg. INT 3 inst use to insert breakpoints in programs for debugging.

INT 0 used to send execution to divide by zero ISP without having to run the actual division program.

INT 2 to send execution to an NMI ISP without needing to apply an external signal to the NMI IP of the 8086.

INTR interrupts - 0 through 255

→ 8086 INTR i/p allow some ext signal to int. execution of a prog.

→ INTR can be masked so that it cannot cause an interrupt

→ If IF = 0, INTR i/p is disabled.

IF can be cleared at any time with CLI (Clear interrupt instruction)

if IF = 1, INTR will be enabled.

IF = 1, by STI ('Set interrupt inst)

→ when 8086 is reset, IF, automatically cleared.

→ before 8086 respond to INTR i/p.

IF = 1 with STI.

8086 was designed this way so that ports, timer, registers etc, can be initialized before the INTR i/p is enabled.

* INTR tells 8086 to get ready to handle interrupt.

Remember that ~~When a trap occurs~~ IF is automatically = 0 as part of response of 8086 to an interrupt.

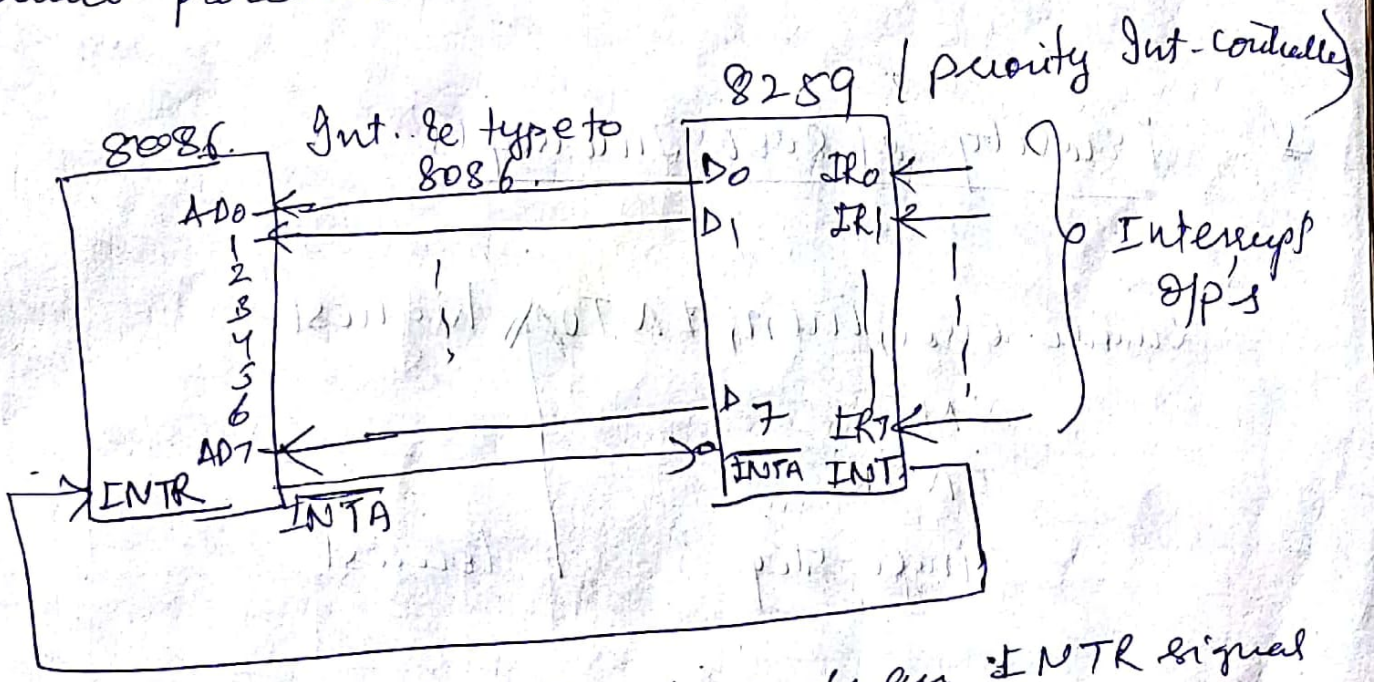
1st, it prevents a signal on INTR i/p from interrupting a higher priority ISP in progress

If you want another INTR i/p signal to be able to interrupt an ISP in progress, you can reenable INTR pin later on STI must at any time that is the reason for automatically disabling INTR i/p at start of ISP to make sure that when INTR = 1, enabled, 8086 will be interrupted by a signal on INTR i/p. If INTR were not disabled during first response, 8086 would be continously interrupted and would never get out to an actual ISP.

* IRET inst at end of ISR restores flags to the condition they were in before the procedure by popping the flag reg. off the stack.

→ This will reenable INTR i/p.
If INTR is still high, it will interrupt 8086 again.

* If you do not want the 8086 to be interrupted again by same i/p signal, you have to use external hardware to make sure that the signal is made low again before you reenable INTR with the STI inst or before the IRET from the INTR service procedure.



→ When 8259 receives IR, it sends an INTR signal to 8086
→ if INTR = 1 with STI inst, 8086 get interrupt from external device,
8086 first does two INTA cycle - 8086 floats data bus lines AD0-AD7 in 1st M/C cycle and sends out an ~~int~~ INTA to 8259 to get ready

2nd M/c cycle' - During 8086 send out another pu on INTA o/p pin, then 8259 puts the interrupt type on lower eight lines of data bus, where it is read by 8086.

→ When 8086 receives interrupt type, it pushes Flag register, clear TF & IF, push CS and IP → Stack, then use the type it read from ext device to get CS and IP for ISP from the Int. pointer table in memory.

* Priority of 8086 Interrupts.

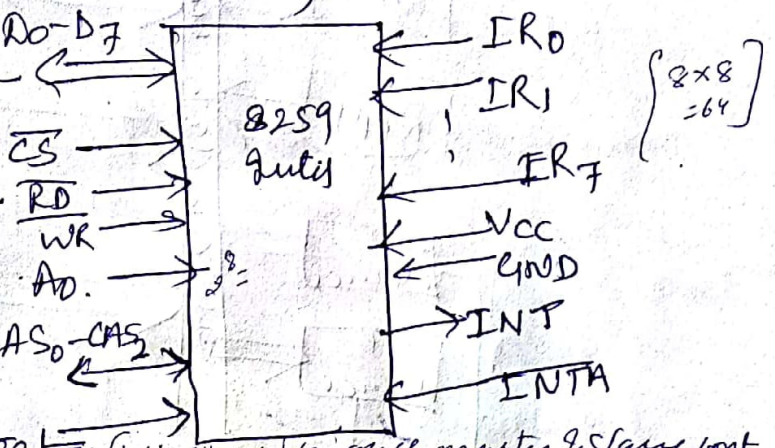
Divide error, Int n, INTO — highest
NMI
INTR
Single-step — lowest

→ Hardware interrupt applications:
* using Interrupts for counting & timing

Assignment 3rd q

* Programmable Int. Controller 8259 (used when I/O transfer data using Int's and they are to be connected to same Int. line of µp.

Control, status & int vector info are transfered with this bus.



RD [Enables 8259 to send various status signal on the data bus for CPU]

WR A low on this pin, enables 8259 to accept command word from CPU.

SP/EN (used to distinguish master & slave cont. [for this purpose, signal is used as an I/P signal. for master 8259, it is 5V; for slaves 8259 it is kept at 0)

A0 address line, acts in conjuction with RD, WR and CS. 8259 uses it to interpret command words the CPU writes and status the CPU wants to read.

CAS0-CAS2: cascade lines (in large systems to drive data bus buffers are used) 8259 is used in buffered mode)

SP/EN: Slave program/ enable buffer. If it is retained as I/P to cascade control. (for slave 8259 = 0, Master = 1) data bus (when SP/EN is used as O/P to enable the data bus buffer.

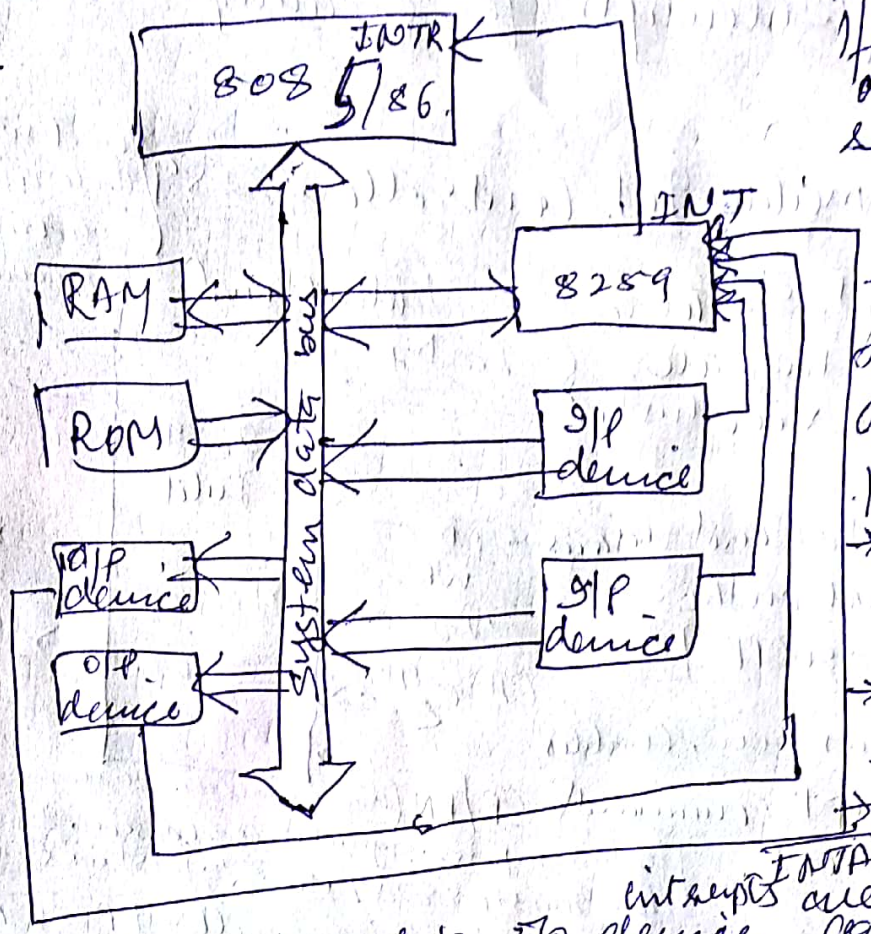
8259 Master (5V) → 8259 slaves (0V)

INT: used to interrupt CPU.

PIC is used when several I/O devices transfer data using interrupt and they are to be connected to the same interrupt line of the µp. When no of I/O devices < no. of interrupt lines of µp, such controllers are not required. 8259 is a single chip programmable int controller. It is compatible with 8086, 88, 85 µp.

→ 28 pin DIP IC, uses N-mos tech.
→ requires +5V supply.

8259 and I/O devices



If more than one I/O devices send interrupt requests at the same time the intrupt controller determine their priority.

→ sends an intr req. through INT line.

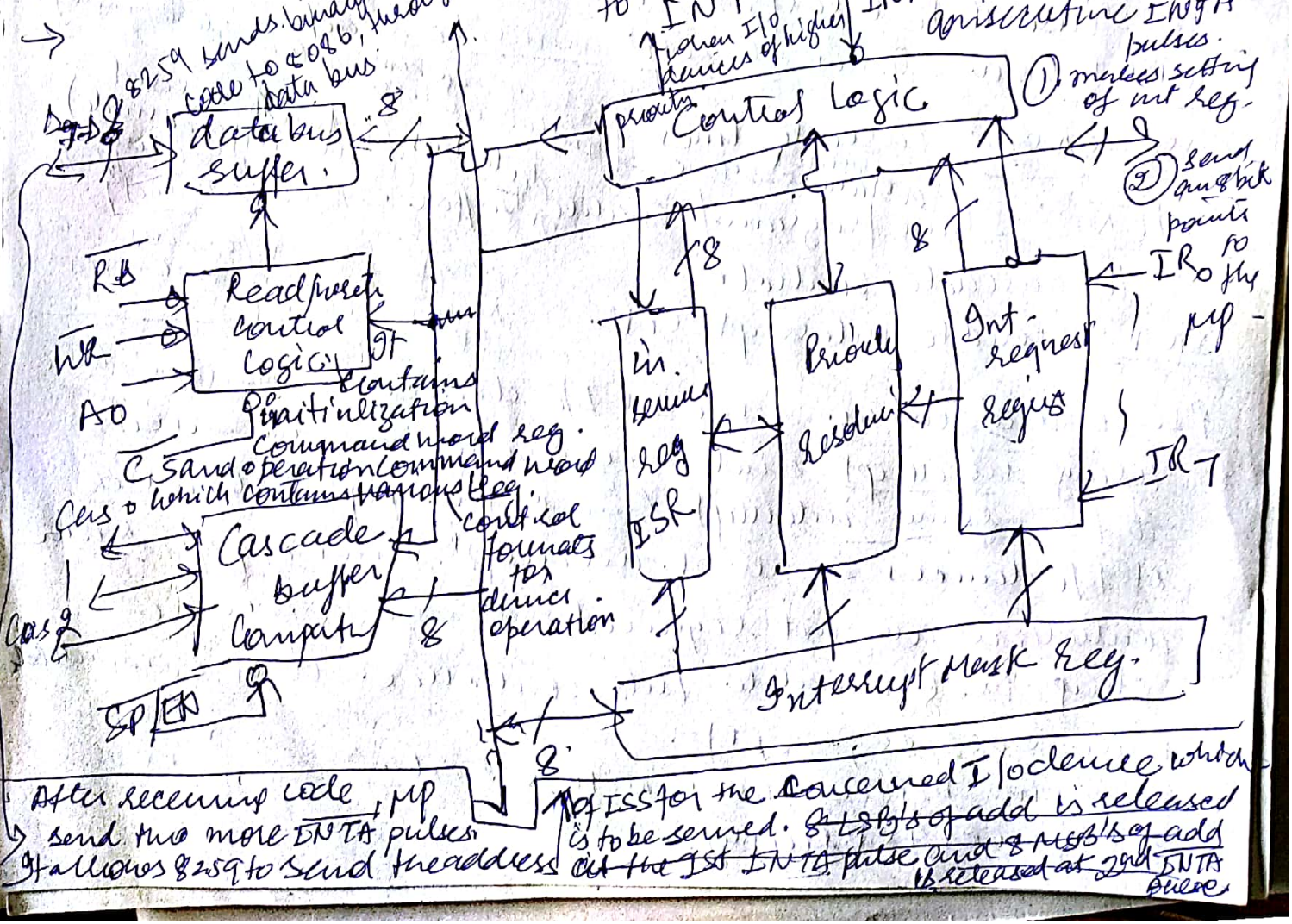
→ µp then send INTA line

→ on receipt of INTA, all lower priority interrupts are inhibited

→ Priority can be assigned to I/O device connected to IC

→ 8259 sends binary code to 8086 through data bus

→ 8086 sends two consecutive INTA pulses.



After receiving code, µp send two more INTA pulses. It allows 8259 to send the address of ISR for the concerned I/O device which is to be served. 8 LSBs of add is released on the 1st INTA pulse and 8 MSBs of add is released at 2nd INTA pulse.

→ 8259 sends a binary code to the µp through data bus,
→ After receiving code, the processor sends two INTA pulses,
→ on the receipt of 1st INTA pulse, 8259 makes necessary setting of its internal reg.
→ On the II nd INTA pulse, 8259 sends an 8 bit pointer to the processor.
→ 8 bit pointer is used by the processor to compute the starting memory add of ISS for the concerned I/O device.

IRR → It stores int requests (8-bit reg.)
→ When an intr. request is received, the corresponding bit in IRR is set.

ISR: which intr is currently being serviced, during the service of an intr request, if another higher priority intr becomes active, it will be acknowledged and the control will be transferred from lower priority intr (ISS) to higher priority ISS. Thus, more than 1 bit of ISR will be set indicating the no of intr being serviced.

Priority Resolver: determines the priorities of the int's set in IRR. Takes info from IRR, IMR and ISR to determine whether the new interrupt request is having highest priority or not. If the new

interrupt req. is having the highest priority it is selected and processed. The corresponding bit of ISR will be set during interrupt ack. m/c.

IMR contains a specific bit for each interrupt. It is used to mask or enable individual intr. i/p. An intr. i/p can be masked by setting the corresponding bit to 1 in IMR. An intr. which is masked by software is not recognised and serviced even if the corresponding bit is set in the IRR.

Prog. counter / Timer

* Applications: used in real time app. for timing & counting functions such as:
  - BCD & binary counting
  - generation of accurate time delay
  - square wave of desired freq.
  - rate generation
  - hardware/software triggered strobe signal
  - one-shot signal of desired width. etc.

NMOS 8254 Software-Programmable Timer/Counter

8253 → freq range of d.c to 2.6MHz

dc to 10MHz ← 8254 → Contain 3, 16-bit counters which can be programmed to operate in several diff. modes.

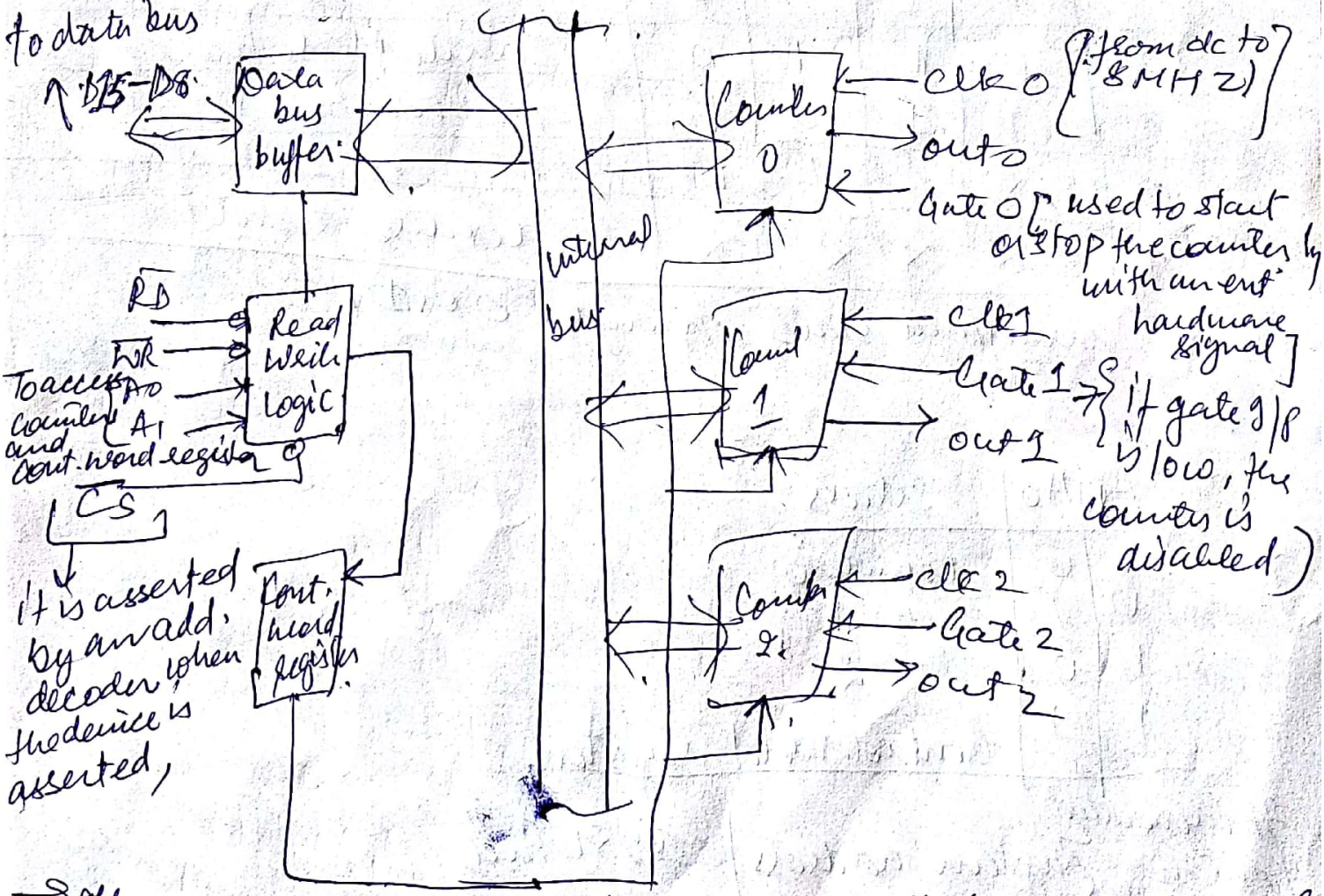
HMOS → identical in function

differences ① 8253 → Max. o/p clk freq → 2.6MHz
8254 → 8MHz (10MHz for 8254-2)

② 8254 has a read-back feature

↓

allows you to latch the count in all counters and the status of the counter at any point



→ These counters can be loaded, started and stopped with instructions, so called Software prog.

→ To program, you send count bytes and control bytes to the device just as you would send data to a port device

Read/Write Logic It accepts signals from system bus and generates control signal for operation of 8254.

$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	
0	0	0	0	1	Read counter No 0
0	0	1	0	1	Read count 1
0	1	0	0	1	Read Count 2
0	0	0	1	0	Load count 0
0	0	1	1	0	" " 1
0	1	0	1	0	" " 2
0	1	1	1	0	Write Mode word
0	1	1	0	1	No operation 3-state
0	X	X	1	1	" " " "
1	X	X	X	X	disable 3-state

X – undefined state, may be 0 or 1.

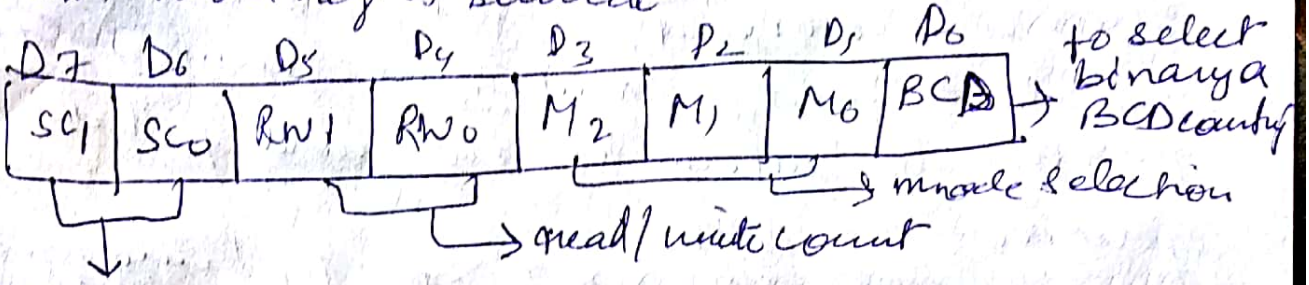
A_1	A_0	Selects
0	0	counter 0
0	1	" 1
1	0	" 2
1	1	Control word register.

It is generated from the address decoder circuit. We are using upper half of data bus

System address	8254 Part
FF01	count 0
FF03	" 1
FF05	" 2
FF07	Control word reg

If lower half then, even addresses

Control Word register: when $A_0, A_1 \rightarrow 11$, then control word reg is selected



To select counter:

SC_1	SC_0	
0	0	counter 0
0	1	1
1	0	2
1	1	Read back command

RW_1	RW_0	
0	0	counter Latching operation
0	1	Read/write least significant byte only.
1	0	" " Most " " "
1	1	" " least S Byte first, then MS byte

Mode:

M_2	M_1	M_0	
0	0	0	Mode 0 Interrupt on terminal count
0	0	1	" 1 Programmable one shott
X	1	0	" 2 rate generator
X	1	1	" 3 Square wave generator
1	0	0	" 4 Software triggered mode.
1	0	1	" 5 Hardware " "

BCD 0 → binary counter 16 bits

1 → BCD counter (4 decades)

* Reading while counting:-

There is a command for latching the content of a counter to read its count while count is still going on. Bit pattern for cont. word for latching operation:

SC_1 SC_0 | RW_1 RW_0 | X X X X → don't care.

SC_1 SC_0 → Specify counter to be latched.

RW_1 RW_0 = 0 0 → counter latching operation

→ On receiving latch command:

↓ 8254 latches the content of selected counter into storage register;

↓ µp reads the content of register by using a normal read command.

↓ read operation is done as specified by RW_1 or RW_0 bits.

Operation: 8254 contain 3 counters, it is available of µp bit.

(1) counter 0 used for single step operation

(2) other two available for users.

used for various applications, like generating programmable baud rate while using with 8251 (microcontroller).

→ clk for 8253 should be less than 2.6 MHz whereas clk freq of 8085 is 3 MHz.

So, 7474 edge triggered F/F has been used to divide this clk freq by two. Thus a clk of 1.5 MHz is available on set, and so connected to clk terminal of particular counter.

* 8254 can be operated in diff modes.

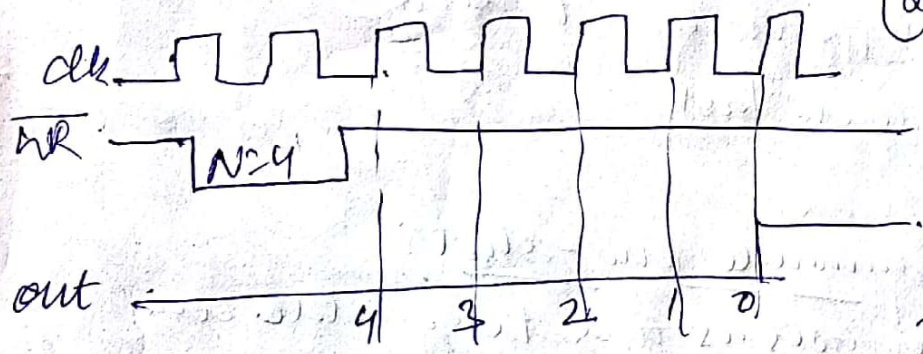
(1) Mode 0: Interrupt on terminal count.

→ used for generation of accurate time delay.

→ One of the counter is initialised with suitable count for the desired time delay.

→ When counting is over, the counter inte-CPU.

→ On interruption, µP performs the required task which is to be performed after the desired time delay.



(1) First load the control word into CWR.

(2) In this mode, the O/P of counter becomes initially low, after the mode is set.

(3) After mode selection, the count is loaded by count N.

(4) Count register is loaded, counting starts if GATE is high., for mode 0 operation, GATE is kept high.

(5) While counting is going on the counter O/P OUT remains low.

(6) When terminal count is reached count reaches 0, the O/p becomes high and remain high until count is reloaded or a new count is loaded.

(7) If gate made low, it stops counting, after that when gate = 1, it starts counting from the resumed value, The count value can be changed any time.

→ After loading new value, it restarts from the new value count.

if 1 byte, it starts counting immediately after loading the new value of the count.
if 2 bytes, counter stops counting after the 1st byte of the new count is loaded, the counter starts counting only after the 2nd byte of the new count is loaded.

→ OUT terminal intercept the pulse.

application: ① is used to generate precise time delay.

② It can be used when a computer wants to execute one task exactly 't' second after performing another task.

* Mode1: Programmable one-shot:

→ In mode 1, counter acts as a retriggerable and programmable one-shot.

→ 0 to 1 transition of signal applied to GATE acts as a trigger signal.

→ OUT becomes initially high,

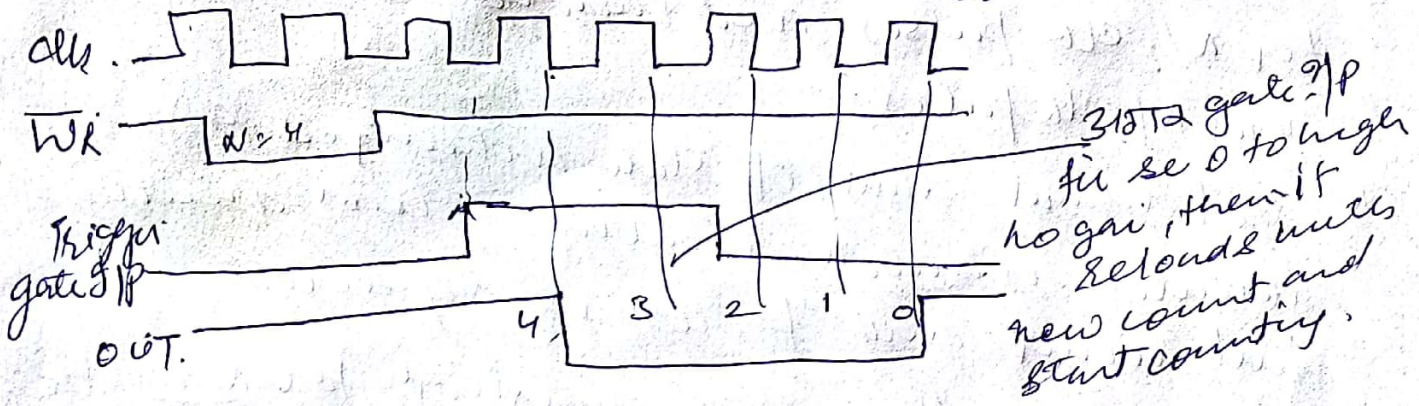
→ After mode set operation, the counter is loaded by a count value of N.

→ the counter decrements count, and OUT goes low for N clk cycles for every low to high transition of the GATE i/p.

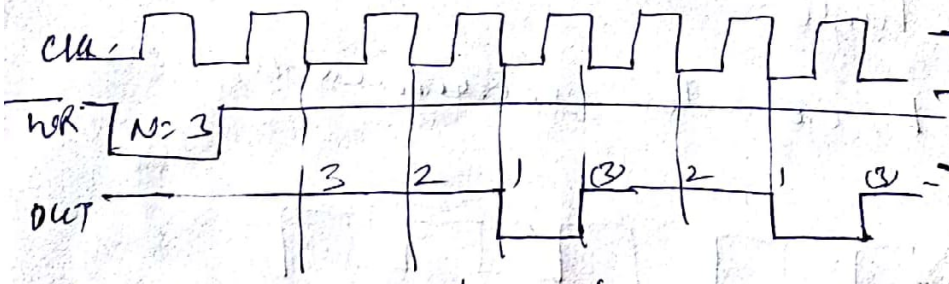
→ o/p goes low at first neg. edge of clk after the rising edge of gate i/p.

→ o/p remain low while counting is going on. It goes high on terminal count. Width of o/p pulse can be varied by changing N. The width of o/p pulse is prog. so Prog. one-shot

* If the GATE i/p is made low to high again, the counter is reloaded by count value N. The counter starts decrementing the count once again, the o/p goes low for N clk pulses again. Thus one shot mode of operation is retriggerable.



* Mode 2 :- Rate generator : In this mode, counter acts as a simple divide by N counter.



And then goes low for one clk pulse.

→ After this o/p becomes high again, and count N is automatically reloaded into counter.

→ again o/p remains high for (N-1) clk pulses and goes low for 1 clk pulse.

→ whole process repeated until gate i/p is 1.

→ If gate i/p = 0, disable counter and o/p stay high

→ When gate returns to high the counter resumes its normal operation and starts from initial count

* Mode 3 : Square Wave Generator (Counter acts as a)

(1) After mode set operation, the counter is loaded by count of value N.

(2) Gate is kept high.

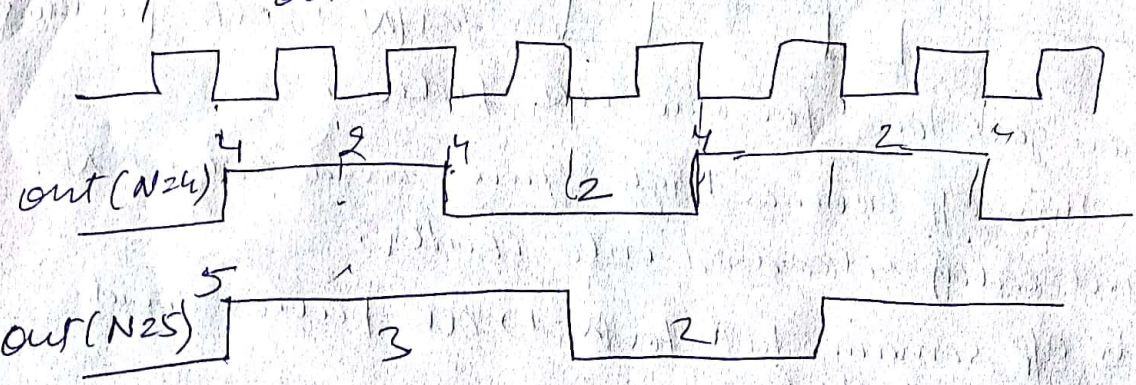
(3) For even values of N, the o/p remains high for N/2 clk pulses and then goes low for next N/2 clk pulses.

(4) On terminal count, o/p state is changed and counter is automatically reloaded with full count and process is repeated.

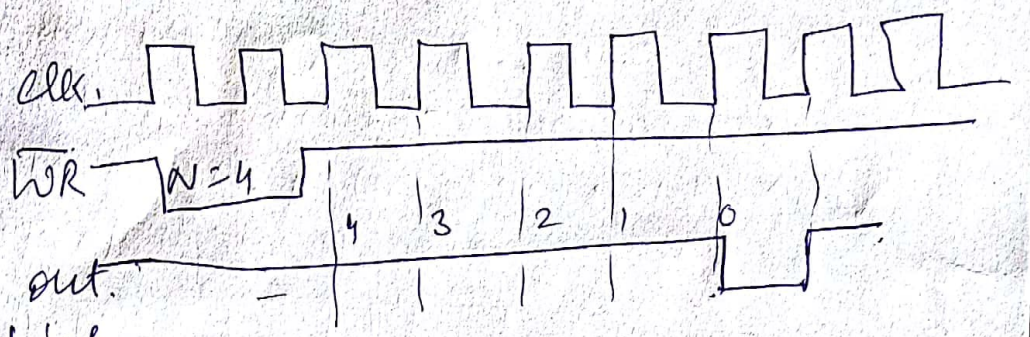
(5) For odd values, op remains high for $\frac{N+1}{2}$ clk period and low $\frac{N-1}{2}$ clk pulses.

(6) Counter is disabled, when Gate is made low and o/p stays high.

(7) When gate = 1, counter resumes its normal operation



* Mode 4: Software Triggered ~~Mode~~ Strobe



(1) Gate kept high.
(2) Initially, o/p high.
(3) begin counting after count is loaded.
(4) When count reaches terminal count, o/p goes low for 1 clk period; it returns to high then.
(5) The o/p signal may be used as strobe.
(6) This is refered to as Software triggered ~~Mode~~ Strobe because the generation of the strobe signal is triggered by loading the count into the count register.

* Mode 5 Hardware triggered Strobe:

(1) gate i/p acts as a trigger.
(2) After mode is set, the o/p becomes initially high.
(3) count value is loaded.
(4) counter starts decrementing, according to low to high transition of gate i/p.
(5) on terminal count the o/p goes low for 1 clk period, then it goes high again.