☑ Min^m mode ⇒ only one µP.
→ there are also chips (not µP)

☑ Max^m mode: there will be multiple µPs
i.e: 8087 & 8089

☑ Here you will also understands the 40 pins of 8086.

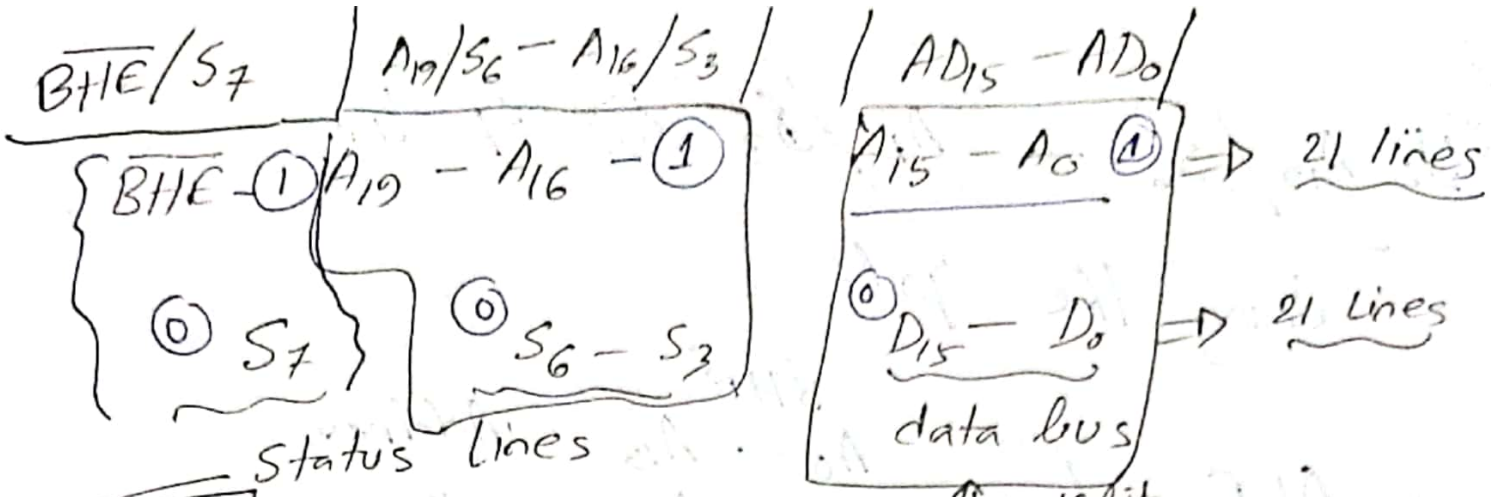
☑ The 1st pin MN/$\overline{MX}$.

☑ Multiplexing ? ⇒ combining 2 ~~sign~~ signals into 1. Adv: you reduce the number of lines.

→ In every operation: µP gives the signal via address bus for finding the location then transfers data; via multiplexing we are reducing 21 signals/pins: otherwise we had 61 pins.

☑

S_4	S_3	seg
0	0	ES
0	1	SS
1	0	CS
1	1	DS



S_4/S_3 indicating segment currently being used.

// S_5 => 0 => IF = 0 } S_5 tells about IF
1 => IF (by default)

// S_6 => 0 => 8086 BM
1 => other BM (max mode) } indicates who is controlling the bus

// S_7 = Reserved (still res)

the one controls the bus is called bus master.

1. ALE = Address latch Enable
ALE indicates $AD_{15} - AD_0$ lines are carrying data or address.

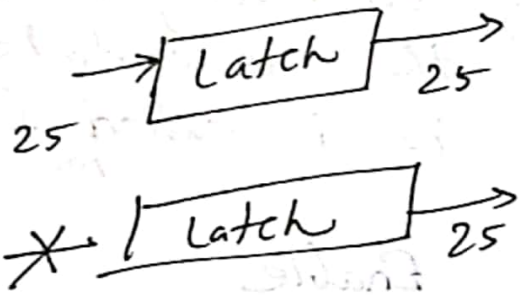
// ALE = 1, the $AD_{15} - AD_0$ bus carrying Add.
// ALE = 0, " " data.

A_{19}/S_6 — A_{16}/S_3 ⇒ 4 lines doing the job

// ALE = 1, $\overline{BHE}$ $A_0 - A_{15}$ Address
" $A_{16} - A_{19}$ "

// ALE = 0; $D_{15} - D_0$, $S_6 - S_3$, S_7 } data or status.

What is latch?



Add & data needs to be demultiplexed. since their working function are different
→ student comes into class via a single door, but seats in 5 sits

still they will give 25.

For example. AC remote control if we reduce the temp to 21; the latch will still hold the value 21. until you changed it again.

→ A bus just transfers values.
→ Latch holds the values.

→ MP calculates the result.

→ latch that is connected between MP & display will hold the result.

→ $\overline{OE}$ = output enable. ; formality.

STB = Strobbed is a single line. ($\overline{BHE}$ A_{19} - A_0)

ALE ⇒ STB = 1, (enables latch) is allowed to enter to the latch. (Flip-flop inside)

ALE ⇒ STB = 0. the entry to the latch shuts. the previously stored value will come out.

→ Any circuit of 8086 will need [3] 8282 latches since we have 21 lines. → (24 bits) 3 lines will be granted.

Now, we need pure data bus | 8286 DATA TRansreciever

→ A buffer. can be enable or disable.

→ When enable it works as a conductor.

→ When disable it goes into a state called high impedance state.

bidirectional buffer

→ i.e. it requires infinite resistance.
that means it works as an insulator.

→ tri-state ⇒ i.e. ~~neigl~~ neither at logic 1 nor logic 0.,
where buses carrying nothing.

→ if $\overline{OE}$ permanently grounded,
it will pass both add & data.

→ so, ~~if~~ we enable $\overline{OE}$ line only
when $\overline{DEN}$ = data enable ; data
~~is~~ has arrived.

$\overline{DEN} = 0$; $\overline{OE} = 0$.

DT/$\overline{R}$ ⇒ T pin $T = 1$, Tx mode
$T = 0$, Rx mode } DT/$\overline{R}$

/ $\underline{ALE = 1}$
21 lines carries address
Add. will go to 8286. but will do nothing

$\overline{DEN} = 1$. so data will not work.
//

// ALE = 0, $\overline{DEN}$ = 0

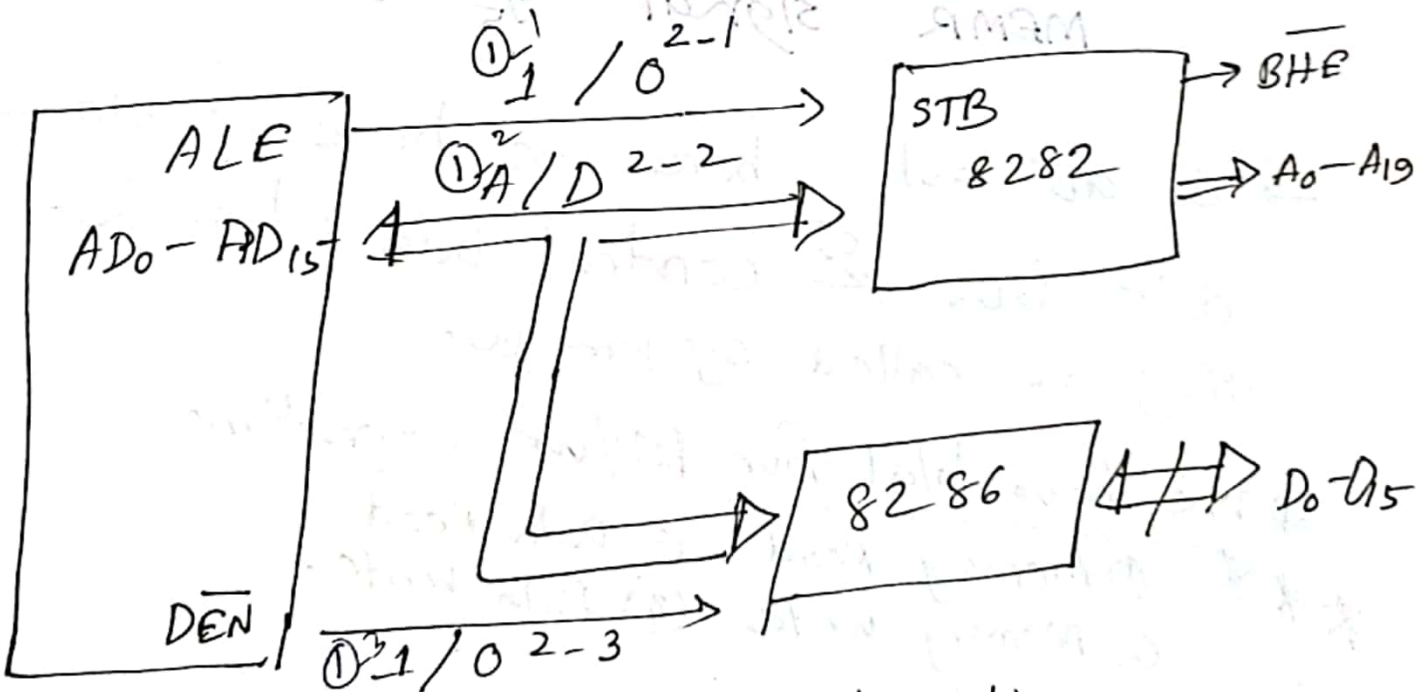
. data will go 8286, data enable will be activated, ~~but~~ DT/$\overline{R}$ depends. is it 0/1. But 8282 latch will be shut down.

So, ALE & $\overline{DEN}$ both are same. but their are not the same in timing diagram.

→ The Question is who will change 1st ?

• if $\overline{DEN}$ = 1, 1st,

21 lines carrying the address cannot ~~can~~ do anything inside 8286.



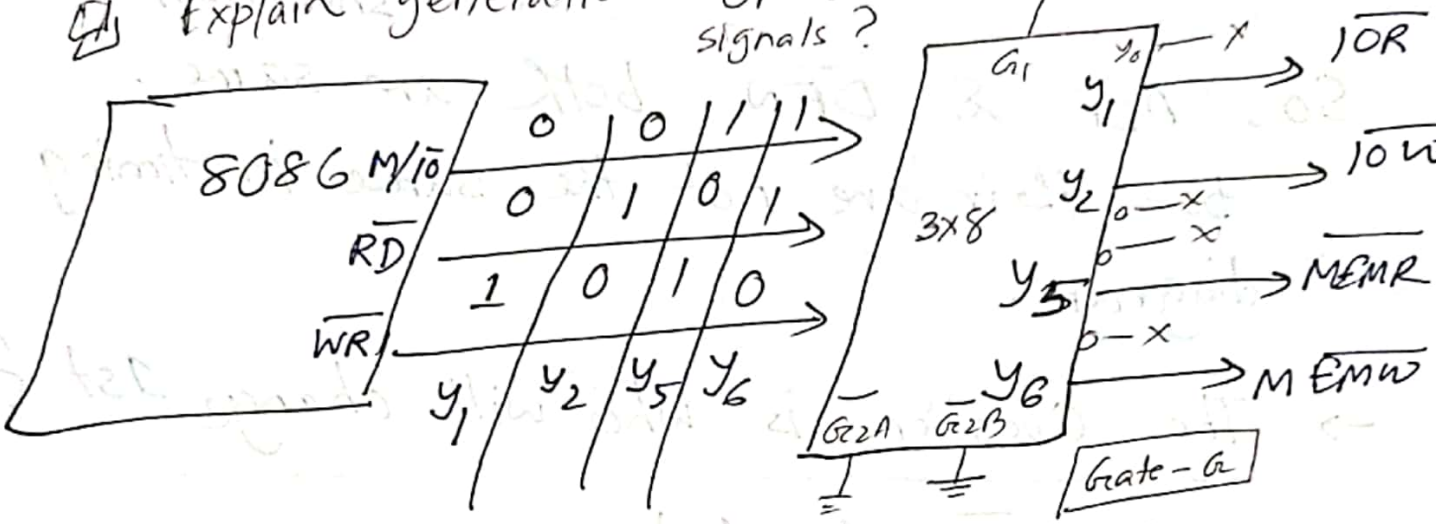
this is the demultiplexing of Add/data lines.

8286 → 2 ~~tatches~~

8282 → 3 latches

	$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$
IOR	0	0	1
IOW	0	1	0
MR	1	0	1
MW	1	1	0

Explain generation of control signals?



// $\overline{RD}$ → 0 1 0 1 only read operation.

// For RAM, read we should activate $\overline{MEMR}$ signal y_5 line.

→ So, at last; here we have a add, bus. data bus & control bus : together they are called system bus.

⇒ We have total four RD/WR operations

*** ① Memory Read ③ Data Read
② Memory write ④ Data write.

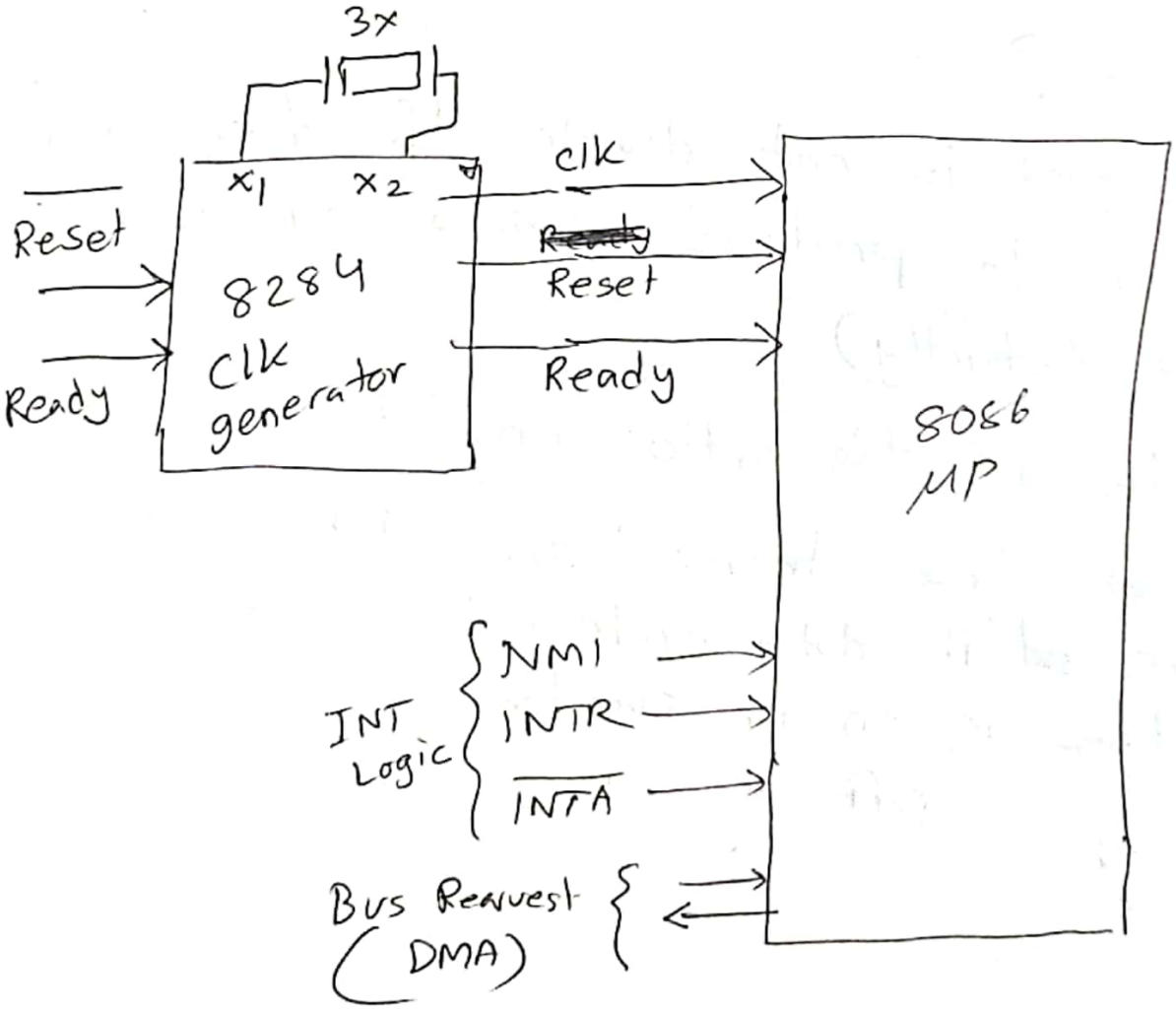
8284 clock generator

→ Any µP requires a clock.

→ Each clk pulse is a trigger to µP, to change the state or to do something new. → µP is a dead device (ಶವ ಥರ)

→ initiates a new activity in µP.

→ 8086 works at 6 MHz. (Standard value)



→ clk is connected to an oscillator.
via oscillation clock is produced.

→ 18MHz divided into 3 signals.
clk, Reset, Ready.

→ crystal oscillator is of 18 MHz gets inside & then divided into 6MHz.

→ Why we are conneeting 18 MHz instead of 6 MHz?

Ans: our goal is not divide the frequency by 3
our goal is to produce 6MHz at 33%
duty cycle (strictly)

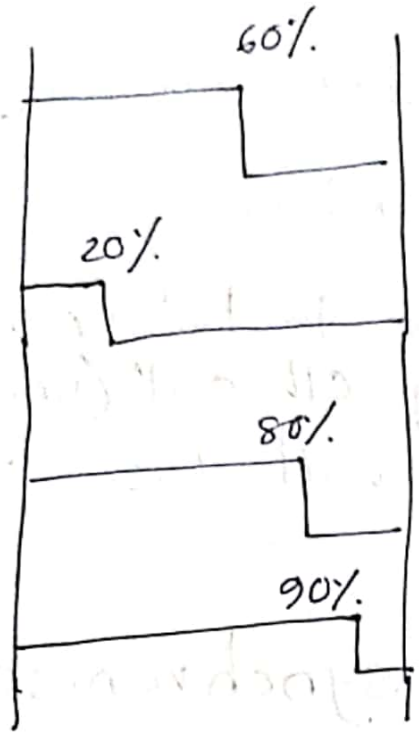
→ duty cycle is the ratio on total time

→ At what time transition takes place
define it duty cycle.

→ clk pulse is on for some time
off " "

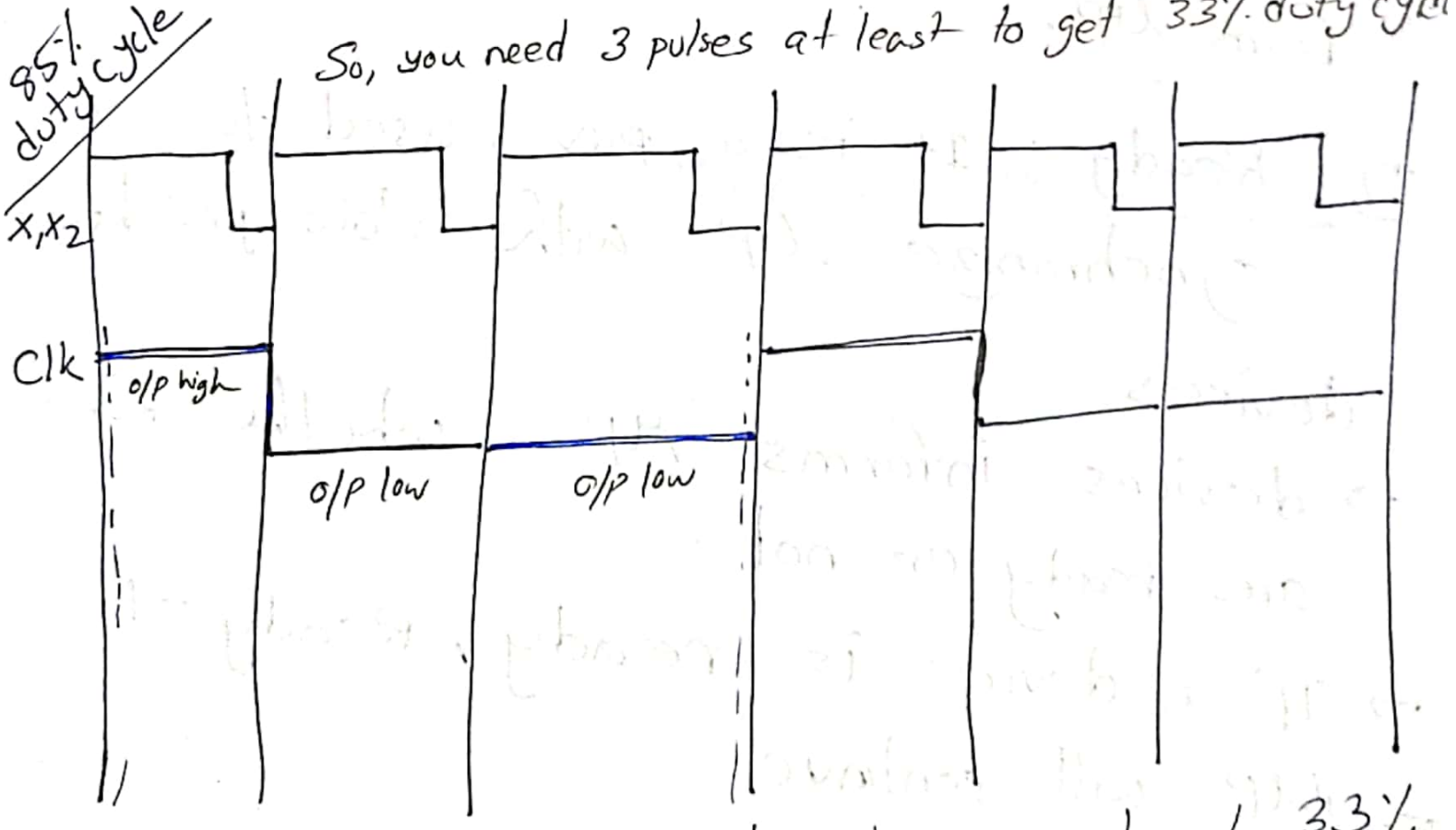
→

various duty cycle



All of them has same time period i.e. same frequency.

Now, we will convert random (85%) cycle into 33% duty cycle.



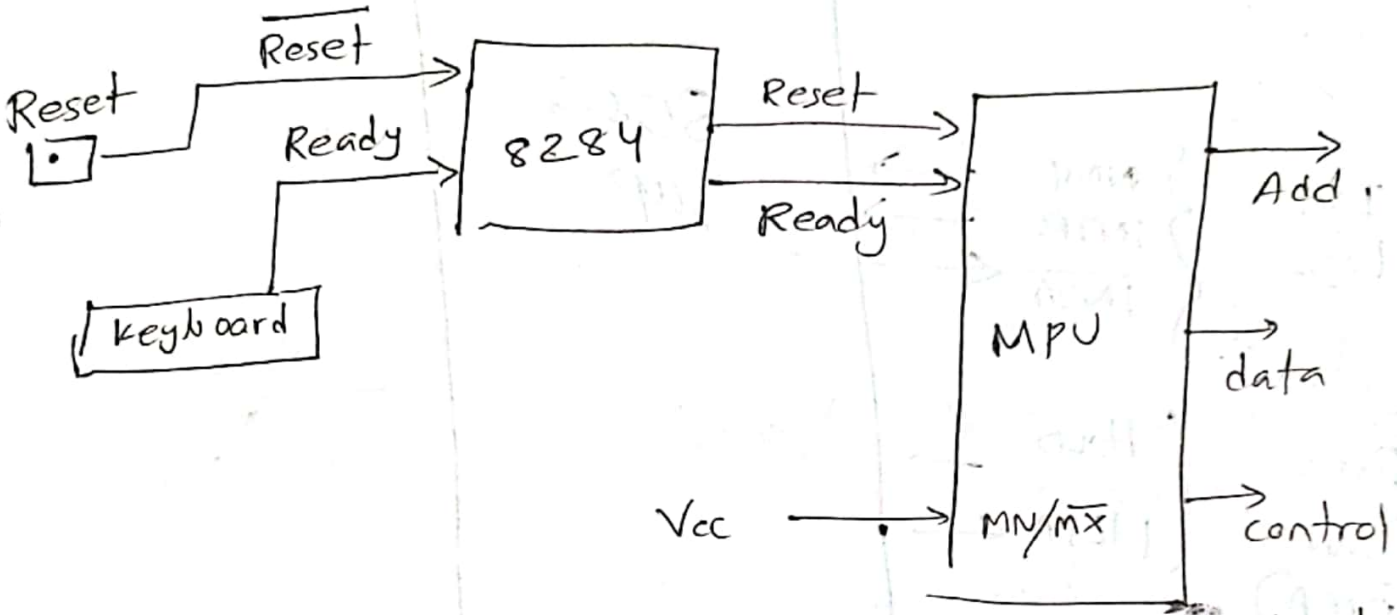
→ high for 1/3 time i.e. we have produced 33% duty cycle clk.

- Clk is given to MPU via 8284. MP does not distribute the clk pulses. As well as to other device eliminating clk out (unlike 8085)

- Reset: the signal will be applied to 8284.
  → 8284 will give synchronize signal to µp & other connected devices eliminating the "Reset out" signal from µp.

- Ready: It is a pin, used to synchronize µp with slow peripheral devices
  → devices informs µp, whether they are ready or not.
  → If a device is ready, Ready = 1.
  → µp will continue.

→ If READY pin = 0,
µp will enter into WAIT state & will keep waiting till the device is ready.



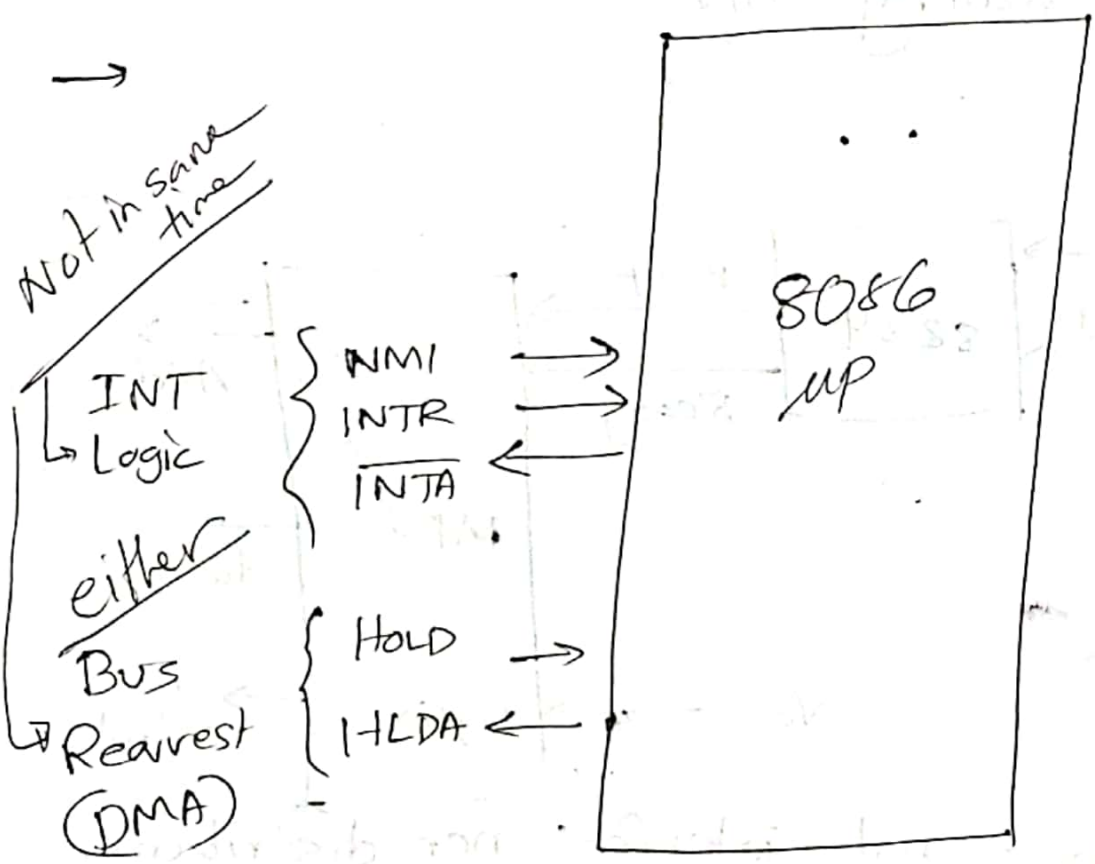
→ So, µp does not interfer nor distribute the Ready/Reset/clk signals.

→ This is the whole working procedure of Active µp in min^m mode.

→

Now, there are things may disturb µP.

→



→ either INT Logic or Bus Rearvest.

INT Logic

Whenever 8086 gets NMI, (NMI is vector) vector num is fixed which is

2.

→ INTR is non-vector. (not fixed)

→ so, µP doesn't know what to execute.

→ So, MP will ~~INTA~~ provide $\overline{INTA}$ via which MP will ask the vector number.

→ $\overline{INTA}$ is used to Ask for a vector numbe

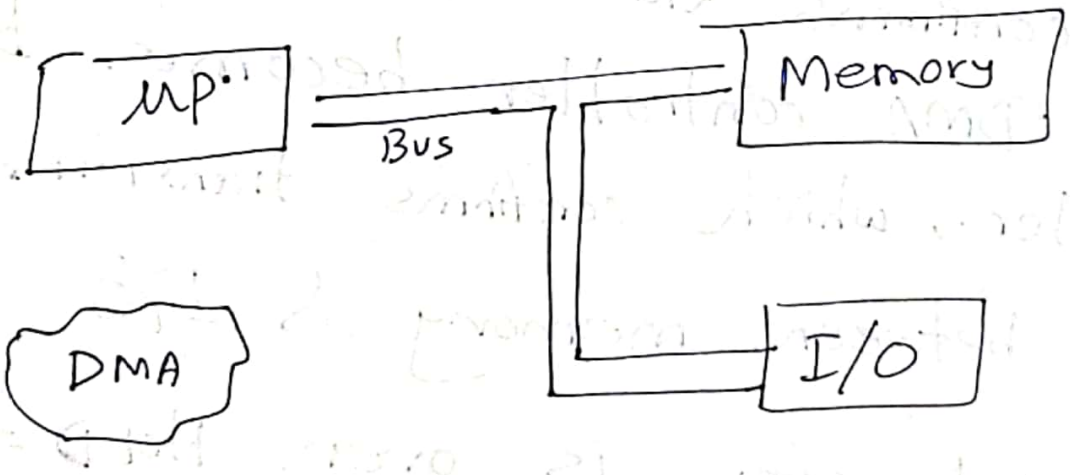
→ two $\overline{INTA}$ is given.

during 1st $\overline{INTA}$, device will calculate the vector number

during 2nd $\overline{INTA}$, device will give the vector number to MP for loading.

→ 2 $\overline{INTA}$ pulse is given back to back

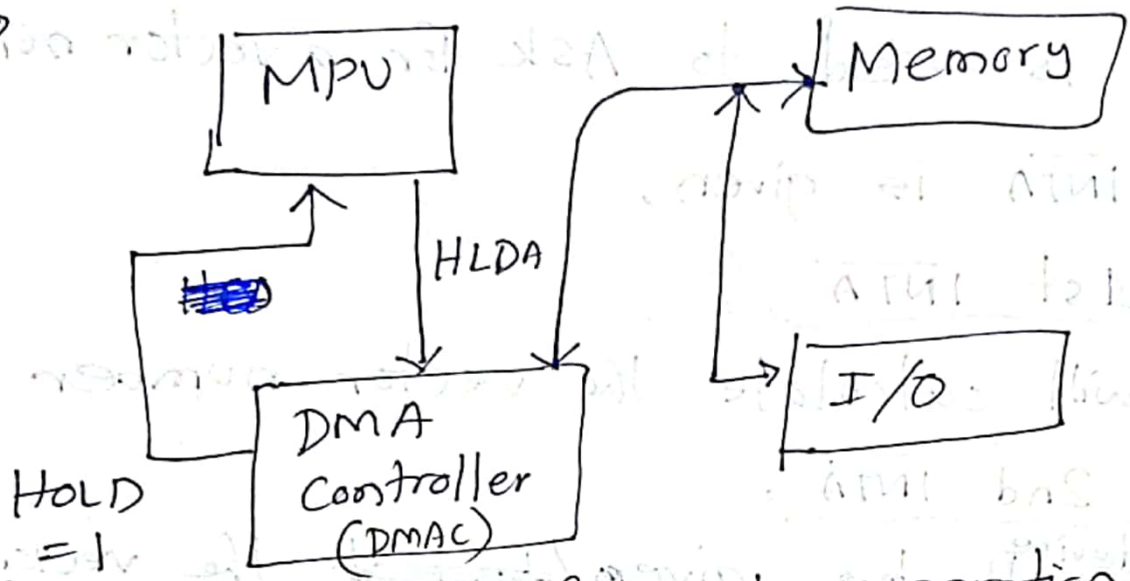
DMA : Direct Memory access.



→ DMA controller can be bus master.

→ But by default, MP is the bus master.

→



↳ telling MP to hold its operation.

→ Then, MP will release the control of the system bus,

→ MP confirms via HLDA.

→ Then DMA controller becomes bus master., which confirms transferring data between memory & I/O

→ When transfer is over, HOLD = 0, then MP becomes the bus master again.

MAXIMUM MODE

→ Multiple µPs operation.
→ 8087 arithmetic co-µp.
→ 8089 I/O µpu.
→ Connecting all together works more efficiently & creates excellent system. i.e. multiprocessing.,
→ cost is high.

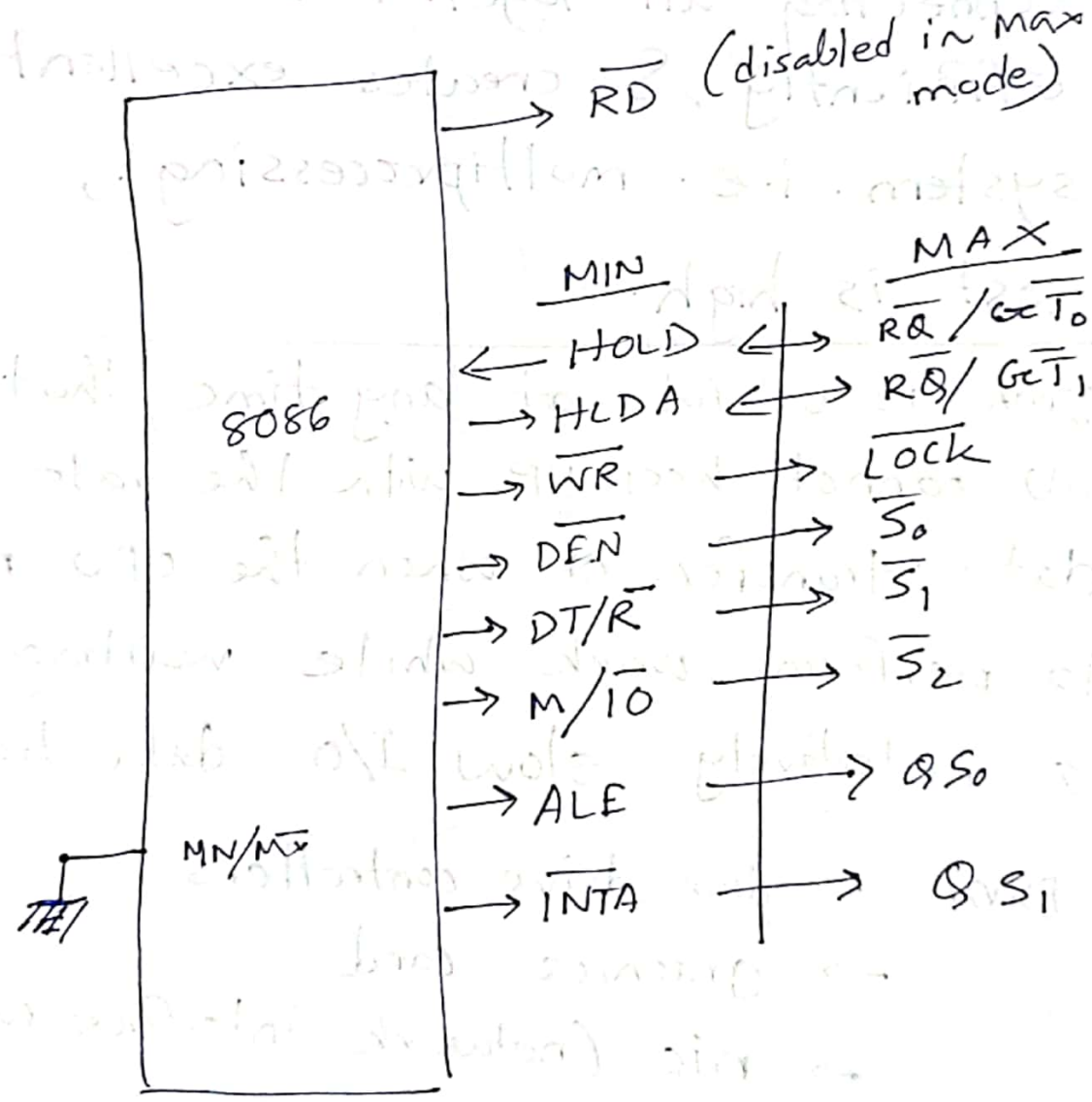
→ DMA is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer.

→ DMA ⇒ disk drive controllers.
→ graphics card
→ nic (network interface card)
→ sound cards.

Q. How does µP knows whether it is ② running in MN/MX ?

→ via pin MN/$\overline{MX}$.

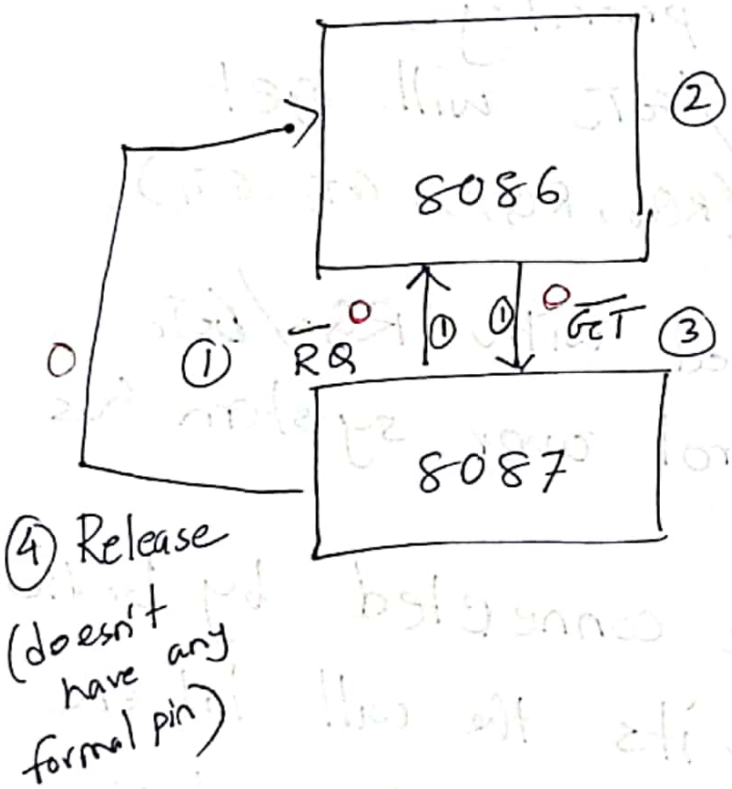
→ $\overline{MX}$ = 0



→ during MAX, apart from RD pin, all other 8 pins, changes their functionality. (3)

RQ/GT → Request & Grant

→ There is only one system bus, so all ~~the system~~ other µp cannot take control over the system bus at once.



→ All of them are active low signals.

→ since all of them are active low, so, all the communication will be done via sending low pulses. (0).

☐ Why they are bidirectional ?

☐ Why there are two RQ/GT pins ?

→ bec. 8086 can be connected with two μPos. (8087 & 8089)

☐ What will happen if the two co-Processors gives signals simultaneously?

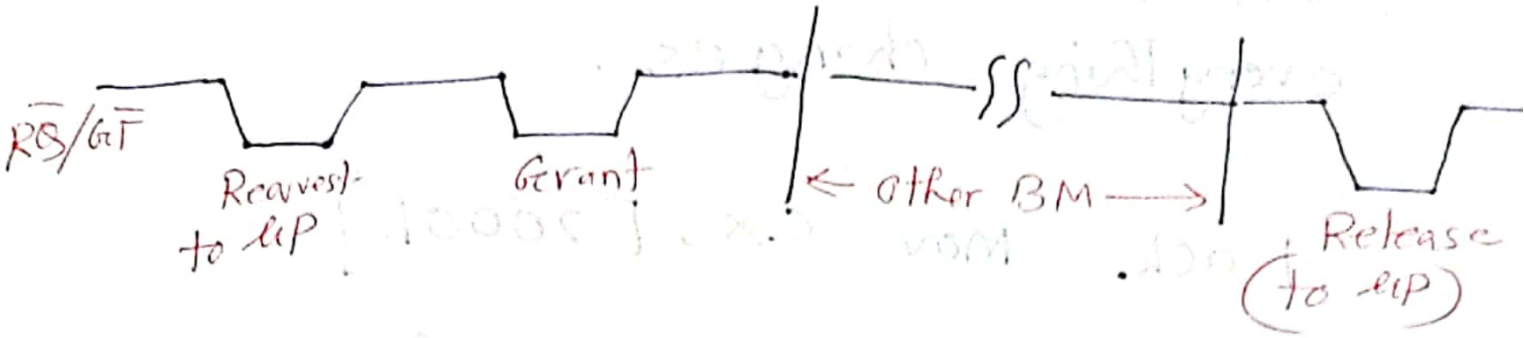
Ans! There will be a priority.

Here, there is RQ_0/GT_0 will get highest priority. (RQ_0, RQ_1 & GT_0, GT_1)

→ Whoever is connected with RQ_0/GT_0 will get the control over system bus first.

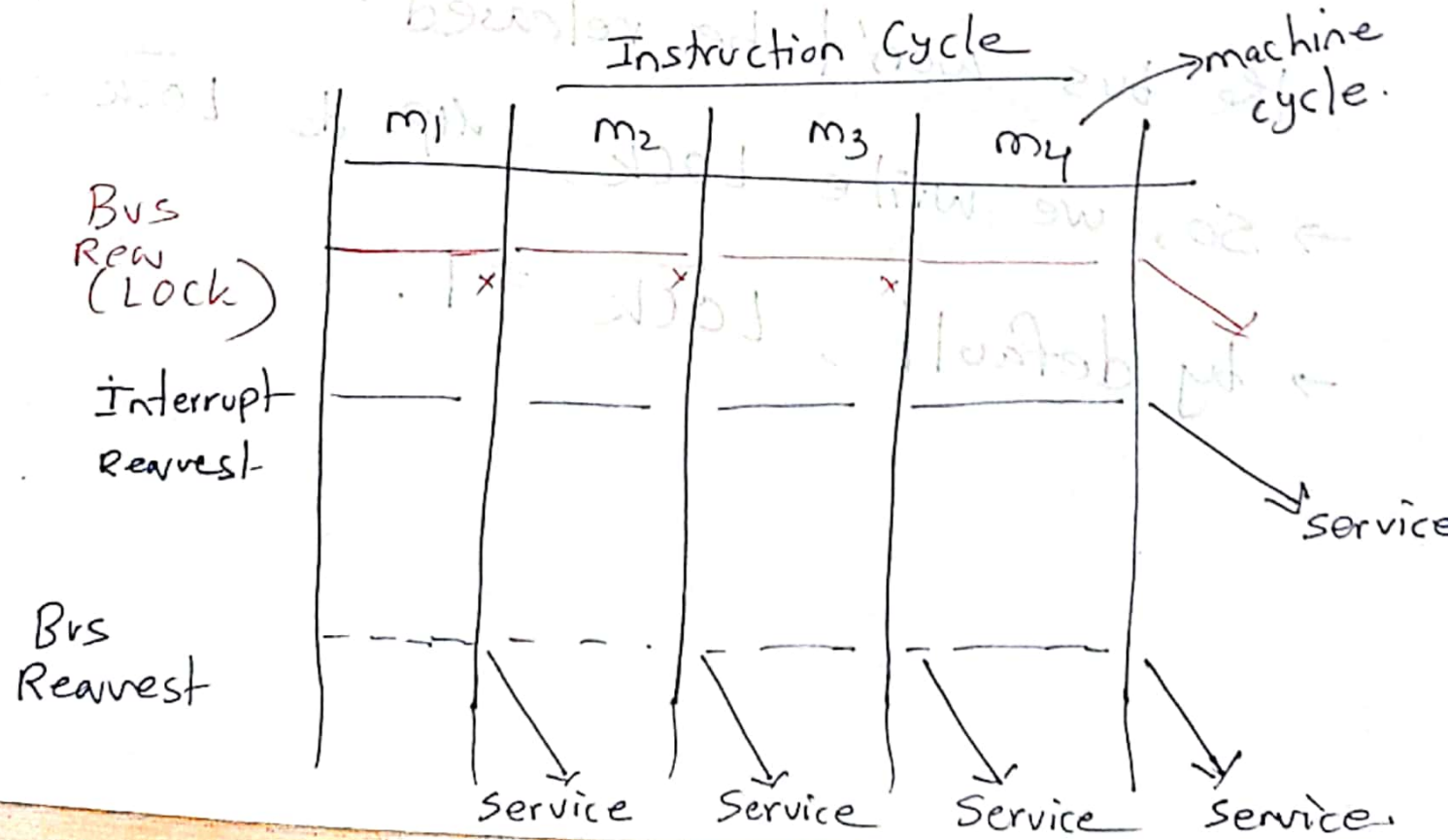
→ RQ_0/GT_0 - can be connected by both the co-Processor, its the call taken by programmer who is making this circuit.

→ But, 8087 is more important.



☐ $\overline{LOCK}$:

When µP gets a Bus reqvest, under normal circumstances, it releases the bus after finishing the current machine cycle. i.e! system bus control can be taken from µP within an instruction.



→ But if write the word $\overline{Lock}$, everything changes.

```
Lock  MOV  CX, [2000H]
```

→ The µp will release after finishing the whole instruction. i.e. you have Lock the bus. via activating $\overline{LOCK} = 0$

→ When we write Lock, two things happens:

① µp will not release the bus during an instruction cycle.

② It will inform all devices that the bus won't be released.

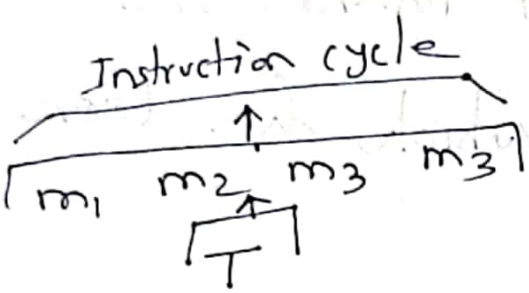
→ So, we write Lock, µp do $\overline{LOCK} = 0$.

→ by default, $\overline{LOCK} = 1$.

→ In a machine cycle there is 4 T-states.

→ In an instruction cycle, there is several machine cycle. (typically m_1, m_2, m_3, m_4)

⇒ T-states is a smallest thing part of machine cycle.



⇒ In one instruction, it requires several machine cycle & one machine cycle there are several T-states.

⇒ two events can disturb µp.
(1) Bus Req (2) Int Req.

⇒ Interrupt request can be ignored.

⇒ Bus " "
via $\overline{LOCK}$ only.

- When μP gets an INT Request, it will finish the current instruction cycle which has been ~~finishe~~ fetched, & then go to ISR & then come back to next instruction.

So, this is why it finishes the current instruction which has been fetched.

- During BUS REQUEST, μP release the bus & go to HOLD state; (it is like PAUSE & UNPAUSE a video)

→ So, during bus request, μP needs to complete the ~~cur~~ current machine cycle (not the whole instruction cycle)

In a machine cycle, there are 4- T states.

1st T state : µp gives the address
2nd " : µp " " READ signal.
3rd " : µp gets the data.
4th " : µp stores the data.

→ While 2nd state, µp gets a Bus request, it cannot release the Bus then, since 1st & 2nd T-states work will be destroyed : And the data will go to 8087 (if 8087 Req for bus control) but 8087 is not even interested for getting thais data. it doesn't even know. from where it is coming. So. therefore data will be lost. And ~~if~~ there is no chance µp can start over 3rd T-state. (T_3)

So, you cannot Breakdown a machine cycle.

$\overline{S_0}$ / $\overline{S_1}$ / $\overline{S_2}$: status signals.

→ 3 bits.

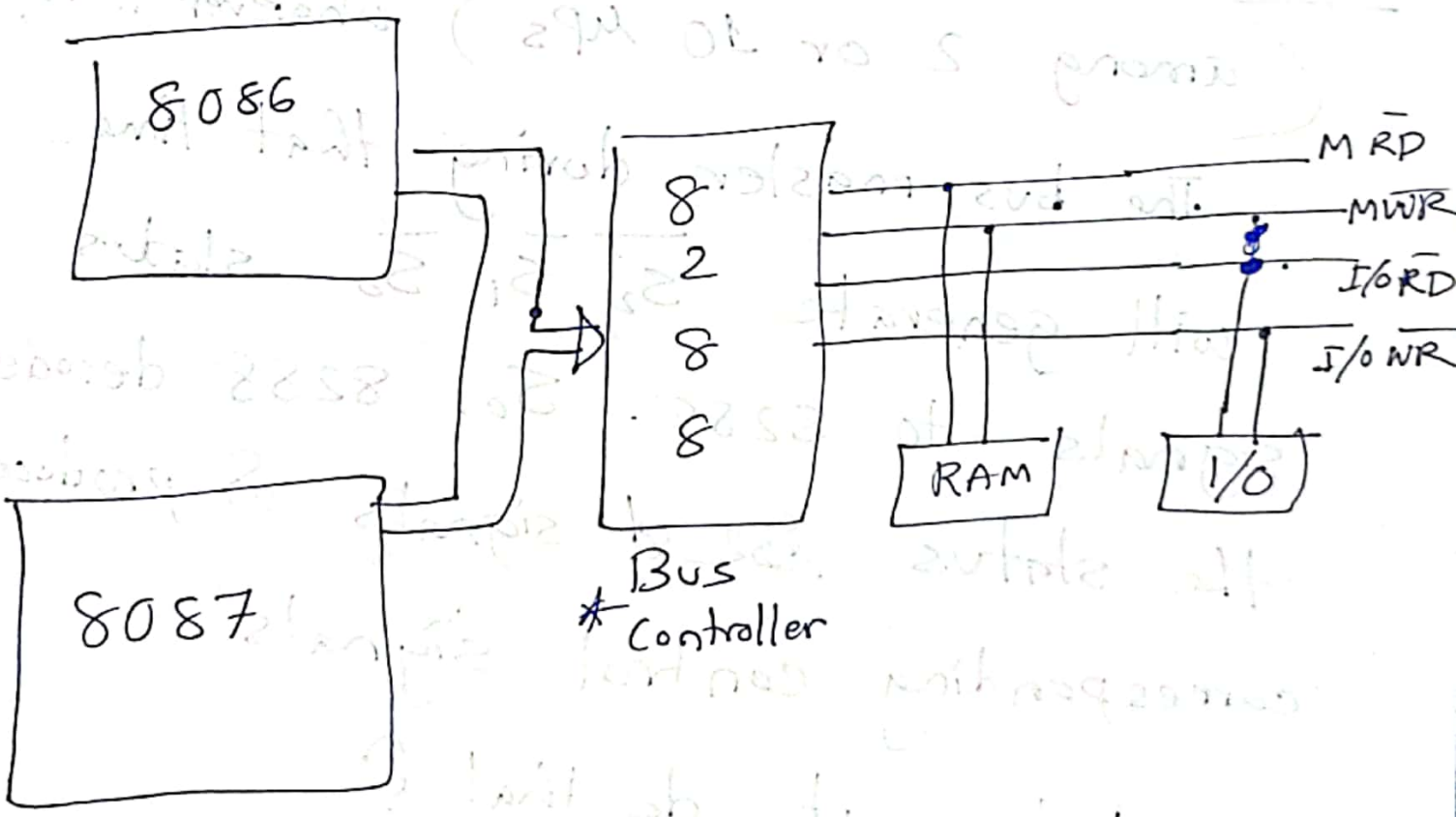
→ gives us 8 combinations.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0	0	0	INTA
0	0	1	I/O Read
0	1	0	I/O WRITE
0	1	1	HALT
1	0	0	Instruction fetch
1	0	1	Memory READ
1	1	0	MEMORY WRITE
1	1	1	IDLE

In Max.m mode, not 8086 nor 8087/8089 is allowed to generate control signals.

→ if we allow, one then authomatically all other µPs will also generate.

→ That will become harder circuit & generate bouquet of signals to generate



* is purposely designed to generate control signals

So, 8288 makes the circuit ~~easier~~, simple, since it will ~~go~~ takes the responsibility of generating control signals.

How does 8288 know which control signal has to be issued?

Ans: It knows via the bus master (among 2 or 10 MPs) whoever it is; The bus master during that time will generate $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$ status signals to 8288. $\overline{S_0}$, 8288 decodes the status ~~code~~ signals & produce corresponding control signals.

→ how does it do that?

via the table $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$

- HALT → No instruction (waiting)
- IDLE → µp is ON, but doing nothing.

Q.1 : What if both µPs gives the status signals at same time?

Ans: This will never happen. Giving the status signal is done by the bus master. Here, only one bus master can work at a given time. (Both cannot be) By default, 8086 is the Bus Master. If 8087 wants to be Bus Master, it will generate $\overline{RQ}_n$. 8086 will release & give $\overline{GT}_n$ signal. Then, 8087 will do whatever it wants to do & then it will release the bus.

Q.2 How does 8288 know who is the bus master at a given time?

Ans: ~~it will~~ . 8288 will never know & will never try to know. who is the bus master.

→ 8288's job is to decode the signal & generate control signals.

→ That is why 8288 is compulsory in max^m mode., without it there will be ~~==~~ NO control signals at all.

☑ $\overline{S_0}$ & $\overline{S_1}$: is useful while interfacing with 8087 (done in mid-2)

☑ There are 21 multiplexed lines

$AD_0 - AD_{15} + A_{16}/S_3 - A_{19}/S_6 + \overline{BHE}/S_7$

⇒ When $\underline{ALE = 1}$, they carry address

So, $A_0 - A_{15} + A_{16} - A_{19} + \overline{BHE}$

⇒ When $ALE = 0$, they carry data,

So, $D_0 - D_{15} + S_3 - 6_6 + S_7$.

→ So, afterwards data & addresses are demultiplexed : 8282 latch collects addresses & 8286 gets data.

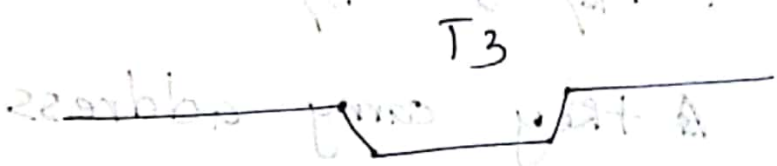
For latch we need $\overline{OE}$; which is grounded (Active low) 8282

☑ Now, for control signals.

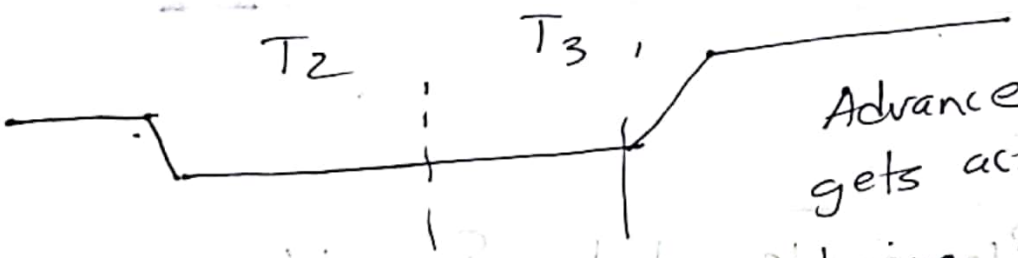
8p will give status signals $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$.

8288 will decode that status & produce all the control signals.

☑ $\overline{MWTC}$ & $\overline{AMWTC}$ ⇒ Advance memory write.



normal WRITE operation gets activated during T_3 state.



Advance WRITE gets activated during 2nd T. state.

So, 1-T-state advance. hence it is named as Advance WRITE.

☑ What is the advantage?

→ It is longer WRITE signal, so it is giving device more time to the slower devices, i.e. they get prolong period for the operation.

- Normal WRITE → for faster devices.
  Advance " → " slower devices.

- 8288 controls everything : i.e. all the system bus

- For 8282 latch,

  via ALE pin, 8288 tells the latch to capture the addresses (in min^m mode it was done by up)

But in Max^m mode, there are multiprocessors so, whoever is the bus master (at a given time) will give the address to the bus.

→ ALE is provided by 8288, not by any up. Otherwise circuit could have been more complex.

→ But, how does ALE know when to make ALE = 1 & ALE = 0 ?

☐ ALE = 1 during the 1st T-state of bus timing diagram.

→ Whenever a µp starts a new machine cycle; it will put address.

→ Suppose, 8086 is the bus master. it wants to do "memory read" operation.

→ at 1st T-state, it will put addresses on the address bus, at that time

ALE = 1 (8288)

So, How does it know? (To Generate ALE = 1)

Ans: Whenever it notices that some bus master is starting a new machine cycle. an operation is called a machine cycle.

→ with the help of status signal.

The moment 8288 notices

$\overline{S_2}\ \overline{S_1}\ \overline{S_0}$ = 101 (memory read)

ALE will become 1. (A new machine cycle)

II. How ALE becomes 0?

→ When the bus carries data, ALE should be 0.

→ A machine cycle has 4 T-states

→ only during 1st T-state, the system bus carries addresses. so, at that time ALE = 1.

→ 8288 always gets to know which machine cycle is going on? Which T-state (clock pulses) is active?

→ That is why clock is given from 8284 to 8288.

→ So, ALE can't be high for the whole T-states. (it has to high for only 1st T-state)

⇒ machine cycle is notified via $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$ signals

T-state " " " clock 8284 generaton

So, rest of T-states T_2 T_3 T_4

ALE = 0.

8286 Transceiver

→ it has to notified when to tx & Rx

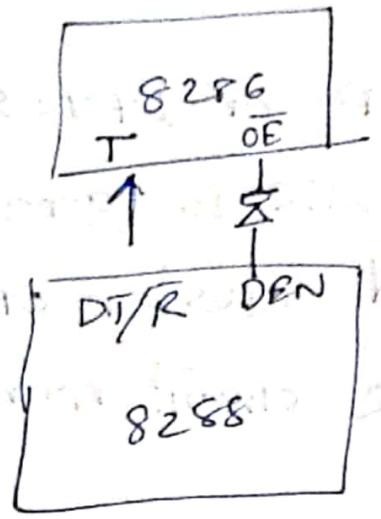
→ This is been notified via (T) pin from 8288 bus controller.

T stands for Transmit

T = 1 for tx (write)

T = 0, for Rx (Read)

→ In min mode it was generated by µP. If it happens, then we have to allow 8087 & 8089 to generate as well as a result we will have cluster of signals & hence circuit will be more complex. So, we have removed all & given to 8288 only.



→ remember, 8288 provides the signals, it does not decide.

→ It is decided by the bus master at a given time. by giving the appropriate status.

So, if $\bar{S}_2\ \bar{S}_1\ \bar{S}_0 = 101$ (memory read)

$\boxed{T = 0}$ Receive mode.

if $\bar{S}_2\ \bar{S}_1\ \bar{S}_0 = 110$ (memory write)

$\boxed{T = 1}$ Transmit mode

Since here, DEN is active high signal, should be put a NOT gate before going into 8286, so that it behaves as a Active low signal.

// So, we still need ALE or DEN signals. But, instead of µp, 8288 is generating since, we need to avoid cluster of signals which make the circuit more complex.

⇒ How does 8288 know which signal to produce?

Ans: the status lines $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ will tell.

⇒ How does 8288 know when to produce signal?

Ans: The clock generator will tell.

// INTR → goes in $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$

INTA → goes out via 000.

// Bus Reavues & Interrupt Reavest can disturb.

- In max. mode, bus request is done via other µPs.
  For that we have specialized signals
  $\overline{RQ_n}$ / $\overline{GT_n}$.

- $\overline{TEST}$ $\overline{QS_1}$ $\overline{QS_0}$ } only for 8087. can use them. (not by 8089).
  - used for to synchronized the prefetch 6-byte of instructions.