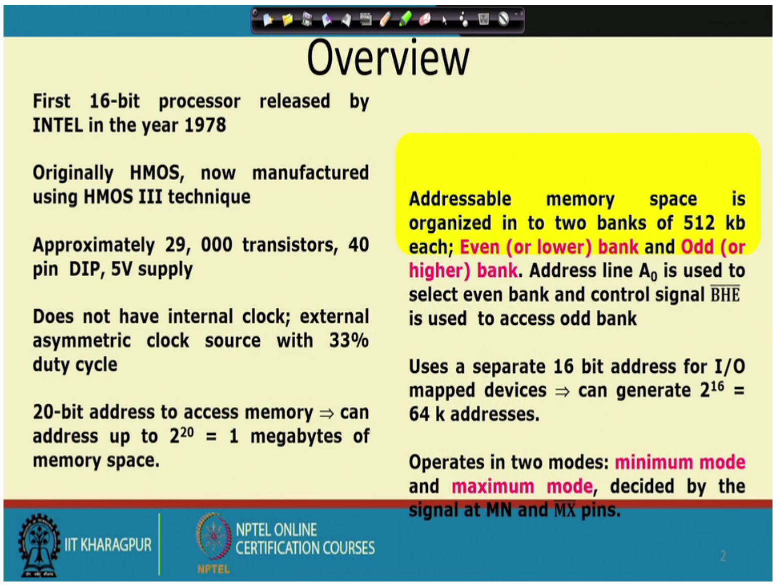
Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture – 63
8086 Microprocessor

So, next we will be looking into another microprocessor which is slightly more complex compare to 8085, which is the next family of microprocessor from INTEL known as 8086.

(Refer Slide Time: 00:28)



Overview

First 16-bit processor released by INTEL in the year 1978

Originally HMOS, now manufactured using HMOS III technique

Approximately 29, 000 transistors, 40 pin DIP, 5V supply

Does not have internal clock; external asymmetric clock source with 33% duty cycle

20-bit address to access memory ⇒ can address up to 2^{20} = 1 megabytes of memory space.

Addressable memory space is organized in to two banks of 512 kb each; Even (or lower) bank and Odd (or higher) bank. Address line A_0 is used to select even bank and control signal $\overline{BHE}$ is used to access odd bank

Uses a separate 16 bit address for I/O mapped devices ⇒ can generate 2^{16} = 64 k addresses.

Operates in two modes: minimum mode and maximum mode, decided by the signal at MN and $\overline{MX}$ pins.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the first 8086 process first 16 bit processor, 8086 is the first 16 bit processor. So, internal processing is all in terms of 16 bit data unlike 8085 where it was all 8 bit operation. So, here all the operations are 16 bit operation, so it was originally release by INTEL in 1978, it was designed on HMOS technology, but later on it was upgrade it to HMOS 3 technology.

So, is a roughly 29000 transistors there is a 40 pin dual in line package. So, it I like 8085 only it have 5 volt power supply, does not have internal clock. So, this external asymmetric clock source is 33 percent duty cycle, so you should have you have to collect connect the clock from the external source.

So, unlike 8085 where you can just connect a crystal and it acts as the it can generate the clock internally in 8086 that is not possible. So, you have to give clock signal externally with 33 percent duty cycle means 33 percent of the time the clock signal is high and 67 percent of the time the clock signal is low, 20 bit address.

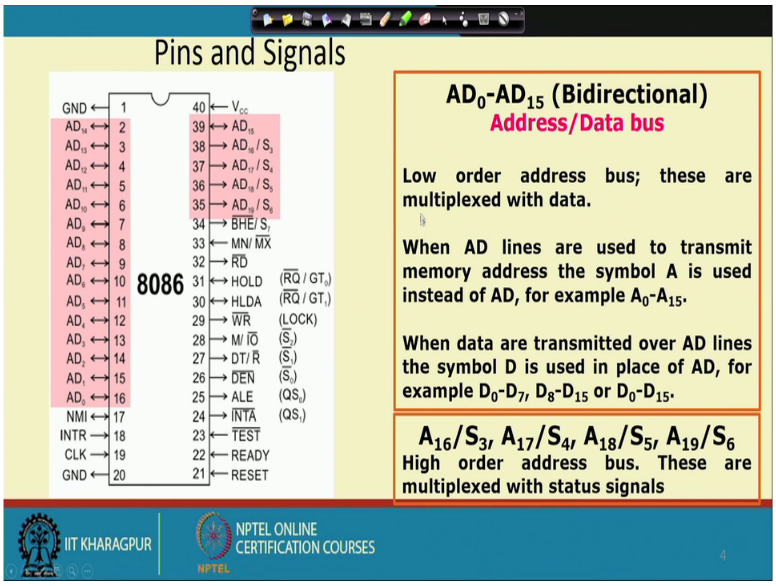
So, address bus in case of 8085 was 16 bit in case of 8086 it is 20 bit address. So, naturally you can access address up to 2 power 10 there is 1 mega byte memory space. So, addressable memory space is organized into two banks of 512 kilo byte each. So, one is the lower bank that which is the even on lower bank and odd or a higher bank.

So, address line A0 can be used to select even bank and control single BHE bar is used to access the odd bank, so these are special signal that are available. So, that way we can we can divide this memory into two 512 kilobyte blocks. So, there is A 16 bit address space for the I O mapped devices unlike 8085 we had 8 bit I O address. So, here we have got 16 bit I O address. So, total number of I O devices that can be connected is 2 power 16 that is 64 kilo. So, many addresses are available, so we can connect from any devices.

Another important thing that this 8086 has is that it operates in 2 modes one is known as minimum mode another is known as maximum mode. So, minimum mode operation is more or less similar to 8085 type of operation, where this process are happens to be the master and you can connect a number of you can connect memory chip and other peripheral devices to it.

However, in many systems what is required is that we have multiple master, so the they co ordinate between themselves to get some operation done then gets some job done. So, that is the maximum mode of operation. So, there is a pin called min max bar so, if this pin is equal to 1, so the system will operated in minimum mode and if it is 0 then it will operate in the maximum mode.

(Refer Slide Time: 03:28)



Pins and Signals

GND 1 | 40 V_{CC}
AD_{14} 2 | 39 AD_{15}
AD_{13} 3 | 38 AD_{16}/S_3
AD_{12} 4 | 37 AD_{17}/S_4
AD_{11} 5 | 36 AD_{18}/S_5
AD_{10} 6 | 35 AD_{19}/S_6
AD_9 7 | 34 $\overline{BHE}/S_7$
AD_8 8 | 33 MN/$\overline{MX}$
AD_7 9 | 32 $\overline{RD}$
AD_6 10 | 31 HOLD ($\overline{RQ}/GT_0$)
AD_5 11 | 30 HLDA ($\overline{RQ}/GT_1$)
AD_4 12 | 29 $\overline{WR}$ ($\overline{LOCK}$)
AD_3 13 | 28 M/$\overline{IO}$ ($\overline{S_2}$)
AD_2 14 | 27 DT/$\overline{R}$ ($\overline{S_1}$)
AD_1 15 | 26 $\overline{DEN}$ ($\overline{S_0}$)
AD_0 16 | 25 ALE (QS_0)
NMI 17 | 24 $\overline{INTA}$ (QS_1)
INTR 18 | 23 $\overline{TEST}$
CLK 19 | 22 READY
GND 20 | 21 RESET
8086

AD_0-AD_{15} (Bidirectional)

Address/Data bus

Low order address bus; these are multiplexed with data.

When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A_0-A_{15}.

When data are transmitted over AD lines the symbol D is used in place of AD, for example D_0-D_7, D_8-D_{15} or D_0-D_{15}.

A_{16}/S_3, A_{17}/S_4, A_{18}/S_5, A_{19}/S_6

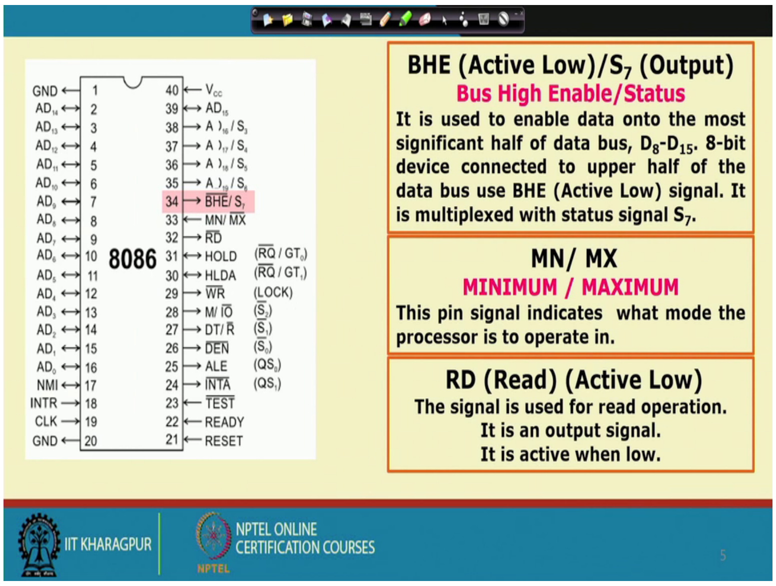
High order address bus. These are multiplexed with status signals

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, as usual we will be starting with the pins and signal. So, we have got this AD 0 to AD 15 which is bidirectional address data bus, so this AD 0 to AD 15 ok. So, these are the address data bus multiplexed address data bus, so just like in 8085 we had got this lower order address data bus was multiplexed, here entire address data bus is multiplexed. The AD lines are used to transmit memory address the symbol A is used instead of AD for example, A 0 to A 15. So, when data is transmitted so we will be reading as D lines, so AD D0 to D7 D8 to D 15 or D 0 to D 15 like that.

So, the rest of the since this is a 20 bit address space as we said, so total 20 address lines are necessary; So, this AD 16 to AD 19 so though they are written as AD, but you can better read them as A 16 to A 19 because, there is no data bus line like D 16, D 17, D 18, D 19. So, data bus is 16 bit only it ends at D 15, so you can be better read them A 16 by S 3. So, the they are also multiplexed, but it is multiplexed with the status line S 3 S 4 S 5 and S 6.

(Refer Slide Time: 04:46)



BHE (Active Low)/S_7 (Output)

Bus High Enable/Status

It is used to enable data onto the most significant half of data bus, D_8-D_{15}. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S_7.

MN/ MX

MINIMUM / MAXIMUM

This pin signal indicates what mode the processor is to operate in.

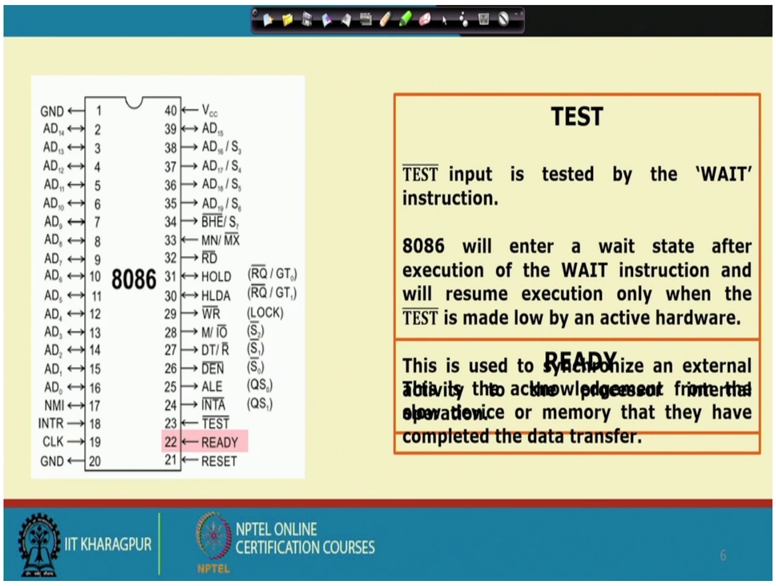
RD (Read) (Active Low)

The signal is used for read operation.
It is an output signal.
It is active when low.

There is another signals which is BHE bar which is bus high enable or the S 7 is the status BHE stands for bus high enable, it is used to enable data onto the most significant half of the data bus. So, D 8 to D 15 so when we want to say that we have got this B the data is available on the higher order data bus D 8 to D 15, in that case this BHE bar signal will be low telling the process device which is connected to D 8 to D 15 that data is available there. So, this that that is what is written here that 8 bit device connected to upper half of the data bus can use this BHE signal which is active low and it is multiplexed with the status signal S 7, so BHE bar is multiplexed with S 7.

Then there is a min max bar pin so these 1 pin number 33 min max bar pin. So, it will indicate whether the processor is in minimum mode or in maximum mode then there is a read bar line which is active low. So, it when it is doing a memory read or I O read type of operation then this read bar line will be low, then is another important line which is known as test.

(Refer Slide Time: 05:56)



GND 1 · AD14 2 · AD13 3 · AD12 4 · AD11 5 · AD10 6 · AD9 7 · AD8 8 · AD7 9 · AD6 10 · AD5 11 · AD4 12 · AD3 13 · AD2 14 · AD1 15 · AD0 16 · NMI 17 · INTR 18 · CLK 19 · GND 20
8086
40 Vcc · 39 AD15 · 38 AD16/S3 · 37 AD17/S4 · 36 AD18/S5 · 35 AD19/S6 · 34 BHE/S7 · 33 MN/MX · 32 RD · 31 HOLD (RQ/GT0) · 30 HLDA (RQ/GT1) · 29 WR (LOCK) · 28 M/IO (S2) · 27 DT/R (S1) · 26 DEN (S0) · 25 ALE (QS0) · 24 INTA (QS1) · 23 TEST · 22 READY · 21 RESET
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

TEST

$\overline{\text{TEST}}$ input is tested by the 'WAIT' instruction.

8086 will enter a wait state after execution of the WAIT instruction and will resume execution only when the $\overline{\text{TEST}}$ is made low by an active hardware.

This is used to synchronize an external activity to the processor internal operation.

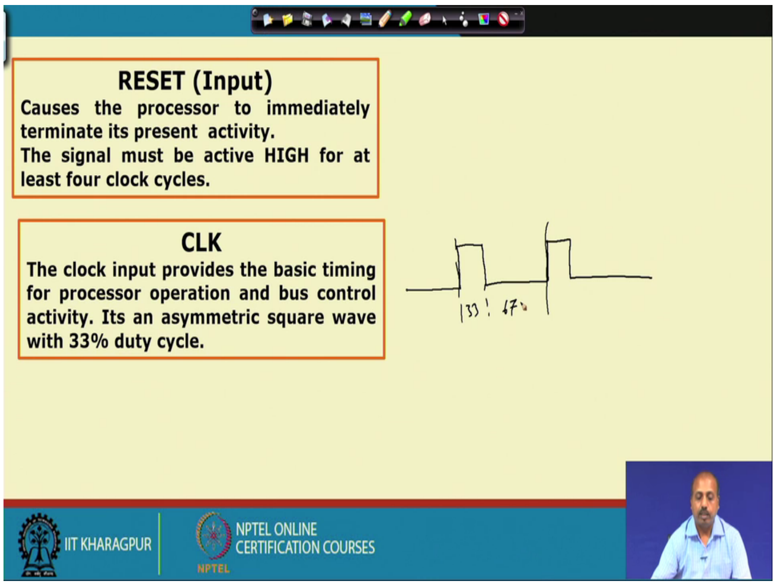
READY

This is the acknowledgement from the slow device or memory that they have completed the data transfer.

So, this test line so test bar pin is used tested by the wait instruction. So, sometime what happens is that while there are multiple there are many operations that are going on in the system, so we need to wait for some event or some particular operation to be over. So, in that case we the processor 8086 has got a special instruction for called wait and what this wait instruction does is that it suspends the operation of this processor till this test bar line becomes equal is made low.

So, this 8086 will enter into a wait state after execution of this wait instruction and we will resume execution only when test bar is made low by some active hardware. So, that way it is this is there is read bar there is a ready signal, ready signal is used for external memory can tell whether the some device is slow or not then it will be the ready single, so it will extend the memory operation.

(Refer Slide Time: 07:04)



RESET (Input)

Causes the processor to immediately terminate its present activity.
The signal must be active HIGH for at least four clock cycles.

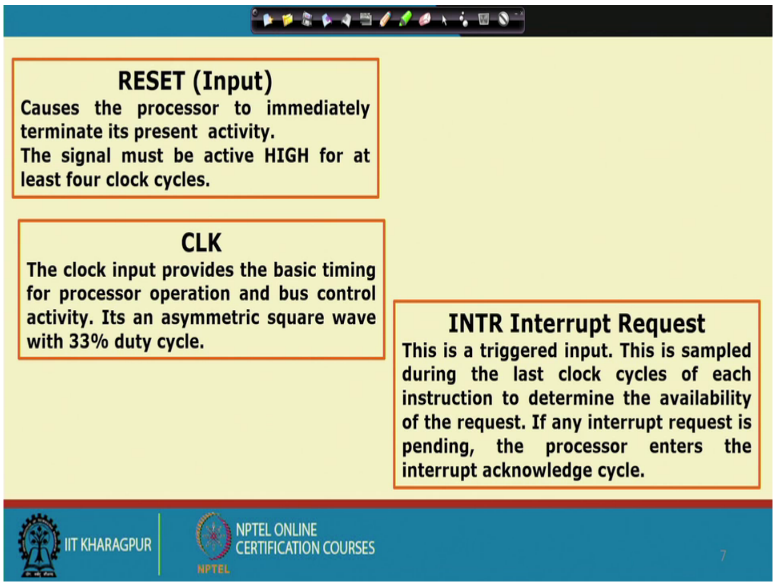
CLK

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

Then there is a reset signal which causes the processor to immediately terminate it is present activities, so just like 8085 reset signal. The signal must be active high for at least 4 clock cycles to be acknowledged because, if it is not so then maybe it is a very small amount of time and the reset was actually not intended just a spike came. So, just we avoid that situation so it is the caped it is it is required that the reset should be active for 4 clock cycles.

Then the clock we have already said the clock will provide the basic timing for the processor operation and bus control activity, asymmetric square wave with 33 percent duty cycle. So, it is like this so this is the 33 percent of the time the clock signal is high and 37 percent of the time the clock signal is low. So, it is like this so if this is the total time period from here to here, so this is the 33 percent and this is 67 percent. So, 36 percent the signal should be low anyway so that is how this clock signal should be provided.

(Refer Slide Time: 08:15)



RESET (Input)

Causes the processor to immediately terminate its present activity.
The signal must be active HIGH for at least four clock cycles.

CLK

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

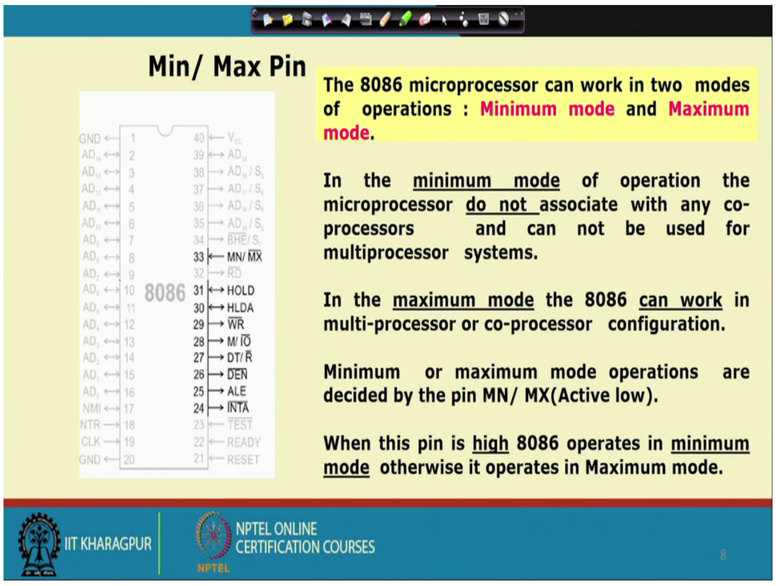
INTR Interrupt Request

This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, there is one interrupt line unlike 8085 which where you have got a number of interrupt lines, so here we have got only 1 INTR pin which is interrupt request. So, this is the triggered input and sample during the last clock cycle of each instruction to determine the availability of the request. So, as true for 8085 also so this interrupt line is sampled on the last clock cycle of an instruction execution. So, that way it decides whether an interrupt is interrupt has arrived or not and if an interrupt has arrived, so it will go into a interrupt acknowledged cycle.

(Refer Slide Time: 08:54)



Min/ Max Pin

The 8086 microprocessor can work in two modes of operations : Minimum mode and Maximum mode.

In the <u>minimum mode</u> of operation the microprocessor <u>do not</u> associate with any co-processors and can not be used for multiprocessor systems.

In the <u>maximum mode</u> the 8086 <u>can work</u> in multi-processor or co-processor configuration.

Minimum or maximum mode operations are decided by the pin MN/ MX(Active low).

When this pin is <u>high</u> 8086 operates in <u>minimum mode</u> otherwise it operates in Maximum mode.

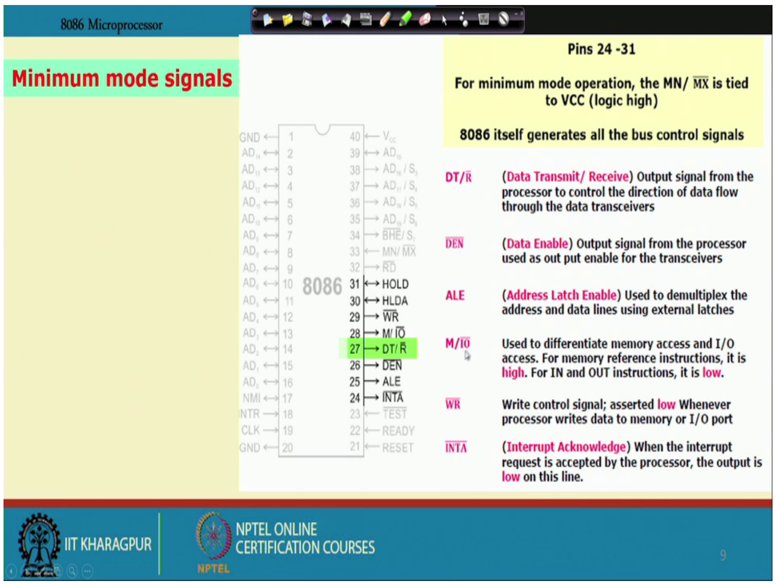
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, coming to this min max pin, so this min max pin there are 2 modes of operation for 8086 1 is known as minimum mode another is known as maximum mode. So, minimum mode is just like a single processor system, so in minimum mode of operation the microprocessor do not associate with any coprocessors and cannot be used for multiprocessor system. So, if there are multiple processor, so we get a multi processor system many times this 8086 it can connect some coprocessor with it; for example, there is a very well known coprocessor 8087 which is a math coprocessor.

So, that whatever mathematical operations are required so 8086 will transfer those instructions to 8087 for execution that way operation can be made faster. So, while doing those operations the coprocessor also needs to access the memory and other, so we have to give the control of the bus through that device in that case or through that coprocessor in that case. So, in minimum mode of operation, so this is not possible here in the 8086 itself is the sole master of the whole system.

In a maximum mode of operation so it can 8086 can work in multi processor or coprocessor configuration and minimum maximum mode is configured by this min max bar pin. So, this min max bar line being equal to 1, so it will min it is the minimum mode and it is it is equal to 0, so it is a maximum mode of operation.

(Refer Slide Time: 10:25)


8086 Microprocessor

Minimum mode signals

Pins 24 -31

For minimum mode operation, the MN/ $\overline{MX}$ is tied to VCC (logic high)

8086 itself generates all the bus control signals

DT/$\overline{R}$	(Data Transmit/ Receive) Output signal from the processor to control the direction of data flow through the data transceivers
$\overline{DEN}$	(Data Enable) Output signal from the processor used as out put enable for the transceivers
ALE	(Address Latch Enable) Used to demultiplex the address and data lines using external latches
M/$\overline{IO}$	Used to differentiate memory access and I/O access. For memory reference instructions, it is high. For IN and OUT instructions, it is low.
$\overline{WR}$	Write control signal; asserted low Whenever processor writes data to memory or I/O port
$\overline{INTA}$	(Interrupt Acknowledge) When the interrupt request is accepted by the processor, the output is low on this line.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

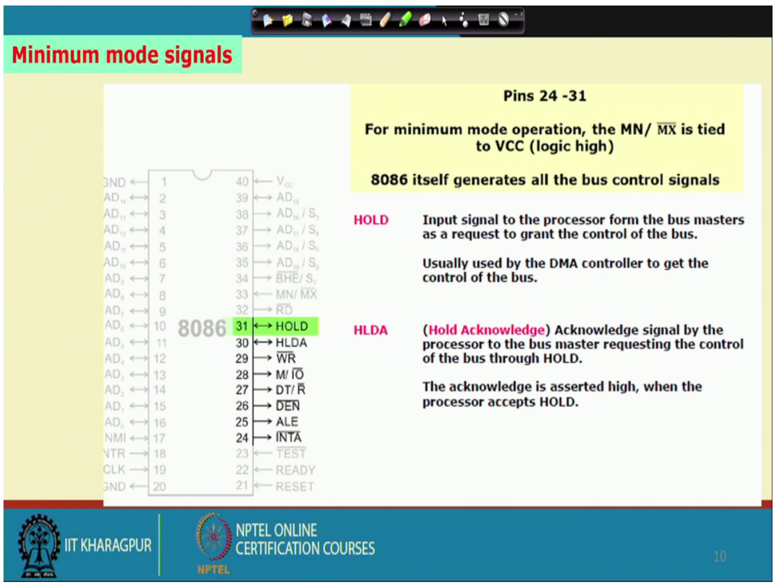
9

So, minimum mode signal, so these are the important signals that we have in minimum mode of operation 1 is DT R bar. So, DT R bar is the data transmit or receive. So, this is the output signal process from the processor to control the direction of dataflow through the data transceiver. So, if it is using some if it is using some asynchronous transfer then this DT R bar can be useful then den bar data enable. So, this is output from the processor as output enable for the transceivers, so these are actually for serial transmission.

Then ALE so this is address latch enable for this is for demultiplexing the address and data bus for using external latches. Then M by I O bar so in case of 8085 we have I O by M bar. So, here it is the name is the convention is just reversed for memory access this M by I O bar line is equal to 1 and for I O access this M by I O bar line is equal to 0 and then we have got this write bar write control signal. So, it is asserted low whenever processor writes data to memory or I O code and there is INTA bars line.

So, it is interrupt acknowledge when the interrupt request is accepted by the processor the output will be low. So, that way that acknowledge signal will go to the device and device will know that the processor has done into an interrupt acknowledge mode, so it will provide the interrupt service routine address some more minimum mode signals there is a hold signal.

(Refer Slide Time: 12:04)



Minimum mode signals

Pins 24 -31

For minimum mode operation, the MN/ $\overline{MX}$ is tied to VCC (logic high)

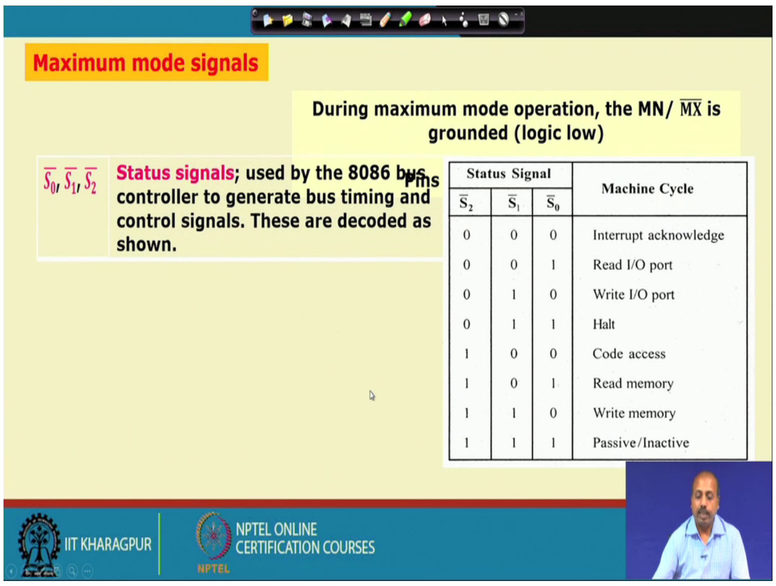
8086 itself generates all the bus control signals

HOLD	Input signal to the processor form the bus masters as a request to grant the control of the bus. Usually used by the DMA controller to get the control of the bus.
HLDA	(Hold Acknowledge) Acknowledge signal by the processor to the bus master requesting the control of the bus through HOLD. The acknowledge is asserted high, when the processor accepts HOLD.

So, basically this is for relinquishing the control of this data and address bus. So, there is a mode of transfer which is known as DMA direct memory access, where this another bus controller it is used for transferring the content directly from memory one memory location to other memory block transfer or from I O device to memory. So, like that so this DMA controller it has got a number of secondary device has connected to it and then through this through this DMA controller. So, we can transfer data blocks from this I O devices to the memory without any intervention of the processor.

So, this for that purpose the processor 8086 needs to release the control of this address and data buses and for that we have got this hold and hold acknowledge lines. So, if a request comes on the hold signal hold line; that means, the processor will understand that some other master is asking for a bus access. So, it will be giving this hold acknowledge, so it will release the control of the bus and it will be asserting this hold acknowledge line. So, that way these are the requesting processor will understand that this hold has been granted.

(Refer Slide Time: 13:20)



Maximum mode signals

During maximum mode operation, the MN/ $\overline{MX}$ is grounded (logic low)

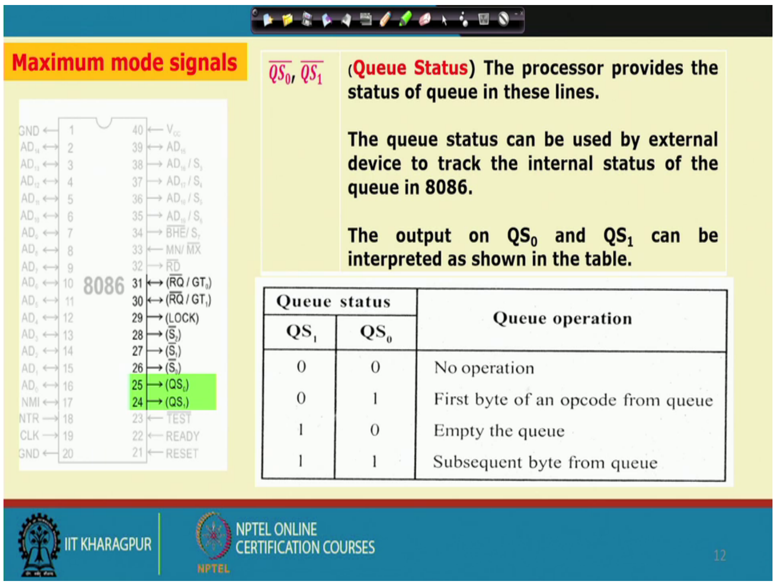
Pins	
$\overline{S_0}, \overline{S_1}, \overline{S_2}$	Status signals; used by the 8086 bus controller to generate bus timing and control signals. These are decoded as shown.

Status Signal			Machine Cycle
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

On the other hand for the maximum mode of operation, so this min max bar line has to be grounded and this status signals S 0 S 1 and S 2 they are used by 8086 bus controller to generate bus timing control signals. These are so the decoding is like this if it is 0 0 0 so, it is interrupt acknowledge 0 0 1 so this read o code. So, like that so that way this is actually giving some information about the, what is the what is the mode what is the operation that this 8086 is doing now, so this status line gives that indication.

(Refer Slide Time: 13:56)



Maximum mode signals

$\overline{QS_0}$, $\overline{QS_1}$ (Queue Status) The processor provides the status of queue in these lines.

The queue status can be used by external device to track the internal status of the queue in 8086.

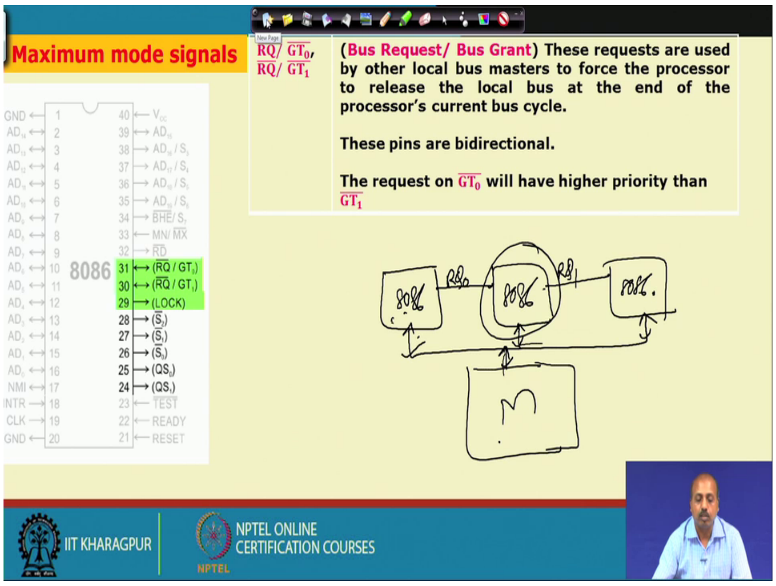
The output on QS_0 and QS_1 can be interpreted as shown in the table.

Queue status		Queue operation
QS_1	QS_0	
0	0	No operation
0	1	First byte of an opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

There are 2 mode signals QS 0 bar and QS 1 bar so they are queue status. So, we will come to this queues slightly later, so what happens is that internally 8086 process possesses one instruction queue and in that instruction queue instruction queue all the instructions when they are fetched from the second from the memory so they are kept ok. So, that speeds up the operation of this processor so we will see that.

So, what is the status of that queue, so that is actually indicated by these QS 0 QS 1 line. So, if it is 0 0 means queue is no operation is being done on the queue if it is 0 1. So, it is a first byte of an opcode from the for queue has been taken 1 0 the queue is empty and 1 1 it is subsequent bite from the queue. So, these are the various queue operations that are taking place, so this is the processor gives this indication to the outside. So, the external device can track the internal status of this 8086 queue by means of these lines.

(Refer Slide Time: 15:04)



Maximum mode signals

$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$

(Bus Request/ Bus Grant) These requests are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle.

These pins are bidirectional.

The request on $\overline{GT_0}$ will have higher priority than $\overline{GT_1}$

Then this RQ and GT lines, so bus request and grant bus grant lines. So, these are used by other local bus masters to force the processor to release the local bus at the end of the processors current bus cycle. So, this if a if a device just like in minimum mode we had the hold and hold acknowledge line, in maximum mode we have got this request and grant lines.

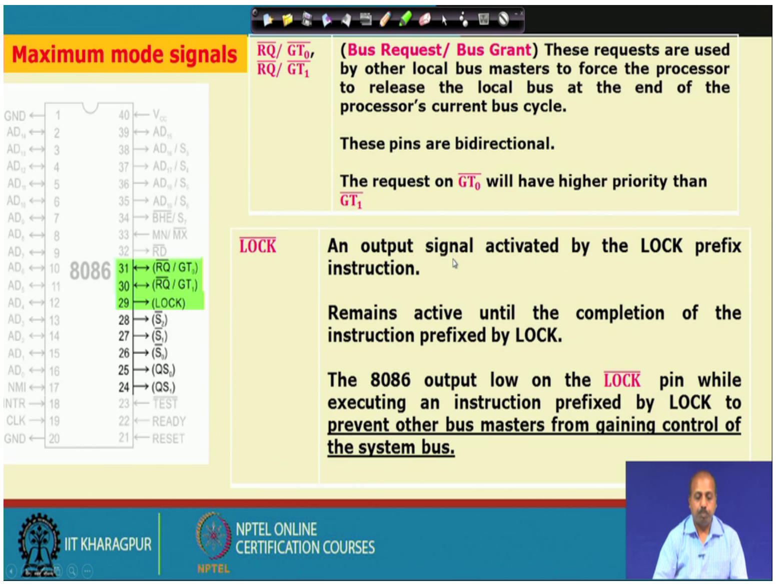
So, any processor which is asking for this bus access so it will give a request on this request line and getting this requests if the processor will finish the current bus cycle. So, whatever it was doing so that cycle is finished and then the bus is released. So, when the bus is released then this corresponding grant signal is given.

So, we have we have got 2 lines RQ GT 0 and RQ GT 1 out of this 2 GT 0 has got higher priority than the GT 1. So, if we have got 1 device connected to GT 0 another device connected to GT 1, so GT 0 device will have more priority. So, in this way I can connect a more than 1 processor like it may. So, happened that we design a system like this, so we have got 3 8086 processors 3 8086 processors and then they are all accessing the they are all accessing the same memory, the memory part is same and they are feeding the they are controlling this memory by means of this address and data lines so it is like this.

Now, whenever any processor needs to access the memory, so it has to tell the other processors that I need to access it. So, it will it may be this is connected over this RQ 0 line and this is connected over RQ 1 line. So, if this is the main master then when this fellow needs to use the bus, so when RQ 0 line it will send a request to this accordingly if it is granted then this processor will be allowed to access this memory this bus will be control by the this first processor.

On the other hand if this device needs this processor needs to use the bus then it will send a request on this RQ 1 line and if it is granted then it will be using the bus by this line. So, this is how this request and grant lines will work just like this hold and hold acknowledge in case of minimum mode of operation.

(Refer Slide Time: 17:32)



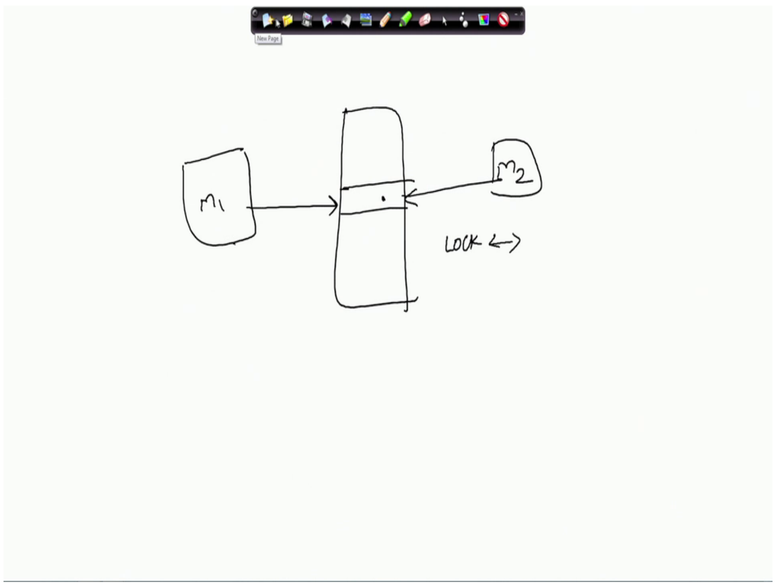
Maximum mode signals

$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$	(Bus Request/ Bus Grant) These requests are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. These pins are bidirectional. The request on $\overline{GT_0}$ will have higher priority than $\overline{GT_1}$
$\overline{LOCK}$	An output signal activated by the LOCK prefix instruction. Remains active until the completion of the instruction prefixed by LOCK. The 8086 output low on the $\overline{LOCK}$ pin while executing an instruction prefixed by LOCK to <u>prevent other bus masters from gaining control of the system bus.</u>

So, there is another pin which is known as lock bar pin, so this is an output signal activated by lock prefix instructions. So, lock prefix instruction means that we will see that there are some instructions in 8086 where you can put a lock prefix. So, you can you can write lock in front of them. So, it means that it will remain active until the completion of the instruction prefix by lock. So, this 8086 output low on the lock bar pin while executing instructions prefix by lock to prevent other bus masters from gaining control of the system bus.

Like if suppose in the previous example that we took, so we had 3 8086 processors. So, when one processor is trying to request for the bus from another processors, so first instead of sending the request directly it may check the check the lock bar pin of the other processor. So, if that lock bar is 0 that means, the processor is executing some lock type of instruction where the prefix instruction, where it will not release the bus because then the memory content may become inconsistent. The reason is that if I am if there is a memory location if they if there is memory location like this say.

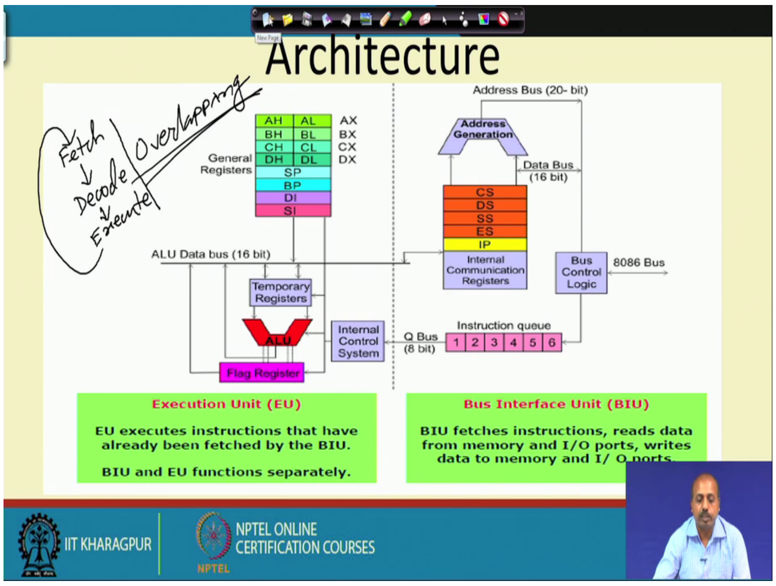
(Refer Slide Time: 18:54)



If this is say the memory and then there is 1 location and we have got 2 masters, master 1 and master 2 and both of them want to modify this location. Now, naturally so this is a serious condition because, if the modification is not done completely by 1 processor then other fellow may write something and the original content will be lost. So, that way lead that that may lead to some inconsistent step, so this is a very important problem when we consider the operating system design. So, what is done is that this updation the writing on to this location.

So, that instruction is prefix by a lock part ok, so that as a result if M 1 is executing this. So, this M 2 while requesting for the memory; So, it will first see the lock line of this M 1 and if this lock line is low that means, it will not be allowing to access the to access the buses. So, it should not send the request lines. So, that is the purpose so we can use it for this lock type of instructions. So, next we will be looking into the internal architecture of 8086 and how does it differ from 8085.

(Refer Slide Time: 20:11)



Execution Unit (EU)

EU executes instructions that have already been fetched by the BIU.

BIU and EU functions separately.

Bus Interface Unit (BIU)

BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/ O ports.

So, internally this 8086 can be divided into 2 distinct modules, 1 is known as execution unit another is known as bus interface unit. So, in the execution unit is the portion where it actually does the execution of the instructions, so these actual operations are carried out in this part. So, we have got say in this part we have got a set of registers AX BX CX DX SP BP SI DI, so these are few registers that we have here and out of this AX is A 16 bit register. So, all these registers are 16 bit registers so they are called general purpose resisters. So, GPRS and this AX BX CX DX, so they can be divided into 2 parts high and low like A high A low B high B low like that this SI DI SP BP.

So, they are they are having some special purposes so we will come to that later. Then this is the ALU which will be doing the operation and there are temporary registers from which this ALU input is fed. So, this data bus is 16 bit data bus and this ALU is A 16 bit ALU. So, it can do the operation 16 bit operations so we can get the operands from these register or you can get the operands from outside through this bus and that way it will be coming to this ALU.

So, there is a control system so that will be controlling the operation that this ALU does and the outcome of the ALU operation. So, that is by viewing in terms of some flag registers or flag bits may be set here, so that maybe so that may be that may be checked by subsequent instructions and all.

So, this EU it will execute instructions that have already been fetched by the bus interface unit. So, if you remembered 8085 so what happening is that one instruction is fetched then it is decoded then it is executed. So, it was going in a cycle like fetch, then decode and then execute and after the execution of the first instruction is over then only the second instruction is fetched, so it was going like this.

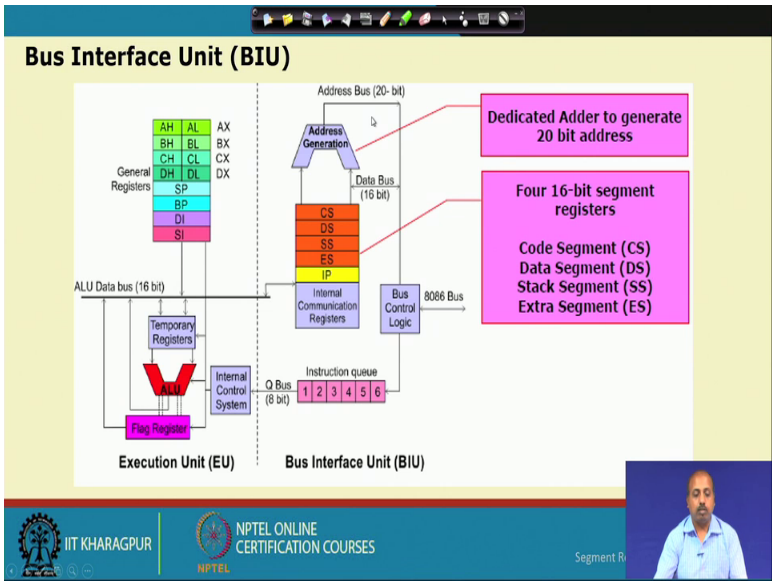
In case of 8086 so this structure is bit modified, so where there is this fetch part and decode parts. So, they are this fetch part particularly so that is separated into a different module. So, we have got a this bus interface unit, so which will fetch the instructions and data from the memory for I O I O codes writing the data. So, they will be done by this bus interface unit and in this bus interface unit also. So, we have got a few registers CS DS ES SS and the IP.

So, they are all 16 bit registers there are some internal communication registers so that are that is internal not visible to the programmer there is an address generation unit. So, this so that takes 216 bit values and performs a 20 bit addition. So, result of this ALU is a 20 bit ALU, so it will be producing a 20 bit output and the 20 bit address goes via this bus control logic to 8086 bus so and internally there is a queue there is in there is an instruction queue.

So, whenever we are doing some operation in this part, so these buses are this bus is free. So, at that time this bus interface unit it fetches subsequent instructions from the memory and once the instructions have arrived so they are kept on to this queue. So, this is a 6 by q. So, subsequent bytes are fetched from the memory and they are kept into this queue.

So, this helps in some amount of overlapping of this fetch decode execute phase, so we have got some overlapped execution of. So, it is it is a fetch decode execute overlapping. So, this is a very important concept that we will find in later processors also it started with 8086.

(Refer Slide Time: 24:18)


Bus Interface Unit (BIU)

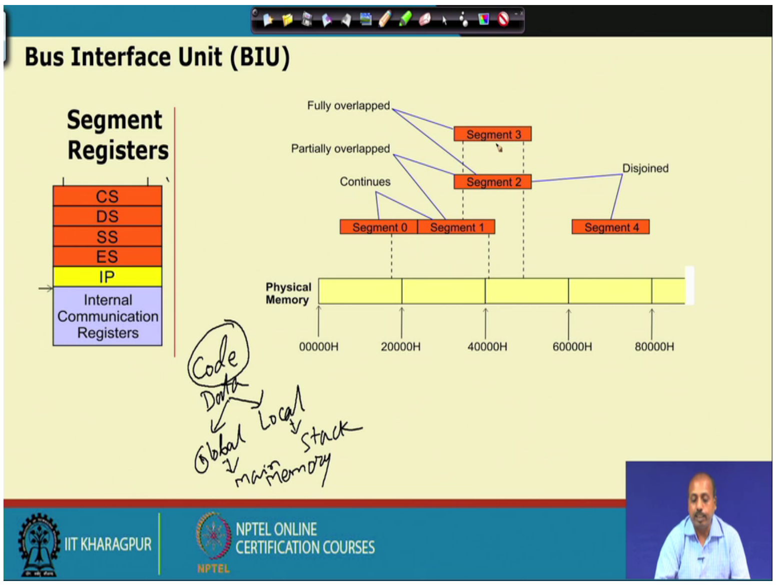
Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

So, we next we will be looking into detail like what is going to happen. So, we will start with the bus interface unit. So, as I said that this address generation is done by a 20 bit address. So, this 20 bit ALU so this will generate 20 bit address and this CS DS ES SS. So, they are all 16 bit registers so they are called segment registers CS is called the code segment register DS is called data segment registers there SS is stack segment register and ES is extra segment register. So, we will looking into detail of this segment registers and how are they generating the address.

(Refer Slide Time: 24:52)


Bus Interface Unit (BIU)

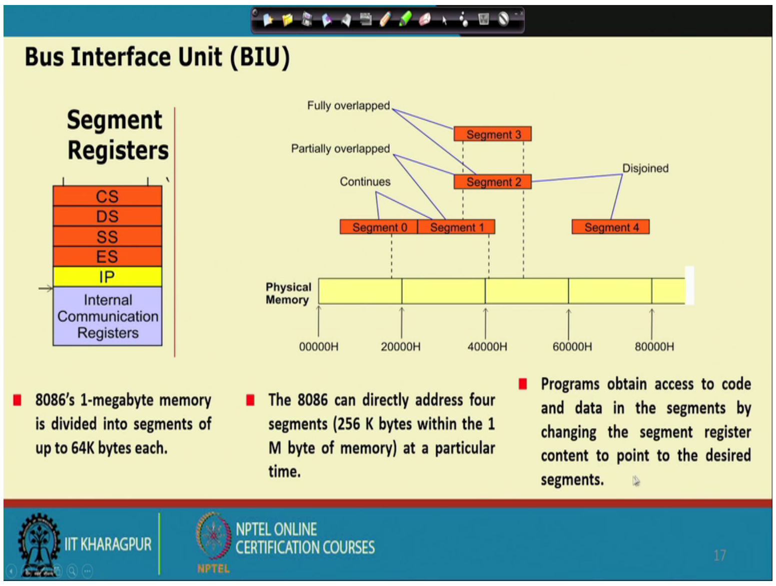
So, these are the segment registers CS DS ES SS and ES and then you can if you look into a code then any program. So, if you look into any program that program can be divided into the different parts, like we have got the code part we have got the data part which is this data can again be divided into global data and local data.

Now, for any program this code part is the they are unchanged they remain same and this data the global data and local data. So, normally this local data is created onto the stack and this global data is put on to the main memory; all are part of main memory but they are so they are allocated at the time of this program loading itself this global data and local data is as you are making procedure calls this local data is filled up.

So, we have got this segment registers and you can divide this physical memory into a number of segments, like a say this segments like say 0 0 0 0 to 2 0 0 H, then 4 0. So, we are divided into a few segments now you can name this segments likes say this is segment 0 this is segment 1 so that way, so this segment actually this is yeah. So, this is segment 0 and segment 1 so this is a continuous segment then this statement 2 and 3.

So, that way we can think of the memory to be organized into a number of segments and once we have done that for this code may be located 16, in segment 0 data may be located in segment 3 stack may be located in segment 4. So, like that we can control in which portion of the memory which part of the program will be loaded.

(Refer Slide Time: 27:03)



Bus Interface Unit (BIU)

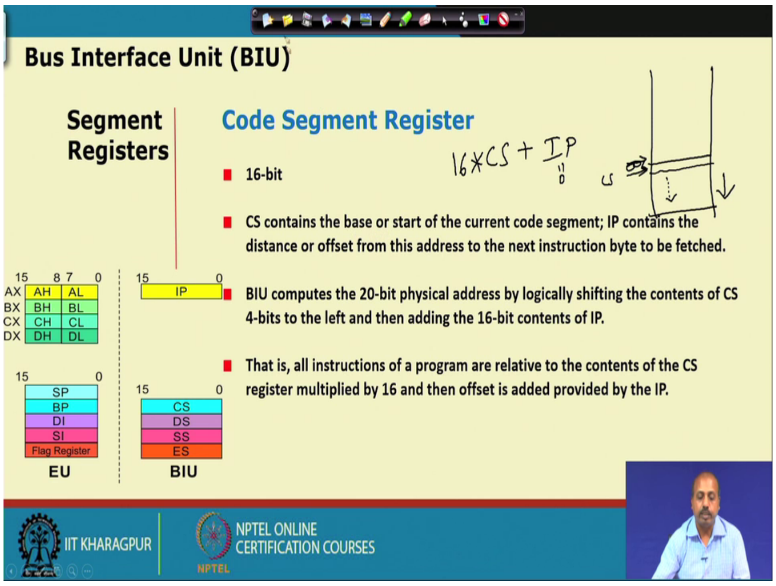
- 8086's 1-megabyte memory is divided into segments of up to 64K bytes each.
- The 8086 can directly address four segments (256 K bytes within the 1 M byte of memory) at a particular time.
- Programs obtain access to code and data in the segments by changing the segment register content to point to the desired segments.

So, this is this facility this segmenting facility is available in 8086 and so the total 1 megabyte of memory it is divided into segments of up to 64 bytes each and this it can directly access 4 segments of 64 kilobyte segment. So, these 256 kilobytes can be accessed by this any 8 program 8086 at a particular time ok. So, how does it access?

So, program obtain access to code and data in the segments by changing the segment register content to point to the desired segments. So, if I say that this segment 0 will be corresponding to the code, then this CS register will be made to point segment 0. Similarly if I say that segment 4 will be the data segment then this DS register is made to point to segment 4.

So, each segment is 64 kilobyte so I have got at most 4 segment. So, at any point of time so you can have total 256 kilobyte of space accessed by 8086 and of course you can you change the content of this segment registers all the segment registers to have more number of segments. Like at after sometime if you find that DS should better point to segment 3, so you can change the content of DS so segment 3 becomes the data segment so that is possible.

(Refer Slide Time: 28:14)



Bus Interface Unit (BIU)

Segment Registers

Code Segment Register

- 16-bit
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
- That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.

$16 \times CS + IP$

So, this code segment register is 16 bit register, so this is it contains the base or start address of the current code segment and IP contains the distance or offset to from this address to the next instruction byte to be fetched. So, what happens is that this address when it is formed, so this address is formed as so this address is formed as 16 into the code segment value plus the instruction pointer value.

So, whatever so if this is the memory and I have initialize the code segment to be equal to this and say the address is increasing in this form this is the code segment. So, CS register points to the first 1 so this value this location address is in CS register and the IP value is made equal to 0 say.

So, when I am accessing first location so the 16 into CS plus IP. So, it will be accessing this location then IP value will be incremented in subsequent accesses it will, so it will accesses these locations. So, this IP contains the offset and this CS contains the base value. So, that way this access will be done by the code segment register.