Microprocessors and Interfacing

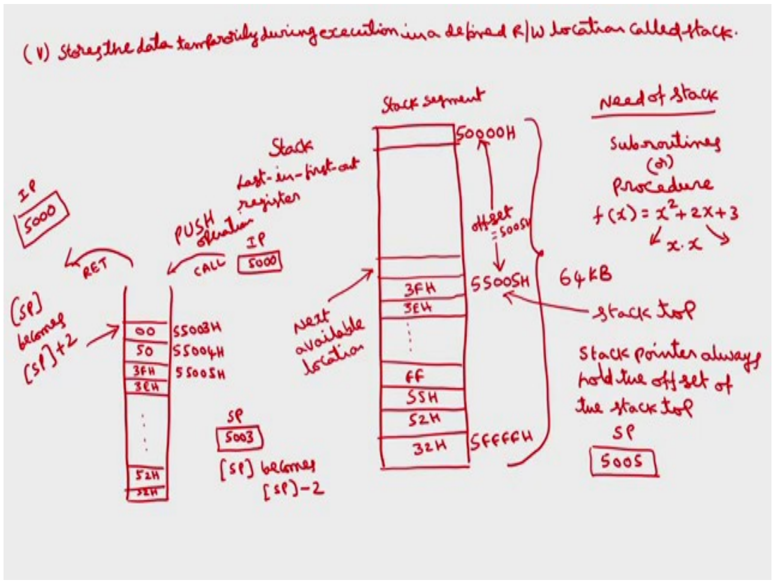
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture - 03
Functional Diagram of 8086

So, we are discussing about the second microprocessor operation which is internal data operations. So, in that we have already discussed about storing of the data for which we require some registers, then the second operation is performance the arithmetic and logical operations so, we require a ALU.

The third one is test for the conditions ok, test for the condition we need some flags already I have explained the I mean conditions of the various flag to set and reset. Then the fourth operation is sequence the execution of the instructions for which we require instruction pointer that also we have discussed in the last class.

(Refer Slide Time: 01:14)



Then the last fifth operation of these internal data operations is stores the data temporarily during execution in a defined read write location called stack, read write location called stack. So, first I will explain the stack, the purpose of the stack pointer and stack in microprocessors.

Stack is basically we know that in 8086 there is a stack segment. So, if I assume that this is a stack segment. So, this is the starting of the stack segment, this is the ending let us assume that this is 64 kilobytes of the stack segment is there. So, if the starting address is 50000 H, ending address will be 5 FF FF H and stack is a first last in first out register, stack is last in first out.

So, we have the data which is inputted last will be outputted first. So, there is one pointer called stack pointer. So, if the data is filled from here to suppose here some 32 H is there some 52 H 55 H some FFH. So on up to if the data is full up to here, if this address location is something kind 55005 H.

So, the next available location for this stack is this, this is next available location. And this is called stack top, the location up two which data is fully is called stack top. So, this will be stack top; the location up to which the data is full and this is next available location.

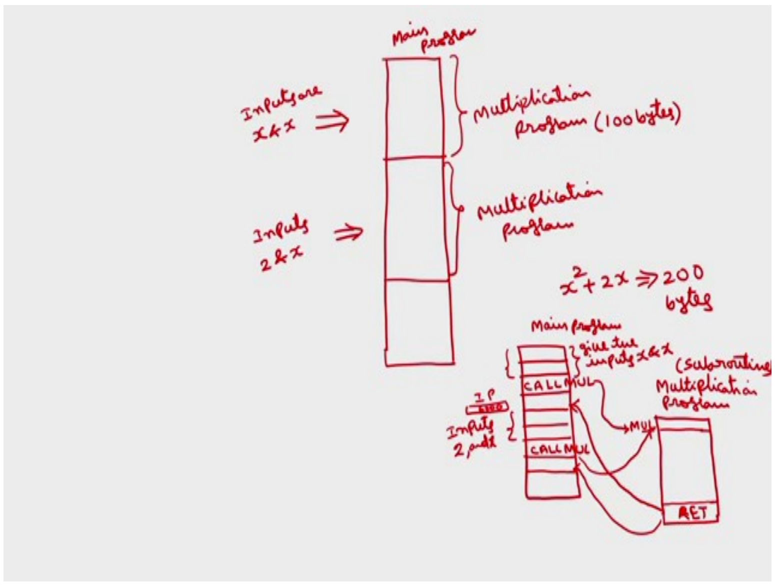
Then the stack pointer this is 16 bit register. So, stack pointer always holds; stack pointer always holds the physical address or the logical address, the offset stack pointer always holds the address of the location which used to be synchronized. So, always holds the stack top offset of the stack top.

In this case, so, the offset will be the first the distance from here to here. So, this offset is 5005. So, this offset will be hold by stack pointer. The contents of stack pointer will be the offset from the starting of the stack segment to the stack top. This is stack segment. So, the stack pointer location will be 5005.

Now, what is the need of the stack this is the (Refer Time: 06:05) what is the need of stack? So, the stack is required; so, whenever you want to I mean execute some sub routines or procedures. So, you might have heard these sub routines in your C programming sub routines also called as procedures.

That is, if your particular operation repeatedly occurs like if I want to write the program for the f of x is equal to x square plus 2 x plus 3 here multiplication operation is coming twice x square can be written as x into x. This is one multiplication whose inputs are if I write a multiplication program the inputs are x and x and this is another multiplication with inputs 2 and x.

(Refer Slide Time: 07:11)



So, if you want to write two programs you have write the multiplication program, suppose from here to here multiplication program is there. For this multiplication program the inputs are we are giving x and x. So, x will you to x square then to perform to x again we have to write the same multiplication program another time multiplication program, this time the inputs we have to give as 2 and x.

Suppose if this multiplication program takes 100 bytes say, then to obtain x square plus 2 x to obtain x square plus 2 x requires 200 bytes. This is waste of memory, because the program is same only the inputs are different. So, instead of writing this multiplication program 2 times; so, you write this multiplication of program in a sub routine and you call this a multiplication program 2 times, that will save the number of memory locations that is what is called sub routine.

So, in any particular program if a particular operation is repeating several times so, instead of writing that program several times in the main program you write a sub routine and you call that as many times as you require. So, now what happens to you in sub routine sub concept, this is my main program. So, in the main program what I will do is this is my main program, I will first initialize these two inputs; I will give the inputs as x and x using some instructions. Then you write a multiplication program somewhere else, this is multiplication program.

So, whatever the data that you give this will multiply and it will feed back the result to the main program. So, this particular multiplication program this is called sub routine or procedure. So, here we will call the multiplication program, and this is the starting address multiplication. So, whenever the call instruction is there the microprocessor control after this will goes to this one.

So, whatever the input that is given before this, this will multiply those two inputs and the last instruction of this any sub routine is return this is RET return. So, here it has to return back to the instruction which next to this, then again you will give the inputs x and 2, again we call the same multiplication program which is called sub routine. Again here also the microprocessor control will goes to its multiplication program and return back to the instruction which is next to the multiplication program.

So, in this way we can save the number of memory locations so, but here the problem is so, whenever the microprocessor is executing this call instruction, the instruction which is to be executed next is this. So, this will be having some instruction pointer will holds to some address.

So, suppose if the offset of this particular memory location is some 6000 H the instruction pointer holds to this 6000 H, then if I do not save this instruction pointer contents and if I go to this sub routine. While returning how does the microprocessor knows that it has to return back to the 6000 H? So, that is why what you have to do is the 6000 location has to be stored in some temporary location which is called the stack.

So, whenever a call instruction is executed and never a subroutine is called in the main program. So, what you have to do is before you are going for the subroutine you have to store the contents of the instruction pointer in a temporary location called stack. So, in order to store this I P contents here in the stack. So, if I take this stack pointer in the previous slide if I want to move this 6000.

So, we can move this contents into the stack using the operation called push operation. So, if these 6000 contents has been return in to this one. So, what happens is before this execution of this one before these contents has been transferred here, these are the contents of the stack pointer this is 32 H 52 H and so on. So, this is 3 EH 3 FH.

Now, contents of the stack pointer will be this is stack top 55005 H. So, once if the call the instruction from the sub routine. So, the what will be the inputted here will be the contents of IP is 5000. So, in that the 5 0 will be first stored in this location which is 55004 H. The higher order 8 bits and then lower order 8 bits will be 0 0 will be stored here then stack pointer contents will be 5 5 0 0 3 H.

Now, what will be the contents of the stack pointer now up to here? The stack is full. So, the contents of stack pointer becomes now 500 3H. So, before the instruction pointer contents are pushed into the stack, the contents are 50005. Whereas, after this instruction pointer contents has been pushed SP becomes this. So, means the SP contents will be decreased by 2 SP becomes SP minus 2.

So, this contents of the instruction pointer which will be pushed into this stack pointer or stack memory, this will be internal to the this call instruction. Whenever call instruction is executed, the contents of instruction pointer will be automatically pushed into the stack. So, this in detail I will discuss while discussing about the push and pop instructions of the 8086.

So, in addition not only the contents of the instruction pointer, if the instruction pointer contents has to be saved this is internal to the call instruction. So, in addition to this what happens is if I use here all the register for some purpose. So, you know that we have the 8086 a x b x c x d x.

So, we have different registers if I use for some purpose here also, we have to write this multiplication program which is subroutine also using the same registers. So, if want I mean use the same register for some other purpose. So, the contents old contents of this registers will be lost.

So, in order to avoid that even the all the register contents also we can push into the stack like this instruction pointer then you go to the subroutine while return back. So, we take all the contents back which is called pop operation. So, in pop operation what happens is this is the push operation. So, in push operation what happens is the contents will be pushed on this is after push operation, this is push operation.

Whereas, in case of a return instruction so, the last instruction in the subroutine is return. So, inherently in return instruction there is a pop operation what happens is in pop. So, this in pop what happens is these contents will be taken back into the instruction pointer, instruction pointer contents will become same 5000 H. This operation occurs so, whenever the call instruction is executed, this occurs whenever return instruction is executed.

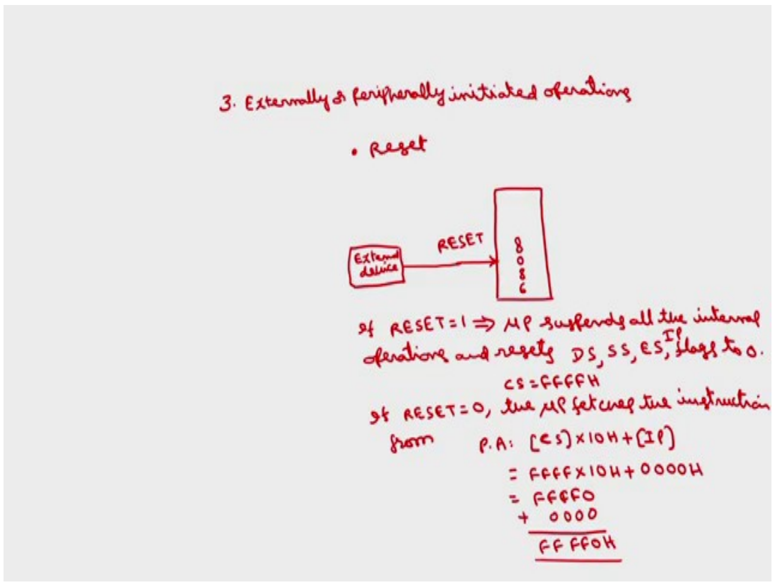
So, if I take that this; so, as I have told this push is last in first out stack operation I mean operates on the last in first out operation I mean a principle. So, while inputting first few 5 0 has been inputted and 0 0 has been inputted. So, while outputting first 0 0 will be outputted then 5 0 will be outputted. So, after that again what happens is, this stack top these two will be empty locations now stack top will be points to the previous location 5 0 55005 now stack pointer contents becomes 5005 only.

So, in the return instruction what happens is stack pointer will be stack pointer plus 2. So, this is about the stack operations. So, the need of the stack is whenever a particular operation is repeating repeatedly coming; so, instead of writing that many times the program in the main program. So, we can write a simple program in a subroutine and you can call as many times as you require ok.

So, in that process we need to store the contents of the instruction pointer and the registers temporarily in summary of some location memory location which is called as stack. So, while storing which is called as push operation the contents of stack pointer will become stack pointer minus 2, because we are going to save this and in case of pop operation stack pointer becomes SP plus 2. So, this I will discuss this in detail while discussing about push and pop instruction set of 8086.

So, this is the 5th operation that will be performed. So, here about the 5 operation that takes place inside the microprocessor which are called internal data operations. And, the third type of the operations are externally or peripherally initiated operations.

(Refer Slide Time: 18:43)



3. Externally & peripherally initiated operations

- Reset

If RESET = 1 $\Rightarrow$ μP suspends all the internal operations and resets DS, SS, ES, IP, flags to 0.

CS = FFFFH

If RESET = 0, the μP fetches the instruction from

$$\begin{aligned} P.A. &= [CS] \times 10H + [IP] \\ &= FFFF \times 10H + 0000H \\ &= FFFF0 \\ &+ \ \ 0000 \\ &\underline{\overline{FFFF0H}} \end{aligned}$$

So, what are the operations that will be initiated by the external devices? What are the external device to the microprocessor? I/O and memory ok. So, there are some few operations which is the first operation is reset operation. So, reset so, later we are going to discuss about the pin diagram of 8086 in that there is one reset pin. So, this reset pin is input to the microprocessor.

So, whenever the microprocessor receives the reset pin 8086 by external device, this will be connected to an external device which has memory or I/O; so, any I/O device external device. So, whenever reset is 1, in the microprocessor received a high signal on the reset. So, the microprocessor immediately what happens is microprocessor suspends all the internal operations resets.

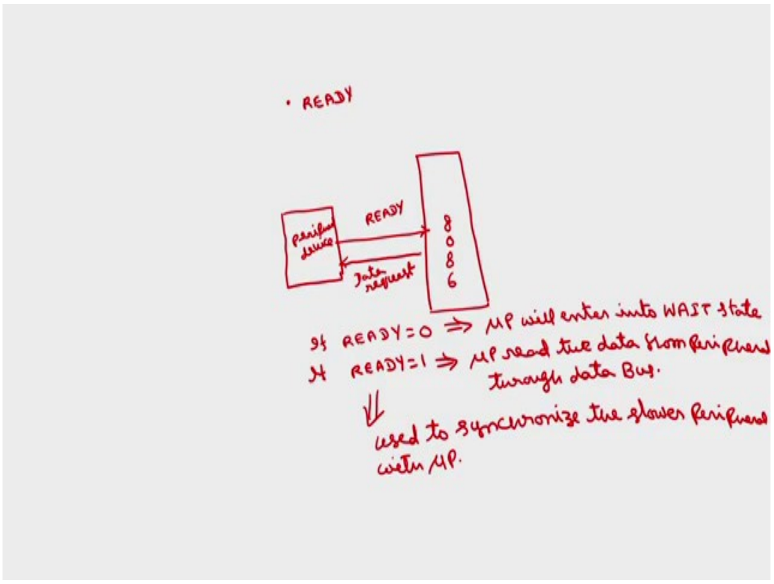
So, all the registers like data segment, stack segment, extra segment and all flags to 0. It just mean suspend all the internal operations and makes data segment, stack segment extra segment flags to 0. This all these semi registers will be reset, but code segment will be set to FFFF H. If the reset becomes 0, if the external device releases the reset signal, then what happens is the microprocessor fetches the instruction from which location?

So, we know that code segment, the physical address of any location is physical address is given by code segment contents into 10 H plus instruction pointer; this we have discussed in the earlier class this is the physical address. So, when the reset is when what are the contents of the code segment is FFFF into 10 H model the contents of instruction pointer even instruction pointer also here set to 0, this instruction pointer is 0 0 0 0 H.

Therefore, instruction becomes FFFF0 plus 0000H, this will be FF FF 0 H. So, the microprocessor fetches the instruction from this address location, this is about the reset operation. So, whenever the microprocessor receives a reset signal from the external device; it suspend all the internal operations and sets data segments, stack segment, extra segment instruction pointer flags to 0 whereas, code segment will be set to FFFFH

So, if the reset becomes 0 the microprocessor then fetches the instruction from the physical address which can be calculated by using code segment to 10 H plus instruction pointer which becomes FFFF0H. This is the first term I mean externally peripheral or peripherally initiated operation.

(Refer Slide Time: 23:17)



• READY

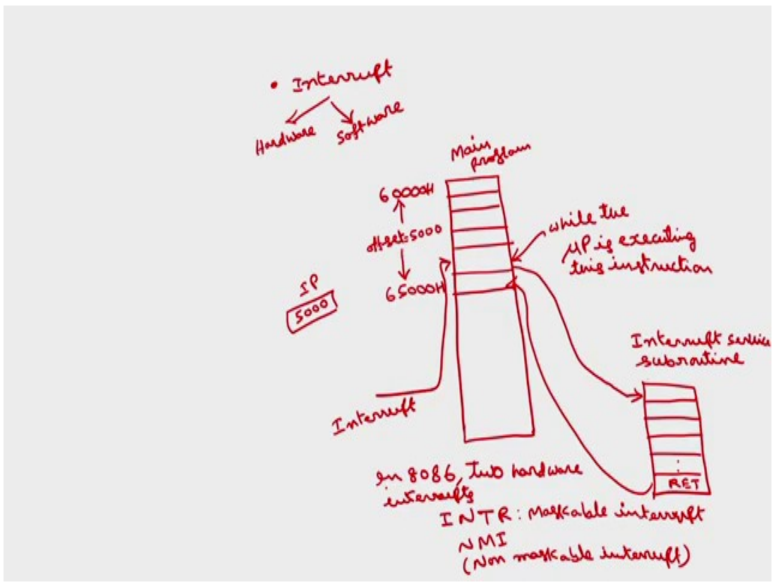
If READY=0 ⇒ μP will enter into WAIT state
If READY=1 ⇒ μP read the data from peripheral through data Bus.
⇓
used to synchronize the slower peripheral with μP.

And, the second externally or peripherally interrupted I mean initiated operation is ready. So, microprocessor is having another signal called ready signal. So, any peripheral will be connected to the microprocessor to ready signal, peripheral device if it is connected ready signal; so if ready signal is 0, microprocessor receives a low on ready signal. So, microprocessor will enter into the wait state. So, this is something like if the microprocessor want to take the data from a peripheral device normally the peripheral device is slow when compared with the 8086. So, because of this the first microprocessor will initiate for the data through some request data request.

So, if the peripheral is ready to give the data it will send high on the ready signal; so, till the data is ready, because the peripheral is lower when compared with the microprocessor. So, this will make ready is equal to 0. So, microprocessor simply waits in the wait state. So, if the ready is one, microprocessor read the data from read the data from the peripheral through data bus.

So, basically this ready operation will be used to synchronize this lower peripheral with microprocessor, basically ready signal will be used for synchronizing this lower peripheral with microprocessor. This is the second externally or peripherally initiated operation

(Refer Slide Time: 26:04)



The third one is interrupt. So, whenever the microprocessor receives the interrupt. So, this is the main program in the microprocessor 8086. So, interrupt can be of two types it can be software interrupt or hardware interrupt, in any case the operation is same the interrupt can be hardware or software.

Software interrupt as the name implies there are some instructions which will access interrupt hardware there will be pin corresponding to each hardware interrupt. So, depends upon the position of that particular pin whether we have to make high or low to high transition high to low transition. So, accordingly the microprocessor will be interrupted.

Suppose if the microprocessor executing the programs one by one. So, while the microprocessor is executing this program, this instruction and let us assume that if the starting of this one is some 60000 H and if the address of this one is 65000 H. So, while the microprocessor executing this instruction. So, what will be the contents of the instruction pointer as we have discussed in the earlier class, the instruction pointer always holds the offset from the starting of the segment

So, what is the offset here is 5000. So, this 5000 will be hold by the instruction pointer, instruction pointer holds the offset from the starting of the segment. So, the microprocessor is supposed to be execute this instruction after the this instruction, but while executing this instruction the microprocessor has received the interrupt. Let us assume that, while the microprocessor is executing this instruction microprocessor has received the interrupt.

So, what the microprocessor will do is first it will execute the current instruction after that. So, instead of going for the next instruction the microprocessor control will goes to after the execution of this instruction, there is one subroutine called interrupt service subroutine, interrupt service subroutine which is similar to the subroutine which I have just discussed for the call instructions, where you can write the some program.

So, normally this interrupt will be activated whenever you want to perform some emergencies task. microprocessor is executing some program so, whenever you want to perform some emergency task you interrupt the microprocessor, you disturb that current I mean running of the program then you go to this interrupt service subroutine you write the instructions here to perform the emergency task. Then the last instruction on this one is similar to any subroutine return.

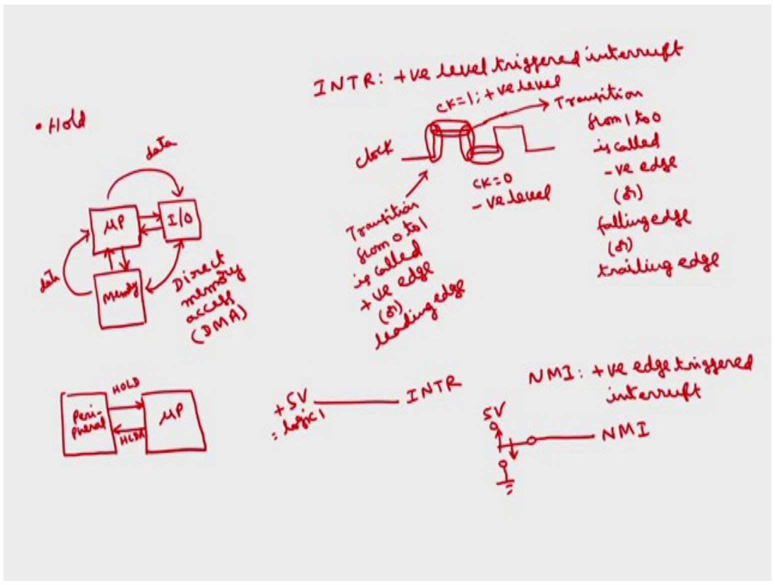
So, while it is going for this interrupt service subroutine this instruction pointer continues has to be saved into the stack, as we have discussed just now. Then in the return what happens is the contents of the instruction pointer which are stored in the stack will be again taken back into the instruction pointer. So, the control will automatically goes to the instruction which is next to the this interrupt next to the instruction which is next to the interrupt

So, you see the process this is exactly similar to this call and return instructions. So, after execution of this instruction this will goes to subroutine. So, you see how the interrupt will occurs ok. So, in 8086 there are many software interrupts that I will discuss there are two hardware interrupts in 8086, in 8086 there are two hardware interrupts they are called INTR and NMI, NMI stands non maskable interrupt. So, this is where this one is interrupt just simply this is maskable interrupt.

So, what is meant by non-maskable and maskable? So, this non-maskable as the name implies, it cannot be disabled this can be used to interrupt the microprocessor at any time. Whereas, INTR can be disabled by the programmer suppose the microprocessor is right I mean executing the main program itself some emergency task and the main program itself.

That time it can disable this INTR through some program, but NMI cannot be disabled even if you want to execute some emergency program here, the NMI can be used to interrupt the microprocessor and it can be asked it can be forced the microprocessor to execute some other which interrupt service subroutine. So, it is the difference between maskable and maskable and maskable, maskable means which can be disabled. So, NMI means which is not disabled. But when does this INTR will be enabled NMI will be enabled. So, this INTR is called positive level triggered interrupt.

(Refer Slide Time: 32:17)


• Hold
data
MP
I/O
data
Memory
Direct memory access (DMA)
HOLD
Peripheral
HLDA
MP
INTR: +ve level triggered interrupt
CK=1; +ve level
Transition from 1 to 0 is called -ve edge (or) falling edge (or) trailing edge
clock
CK=0 -ve level
Transition from 0 to 1 is called +ve edge (or) leading edge
+5V = logic 1
INTR
NMI: +ve edge triggered interrupt
5V
NMI

INTR is positive level triggered. So, if you want to interrupt the microprocessor through INTR. So, if I take the clock signal will be having totally four divisions. This is called positive level the region remains a duration over which this clock is equal to one, clock is equal to one logic one is called positive level, and this is called where the clock is equal 0 is called negative level. The transition from 0 to 1 is called positive edge or leading edge transition from 0 to 1 is called positive edge or leading edge whereas, the transition from 1 to 0 transition from sorry not this.

So, you see transition from 1 to 0 is called negative edge or falling edge or even its also called trailing edge. So, when does the micro based interrupt will be enabled so, if I take this INTR during this portion only. So, whenever you give INTR line this is INTR line, this is INTR line. If I apply this 5 volts to this one that is called positive which is logic one, then only this INTR will be enabled. Whereas, NMI non-maskable interrupt is positive edge triggered interrupt. That is if you want to enable this NMI you have to give a transition from 0 to 1.

So, if this is NMI signal if this is my ground, this is my 5 volts line then you take this NMI signal first to connect to 0 then you connect to 5 volts, then only this NMI will be enabled 0 to one transition which is called as positive edge whereas, INTR is positive level if I make simply plus 5 volts here INTR will be enabled. So, these two of this hardware interrupts INTR and NMI in 8086. So, INTR will be a positive level trigger NMI will be positive edge trigger this is one difference.

So, another difference is INTR can be maskable NMI cannot be maskable. So, this is how the interrupt whenever the microprocessor want to be an execute some emergency program by the external device request. So, it will execute that program through the interrupt. The fourth external operation will be hold operation. So, we know that the microprocessor block diagram of this microprocessor microcontroller is we can have the microprocessor memory I/O.

These two are bidirectional, this is simplified block diagram of any microcomputer as I have told microprocessor is useless unless otherwise is it is connected to memory and I/O. So, because the microprocessor (Refer Time: 37:03) address and data buses. So, if the memory wants to transfer the data to the I/O. So, the regular process will be, so first memory will transfer the data to microprocessor, then from microprocessor data will goes to I/O.

Because why this microprocessor you are having address and data buses for I means data transfer address and data buses are required. So, for that is why the memory will first place the data into microprocessor then from microprocessor the data will goes to the I/O, this is a time consuming process. For small data transfers we can use this process, if the data to be transferred is huge. So, in that case this is time consuming process.

So, instead of that if I directly transfer the data from memory to I/O or I/O to memory this is called direct memory access DMA. So, we can do this direct memory access to save this time. So, we can transfer the data directly from memory to I/O, but what is required here is if I want to transfer the data from memory to I/O. So, I have to take the control of the buses from the microprocessor; so, for that the peripheral device memory or I/O even I/O can be transfer the data to the memory directly this is a bidirectional.

So, these two peripheral device any peripheral device; so, if any peripheral device want to make DMA data transfer this is microprocessor any peripheral this can be either memory or I/O. So, first it has to take the control of the address and data bus of the microprocessor further the peripheral will request the microprocessor through a signal called hold signal.

So, hold signal is a request for the microprocessor to hand over the control of address and data buses to the peripheral for a DMA data transfer. So, whenever the microprocessor is ready to give this address and data buses. Since we will acknowledge through hold acknowledge signal HLDA, this is hold and this hold acknowledge HLDA.

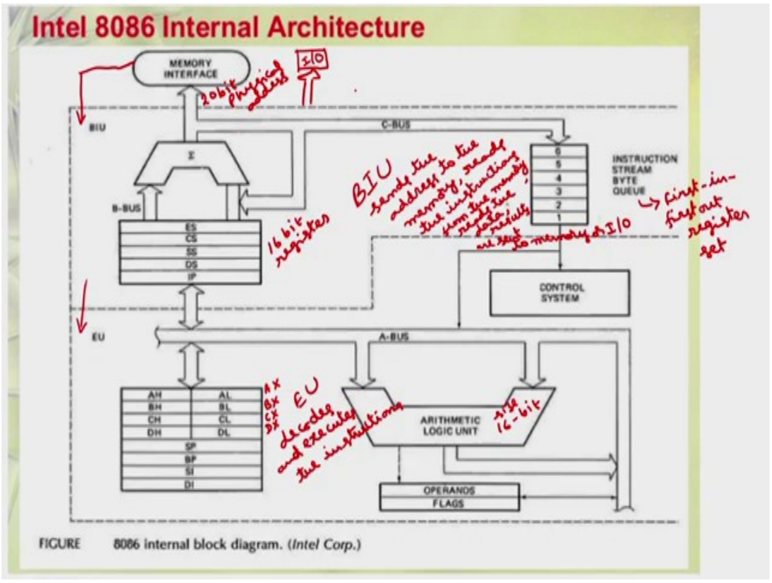
So, once if the peripheral receives the hold acknowledge after that the control of the address and data buses will be given to the peripheral then the peripheral it can memory or I/O, it will directly transfer the data to the other peripheral. So, after the data transfer is over again the control of the address and data buses will be given to the microprocessor.

So, this is what is called direct memory access so, we will discuss this DMA also there is one IC exclusive I C for direct, I mean direct memory access DMA. So, we will discuss more details that time. So, this is the direct memory access. So, in this we will use hold and hold acknowledge signals, this is also externally initiated signal. So, these are all this I mean the microprocessor operations, till now we have discussed about. So, all the I mean 3 types of the microprocessor operations.

The first one is the microprocessor initiated operation such as memory read, memory write, I/O read, I/O write, second one is internal data operations. So, like storing of the data, performing arithmetic logical operations, then checking for or testing for the conditions sequence the execution of the instructions stores the data temporarily during the execution in a memory location called stack then there are some externally or peripherally initiated operations. So, which are reset ready interrupt and hold ok. So, and the corresponding the hardware elements and the signals that are required for the 8086 we have discussed.

So, with this discussion now we can move to the architectural diagram of 8086 and then we can discuss about the different pins of 8086, this is all the brief introduction to the 8086.

(Refer Slide Time: 41:15)



Intel 8086 Internal Architecture
MEMORY INTERFACE
20bit physical address
I/O
BIU
C-BUS
Σ
B-BUS
ES
CS
SS
DS
IP
16 bit registers
BIU sends the address to the memory, reads the instructions from the memory, reads the data, results are sent to memory & I/O
6
5
4
3
2
1
INSTRUCTION STREAM BYTE QUEUE
→ first-in-first out register set
CONTROL SYSTEM
EU
A-BUS
AH
BH
CH
DH
AL
BL
CL
DL
SP
BP
SI
DI
AX
BX
CX
DX
EU decodes and executes the instructions
ARITHMETIC LOGIC UNIT
size 16-bit
OPERANDS
FLAGS
FIGURE 8086 internal block diagram. (Intel Corp.)

Now, coming for this 8086 architecture so, this is internal architecture of Intel 8086 microprocessor. So, which is taken from the Intel corporation. So, these are the different blocks in 8086. So, basically this entire architecture is divided into two parts one is called bus interface unit BIU, this is the BIU this above one doted one this one is called BIU. And this one is execute unit E U.

So, what is the purpose of BIU what is the purpose of EU. So, BIU actually it will send the address to the external device to the memory. Because where the programs will be stored in the memory ok, in order to execute that instructions or the programs. So, the microprocessor has to fetch the instructions and then it has to be executed.

So, for that BIU bus interface unit BIU stands for bus interface unit send the address to the memory. So, you see the memory outside memory interface and we can have I/O also. So, we have memory interface we can connect to the I/O devices also here this connection can help this can be connected to I/O also.

So, BIU snds for the address to the memory location. So, from that particular memory location it will fetch the instructions one by one, read the instructions from the memory location. Then, in order to execute the instruction it may require the data also read the data.

So, this reading of data can be either from the memory or I/O also. So, reads the data then execution will not be takes place here execution as the name implies execution will takes place here. So, after the execution here the results will be sent back to the either memory or I/O. So, these are the various operations in bus interface unit it sends the address so that in that particular address we will be having some instruction that the instruction will be fetched into the BIU, then whatever the I mean data that is required to execute that instructions it will fetches from the memory or it can be from I/O. So, after that the data will be given to execute unit execute unit will execute now after that the results will be sent back to BIU. So, the BIU basically sends the data results to the either memory or I/O.

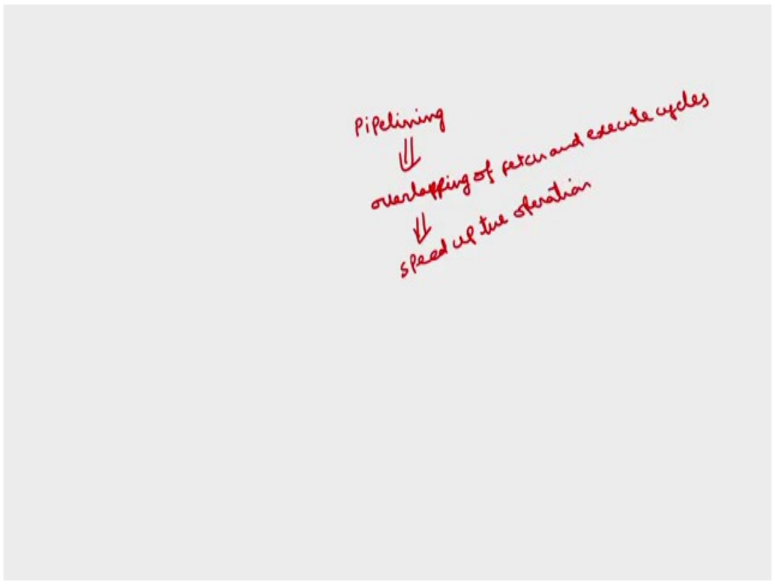
So, basically this BIU is going to interact with the outside the world, and as the name implies execute unit. So, it decodes the instructions, decodes and execute the instructions. So, here one important thing here is here in this bus interface unit there is a 6 byte long instruction queue. So, what happens is microprocessor will fetches the instructions. So, if the microprocessor fetches the first instruction after fetching the first instruction from memory, from memory it fetches the instruction.

This will send this instruction to the execute unit for decoding an execution. So, while the microprocessor is executing the first instruction. So, this bus interface unit fetches several instructions up to 6 bytes 6 instruction bytes, it pre fetches and stores in a temporary location called queue. So, this queue is first in first out register set. So, each one register total we have 6 registers. So, this queue is first in first out register set.

So, while the microprocessor is executing, execute unit is executing the first instructions. So, it will fetch the 6 instruction because this will take the time. So, during that time the microprocessor means the bus interface will not be ideal. So, what it will do is it will fetch the 6 instructions as many as 6 instruction bytes it will store into the queue.

So, what is the advantage of prefetching the 6 instruction bytes while the first instruction is being executed by the execute unit. So, in this way what you can do is we can overlap the fetching operation with execute operation. So, this is what is called pipelining.

(Refer Slide Time: 46:42)



Pipelining

$\Downarrow$

overlapping of fetch and execute cycles

$\Downarrow$

speed up the operation

So, 8086 is operates in pipelining mode, this pipeline is nothing but overlapping of overlapping of fetch and execute cycles is called pipelining. So, what is the advantage of the pipelining, because we are going to overlap the fetch cycle and execute cycle you can speed up the operation, there were there by we can speed up the operation. This is the advantage of the 6 byte instruction queue.

So, while the microprocessor executing the first instruction we can store 6 bytes here. So, now, the microprocessor will take the instruction from instruction queue. So, this instruction will be fetched into the instruction queue if this instruction is having a space for at least two bytes, then only this I mean BIU will fetch the instructions from the memory.

So, from the memory the instruction go to instruction queue from instruction queue it will go to the execute unit then after execution again which also will send back to the either memory or I/O device. And the remaining blocks most of the things we have already discussed. So, we have this segment registers extra segment code segment, stack segment, data segment, instruction point the purpose of all these registers we have already discussed.

And this is arithmetic logic unit sorry this is actually this physical address calculation unit as I have told how to compute the physical address from the offset or the logical address. So, this particular code segment or any segment contents the contents of this segment register into 10 H plus the contents of instruction pointer are any base pointer

So, in that way you can compute physical address this is twenty bit physical address, this is twenty bit physical address this is twenty bit physical address. Whereas, these registers are 16 bit registers. So, to obtain this 20 bit physical address from 16 bit registers, we can use this operation the operation that I have already explained. So, the contents of any segment register place 10 H into instruction pointer or any base pointer index pointer depends upon the type of segment that we are going to use.

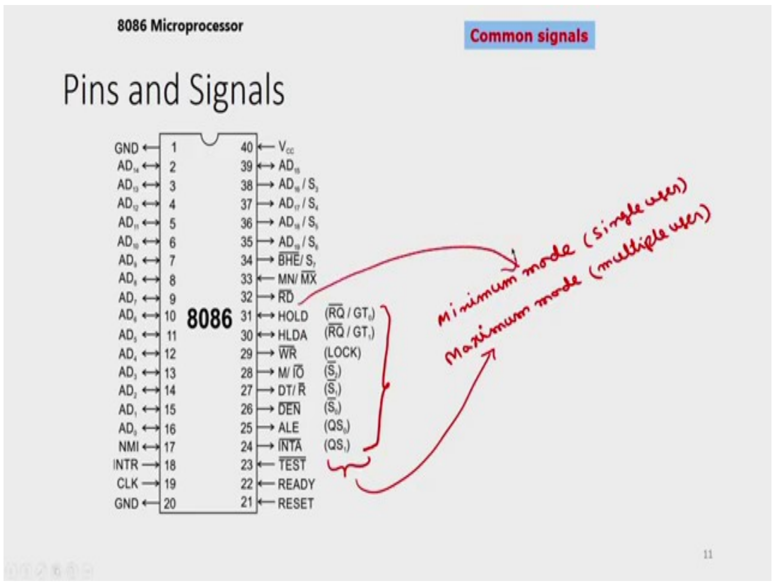
So, its about this bus interface unit blocks and then coming for this execute blocks execute unit. So, you have different registers as I have told AH AL this can be used as AX as a whole this can be used as BX as a whole 16 bit if you want to store the 16 bit data. We can use BH and BL as BX CH and CL as CX DX. If you want for our 16 bit operations we will use AX BX CX DX for 8 bit operations you can use AH AL BH BL CH CL DH DL.

And then we have stack pointer, just now I have explained the stack pointer operation, stack pointer always holds the offset from the stack I mean start of the start segment to the stack top and base pointer base pointer will be basically used in calculation of the physical addresses for data segment and extra segment and all. Similarly we have source index and destination index these will be useful in string operations which I have discussed in the last class.

So, these are all about the register array then arithmetic logic unit which performs all arithmetic and logical operation the size of arithmetic logic unit is here 16 bit because this is 16 bit microprocessor. So, the size of the ALU is going to decide so how many bit microprocessor is that.

Then we have some flags which I have already explained. So, in order to I mean, compute, all this operation we require some control system. This is about the internal architecture. So, most of the blocks we have already discussed while discussing about the various operation microprocessor operations

(Refer Slide Time: 51:11)



Then coming for Pin diagram; so, totally there are 40 pins 8086 is a 40 pins I C. So, these are the 40 pins there are some common signals, there are some specialized signals. So, this from here to here the microprocessor can operate in two modes one is called minimum mode and maximum mode. Minimum mode is single user and maximum mode is multiple user. So, multiple users can access the 8086 at the same time which is called multiprocessing. So, in that case you have to operate this 8086 in maximum mode for a single user in minimum mode.

So, in any case except these 8 signals, the remaining signals are common whether it is a minimum mode or maximum mode. So, these signals these set of signals are for only maximum mode. Whereas, this set of signals are minimum mode signals. So, we will discuss the functions of each I mean pins. So, in the next class 1st I will discuss about the minimum mode signals.

So, maximum mode signals we will discuss later. So, the functions of each pin most of the signals we have already discussed like NMIS INTR is a reset hold, hold acknowledge ok. So, whichever the signal that we have missed we will discuss in the next class after that we will go for the instructions of 8086.

Thank you.