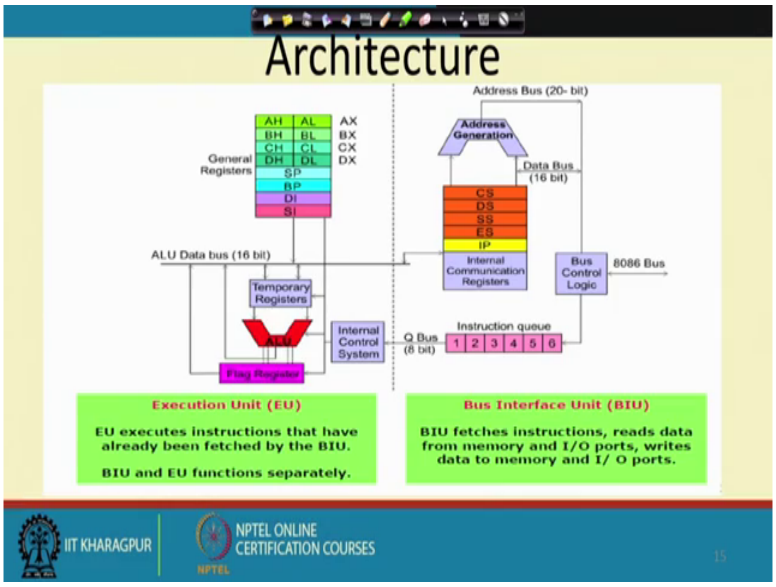
Microprocessors and Microcontrollers

Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

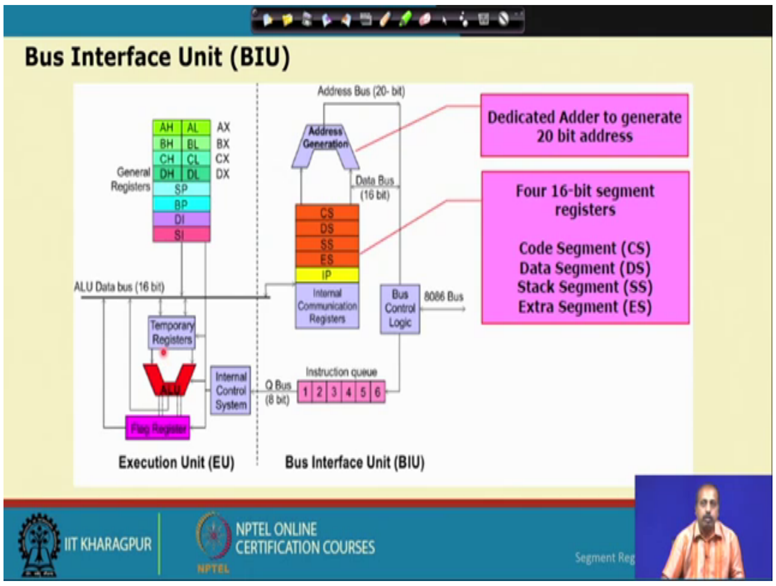
Lecture – 60
8086 (Contd.)

(Refer Slide Time: 00:19)



So, 8086 architecture, so it consists of two units as we have noted in the last class, there is a bus interface unit and there is an execution unit. So, bus interface unit is on this right side. So, this actually point containing the logic by which it can access the bus that is the address bus, data bus and all that. And on the left side, you see that execution unit where the actual execution are taking place. So, we will look into this in detail.

(Refer Slide Time: 00:44)



Bus Interface Unit (BIU)

Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

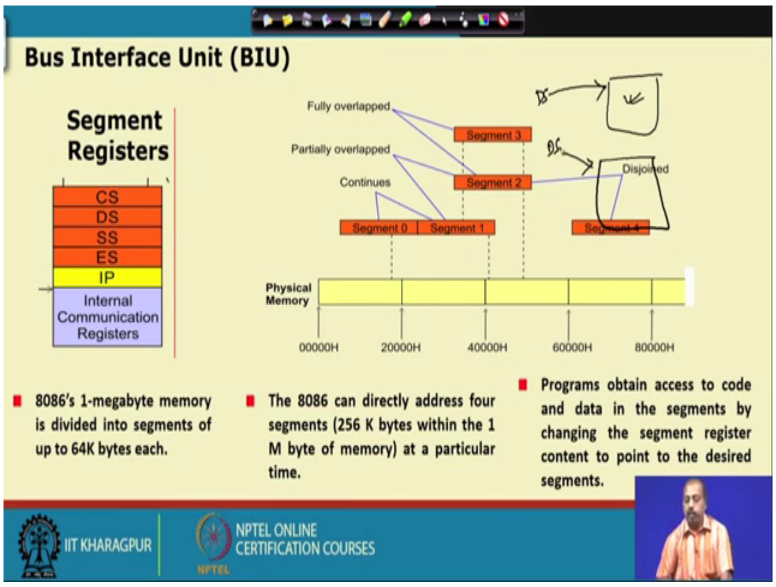
Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

So, this bus interface unit so to be more specific. So, there are some 16-bit registers CS, DS, SS and ES. So, they are called segment register. So, code segment register, data segment register, stack segment register and extra segment register. So, we will look into the functionality shortly. And there is another 16-bit register which is the instruction pointer or IP. So, this IP is similar to program counter that we have in 8085. So, here in 8086, this IP and this CS, they two will be constitute in the 20-bit address bus. Similarly, when it is accessing data the DS or SS or ES along with that some other register may be combined, and you will get the data address or the stack address like that.

So, will see that that memory addresses are generated by a combination of this segment register and some offsets. So, for program access the code access, so this CS and IP, so those two will be added for others some offset value will be added with the other segment registers. And there is a separate address generation ALU ok, so this is one 20-bit adder and this adder is dedicated for this purpose. So, you see that you look into this diagram you see there is one ALU on the execution unit which does which also has the capability to do addition, but this ALU is not given the task of generating the address. So, for address generation, a separate address generation adder is put into this module.

So, so CS value and this IP value when they are coming then that will be combined together to get 20-bit address and that 20-bit address will be put onto this address data on the this address bus and through this bus control logic. So, it will be connecting to 8086 bus. So, this 8086 bus you know that this is a multiplexed bus 20 bit 20 bit bus out of which 20 bits can constitute the address; 16-bits can constitute data, and there are some status bits that will be multiplexed with higher order address lines to get 20-bits here ok. So, we will look into these one by one.

(Refer Slide Time: 03:02)



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 bus interface unit the operation is like this. So, if this is a physical memory address that is a physical memory starting at say 0000 to say 200000 so 4, so that way it is increasing now. So, it is divided into segment. So, physical memory is divided into segments. So, may be this 0 to 2000H so that is one segment. So, this is the next segment. So, we call this segments ok.

Now, this segments may be continuous, so segments may be disjoint, segments may be overlapping. So, this definition of segment is with respect to I should say the programmer. So, programmer may define the segments to be common, segments to be disjoint, segments to be overlapping like that. And the segment register settings can be controlled by which you can you can have a difference between this overlapping or this disjoined or that way. So, this overlapping pattern can be controlled by putting this segment register values. So, in this a particular program may consists of a number of segments like segment 0, segment 1, 2, 3, 4 out of which so this segment 0 and segment 1, they are continues. After segment 1 starts segment 2 and segment 1 there is a partial overlapping. So, segment 1 also partially overlaps with segment 3, 2 and 3 are completely overlapped. So, then 3 and 4, 2 and 4 are disjoined.

So, like this we can have different style different type of overlapping. So, total memory that 8086 supports is so 1 megabyte, so 20-bit address bus total memory is one megabyte, and it is divided into 64 k segments. So, each segment has got a maximum size of 64 k, so 64 kilo byte. So, now you can think about different types of overlapping. So, it is not mandatory that your memory should be divided into two segment or four segment or eight segment like that. So, depending upon the programmers requirement it can be done but each segment the size will be 64 k irrespective of overlapping or non overlapping.

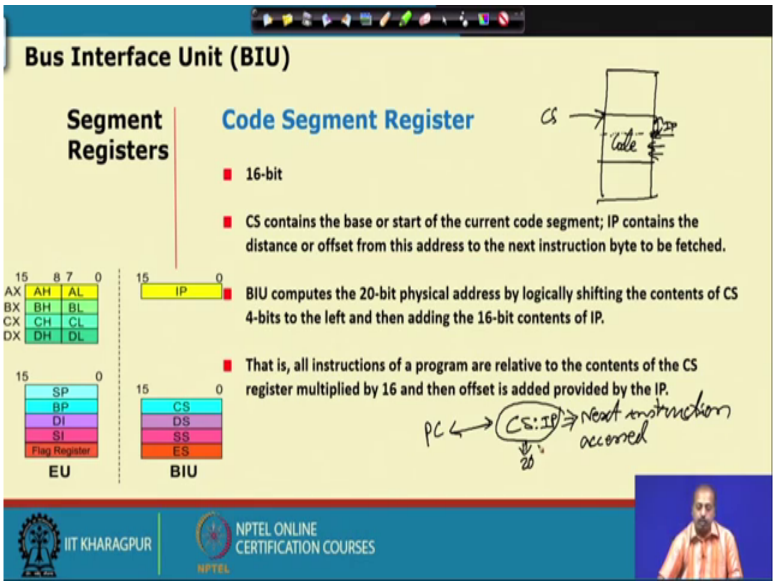
And then there are four such segment registers CS, DS, ES, and SS. So, it can use this 4 segment. So, if the all the segments are distinct from each other disjoint from each other, then this 64 k multiplied by 4, so you can get 256 kilo bytes in one megabyte memory. So, there are four segments, so this each segment is 64 kilo byte, so I can have a total of 256 kilo byte of memory that can be addressable at one point of time. So, naturally there are the other segments. So, out of this one megabyte only 256 kilo bytes are accessible, so rest are there.

So, basically the purpose is that I can have a multiprocessor multiprocessing system, where different user programs are loaded in different part of the memory. So, this provides generic structure like you can you can give 256 kilo bytes to per user. So, you can support four users directly, so that way so that may be the simplest possible organization. And you can think about any other complex combination of this segment register, and the segment definitions.

So, programs obtained access to code and data in the segments by changing the segments register content to point to the desired segment. What I say what I what it means is that maybe I have in my program I have got say two data segment. So, this is one data segment and this is another data segment. So, at some point of time, I want to access this data segment. So, in that case somehow this DS register should be made to point to the beginning of this. Sometime later in the program if you find that need to access this data segments. So, you can change the DS register, so that it points to this data segment now.

So, you see that one point of time. So, you can access four different segments by through this segment register. But if you want, so you can change the content of those segment registers in the program as a result it can access different regions of the program different regions of the memory during execution.

(Refer Slide Time: 07:35)



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.

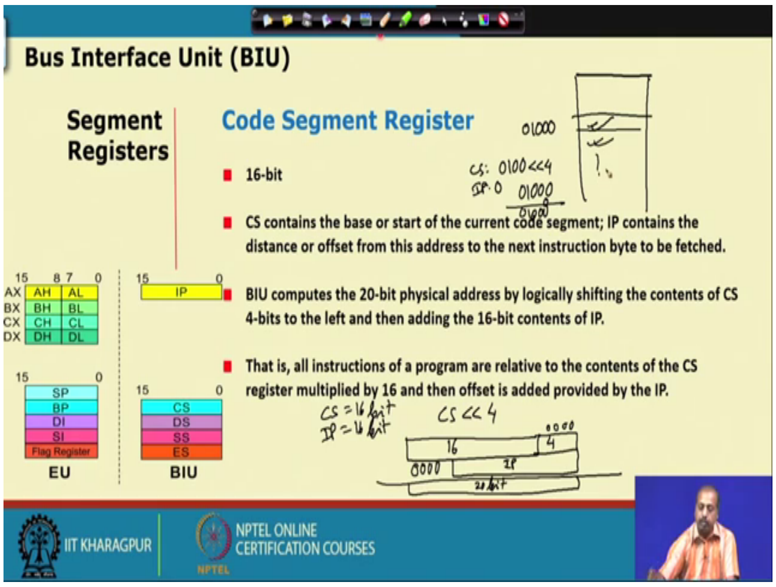
So, the next is about the segment registers we will start looking into more detail. So, there are four segment registers as I said the first code segment register is CS or it is called code segment register. So, this is a 16-bit register. So, this code segment register is basically for pointing to the segment of the memory that contains the code of the program. So, the so naturally the code is not modifiable. So, it is actually trying to distinguish between code program memory and data memory. So, in the sense that this program memory will be pointed two by this CS register, the codes segment register; whereas, this data memory will be pointed two by this data segment register that extra segment register like that.

Of course, you can over write options it is not mandatory that this code segment register cannot be used for the data access, so that can be done but there should be explicit over writing that way. So, if you do not take the overwriting then the code segment register is 16-bit register, it contains the base or start of the current code segment. So, an IP the instruction points there it will contain the distance or offset from the of the of the from these address to the next instruction byte to be fixed. So, if we say that this is my this is my memory and in that memory so my code segment is somewhere here. So, this is my code segment part this is the code part.

So, what is what it means that the CS register will point to the beginning of this segment? And within that at some point of time if I am at this statement, so this is that this off set distance from the beginning of the segment, this distance this distance is contained in the IP register. So, in after this access is over so this IP will be updated, so that IP register will point to the next instruction, then it will point to the next instruction that way. So, this CS colon IP, so it is normally retained as a pair, so CS colon IP, so CS colon IP it contains the next instruction to be accessed. So, it is the next instruction to be accessed, so that way you can say that the equivalent of this program counter that we have in 8085 is the CS colon IP pair the CS colon IP pair this it will be converted into a 20-bit value will see how it is done.

So, this bus interface unit. So, this will be converting this it will be getting this 20-bit physical address by logically shifting the content of CS register 4-bits to the left and adding the 16-bit contents of IP. So, we know that CS registered is 16-bit and this IP is also 16-bit. So, this is CS is 16-bit and IP is also 16-bit.

(Refer Slide Time: 16:24)



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.

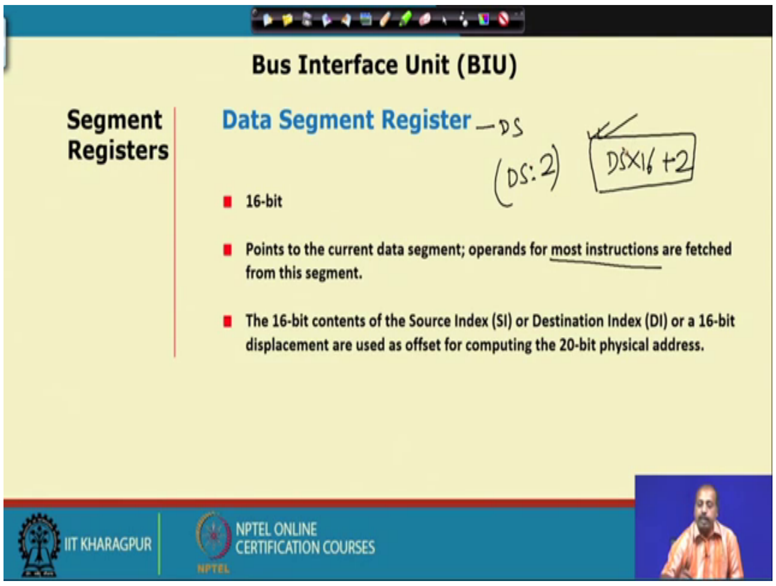
So, first what is does is that CS left shifted by 4-bit positions, so you get this 20-bit pattern now. So, this is the 16-bit CS value that is that has left shifted. So, you get another four bits which are all zeros this bits are all zero, but overall number is 20-bit.

So, with that that way the 16-bit IP will be added. So, 16-bit IP will be somewhere here. So, these bits will be taken as zero, this is the twenty 16-bit IP, so that will be added and the resulting value will be 20-bit this resulting value will be 20-bit and that is the physical address ok. So, this is 20-bit physical address will be formed logically shifting the content of CS register by 4 bits, and then adding the 16-bit IP with that.

So, all instructions of a program, so they are relative to the content of this CS register multiplied by 16. And then the offset is provided by IP. So, if so this provides a very good facility for loading the program at different places like if you if you some point of time, you find that the program will be loaded from memory location 1000 then this CS register can be initialized to say to 1000 right shifted by 4 and then IP so that way it is the while actually running the program, so it will be left as CS will be left shifted by 4 bits. So, it will be doing fine.

Whereas, so the if it is so what I mean is suppose this program is loaded from memory location started at starting at memory location 01000 that way 20-bit address, so programming starting from this address. So, for this purpose the CS register should be loaded with the pattern 0100 ok, so that whenever you are using the IP register is initially zero, so that when the first instruction is being accessed. So, this should be less shifted by four bits. So, as result you will get the pattern 01000 with that this IP value will be added. So, you get 01000, so it will be accessing the first location then IP is incremented it will be accessing the next location, so that way it will go. So, this way we can have this 16-bit offset program re location can be done. So, program relocation is just changing the CS value. Loading the program at different places. So, by just changing CS value the program relocation will becomes simple.

(Refer Slide Time: 13:14)



Bus Interface Unit (BIU)

Segment Registers

Data Segment Register – DS

$(DS:2)$ $\boxed{DS \times 16 + 2}$

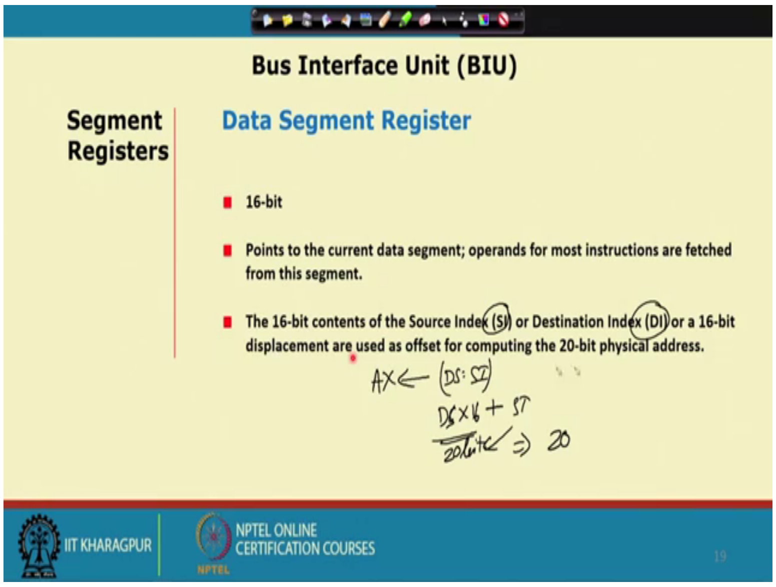
- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

So, so next register segment register that we have data is the data segment register. So, this data segment register is again our say DS this is this is also called that register, this is also called the register DS. So, this DS register is a 16-bit register and it points to the current data segment. So, just like to CS points to the code segment, DS point to the data segment. And operands for most instructions are fetched from this segment. So, please note this term it is the most instructions ok.

So, you can if you say that I want to get the content from memory location say two ok, so I want to get the content of memory location two. So, what will happen is actually the when I say 2, this will actually mean DS colon 2 that is the content of DS will be left shifted by 4-bits that is multiplied by 16 plus 2. So,. So, whatever be the address so this DS, so when I say that I want to get the content of memory 2, so this is actually the location that will be access DS into 16 plus 2. So, this way we can have this segment registers, data segment register pointing to the data segment and it is a most instruction means some instructions they use some other segment register for data access.

So, for example, particular so when you are accessing stack the stack segment register is SS used for access or whenever there is a another segment register ES or extra segment register so that can also be used for data segment. So, this 16-bit content of the source indexed register SI and destination index is DI or a 16-bit displacement can be they are used as offset for computing the 20-bit physical address. So, the so DS, SI and DI, so these registers will find in the execution unit.

(Refer Slide Time: 15:10)



Bus Interface Unit (BIU)

Segment Registers

Data Segment Register

- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

$AX \leftarrow (DS:SI)$

$DS \times 16 + SI$

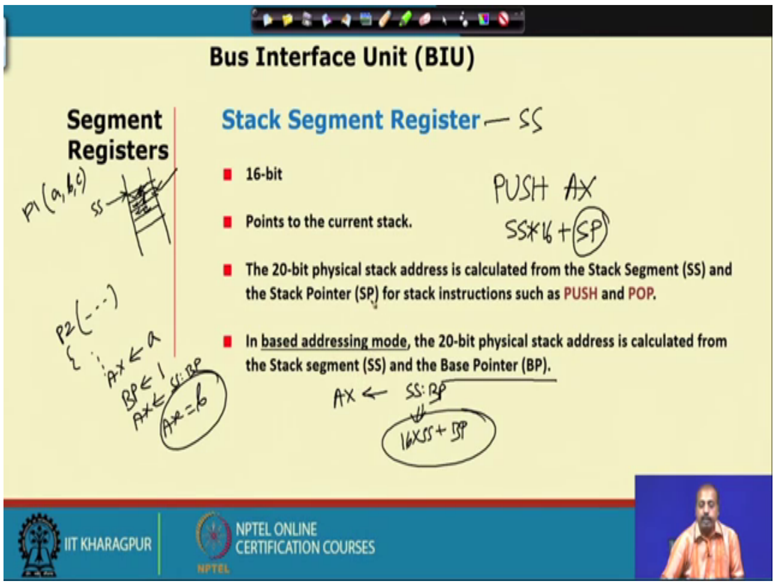
$\underline{20\text{bit}} \checkmark \Rightarrow 20$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

19

So, if you say like say if you want to get into this register AX content of say DS colon SI, so what happens is that this DS multiplied by 16 DS register multiplied by 16, so that that way it is getting it is giving me a 20-bit value with that this SI register content will be added. So, you get a 20-bit address. So, 20-bit data address will be generated, so that will be giving as the address for the data access. Similarly, the another register can it can be used is that DI register destination index. And also you can mention displacement in many other ways. So, you will be look into the instructions will see that there are many other ways by which we can generate this offsets.

(Refer Slide Time: 16:01)



Bus Interface Unit (BIU)

Segment Registers

Stack Segment Register — SS

- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).

Next we have the stack segment register which is commonly known as the SS register. This is the SS register. So, this is again the 16-bit register; it points to the current stack. So, as you can so as you can understand that if I change the stack segment register the SS register content then it can point different locations in the memory. So, as a result it is not mandatory that in my memory that is only one step. So, different user programs may have different stack segment, and they may be loaded, they may be accessed via the SS register in a different fashion.

So, as an whenever we want to locate that stack segment, so you can locate you can load the SS register accordingly to go to that stack segment. So, it is a 16-bit register, it points to the current stack. And again the same thing that the 20-bit physical address will be computed by this stack segment register plus the stack pointer SP for the push and pop instructions like you can have instructions in 8086 like this. So, you can have a instruction like push AX. So, what we want to do is we want to set the register AX into stack. So, what address it will be pushed, so that will be determined by the segment register content multiplied by 16 plus the stack pointer register. So, there is a stack pointer register available in the execution unit of the of the of this 8086, so that value will be added with 16 with 16 to SS and that will give me the address where the AX content will go.

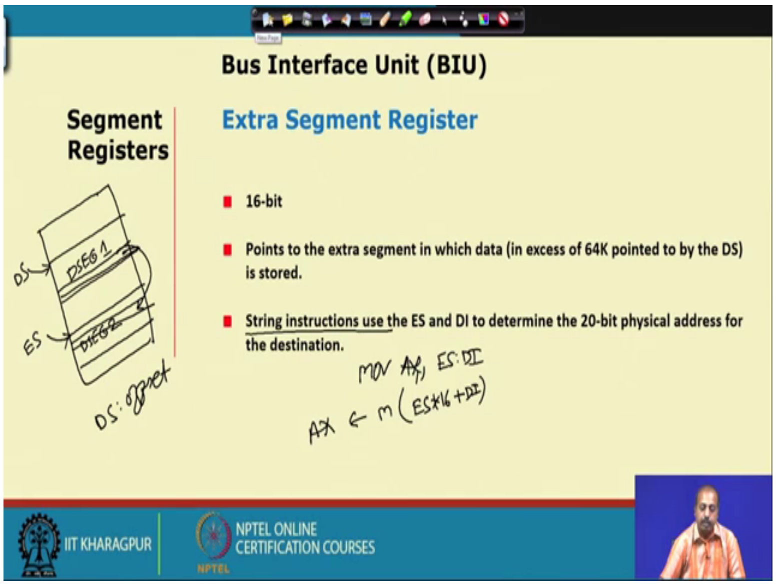
And there is another register which is known as the base pointer or BP. So, this is also used for some cases some instructions in the in a particular address mode which is called base addressing mode. So, there it will be using this stack segment plus this base pointer. So, in this case, what will happen is suppose I want to get the content of this location pointed to by SS colon BP. So, we can we can have instructions for that purpose. So, again the same thing that is 16 into SS plus the BP, so that particular memory location will be accessed and that content will be passed to AX.

So, this is useful whenever you are using this stack for parameter passing type of application like what happens is that suppose I have got a procedural P 1 and this procedural p one has passed some parameters abc to some procedural P 2. So, it has got this abc here. And in the code of P 2, so we are trying to ge[t] access this abc. Suppose we are trying to get the into the AX register, we want to get value of a that we have passed.

So, how to do this thing? So, you need to in case of 8085 we have seen that we have to explicitly use this push pop instructions to pop out the corresponding parameter from the stack, and then do the operation. Here that is not necessary. So, here you can in the stack segment, we have got this abcs all this parameters we have pushed so that we have passed, so they are they are in the stack segment register and they for in into the stack segment in the memory. Now, if the stack segment register SS points to the top of this when the SS colon 0 is the a, then SS colon 1 is b, SS colon 2 is c etcetera.

So, you can initialize this BP the base point at two say one and then you can say AX gets SS colon BP. So, what will happen SS one means so it will be accessing this b so as a result AX will be getting the value B. So, this can be done fine. So, we will see this technique when the parameter passing is looked into this based addressing mode looked into. So, you see that the data access in this case is not by the data segment register, but by the data segment register, but by the stack segment register.

(Refer Slide Time: 20:05)



Bus Interface Unit (BIU)

Segment Registers

Extra Segment Register

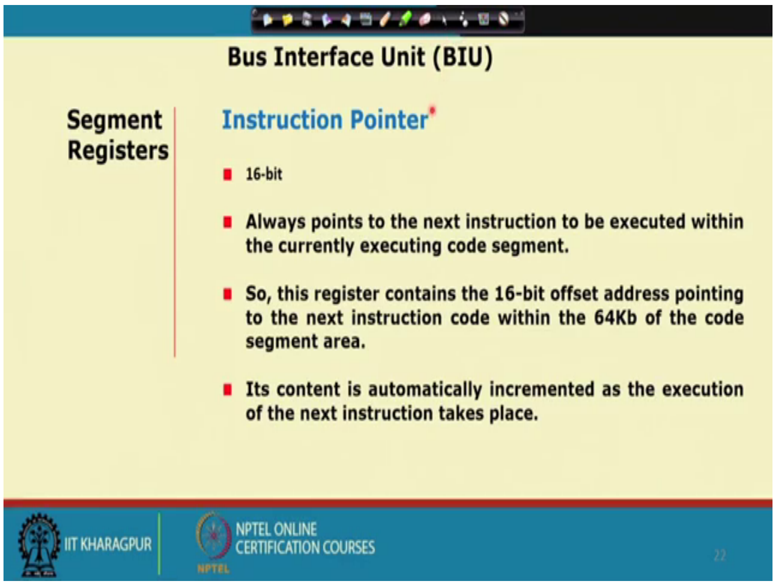
- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.

So, another register, another segment register has been provided which is called extra segment register. So, this extra segment register it is sometimes useful because we may have instead of having a single data segment, we may have two data segment. So, if this is a memory, if this is the memory, so in the memory I may have two different data segments. And maybe this is one data segment, so let us call it say DSEG data segment 1, data segment 1 and this is a another segment which we call data segment 2.

Now, since we have got a single data segment register so DS may be made to point to the beginning of this segment. Now, if you want to transfer some data from here to here, then how to do this, this is very difficult because you see whenever you are accessing data. So, it will be doing like DS colon offset. So, it will be doing DS colon offset. So, within a single instruction, you need to change this segment register value, so that is not possible because for changing this segment register value, you have to first move some value to DS. So, one possibility may be that you move the content of suppose I want to get the content of this location content of this location copied to this particular location in this new segment. So, one possibility is you copy it to some code segment part then from code segment copy in two steps you do this thing, but that will be difficult.

So, instead of that what facility that is provided is that DS let DS point to this. So, you have got another segment register ES which is pointing to this other segment that that is why it is treated as an extra segment. So, this extra segment register is there and again so this an extra segment in which another 64 kilo byte of memory can be accessed. So, so this string instructions they have got this type of application, but in general I can say like say move AX comma say ES colon DI. So, we can have instructions like this, so that way this EX either EX extra segment register divided by 16 plus DI, the content of that particular memory location will be come into the AX register. So, this has got particular usage in the string instructions that we will see later, but otherwise also you can use this extra segment register.

(Refer Slide Time: 22:52)



Bus Interface Unit (BIU)

Segment Registers

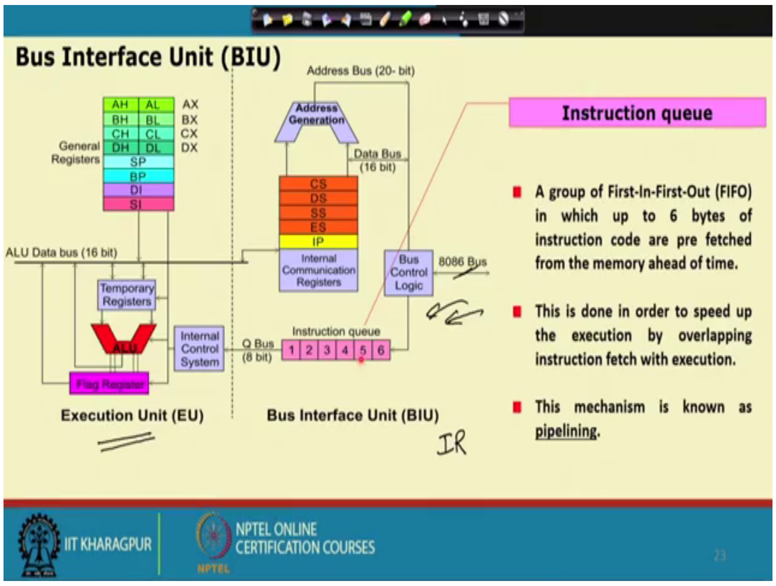
Instruction Pointer

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.

Then the instruction pointer registers, so this is a 16-bit register it points to the next instruction to be executed within the currently executing code segment. So, at the beginning, code segment register has been made to point to the beginning of the segment and the IP register content is made zero, so that in successive instruction whenever one instruction has been fetched, this IP valve is updated and CS value remain same the IP value is updated, so that it can go to the next instruction without affecting the CS register.

So, this register contains the 16-bit offset address pointing to the next instruction code within this 64 kilo byte of code segmented area. And its content is automatically implemented as the execution of the next instruction takes place. So, this is very much possible this automatic incrementation, so this is very much possible because we have got dedicated adder in the bus interface unit. So, you can use that error for doing this interface addition without affecting the ALU that we have in the execution unit.

(Refer Slide Time: 24:04)



Bus Interface Unit (BIU)

Instruction queue

- A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.
- This is done in order to speed up the execution by overlapping instruction fetch with execution.
- This mechanism is known as pipelining.

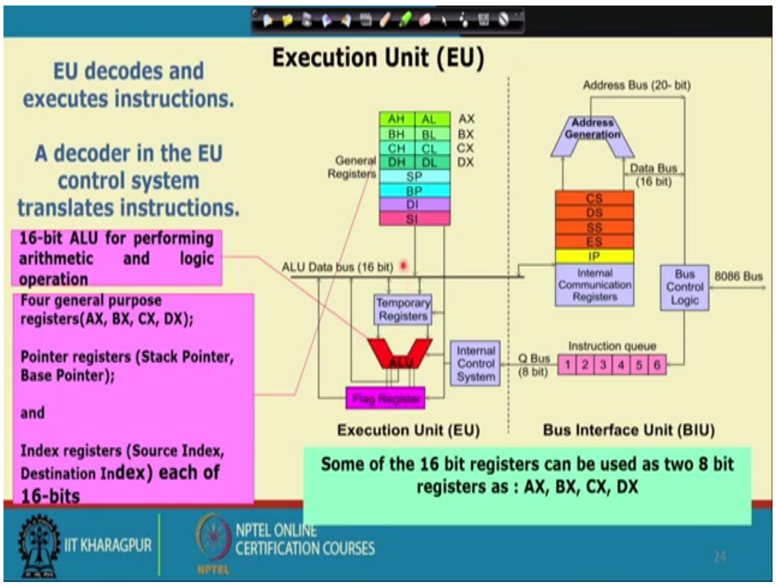
So, the next important component in the bus interface unit we you have is the instruction queue. As I said that there is six byte instruction queue and each of this location is 8-bit wide and there are six such queues. So, you remember in case of 8085 we had one instruction register or IR. And I said that whenever the instruction is brought from instruction is brought from memory into to the CPU, so this instruction is coming to the instruction register.

So, here that has been taken one step further like since the many a time what will happen is that this execution unit, so this will be busy do I executing some instruction. And while it is executing the instructions, it may not be using this bus, because it may be doing the register operations here the maybe it is all operands in the register, so that way it does not need to access the bus. So, this address this bus interfacing unit instead of waiting for the current instruction to be over, so it can ask to get the next instruction next bytes of the program from the program memory. So, this is exactly what is done. So, instruction queue the instruction surface and they are kept in first-in-first-out order.

So, this is a first-in-first-out queue in which or the FIFO queue in which the up to 6 bytes of instructions code are pre fetched are fetched on the memory at a time, ahead of time. So, we will see that many of the instructions of this 8086, they can go up to 6 bytes, so that is why this number 6 has been taken, so that if the six bytes are faced that means, at least definitely one complete instructions is available in the queue. Now, of course, there are instruction which are smaller than 6 bytes, so that does not matter, but this it will always try to fill up this six byte queue. So, this is done in order to speed up the execution by overlapping the instruction fetch and execution. The fetch and executes so they are overlapping when it is internally doing the execution part. So, we can have this fetch part, the fetch part going on through this bus.

So, this way we can have this instruction. So, this is this is that pipelining because that we have shown previously that between the fetch decode and execution there is a pipelining. So, here the pipelining has been taken a bit ahead we can say where this fetch is having a 6 byte pre fetch of this instructions from memory.

(Refer Slide Time: 26:53)



Execution Unit (EU)

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

16-bit ALU for performing arithmetic and logic operation

Four general purpose registers(AX, BX, CX, DX);

Pointer registers (Stack Pointer, Base Pointer);

and

Index registers (Source Index, Destination Index) each of 16-bits

Some of the 16 bit registers can be used as two 8 bit registers as : AX, BX, CX, DX

So, next we will look into this execution unit part. In the execution unit, so the responsibility is to decode the instruction and execute it. So, there is a decoder in the execution unit that is the control system this control systems of this the decoder. So, from this instruction queue, it will get the successive bytes, and it will go do this internal control system. So, it will be determining the control sequence as we have seen in 8085 also at different clock steps the signals are to be generated. So, it will be doing that, it will be generating those internal signals for different clock cycle. And accordingly it will be activating this registers, this ALU, and this bus this the (Refer Time: 27:39) will available on bus etcetera and that way it will be doing the operation.

So, this 16-bit ALU has been provided. So, this ALU is 16-bit whereas, this adder that we have in bus interface this is 20-bit. So, there is a difference. This is 16-bit and that one is 20-bit. So, this 16-bit ALU for performing arithmetic and logic operations then this there are registers we have got general purposes registers AX, BX CX and DX. So, these registers are 16-bit registers, and they can also be thought about to consisting of 8 bit pairs, like AX is can divided into two registers and AL of eight bit each, BX can be thought about two bit two bit registers BH and BL. So, they can be accessed either as 16-bit registers or two eight bit registers.

And there are some more 16-bit registers the stack points are SP the base point are BP and the indexed registers SI and DI. So, SI stands for source index; and DI stands for destination index. So, this SI and DI instruction they are very much used for string manipulation, and this block transfer of data. So, some of this as I said some of this 16-bit registers can be accessed as two 8-bit pairs that we have seen and this SI, DI. So, this source index destination index, so they are also useful for transferring the content of a portion of memory to some other portion, the multi byte data movement, so that the for that purpose also this SI, DI are utilized.