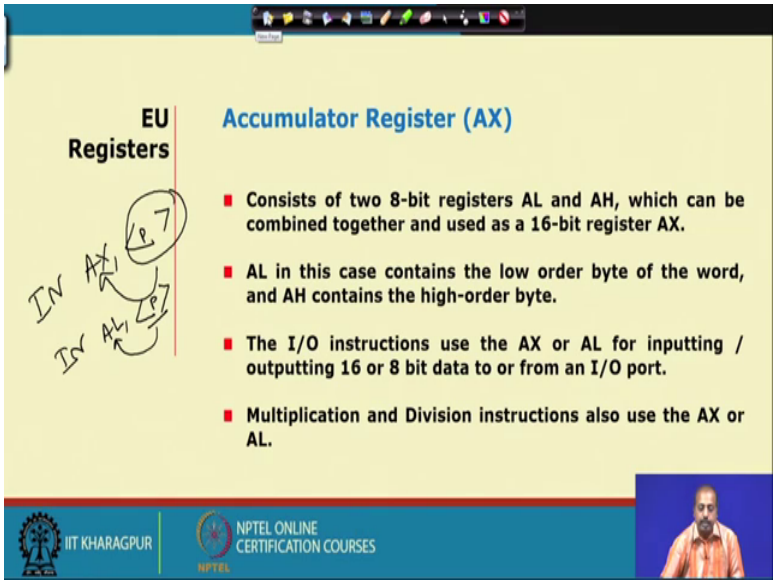
Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E and E C Engineering
Indian Institute of Technology, Kharagpur

Lecture - 61
8086 (Contd.)

Accumulator register or AX.

(Refer Slide Time: 00:19)



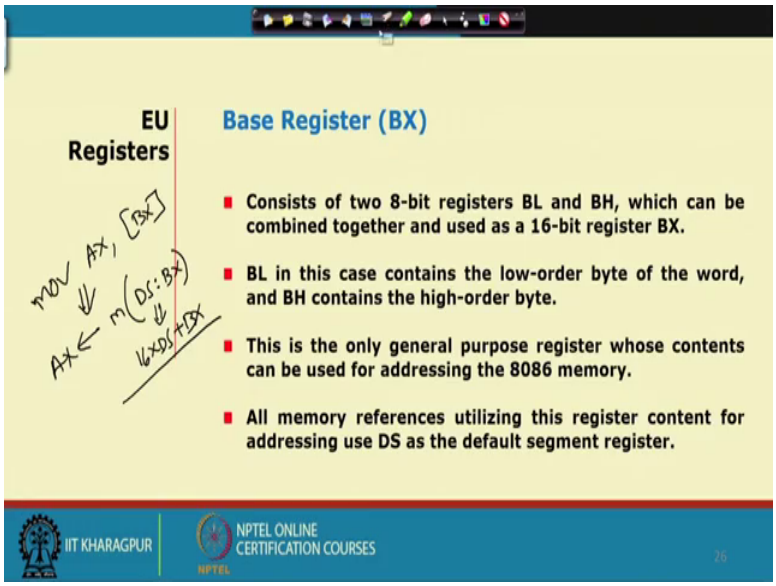
So, we will look into these registers that we have in this execution unit, the first one is the accumulator register as I said that it can be considered to be 2 8 bit registers AL and they can be combined together and used as a 16 16 bit register AX, AL it contains a lower order byte and contains the higher order byte. So, AL is the lower order one, is the higher order one the I O instructions use that AX or AL for inputting or output AC out 16 or 8 bit data to or from the I O port.

So, you can have 16 bit I O port you can have 8 bit I O port. So, if you are having 8 bit I O ports then you have to use this AL or AL register for outputting the value and if you are outputting a 16 bit value then the AX register can be used. So, if you are. So, the instruction itself will be identifying the thing like if you say in AX some port number, some port number should be there. So, if you are reading from this port address. So, this is assumed that this will be a 16 bit port and the 16 bit value will come to the AX register, if this is an 8 bit port then you can write in a similar fashion in AL comma some

a s and again the port address and this port is assumed to be 8 bit port and the 8 bit value will be coming to the AL register.

So, this way we can have both 8 bit and 16 bit access for this and a AX register the multiplication and division also use the AX or AL. So, these they can also use this AX or AL register. So, for 16 bit multiplication so it will be AX register for 8 bit multiplication will have AL register another register b x.

(Refer Slide Time: 02:07)

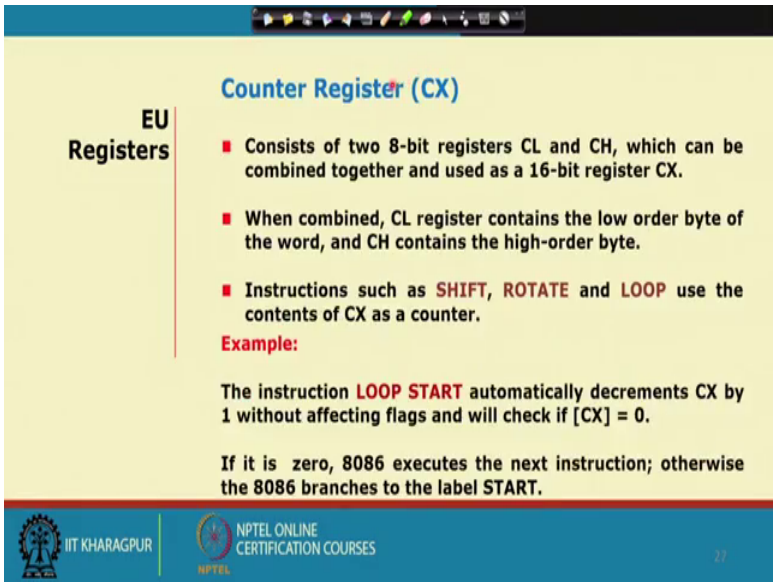


So, BX is a general purpose register, but it has got some other functionality also for the memory addressing that is why so this is called the base register. So, it again it consists of 2 8 bit registers BL and BH. So, they can be combined together as a 16 bit pair into a BX register, the BL is the lower order byte BH is the higher order byte. So, this is the only general purpose register whose contents can be used for addressing the 8086 memory. So, you can use this BX register as a something like a pointer ok.

So, you can use it as a address for addressing this memory. So, all references utilizing this register content for addressing used ES as the default segment register. So, you can have instructions like say, you can have instructions like say MOV can have instructors like MOV a x comma within bracket b x. So, the meaning of this instruction will be something like this that AX register will get the content of memory location d s colon b x ok. So, SS colon B so, DS into. So, this is DSS 16 into DS plus BX. So, that way it will come. So, it will be having this 16.

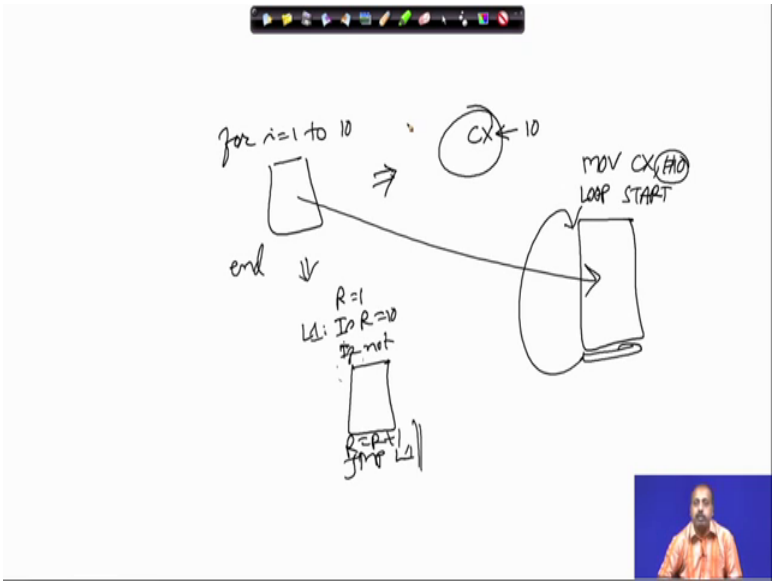
So, this BX register is the only register that can be used for, only general purpose register that can be used for memory addressing and this is h and it is that it is called based addressing, we will see under in based addressing this will be utilized and this will be using the DS register as the segment register. .

(Refer Slide Time: 03:51)



Next we will be looking into the CX register, now you see that each of these registers apart from a x which is accumulator. So, other registers BX CX. So, they have they have been given some special names ok, unlike say 8085 where we have got this ABCD like that. So, it is not that way. So, these ABCDs were a bit arbitrary, but here it is not. So, here this CX register is another 16 bit register consisting of 28 bit register CL and CH CL contains the higher order lower order byte CH contains higher order byte, then we have got a number of special instructions like shift rotate loop etcetera where the CX register is used as a call as a counter. So, actually a typical example can be of this loop instruction, you see and that while using a loop like say if I want to have a loop.

(Refer Slide Time: 04:53)



So, I can I suppose in high level language we have got a loop like this for I equal to 1 to 10 ok.

So, this body of the loop will be repeated and if this is the loop so, in case of assembly when I convert it into assembly level a program for 8086 what I can do I can load this CX register with the value 10 by having instructions like say MOV CX comma hash 10. So, c x register value has got a is having the value 10 and then I can say loop start. So, what it will do it will repeat a loop body, it will repeat this loop body number of times equal to this 10.

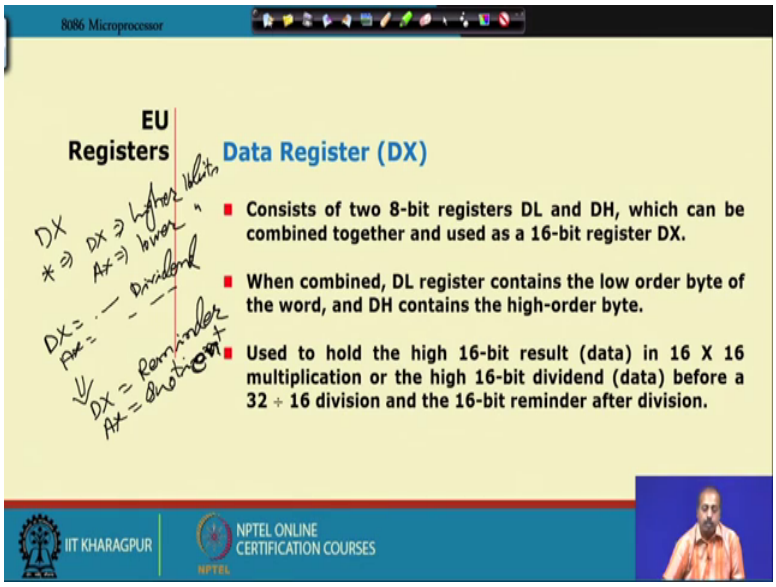
So, I do not have to do anything. So, this loop body can be put here. So, it will be repeating this loop body and we do not have to check this the end of loop condition at the end. So, normally if we are writing this program in 8085. So, it will be looking something like this first initializing some register to some register r 2 10 and then writing the loop body then we have to check for the sorry, before that we have to have some check also before that we should have a check that whether R is equal to this R you have to initialize to say one then we put a check here whether is R equal to 10 and if not if not then I should have the body here and then I have to again increment R R equal to R plus 1 and then jump to this position again say this is say 1 oh 1. So, at a jump to l o 1.

So, the programs will look something like this, the structurally it will look something like this now you see these extra things need not be done in case of 8086 loop

instruction. So, these things will not be required and the CX register. So, this this will hold the iteration count how many times this loop body will be executed. So, that count is kept in the CX register. So, I will come back to this instruction when you go to this 8086 instruction set. So, this loop starts.

So, this will automatically decrement CX register by one without affecting flags and we will check if CX becomes equal to 0, if 0 then the 8 0 its if it is 0 8086 execute the next instruction otherwise 8086 will branches to the label start. So, at the end of the loop body so you can just say loop start so, it will be jumping back to the loops lower the level start if CX is non 0 and if CX is 0 then the loop body is over. So, it will be automatically going to the next instruction. So, we can use it for writing something called 0 overhead looping. So, there is no extra overhead that we have to pay for this looping purpose.

(Refer Slide Time: 07:55)



Then another register is the DX register or data register and this has got again DL and DH to 8 bit registers. So, when combined with DL this DL register will contain the lower order byte and the DH register will contain the higher order byte now used to hold the high or 16 bit result in 16 cross 16 multiplication.

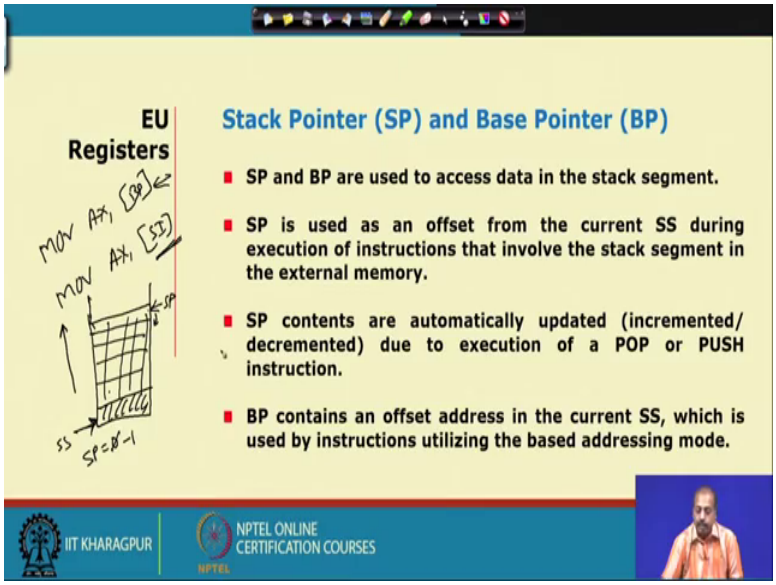
So, if you do 16 cross 16 multiplication the result will be 32 bit and for the 2 bit result this DX register will hold the higher order byte and this A X register will hold the lower order byte, for the multiplication instruction. Similarly, if you are doing the division operation this high 16 bit dividend A x before 32 by 32 by 16 division and the 16 bit

reminder after division. So, this DX register. So, this is particularly suitable for this multiplication and division operation. So, DX register so for multiplication. So, DX will have this higher order 16 bit higher 16 bits and your AX will have the lower 16 bits and for division. So, we have to work we are going to divide a in the 32 bit data by a by a 16 bit data so, this high 16 bit dividend before with this 32 by 16.

So, this DX register we will have this high order 16 bits of the dividend, higher order 16 bits of the dividend and this lower order 16 bits will be in the AX register, this will have the lower order dividend and the after division this DX register will contain the remainder. So, then the division will be done and then there the DX register will have the reminder and this AX register will have the quotient, AX will have the quotient and DX will have the AX will have the quotient and DX will have the reminder. So, that way this DX register is useful particularly for multiplication and division operation, otherwise you can use it for as a general purpose register of course.

So, that is always there, but special function for this multiplication and division operation 2 more special purpose registers stack pointer and base pointer.

(Refer Slide Time: 10:26)



So, both of them are uniform, in the sense that both of them will access the stack segment. So, any instruction that uses stack pointer or base pointer as an operand will be looking into the stack segment of the memory. So, if you say that MOV for AX comma b p. So, if you have an instruction like MOV AX comma b p then it will not look into the

data segment, but it will look into the stack segment whereas, if you look into the instruction MOV AX comma within bracket si. So, then it will be looking into this, it will be using des registered as the segment register.

So, it will be using the data segment whereas, your if you are writing like this in this case the segment register will be the stack segment. So, this SP and BP they are used for accessing stack segment, SP is used as an offset from the current stack segment register during execution of instructions that involve stack segment in the external memory. So, basically we have got this stack segment register and then whenever you are doing this push pop operations then this stack pointer will be utilized and the stack pointer will automatically be updated incremented or decremented due to pop or push instructions.

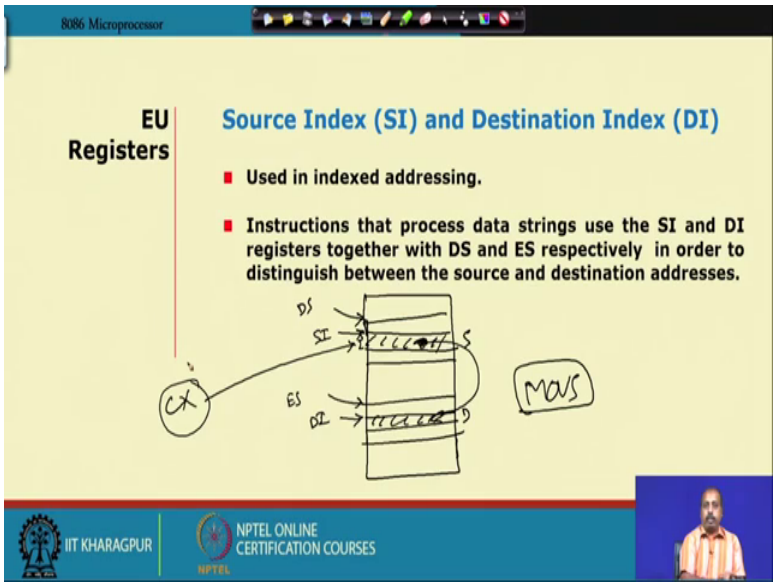
So, when you do a pop then the stack pointer is incremented, when you do a push stack pointer is decremented. So, this stack pointer because this is a stack pointer will be a stack this SS register will be pointing to say this one, the bottom of this memory where this the stack will be made and then as you are doing a push operation the stack pointer is decremented.

So, this so currently the stack pointer value will be same as if you if you if you have got this stack segment register here and memory addresses are decreasing in this direction ok. So, memory addresses are decreasing in this direction and then stack segment register is put to the maximum value and this stack pointer register is made to be equal to 0, now after that whenever you are doing a push. So, this value will become minus 1. So, this. So, value will be decremented by 1. So, you will get it so it is accessing it will be getting this location where the value will come in a push instruction or if you are doing a pop operation in that case the stack pointer value will be implemented, say at some at some point of time may be the stack is full up to this much, all these locations are full and the stack pointer is pointing to this.

Then in that case the stack pointer will be incremented because after doing this operation the stack pointer should come down. So, stack pointer value should increase. So, it will be doing that ok. So, this way we can have this stack pointer and this stack segment accessing they are being accessed for this push and pop instructions and the base pointer. So, this is an offset within the current stack segment. So, that can be used for the base addressing mode.

So, as I said that many a time when you are passing parameters etcetera then to access parameters in the called procedure. So, it is necessary to look into the stack segment and for this purpose this base pointer can be utilized for going to different parameters that you have passed.

(Refer Slide Time: 14:00)



So, there are 2 more registers in the execution unit special purpose register, one is called source index register SI another is the destination index register or DI they are used for indexed addressing. So, as you know that indexed addressing is one of the very popular addressing modes where this, these registers can be used as index. So, you have got this DS register pointing to the beginning of the segment and DS colon si. So, that will be pointing to the particular entry within that segment.

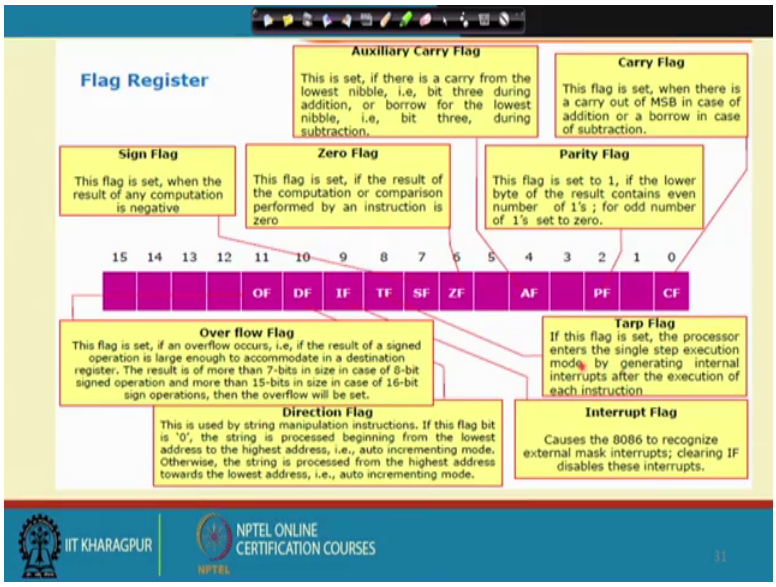
So, this SI and DI registers. So, they are used for this indexing purpose, this instructions that process data strings are use this SI and DI registers together with the SI will be d d s as a segment register DI will be using ES as the segment register in order to distinguish between source and destination addresses. So, if we have got, if this is, if you have got say this as the memory and then in that memory suppose we have got 2 segments.

So, as I was telling this is one segment and this is another segment and you want to transfer a number of bytes from this space to this space. So, this is thus there. So, you have got these are the source and this as the destination, there will be number of bytes that you want to transfer.

So, what you can do this segment register you make it 2.2 by ds this segment register you make it 2.2 by ES the SI register you initialize here and DI register you initialize here and you can initialize and you initialize another register CX the count register to hold the number of bytes that you want to transfer ok, from the source how many bites you want to transfer the number of bites you can load in the CS register and then you can use this string instruction MOVs there is a string instruction MOVs.

So, if you use this, what it will do it will automatically transfer the content of these memory locations to this memory location. So, it will transfer say ca whatever count has been put into this CX register. So, many bytes it will transfer from the source block to the destination block. So, this way this source and destination index registers can be useful for doing the data transfer.

(Refer Slide Time: 16:30)



Another register that we have is the flag register. So, this flag register the bits that are important the 0 with a bit 0 which is the carry flag. So, this flag is set when there is a carry out of the most significant bit in case of addition or a borrow in case of subtraction, this carry flag is said this say this similar to what we have in 8085 also the status train is the PSW register. So, it is similar to that then we have got bit number one is unspecified. So, this may be for their internal purpose, then this bit number 2 is the parity flat. So, this flag is set to 1 if the lower byte of the result contains even number of ones and for odd number of forward number once it is set to 0.

So, if you think the number of ones in the lower order byte is even then it will be 1. So, that is the overall result should be odd parity. So, it following the odd parity principle then again the bit 3 is not used then we have got this auxiliary carry flag at bit number 4. So, this is set if they carry from the lowest nibble that is the minimum of the least significant for 4 bits that we have.

So, bit 3 during addition or borrow for the lowest level that will be 3 during the subtraction. So, if there is a burst carry from the lowest nibble, that is the lowest 4 bits then this bit will be set or there is a borrow for the lowest 3 bits. So, this bit will be set to one there bit 5 is again of not utilized, then bit 6 is the 0 flag.

So, this is the result of computation or comparison performed by an instruction is 0 if the result is 0 like if you are doing the equality check or if you are doing some addition or subtraction operation and the result becomes 0 in that case the 0 flag will be same, then we have got sign flag s f. So, this flag is said when the result of any computation is negative, the result is negative in that case this SF flag will be the sign flag will be set then we have got TF flag. So, it is the tar flag. So, this means that it will be it will be entering into the single step execution mode.

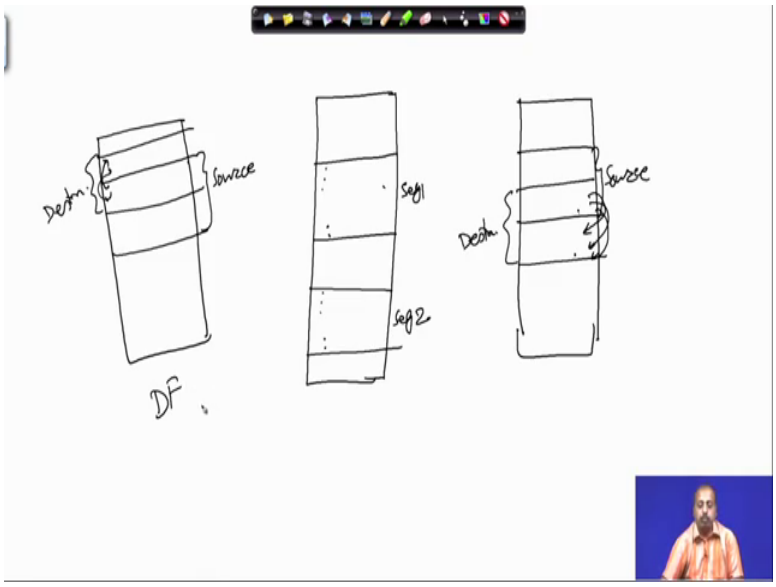
So, single step execution or it is or single stepping. So, this is useful when you are trying to debug a program like in the initial stage of program development there may be we write the program, but we may be the program is not giving me the correct result. So, to understand what may be the possible problem. So, we need to run the program in the in a step by step, one instruction at a time and we can trace through the program to see like how is it going on so, stopping the program after executing every instruction so it may be difficult.

So, if you set this TF bit then it will be ensuring that a single step execution mode by generating internal interrupts after execution of each instruction. So, after execution of each instruction it will generate an interrupt that it waits. So, as if as a debugger routine so we can have that interrupt service routine so, the inter service routine is actually the debugger code that we have. So, now, it can access the internal registers depending upon whatever value the user wants to see they can be shown to the user. So, that way the a debugger design can be facilitated by using this TF bit, then there is I f bit which is the inter flag.

So, this causes a generate seeks to recognize external mask interrupts and clearing it visible this interrupts. So, basically the mask able interrupts that we have in 8086. So, if you set it to 1 then this mask you will interrupts will be reaching the 8086 processor, if you set it to 0 then these interrupts are all disabled.

So, there is no interrupt or reaching the 8086 processor, then we have got this direction flag DF. So, this is again for the string manipulation operation if this bit is 0 the string is process beginning from the lowest address to the highest address that is in auto incrementing mode otherwise the string will be processed from highest address to towards the lower address in auto incrementing mode auto decrementing mode.

(Refer Slide Time: 20:45)



So, it is like this , so it is like this that I want to MOV this code like if this is if this is the memory and we have found that I can transfer the string manipulation instructions. So, they are useful mainly for this transfer of one block of data from one chunk to another, now if these chunks are not all overlapping. So, this is this is this is the segment 1 and from segment one I want to confer it to segment 2. Now, if the segment one and segment 2 they are all overlapping then I can copy this byte here then the next byte at a next location.

So, I can go on doing like this or I can do the other way also I can start copying from the bottom first copy this one here, then copy this one here. So, that is the way I can do it like this, but if the segments are overlapping in nature that is in the memory. So, I have

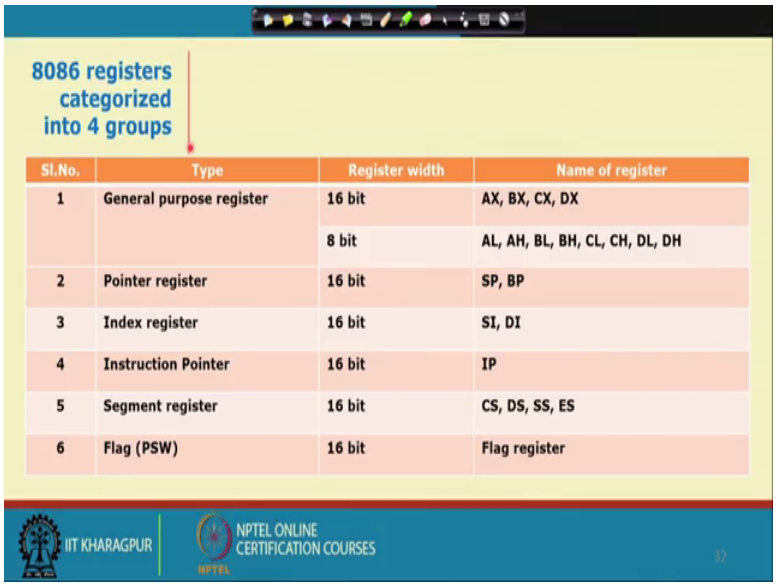
got this source segment like this, this is my source segment or the say one that we have talked about and this is the destination ok. So, this is the destination and this is the source, in that case we need to MOV from the bottom otherwise the content will be overwritten. So, I need to start from the bottom first copy here then there I need to proceed like this or if the overlapping is in the other way. So, over lapping is say like this, this is the source and this is the destination.

So, this is your destination and this is the source, if it happens like this then I have to start copying from the beginning. So, from the first from the first location I have to start copying from the beginning of source I have to start copy. So, this direction can be controlled by that DF flag in that status register. So, da flag if you set it to 0 then it will be copying in one direction and if you set it to one to be copying in the other direction. So, by (Refer Time: 22:49) looking into this addresses of this source and destination blocks. So, you can just figure out like which directions it should be set and depending upon the overlapping pattern so we can select the direction. .

That is that DF flag or direction flag and there is an overflow flag. So, if this flag is set if an overflow has occurred that is if the result of a signed operation is large is large enough to to accommodate in a destination register then the overflow flag will be 0 and the result is more than 7 bits in size in case of 8 bit signed operation and more than 15 bits in case a 16 bit sign in operation the overflow will be set.

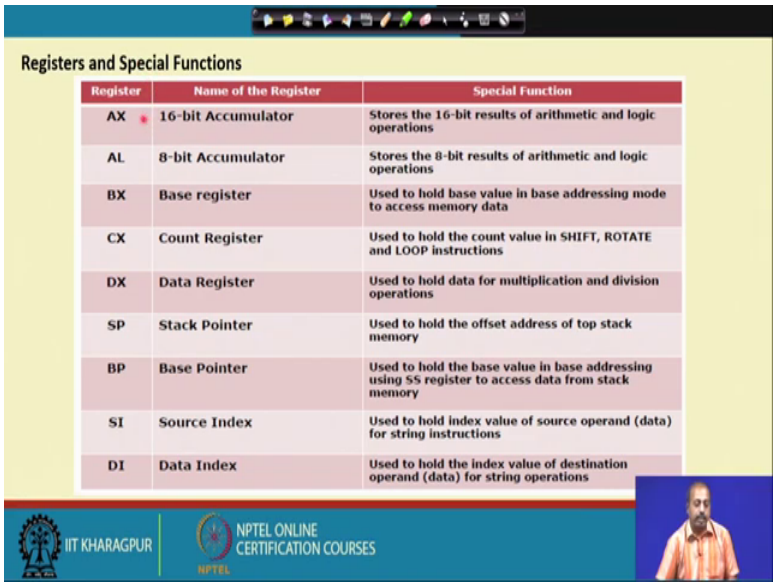
So, it is if overflows overlap like if you in 2s complement number system you know that if the total space is given as 8 bit then your number should come from, number should be realizable as is minus 2 to the power 7 to plus 2 to the power 7 minus 1. So, that if it is not realizable in 7 bits then there will be an overflow. So, all numbers in 2s complement cannot be represented in 8 bit similarly when you are going for 16 bit representation, it goes up to 2 power minus to power 15 to plus 2 power 15 minus 1. So, if you are going beyond that range then the overflow will be set. So, otherwise the overflow bit is 0. So, this flag register it will have all these information and we can use it for different condition checks.

(Refer Slide Time: 24:27)



So, next we will look into a general.

(Refer Slide Time: 25:36)



So, this is the summary of these registers that we have in 8086, they can be categorized into 4 groups, group one are the general purpose registers those general purpose registers can be 16 bit or 8 bit.

So, AX, BX, CX, DX so they are the 16 bit register and AL, BL, BH, CL, CH, DL, DH they are a bit registers then there are 2 pointers 16 bit pointers SP stack pointer and BP the base pointer there are 2 index registers SI and DI source index and destination index

they are again 16 bit then there is an instruction pointer IP ok. So, that is the the instruction pointer register that we have, then there are segment registers CS, DS, ES s and s code segment, data segment, stack segment and extra segment they are again 16-bit registers and then we have got another flag register or PSW processor status word. So, that is also a 16 bit register.

So, these are the registers that we have in 8086 and they are they have got different purposes. So, this purpose you can summarize like this that AX register is a 16 bit accumulator register it has got so we have got also for 8 bit we have got AL that will act as accumulated. So, this is the x will be you will be used for 16 bit operations and AL will be working as accumulator for 8 bit operation, then the BX register base register BX this holds the base addressing base value of the base addressing mode and they. So, therefore, based addressing mode we will see, then the CX is the special function this is for the count it a count register, used as count register for holding the count value for shift rotate loop instructions.

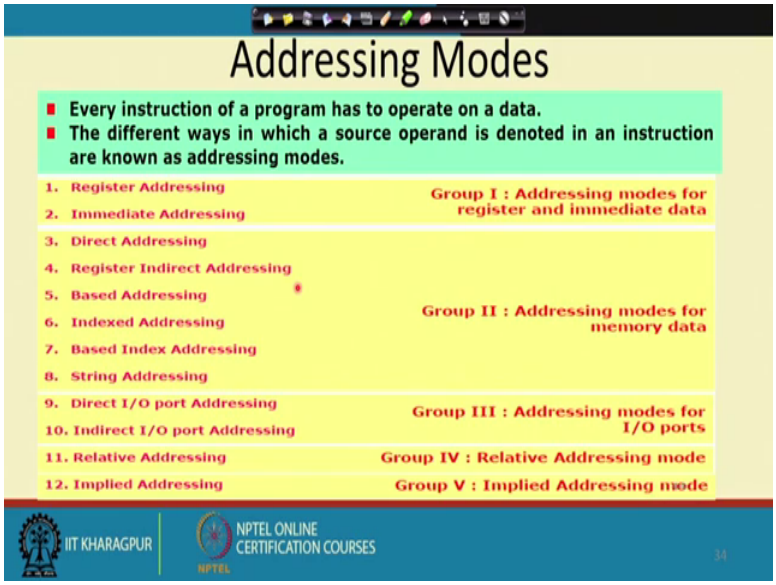
Then that DX register so this has got this is the data register they apart from working as a general purpose register. So, it has got the special function for holding the multiplication and division operation, during multiplication or division operation the higher order it be higher order 16 bits will be stored there, then the stack pointer register sp. So, that is used to hold the offset address of top stack memory. So, this is stack segment register will be pointing to the bottom and the stack pointer will be pointing to the offset within that, then we have got the base pointer.

So, this is used for getting operands away from the stack ok. So, the this is using the stack segment register as a segment register and then we can have this accessing within the stack segment by this VP then this source index register SI. So, this hold the index value of the source operand for the string instructions and DI data index or destination index. So, this is for the destination operand the for string operation next we look into the addressing modes.

So, just like these register so addressing modes 8086 supports a good number of addressing modes every instruction or program has to operate on a data and there are different ways in which the source operand can be denoted in instruction. So, this is that

this is actually the addressing mode. So, that we know in 8085 also we have seen a number of addressing modes.

(Refer Slide Time: 27:50)



So, in case of 8086 so we can have different types of addressing like register addressing, where a register is used as the operand then immediate addressing where the value on which this operation will take place is specified with the instruction itself ok. The constant values are specified, then we have got direct addressing where we have got this memory address given directly in the instruction, then register indirect addressing in which this register will be acting as the address and the register content will be used to access the memory, then this based addressing where it is similar for it is particularly useful for array type of access.

So, there is a base address and with respect to that it will be accessing it some base registers can be used, then we have indexed addressing the SI and DI registers are used for indexing purpose, then based indexed addressing what is base and index. So, they are combined together for based indexed addressing, then string addressing is there, then for port access. So, we have got direct I O port addressing and indirect port a I O, I O port addressing and for jump instructions and this call instructions we have got relative addressing and we have got so this relative addressing may be relative to the current location and we have got implied addressing where the operands need not be specified

separately the operand is operand is implicit in the instruction itself. So, we will see this addressing modes after this, so different modes of operation.