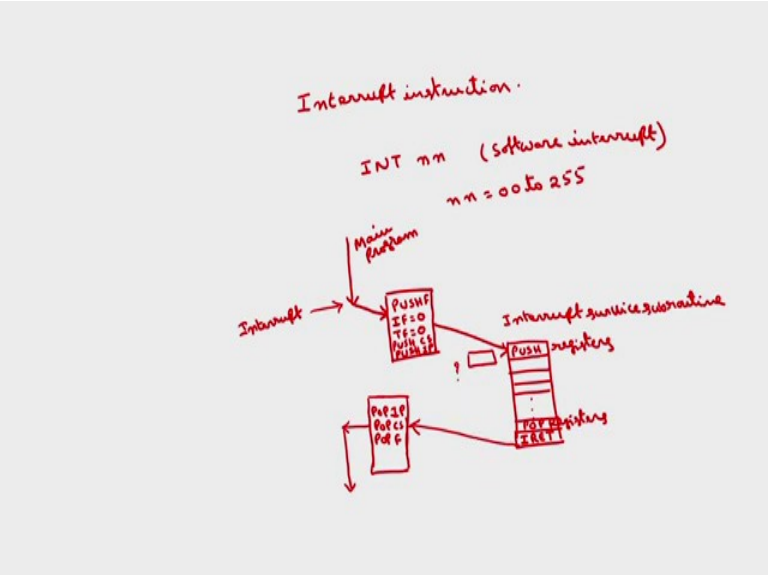
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture - 11
8086 Interrupt and Machine Control Instructions

We were discussing about the different instructions of 8086. So, we have discussed different instructions like data transfer group, arithmetic group, logical group and some rotate and shift instructions, loop instructions, branch instructions. The next group of instructions will be interrupt instructions.

(Refer Slide Time: 00:45)



So, the general format of interrupt instruction is INT n double n you can call. So, this n n varies from n n can be 00 to 255. In fact, F F ok we can write F F. So, that is why I have written 2 n ns. So, whenever this microprocessor execute this interrupt instruction, what are the actions that will be taken by the microprocessor.

So, the general flow of the interrupt will be. So, if this microprocessor is executing the main program this is main program. If at this point interrupt occurs then the microprocessor, what it will do is it will first push the flags, all the flags will be pushed onto the stack, then it will disable interrupt flag interrupt flag will be reset. So, if interrupt flag is reset the remaining all maskable interrupts will be disabled except the NMI which is non maskable interrupt then it also reset trap flag.

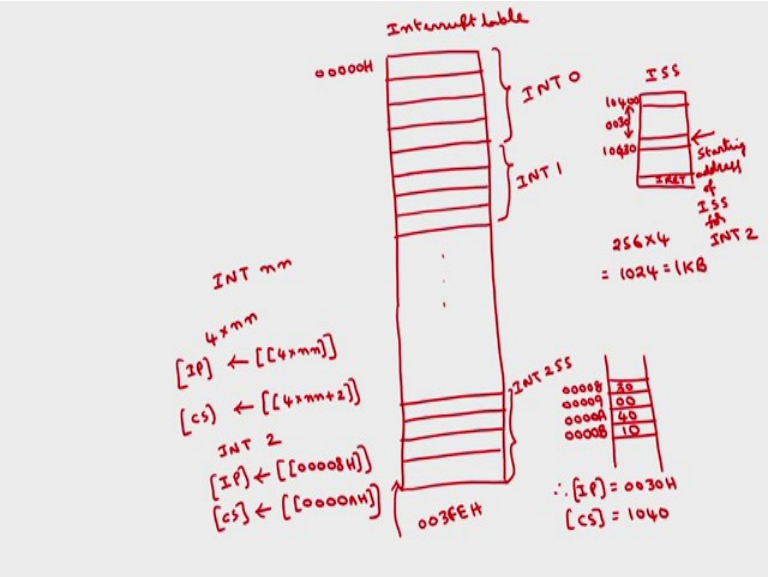
So, it will disable the single stepping operation, then it will push code segment register. Then, it will push instruction pointer, then the control will goes to subroutine interrupt service subroutine. So, in the interrupt service subroutine, first it will push all the registers, then it will execute the subroutine and before it is returning it will pop all the registers. So, this is while it has to push all the registers and why it has to pop the registers. Because, if I use this register for some purpose in the main program the same registers has to be used in the subroutine also.

So, if you do not push if you do not save the contents of this registers and if you view for some other purpose, the old contents will be lost. That is why it is always normal practice to push all the register contents onto the stack, in addition to the flag as well as C S and I P, then at the end of the subroutine we will pop all the registers. And, the last instruction of the interrupt service subroutine is I return. Now, if you take any subroutine general subroutine, the last instruction will be just return. So, the only difference between the return and I return is; I return is interrupt return.

So, while it is returning back. So, it will perform all the reverse operations of that of this push, which is it will pop first IP, then pop CS, then pop flag. Because, we know that the stack pointer is last in first out. So, whatever the contents inputted last will be outputted first. So, this was instruction pointer was pushed last. So, it will be popped first and so on. Then, the main program will continue, this is the flow of the main program. So, whenever a interrupt occurs at this point, these are the various steps that the microprocessor performs to complete the task of the interrupt.

Now, here there are 256 software interrupt they are called software because there is a instruction, this interrupt can be called as software interrupt. In 8086, there are 2 hardware interrupts, which is INTR and NMI, which we have discussed while discussing about the pin diagram of 8086. So, these are called software interrupts. Here the operation is corresponding to each interrupt INT 0 INT 1 so on up to INT 255, we require 4 locations for each interrupt.

(Refer Slide Time: 05:53)



Total 256 interrupts and 256 and each interrupt requires 4 locations. So, it will comes to 1024, which is 1 K, 1 kilobyte of the locations will be used for storing the interrupt addresses, these 4 are for type 0 INT 0. So, whenever this INT 0 is executed. So, as we told that the contents of the code segment will be saved on to the stack, contents of instruction part will be saved on to the stack. Now, what will be taken into the code segment and instruction pointer?.

Like here what will be the address of the starting point of this? What will be address of this and what will be the segment register? To find out that so this 4 locations will be used. So, in general if this INT n n, 4 into n n will be computed and this contents of this 4 into n n will be stored into contents of contents of 4 into n n will be stored into instruction pointer, this is a word. Because, instruction pointer is 16 bit and contents of contents of 4 into n n plus 2 will be stored in code segment register.

For example, if I take INT 2 it can be 0 2 or 2 simply then what will be loaded into instruction pointer will be instruction pointer contents will becomes, contents of contents of 4 into n n will be 00008H. And, contents of CS will be contents of contents of 0000A H. So, these addresses will be stored into the contents of this is 00008H is 4 into mm. So, 4 into 2 is 8 so, this is 8 4 into 2 8 plus 2 is 10 so, 0000H.

The physical address from where the contents of IP will be loaded will be from 00008H. So, if you have 00008H in the memory, if you have something like 3000 some 4010.

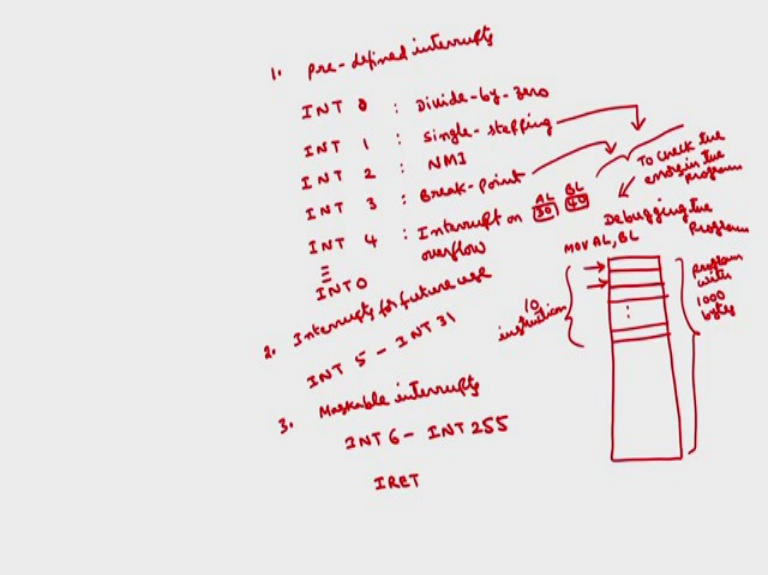
Then, what will be the contents of instruction pointer the contents of contents of this. And, because the instruction pointer is 16 bit it will take a word from 2 consecutive locations. So, the contents of instruction pointer will becomes now, 0030H. And, contents of code segment register will becomes contents of contents of 000A means 1040. So, the physical address from where the interrupt branch address will be in code segment so, if I take this previous this thing.

So, whenever the microprocessor is interrupted, the interrupt service subroutine. Interrupt service subroutine will be executed from the location for this INT to the interrupt service subroutine I will call as ISS interrupt service subroutine. So, this will be in the code segment, whose code segment register is 10400 H this is called 00 with a offset of 0030H, then this address becomes 10030H, 1 0s 4 3 0 H. So, this is the starting point of instruction service subroutine, starting address of interrupt service subroutine for INT 2. So, this is how the branch address of the interrupt will be computed?.

So, whatever this n, that will be multiplied with 4 the contents of contents of that will be stored into instruction pointer and the contents of contents of this 4 n plus 2 will be loaded into code segment register, then within this code segment. So, this is the offset. So, from that address onwards the microprocessor will execute the interrupt service subroutine and the last instruction will be stored here in somewhere I return. So, this 4 locations will be used for storing this IP as well as CS contents and the next 4 locations will be used to 4 store, the IP and CS contents of INT 1 and so on the last 4 will be for INT 255.

So, if this address is say for example, 00000H then the last address will be this address will be 003FEH, because as I have told in the earlier classes 1 K means total 3 F F locations starting with 0. So, it will go up to 03FEH. So, this is called interrupt table. So, where the contents of instruction pointer and code segment corresponding to each interrupt will be stored. So, this is the process of interrupts, then we have total 256 interrupts and there are some special purpose of this instructions like INT 0.

(Refer Slide Time: 13:47)



INT 0 is normally used for this is called divide by 0. Whenever in the division operation a divide by 0 operation occurs automatically INT 0 will be executed. INT 0 means, corresponding address will be computed by using so, 0000H 0 into 4 plus 2 will be the contents of instruction pointer. So, from there some subroutine will be there that will be executed whenever INT 0 is executed, means whenever a divide by 0 operation occurs.

Similarly, the special purpose of INT 1 will be this is called single stepping. This will enable single stepping operation, whenever INT 1 instruction is executed by 8086 it will operate in single stepping. I have already discussed about single stepping. So, this single stepping operation will be used for debugging the program, debugging means checking for the errors to check the errors in the program.

Suppose, if I have a program which consisting of 1000 locations, 1000 bytes, new program consisting of 1000 bytes. If, I execute this entire programs in a once, and if I get after execution if I get some error, then it becomes difficult to I mean trace where the error has been occurred because there are 1000 locations. So, one easier way to I mean check the errors is first you execute the first instruction, after that you check the contents of the registers.

Suppose, if I have something like say first instruction is move AL comma BL. Before, this if I have some AL contents are some 30, BL contents are sum 40. So, what is expected here is after the execution of this particular instruction, I will check the contents

of AL and BL ok. If, AL is having 40 BL is also having 40 then the first instruction is correct. If, AL contents 30 BL contents 40 means, there is a mistake in the first instruction. So, I will check where the I mean mistake is there. If, the first instruction is fine then I will go for the second instruction, I will execute this instruction again I will check the contents of the registers.

If, the all the contents of registers are fine then I will go with third instruction like that I will execute the instructions in 1000 steps. If, I have 1000 bytes in this program, I will execute this in 1000 steps. After, each step I will check the contents of the register and I will check if the contents are fine, then there is no error I will move forward with the next instruction, this is what is called single stepping. So, whenever this INT 1 is executed the microprocessor operates in single stepping operation.

INT 2 is non maskable interrupt. So, we know that there is one hardware interrupt, which is called non maskable interrupt. So, that will be enabled if INT 2 is executed, INT 3 is break point it will create the break point. So, this break point whether is also another method for debugging the program. So, these two are used for debugging the programs, this and this will be used for debugging the programs.

So, difference between the single stepping and break pointing is. So, in case of single step as I have explained I will execute one instruction, and I will check for the results second instruction, I will check the result third instruction, I will check the result and so on. Because, in case of break point, I will set a break point after say some 10 instructions.

So, the programmer can set this number of instruction to be executed at a time by the programmer. So, if I set this to the 10 instructions. So, the microprocessor will execute first 10 instructions at a time. Then, I will check the contents of all the registers. If, the contents of all the registers are fine, it means the 10 instructions are correct. Otherwise, there is a mistake in one of this 10 instructions. It is easier to I mean debug 10 instructions rather than 1000 instructions ok.

So, after if this is correct then I will move to another 10 instructions. So, like that now this 1000 bytes will be executed in 100 steps ok. So, one is executing single instruction, one is executing the entire program in a once, this is in between to this single stepping and executing the program whole in a once ok.

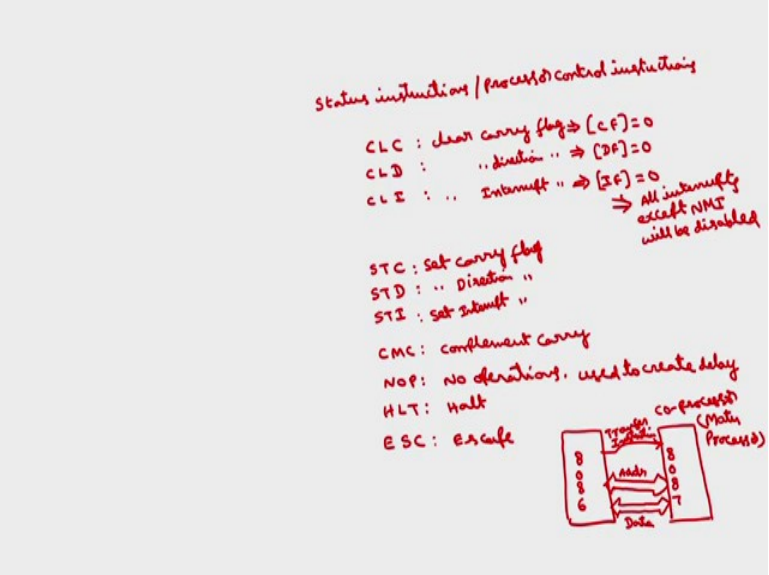
So, in case of breakpoint, we can set some instructions after the few instructions are executed at the same time. I will check for the results. So, like that I will proceed. So, in order to I mean operate this microprocessor in breakpoint method, INT 3 interrupt will be used. INT 4 will be this is interrupt on overflow, this also can be called as INTO interrupt on overflow.

So, whenever a particular arithmetic operation is executed by the microprocessor and if the result is outside the range, as we have discussed in the earlier classes for 8 bit, the result should be signed number result should be from minus 127 to plus 128 ok. So, if the result is beyond this range, then the overflow flag will be set automatically the microprocessor will be interrupted and that will jump to the locations that will be computed by using the procedure just now I have described, this is about INT 4.

So, this 5 interrupts 0 to 4 are called predefined interrupts. Then, we have some interrupts which are used for the future purpose; interrupts INT 5 to INT 31 they are used for the future purpose by the INTEL. And, there are some interrupts, which are used for maskable interrupts. The remaining interrupts INT 6 to INT 255. They can be used as a maskable interrupts. There are 3 groups, one is predefined and second one will be interrupt for future used by this microprocessor and third one is the user defined interrupts ok.

So, they can be used by the programmer, but they are maskable interrupts. So, this is about the interrupt instructions. And, we know that we have one IRET also that which I have already discussed IRET is also another instruction related to the interrupts. So, these are all about the interrupt instructions.

(Refer Slide Time: 22:35)



The next group of the instructions will be status instructions. So, there also I mean implied addressing mode. So, in this case the upper hand is implied like for example, we have clear carry flag CLC stands for clear carry flag. So, whenever this operation is executed carry flag will be reset. Similarly, we can clear direction flag. So, we know that direction flag is cleared then the string operations becomes auto increment, if direction flag is set the string operations becomes auto decrement.

Similarly, we have clear interrupt enable flag; interrupt flag will be reset. Once interrupt flag is reset all the interrupts will be disabled except NMI. NMI as the name implies non maskable interrupt; all the interrupts except NMI will be disabled. Similarly, we have these 3 instructions are for setting the sorry for clearing the flags. And, for setting the flags STC, STD, STI, this is for setting the set carry flag, this is for set direction flag, this is to set interrupt flag. So, there are 6 instructions. And, we can call as this as status instructions or process control instructions also.

And, we have one more instruction related with this one this is complement carry CMC. As the name implies the contents of the carry flag will be complemented. If it is set after this it will be reset, if previously the carry flag is a reset, if I execute the CMC, then the carry flag will be set. It just simply complement the carry. And, we have one more instruction process control instruction no operation NOP no operation. So, as the name implies it does not perform any operation, then what is the use of this instruction is it

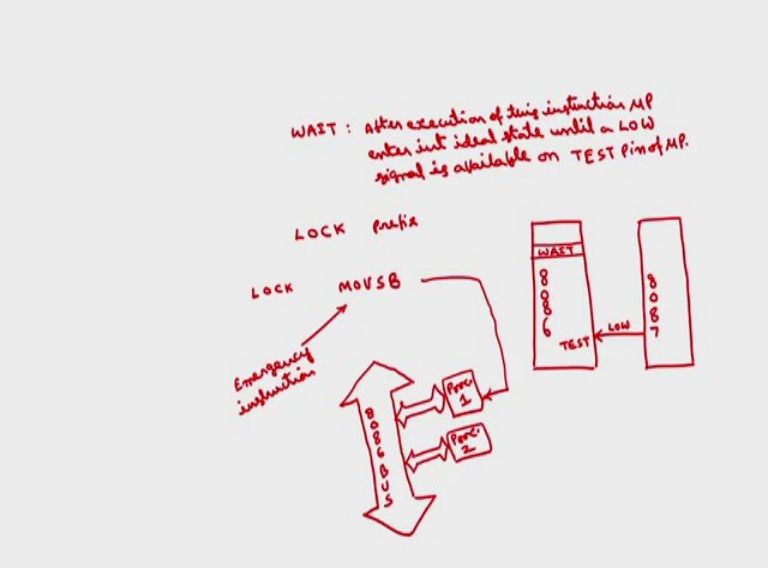
creates the delay. If want to write the delay programs later we will discuss about some delay programs. So, in delay programs NOP instruction will be used.

Then, we have halt instruction halt. So, program will be halted then we have escape instruction ESC. So, whenever the microprocessor execute this escape instruction, then the instruction control will be transferred to the coprocessor. So, it is 8086 microprocessor and 8086 will be connected to some coprocessor, which is called math processor 8087, this is coprocessor also called as math processor, it performs various mathematical operations.

Suppose, the microprocessor want to transfer some instruction to 8086. So, there are some instructions which involves the mathematical operations, then the microprocessor want to control transfer of the instruction to the coprocessor transfer some instruction. So, this transfer of instruction whenever these escape instruction microprocessor execute the escape instruction, then the next instruction to escape will be transferred to the 8087.

So, this 8087 shares the buses with the 8086 address and data buses will be shared by these too because in order to execute this instruction 8087 requires the buses. So, the buses will be shared between 8086 and 8087 which is the coprocessor so, the 8086 can transfer some instructions to the 8087, how does it transfers using this escape instruction. So, whenever it want to mean transfer instruction it simply write escape. So, the next instruction will be transferred to the coprocessor, this is about this escape instruction.

(Refer Slide Time: 28:47)



Then, we have wait instruction. Wait is also another instruction which is processor control instruction, as the name implies whenever the microprocessor execute the wait instruction, it simply goes to ideal state. After execution of wait instruction, the microprocessor enter into ideal state, until a low signal is available on TEST pin.

So, normally this is simply used for synchronizing the slower peripheral with microprocessor, if I take same 8086 and 8087 normally 8087 is slow when compared with 8086 ok. So, here this is having some TEST signal. So, we have discussed about these pins various pins of 8086 in that there is one pin which is called TEST.

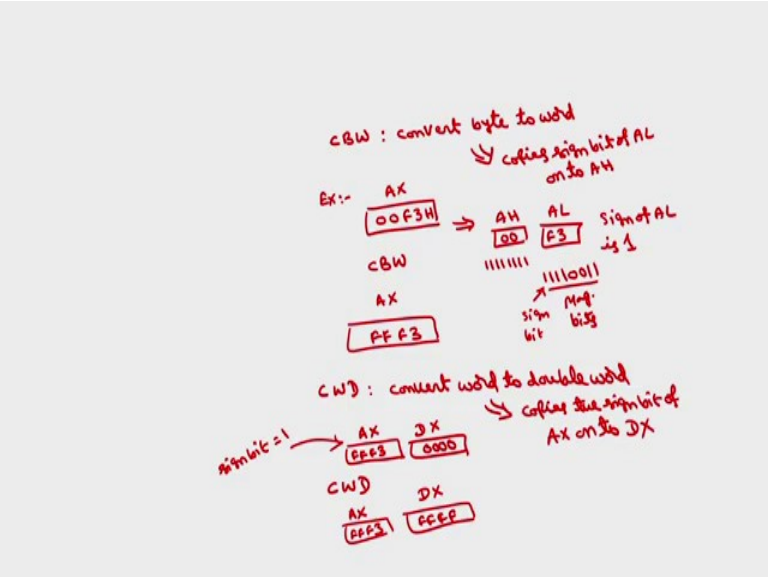
Suppose, if the microprocessor execute the wait instruction here. So, it simply enter into the ideal stage. So, till there is a low signal on TEST signal it will go into ideal state and it will be goes on checking this TEST pin. So, whenever this TEST pin receives a low signal by the external peripheral; that means, so, the I mean the function of 8087 or any other peripheral device is over. So, it will come out of this ideal state and it will execute the remaining instructions this will be WAIT.

Then we have LOCK prefix. This is one more this is not a instruction, but is a prefix. So, LOCK prefix. So, whenever a LOCK prefix is used. So, for example, if I write say a LOCK movestring byte some instruction. If, I add this LOCK prefix, then the microprocessor 8086 can be shared by the different users because this 8086 in maximum mode operation can be operated in multiprocessor configuration.

So, the common bus of the 8086 can be shared by different processors, this will be shared by processor 1, this is processor 1 and so on you can have any number of the processors. So, if the microprocessor 1, processor 1 is executing this instruction, then the meaning of this 1 is while the microprocessor is executing this instruction with LOCK prefix. The second microprocessor will not request for the buses this is buses of 8086.

Normally, this lock prefix will be used for executing some emergency task, some emergency instructions and this is emergency instruction. So, while the processor 1 is executing these emergency instructions. So, it do not want to interrupt by the other processor. So, in that case it simply add a LOCK prefix to that particular instruction. So, that the second processor will not request for the system bus while this emergency instruction is executed this is about the a LOCK prefix. These are the various I mean system control or processor control operations.

(Refer Slide Time: 33:37)



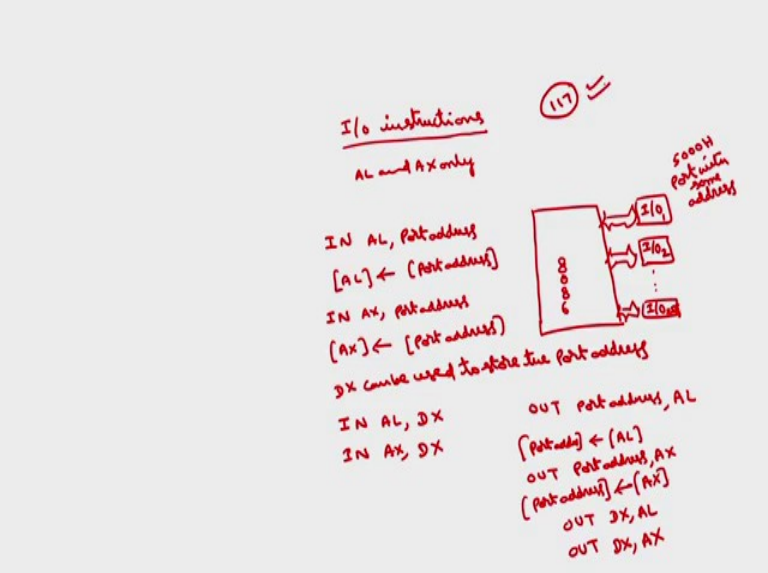
We have few more instructions which is CBW; Convert Byte to Word. So, in order to convert the byte into word so, we will use CBW. This instruction will operate only on the register AX. Suppose so, this instruction basically copies the sign bit of AL onto A H. For example, if I use the contents of AX is 00F3H, if I write CBW.

So, in AL AX is this one means, what are the AH contents AH will be 00 AL will be F3. What is the sign bit of AL 1, because F3 means 11110011. So, in this these 3 are magnitude bits this is sign bit these 7 are magnitude bits in signed magnitude representation or if it is sign 2s complement 2s complement of magnitude bits ok. So, this is sign of AL will be copied onto the AH. So, this AH becomes now FFF3. So, this sign bit will be copied on to the AH means AH was initially 0 0 now, this will copy with 8 1s, so, which is nothing, but F F. So, in order to copy the sign bit of AL onto AH we will use CBW.

Similarly, if you want to convert the sign bit of AX onto AX DX we will use CWD convert word to double word. So, it simply copies the signed bit of AX onto DX by default these 2 instructions operates with AX and DX registers only ok. If I take say for example, AX is the same if I take FFF3, DX is 0000 these 2 are 16 bit registers, if I write CWD. So, what is the sign bit of this?. So, if I take the first F 4 ones will be there. So, the MSB bit is 1 which is sign bit.

So, this 16 ones will be copied onto DX. So, that after this becomes this remains F F F 3 only, then this DX becomes F F F F this is how you can convert the byte to word as well as word to double word. And, the last group of instructions are IO instructions.

(Refer Slide Time: 37:39)



So, if you want to I mean read the data from some ports IO devices memory means location IO is some port we can call as some port ok. So, if we want to I mean transfer the data from microprocessor to IO ports, this is IO device 1, IO device 2. So, like that we can connect to many IO devices depends upon our requirement. So, each IO device will be having some address ok. Each port you can call this IO device as a port, this will be having some address. So, each port will be having addresses. So, in order to transfer the contents of this port onto the 8086 or the data from 8086 to the port, we can use this IO instructions

There are two instructions, which is the one is called IN instruction, another is OUT instruction ok, but in IO instructions we will use only AL and AX only the registers that we are going to use is AL and AX only. If, you want to read the data from the IO port, if the data to be read is 8 bit, then I will take that data into AL by default. If, I want to read 16 bit data from the port that data will be transferred into AX only. Similarly, if I want to transfer the 8 bit data from 8086 to port any port. So, you have to take the data into AL then you have to output that. And, if you want to transfer the 16 bit data from

microprocessor to any port, you take the 16 bit data into AX, then out the data onto the port.

So, there are two instructions one is called IN instruction, IN AL comma you have to give the port address ok. So, if this port address is say some 5000. So, if I give IN AL comma port address that 5000 H, then whatever the contents of this IO will be transferred into the register AL. This is directly implied in the instruction. The AL will be loaded with the contents of the port address.

Suppose, if the port is 16 bit port, then I will take onto the AX. Address remains only 16 bit only, but the contents of that particular port can be either 8 bit or 16 bit. If it is 8 bit, then that will be taken into AL if it is 16 bit this will be taken into AX the contents of this port address. So, there are 2 instructions.

So, there are 2 more instructions where the port address is by default stored in DX register ok. So, DX can be used for storing the port address, this is somewhat indirect addressing mode. So, you can write IN AL comma DX. Similarly, IN AX comma DX the same operation, but here instead of specifying the port address I am specifying DX, then in DX we have port address. So, DX will be used to store the port address. Similarly, if the port is having 16 bit data that will be taken into the AX register. The only difference here is here we are specifying the port address, here we are specifying DX in that DX port address will be present.

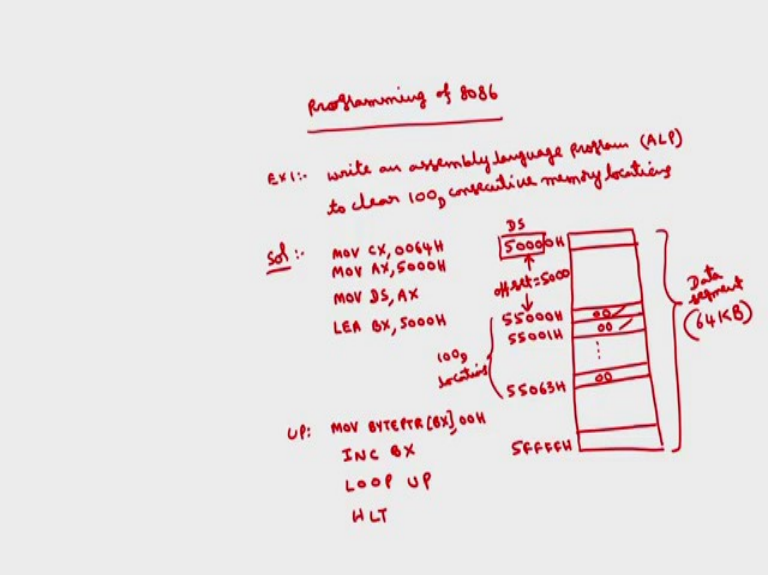
Similarly, we have 4 OUT instructions OUT; this is for reading the data from the port into the microprocessor. If the microprocessor want to send the data to the port we will use OUT instruction. Similarly, we have 4 OUT instructions OUT port address comma AL. The contents of AL will be transferred to the port address. If the port is 8 bit port, then AL contents will be transferred if the output is 16 bit port we have AX.

Similarly, you can take the port address into DX and you can write OUT, DX comma AL OUT, DX comma AX ok. So, there are 4 IN instructions and 4 OUT instructions. So, we will write some programs using this IN and OUT instructions after discussing about a programmable peripheral interface called 8255. 8255 is a device which is used for connecting the input output devices to the microprocessor ok.

So, after discussing about the 8255 architecture we will discuss some programs based on this IN and OUT instructions. So, this is all about the instruction set of 8086. So, there are total 117 instructions, we have discussed about 117 instructions, but the for the sake of clarity. So, I have divided this into different groups.

So, like arithmetic group, logical group, these are IO group. So, we have some processor control group. So, for the sake of I mean better understanding I have divided this into different groups there are total 117 instructions. Now, using this instructions we can write programs ok. So, the second part of this one is programming. So, you take any microprocessor we will be having architecture programming interfacing, I have 3 parts ok.

(Refer Slide Time: 44:55)



So, we will discuss about the programming of 8086. So, I will discuss several programs, let us take the first program as write a assembly language program this is called the language, that will be used for 8086 any microprocessor is called assembly language program, called ALP assembly language program to clear 100 of course, decimal consecutive memory locations.

So, you have to properly first choose the appropriate memory locations. So, before writing this instructions you have to I mean choose the appropriate memory locations. So, by default the data will be stored in data segment. So, this is the data segment register, this is data segment as we have discussed in earlier classes, that the total

memory of the 8086 will be divided into 4 segments. So, this is of 64 kilobytes of the data segment, starting with some address 50,000 H. So, 64 kilo bytes means 5 F F F H.

Suppose you want to clear any 100 decimal, 100 decimal this hexadecimal it will be 64 H any 100 locations start with some offset say you want to store some from here say 55,000 onwards. So, from here to here total 100 decimal locations. So, how to write the program to load 00 into all these locations previously whatever the data after writing this program this 100 locations has to be cleared ok. So, for that so, what will be the data segment, this is data segment; data segment register contents will be it always holds the upper 16 bit of the starting of this one. So, this is data segment register contents ok.

So, data segment cannot be loaded with any address, but you can take this address into any other register ok. So, normally I will take this 5000 H into AX register, then I will move the contents of AX to DS. So, that DS will be loaded with 5000. So, we cannot take this data or address directly into the segment register ok. Now, data segment will points to this. And, what is the offset from there we have to I mean load this the offset from here to here will be 5000.

This is also called effective address. I can take this effective address onto any other registers base register or index registers. So, I will write for that LEA Load Effective Address that is the instruction I am taking the BX register to hold this. So, what is that address 5000 H? From this location onwards total 100 locations I want to clear ok. Move byte PTR I will explain what is byte PTR BX comma 0 0 H. So, what happens is this, 0 0 will be transferred to the memory location, which memory location whose effective address is present in BX, means this 0 0 will be placed onto this.

Why you are writing byte PTR is? If this is word if you want to store 2 consecutive locations this is also possible. So, in that case that is you have to write their word PTR ok. Then in that case you have take 0 0 0 0 H okay. This is also another way to write this program. So, byte means you have to take 0 0 0 0 H. So, total how many locations you have to transfer that you take into CX register, 0 0 6 4 H 100 decimal is equivalent to 6 4 hexadecimal.

So, if I want to write word PTR then you have to take half of this 100 means 50 decimal 50 you convert into hexadecimal that you take into CX register and here you have to give 0 0 0 0 you can write in anyway I am using this byte PTR. So, I am I want to transfer

only 1 byte at a time. So, this 0 0 H will be transferred to the 55, 000 H this one is over now.

Now, you have to repeat this 100 times. So, for that what you have to do is. So, we can write INC BX. So that this BX will points to the next location, which is 5001 ok. So, I have cleared now 5,000, then in the next step you have to clear 5001. If I write here half of this location and if I write here word PTR, then you have to increment twice this INC BX INC BX right. INC BX then this will point to 5001 loop up. So, I have explained this loop instruction, this loop instruction will be executed until CX is equal to 0. CX you have 64 I have transferred I have cleared one location, after that this CX will be inside this loop automatically decremented by 1. So, whenever this CX is not equal to 0 it will jump to the UP this UP I have to write here.

Now, what happens here? Because this BX has been incremented now this BX becomes 5001. So, 0 0 will be transferred to the byte pointed by 5000 1s means 55001 H this will be loaded with 0. After, this is loaded with 0 again when loop instruction executes inside this automatically this CX will be decremented by 1.

So, CX initially 0 0 6 4, after the first byte 0 0 6 3, after the second byte is cleared 0 0 6 2 and so on it will decrement. So, once all of these 100 locations are cleared, then this will come out of the loop, then you have to halt. It is a simple instruction, simple I mean a program to clear 100 consecutive locations ok, you have to choose the appropriate locations? This is one way if you want to I mean write word PTR this word PTR byte PTR they are called they are called assembly directives. So, if you want write this using assembler you have to specify this byte or word ok. So, remaining few more programs we will discuss in the coming classes.

Thank you.