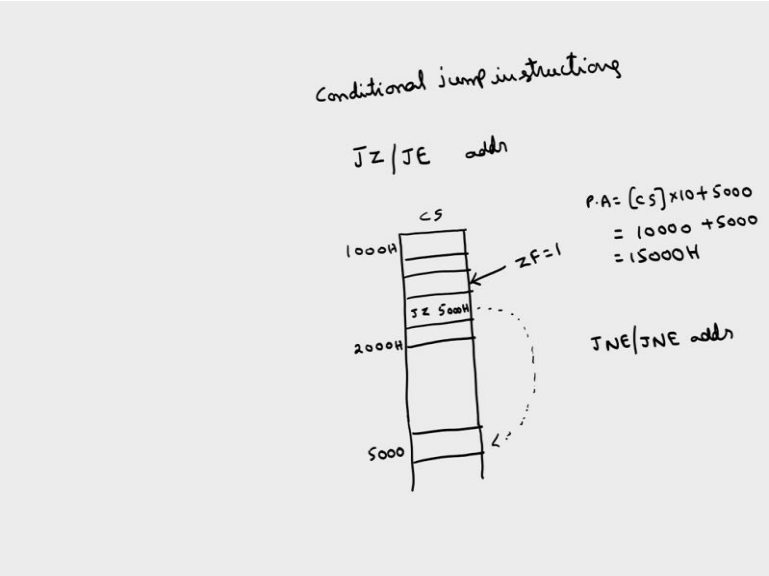
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture - 10
8086 Branch and String Instructions

In the last class we were discussing about the jump instructions. So, we have discussed about unconditional jump instruction, where without checking the condition the microprocessor control will transfer to the address specified in the instruction and the second type of jump instructions are conditional jump instructions.

(Refer Slide Time: 00:49)



As the name implies if some condition satisfies then only it will jump otherwise it will just simply skip that instruction in and will go to the next instruction ok. So, there are 17 conditional jump instructions. So, corresponding to the some flags we have the jump instructions like we have 0 flag jump on 0 this also can be called as jump on equal also both are same, address you have to specify here. So, as the name implies if 0 flag is set then only it will go to that particular address which is specified otherwise it simply skips this instruction and it will go to the next instruction.

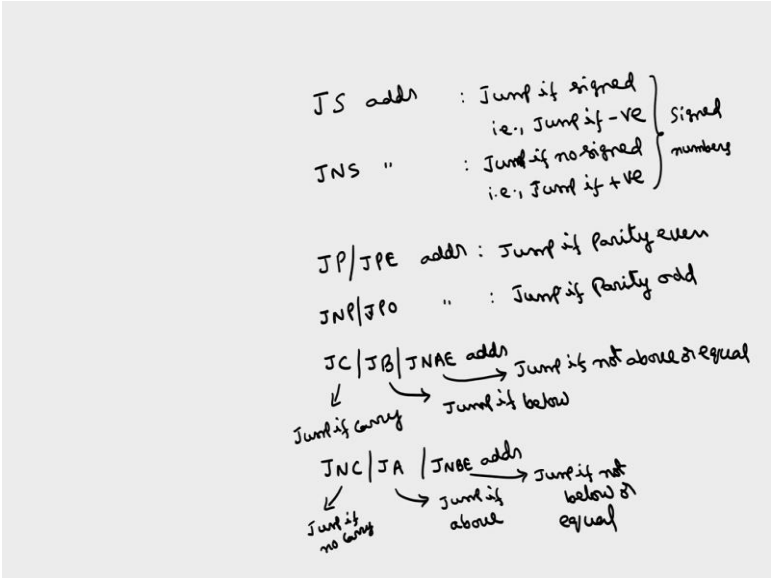
For example if I check here, this is the flow of instructions if I write here jump 0 if I give some say 5000 H. So, if this is the next instruction is in some say 2000 H and code segment starts from say 1000 H. So, if the instruction above here if 0 flag is set after this

jump instruction the microprocessor control will goes to the physical address which can be calculated as. So, this code segment physical address will be code segment contents into 10 plus the address that is specified 5000 this is equal to 10000 plus 5000, which is equal to 15000 H. So, the microprocessor control always from here it goes to the location which is located at 5000 H.

If this 0 flag is set at the instruction just before this jump 0 instruction, if 0 flag is reset; if 0 flag is reset it simply skips this instruction and the microprocessor control will goes to this instruction whose address is 2000 H ok. So, this is about jump 0 or jump equal both are same you can write either jump 0 or jump equal. So, if 0 flag is set the control will goes to the physical address which can be calculated by using this effective address otherwise it will skip the instruction and it will go to the next instruction.

Similarly, we have the opposite to this one is jump not 0 slash jump not equal address this is reverse if 0 flag is reset, then the microprocessor will control will goes to the address which is specified in the instruction otherwise it will skip the instruction the control will go to the next instruction, these are the 2 jump instruction conditional jump instructions related to 0 flag.

(Refer Slide Time: 04:55)



Similarly, we have corresponding to sign flag jump if signed address this will be jump if signed that is jump if negative, if the result of any operation which is just above this instruction which has the negative number, then the microprocessor will control will

goes to the address that is specified. So, by calculating the physical address as I have discussed just now otherwise it will skip this instruction and it will go to the next instruction. Similarly we have jump if no sign means jump if positive, if the result is positive then it will jump otherwise it will skip the instruction and it will go to the next instruction ok.

So, these are all deal with signed numbers, we know that the microprocessor uses signed two's complement representation. So, this is corresponding to the signed flag. So, what are the other flags we have? Parity flag. So, further the instructions are jump if parity or jump if parity even both are same address 16, jump if parity even this first JP stands for jump if parity, parity means by implied it is a even parity or we can directly specify jump parity even. So, if the result of the instruction which is just above this instruction it has even parity then the microprocessor will control will go to the address that is specified otherwise it will skip and go to the next instruction.

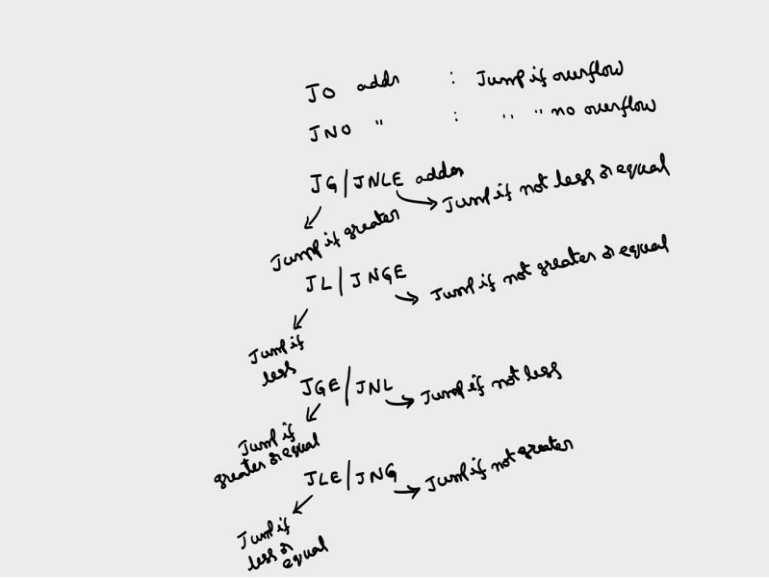
Similarly, we have jump if no parity or jump parity odd this is jump if parity odd. So, corresponding to 0 flag we have 2 instructions corresponding to signed flag we have 2 instructions correspond to parity flag, we have 2 instructions corresponding to auxiliary carry flag there are no instructions then corresponding to carry flag we have 2 instructions jump if carry this is also equivalent to jump if below or also equal to jump if it is not above or equal address.

So, these 3 mnemonics are same we can write either JC or JB or JNAE all the 3 mnemonics are same, this JC stands for jump if carry, if carry flag is set so, if the result of the instruction just above this one results a carry flag. So, the carry flag will be set we have discussed with the condition for the carry flag to be set ok. So, if the carry flag is set, then this microprocessor control will goes to the address that is specified otherwise it will skip the instruction and it will go to the next instruction ok. The same operation this JB stands for jump if below, which is also equivalent to jump if not above or equal jump below means below is equivalent to it is not above or equal these 3 are same.

And what will be the opposite to this one jump if no carry means jump if not below which is also equal to jump if not below jump above or equal address, if this carry flag is reset the microprocessor control will goes to the address that is specified otherwise it will skip the instruction and it will go to the next instruction.

Here these 3 are jump if no carry jump this not below you can call as above sorry jump if above means not below or equal, this is jump if above, this is jump if not below or equal these are the 2 instructions corresponding to carry flag. So, we have discussed total 8 instructions corresponding to 4 flags. So, we have discussed about the 0 flag, sign flag, parity flag, carry flag. And we have 2 more instructions corresponding to overflow flag jump if overflow.

(Refer Slide Time: 10:59)



If overflow flag is set we know the condition for the overflow flag to be set. So, if the overflow flag is set after the result of the instruction which is just above this one, then the microprocessor control will goes to this address otherwise it will skip this instruction it will go to the next instruction.

If jump no overflow this JO stands for jump if overflow that is if they are signed numbers we have a range of the numbers if the result is outside the range then overflow flag will be set this we have discussed in the while discussing about this flags, jump if no overflow. So, we have total 10 instructions correspond to 5 flags for the remaining flags there are no I mean jump instructions.

Now, there are some more jump instructions which will be corresponding to jump if greater or greater means jump if not lower or equal. So, this result of the instruction which is above this one results normally this will be used after this compare instruction. If the source is greater than the destination then this will jump otherwise it will skip the

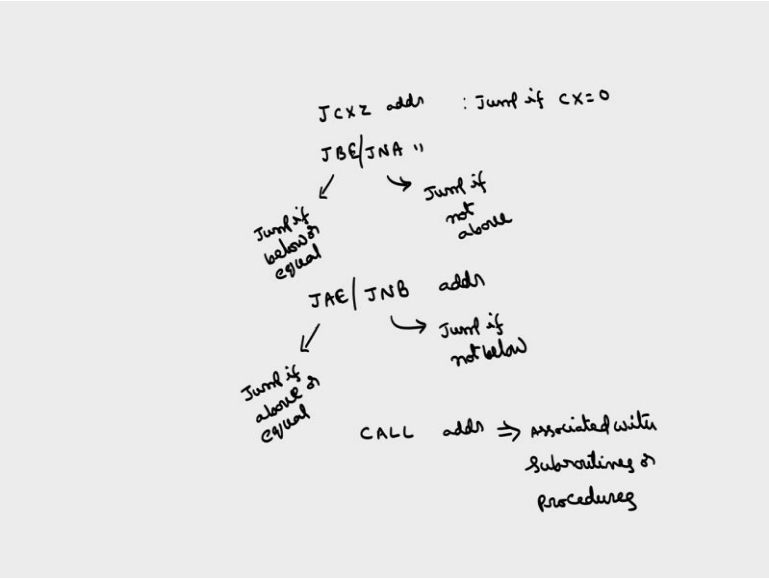
instruction it will go to the next instruction ok, jump if greater this is jump if not jump if not less or equal, basically to compare the 2 instructions, so after the compare instruction this instruction will be used.

So, what is the opposite to this one; opposite to this one is jump less than if less or this is equivalent to jump not greater or equal. So, this will stands for jump if less jump if not greater or equal. Then we have this is only greater less than or equal will be here this is less than and this is greater than equal is here or we can make this greater than or equal here and we can make not less here.

So, (Refer Time: 14:02) the other instruction is jump greater than or equal so, this E I have taken in this side. So, what will be the equivalent of this one jump if not less, because this E I have taken here ok. So, this stands for jump if greater or equal this will be jump if not less.

Similarly, if I take this E to this side we have jump less than or equal or jump if not greater; jump if less than or equal this is jump if not greater. So, totally we have 10 11 12 13 14 and we have 2 more instruction jump instructions which are corresponding to jump if CX is equal to 0 address.

(Refer Slide Time: 15:25)



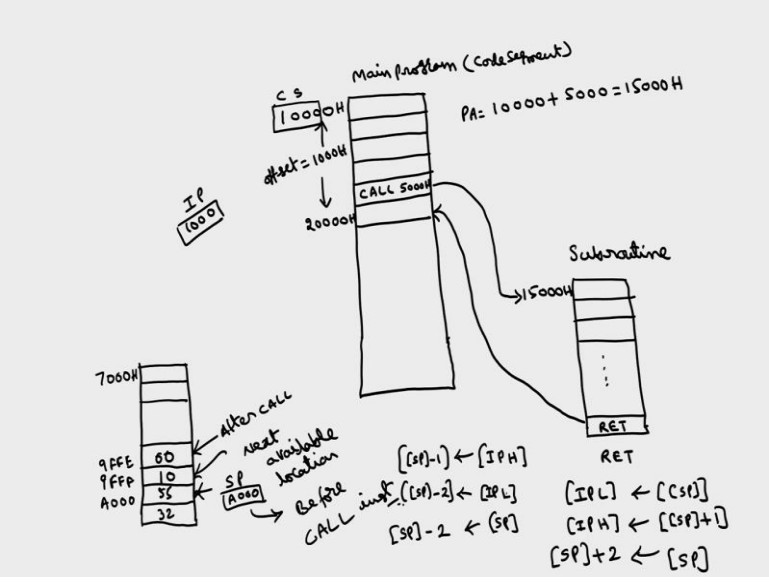
Jump if CX the register CX is equal to 0 otherwise it will come to the next instruction, this instruction will jumps to the instruction that is specified at the address here if CX is

equal to 0 otherwise it will jump to the it will skip this instruction it will go to the next instruction.

Similarly, we have jump below or equal so, we have here. So, here we have below and this is not above equal this is above not below equal you take this E to here you take this E to here we have 2 more instruction which is jump below or equal not above jump below or equal plus jump not above jump if below or equal jump if not above. Similarly we have jump if above or equal jump if not below; jump if not below. So, there are totally 17 conditional jump instructions ok.

Now, the next term branch group will be the call instruction; call you have to specify the address. This address can be specified by using either register or memory location again the memory location can be any of the memory locations which can be pointed by this different addressing mode that we have discussed in the earlier classes. So, here in the call instruction, this call instructions are basically used with subroutines this we have already discussed in the earlier classes. So, the call instructions are associated with subroutines or procedures. I just quickly review this.

(Refer Slide Time: 18:11)



So, basically if any operation repeatedly occurs then you will write a subroutine, this is the main program and this is subroutine this is main program and this is subroutine. So, the operation which repeatedly occurs we will write a subroutine separately this is main program, if I want to call this instruction say call from 5000 H is there.

So, if the starting of this segment is main program will be in code segment, if code segment register contains starting 14 bits of this address this address is seem something like 1000 H this will contain the code segment and if this is this address is some 2000 H. So, what will be the address or physical address of this one? Physical address of the call instruction will be 10000 plus 5; physical address will be 15000.

So, you have to write the subroutine from 15000 onwards, whenever the microprocessor execute this call instruction. So, this will calculate the physical address to where from where this subroutine is to be called that is 15000 H. So, the microprocessor control will goes to this instruction, it will execute this. So, after the execution of the subroutine it has to return back to the main program this is the difference between the interrupt and this main program this subroutine. So, it has to return back the last instruction of the subroutine is return.

So, how does the microprocessor knows that it has to return to the instruction which is next to the call instruction, which is this 20000 H say this is 20000 H if I write physical address 20000 H. For that what this call instruction will do is, inherent to this call instruction there is a push instruction the data that is that is to be stored here will be stored into the offset from here.

So, what is the offset from here? If I write that this is 1000 and this is 2000, if I take this only physical address, the offset will be 1000. So, the microprocessor will transfer this contents onto the stack pointer ok. So, inherent to this call instruction what will be there is, the contents of this is the stack pointer somewhere here, suppose up to this 32 55 up to this data is full then the stack pointer holds the address of the stack top, say this is some A000 H if the starting is say some 7000 H.

Stack pointer holds the stack top till which the data is full, this is next available location. So, this offset which is 1000 will be stored here. So, in that 1000, 00 will be stored here, 10 will be stored here, now the stack pointer will points to this address. If this is A000 this will be 9FFF and this will be 9FFE. So, after this call instruction the stack pointer contents will becomes this is before the call instruction, this is after call instruction.

So, what will be the inherent operation inside this call instruction is the contents of the instruction pointer this offset will be hold by instruction pointer; this 1000 will be hold by instruction pointer. So, the contents of instruction pointer will be pushed on to the

stack and stack pointer will be decremented by 2. So, inside this call instruction the operation will be instruction pointer I this is 10 this is 00 last in first out. So, instruction pointer H will be transferred to this will be SP, if this is SP, SP minus 1 contents of SP minus 1 and contents of instruction pointer low will be transferred to contents of; contents of SP minus 2 and SP becomes SP minus 2.

So, these are the operations which are inherent to this call instruction whenever a call instruction is executed before this microprocessor control transfer to the subroutine it will save the contents of this instruction pointer. So, instruction pointer holds the offset of the instruction which is to be executed next. So, those contents will be transferred to the stack pointer using these 2 operations, then automatically stack pointer will become stack pointer minus 2, then the microprocessor control will goes here ok. So, corresponding to this then the next instruction is return instruction.

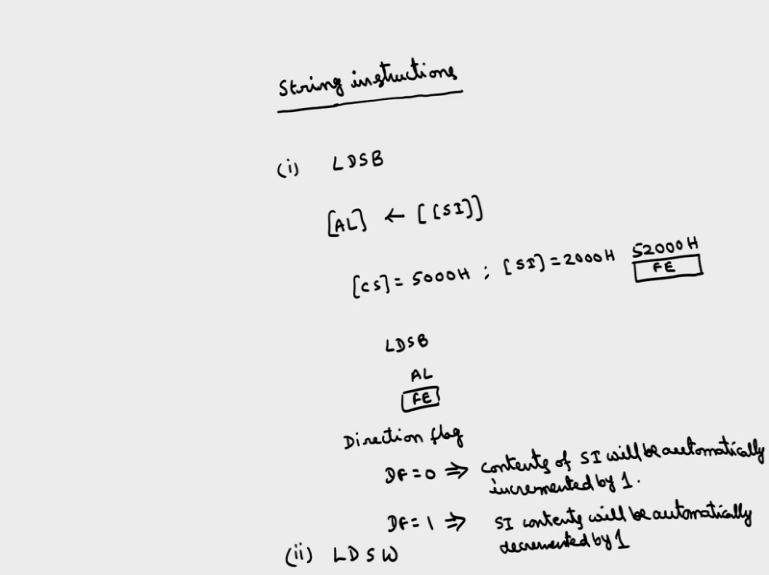
So, in the return instruction RET. So, in order to go this transfer again back to this 2000 instruction from here it has to go this instruction, the operations inside this return instruction which are inherent in the return instruction are opposite to this. So, what is that this is 00 will be transferred to IPL, 10 will be transferred to IPH and automatically SP becomes SP minus SP plus 2. So, this will be IP 0 0 was IPL.

So, the contents of contents of SP contents of SP is 9FFE contents of contents of FE means 00 this 00 has to be transferred to the IPL. And then 10 contents of contents of SP with respect to this one SP plus 1 will be transferred to IPH because we know that this stack pointer operates on the principle of last in and first out then. So, this 2 data will be emptied now this stack top will becomes this A000 which is the previous one. So, the stack pointer becomes now stack pointer plus 2. So, this is about the call instruction and return instruction, this also comes under the branch instructions.

So, we have discussed about the various branch instructions like unconditional jump conditional jump instructions, we have 17 conditional jump instructions and then we have the call instruction to call the subroutine. So, even though this seems to be only a simple instruction inside this we have these operations. So, which I mean saves the contents of the instruction pointer onto the stack then in return the contents of the previous instruction pointer contents will be taken back on to the this instruction pointer

then stack pointer will be automatically incremented or decremented by 2 depends upon the call or return instructions these are about different branch instructions.

(Refer Slide Time: 28:05)



Then the next instructions will be string instructions, as the name implies this deals with string of data an array of data. So, we have different string instructions. So, the first string instruction is load string byte; load string byte. So, this is basically so, we do not mention anything here. So, automatically if it is string byte the contents of contents of SI this is by default will be transferred to the register AL that is example if I take this code segment contents will be 5000 and if SI contents will be 2000 and if the contents of 52000 H is FE then after this instruction the contents of AL becomes if I write instruction load string byte LD string byte the contents of AL becomes FE.

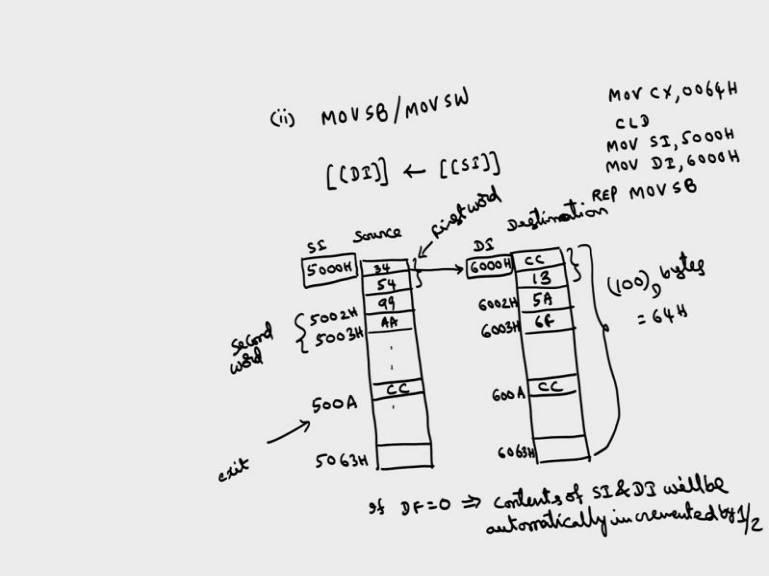
So, this instruction basically take this source index as a default register ok, using this SI it will compare the physical address then that data will be transferred to the AL. Now, if you want repeatedly load the data ok. So, we have direction flag, if direction flag is 0 contents of SI will be automatically incremented after each instruction incremented by 1. On the other hand if direction flag is set contents of SI will be automatically decremented by 1, this is about load string byte.

Here we can have load string word also the second instruction is here load string word here the operation will be here instead of AL this will be AX a word will be taken from 2 consecutive locations and that will be stored into AX and here if direction flag is 0 the

contents of SI will be automatically incremented by 2. If direction flag is set the contents of SI will be automatically decremented by 2 that is the only difference between string byte and string word.

There are 2 differences one is here this will be AX and a word will be taken 2 bytes will be taken from SI and SI plus 1 and if direction flag is reset the contents of SI will be automatically incremented by 2 if direction flag is set the contents of SI will be automatically decremented by 2.

(Refer Slide Time: 32:19)



So, the next string instruction is move string byte. So, this basically we have the contents of contents of SI will be transferred to contents of contents of DI. So, if you want to transfer a string of data from some source to destination we can use move string byte. If I take some source and destination this is source, this is destination. Suppose if you want to transfer from 5000 H to 6000 H total say I want to transfer 100 bytes; 100 bytes means 5063, 6063 this will be total 100 decimal bytes because 100 decimal is equal to 64 hexadecimal and starting with 063 its total 100 bytes.

So, this you have to point to source index this you have to point to destination index, by default in all the string instructions source will be pointed to SI, destination address will be pointed to DI. Then if you want to automatically increment this contents you have to if direction flag is reset as I have discussed in earlier slide also, then contents of SI and DI both will be automatically incremented by 1.

And if you want to transfer this total 64 bytes hexadecimal bytes are 100 decimal bytes then you have to write a set of equations ok. So, by default the count the number of bytes to be transferred, we have taken the register CX this is implied move CX comma 0064 H and to clear the direction flag which we are going discuss later. The instruction is CLD this is a simple implied also CL stand for clear, D stands for direction flag, then you have to load move SI comma 5000 H move DI comma 6000H.

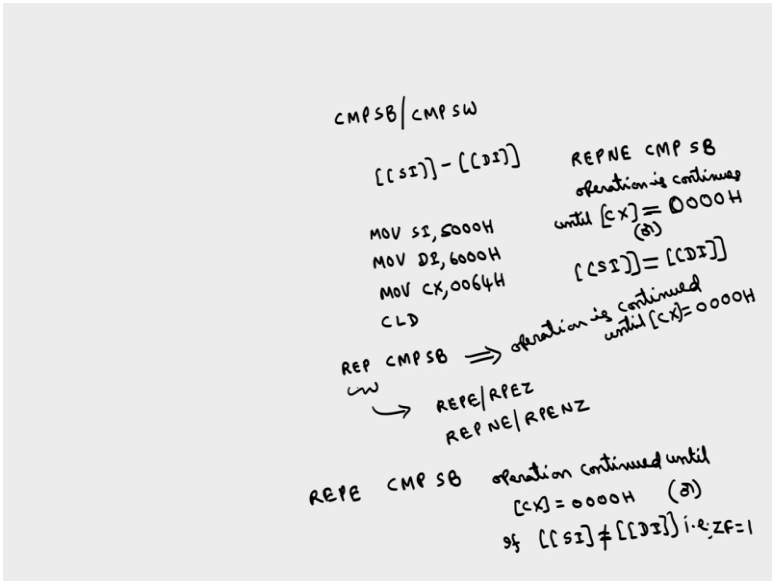
Then move string byte, if I write move string byte this will execute only once ok. So, the contents of 5000 will be transferred to the contents of 6000, if I add a prefix like repeat REP repeat is a prefix, 1 byte prefix which if I add here. So, this operation will continues until CX is equal to 0 by default the register is CX. So, this move string byte will be executed these many number of times that is 100 times. And because this direction flag is reset the contents of SI and DI will be automatically incremented ok. So, then after 100 bytes has been transferred this program will come out of the loop this is how you can transfer 100 bytes from source to destination using move string byte instruction.

So, the requirements are we have to take the number of bytes to be transferred into the CX register, then you have to clear the direction flag if you want to transfer the bytes from bottom to top, this is lower address, this is higher address. Similarly, you have to source you have to load with SI with the starting address of the source, DI with starting address of the destination, then you have to write move string byte you have to write repeat. So, this is repeat is the prefix you have to use to perform the repeated operation, this is how the string of data can be moved from one location to another location.

Similarly, we have move string word also. So, if I write move string word, then if this is 64; 64 words will be transferred means total the locations will be 200 decimal locations. So, a word is nothing, but 2 bytes ok. So, now, to transfer a word so, everything is same the only difference here will be. So, here we will be having 60 more 4 locations here one word will consisting of these 2, these 2 will constitute one word, these 2 will constitute one word. So, like that we have 64 words means we will be having 100 more locations here, 100 more locations here I am not showing here. So, this will be transfers the same thing if clear direction flag is 0, the difference is the contents of SI and DI will be incremented by 2 here.

In case of string word this will be by 2 in case of string byte this will be by 1. So, that the first word will be transferred the second word will be at 5002 H and 6002 H this is first word and the second word will be 5002 and 5003 H. So, automatically the contents of SI will be incremented by 2, DI will be incremented by 2, if DI direction flag is 0 if direction flag is set then this will be decremented by 2. So, the direction will be from top to bottom because now it is bottom to top this is the only difference. So, we have move string byte as well as move string word ok.

(Refer Slide Time: 39:49)



Similarly, we have some other string operations, compare string byte and compare string word as the name implies this will compare for comparison we require 2 locations so, which is if I have SI and DI. So, the operation here will be the contents of contents of SI minus contents of contents of DI operation will be takes place, so in comparison we have subtraction operation only, but the result is discarded based on this result of operation the flags can be either set or reset ok.

So here also the same thing if direction flag is cleared the contents of SI and DI will be automatically incremented by 1, in case of string byte; in case of string word the contents of SI and DI will be automatically incremented by 2. So, this will compare this data. So, here whatever the data is present from here suppose here if you have some 34, 54, 99 AA, this will be having CC, 1D, 13, 5A, 6F. So, this instruction first compare 34 and CC so, 34 minus CC will be performed because 34 is less than CC carry flag will be set.

Now, again in order to perform this repeatedly. So, you have do the same thing suppose if I write same MOV SI comma 5000 H MOV DI comma 6000 H I want to compare 100 decimal data. So, I take CX with the hexadecimal equivalent of 100 which is 0064 H then I want to I mean compare from 5000 to 5063 byte, then you have to clear the direction flag, then you have to write compare string byte. So, this will compare only 1 data, if you want to compare the 100 data you have to use prefix repeat ok, this will compare the 100 data for each data comparison it will set or reset the flags ok.

So, here in case of compare and there is a one more instruction called scan instruction, here this prefix can be either repeat in case of move instruction this is only repeat only ok. Whereas, in case of compare instruction this repeat can be we can use this repeat or repeat equal or repeat Z 0 repeat not equal or repeat not 0. These 2 are same, these 2 are same ok, we can use either repeat or repeat equal or not 0. So, this actually in case of repeat this operation will be continued until continued until CX is equal to 0 contents of CX is equal to 0000 H.

Instead of repeat if I write repeat equal say suppose if I write compare B compare string byte repeat equal then under true conditions this operation will be repeated ok. So, one condition is until CX is equal to 0 is implied, so operation is continued until CX is equal to 0 which is same as this repeat prefix or there is one more condition. So, the comparison result is equal that is 0 flag is set.

If I take the same example suppose somewhere here if you have the data something like say this is 5000 A, this is some 6000 A, if this is having some CC, this is also having CC then in this case this operation will be stopped when these 2 are equal say it is 5000 A itself the operation will be exit from the this string comparison ok. So, if these data's are not equal; if these data's are not equal it will compare all the 64 locations which is I mean 64 hexadecimal or 100 locations.

If any data is same at that point also it will exit there are 2 conditions for exiting this one or continued means is continued if CX is equal to 000 or SI is equal to or it will quit, this compare operation will be quit if this is CX E not equal to 0000 SI is equal to DI ok. So, this operation will be continued until this is sorry not equal to if this is equal then it will quit ok. So, this additional thing which is so, these 2 data should not be equal. So, this

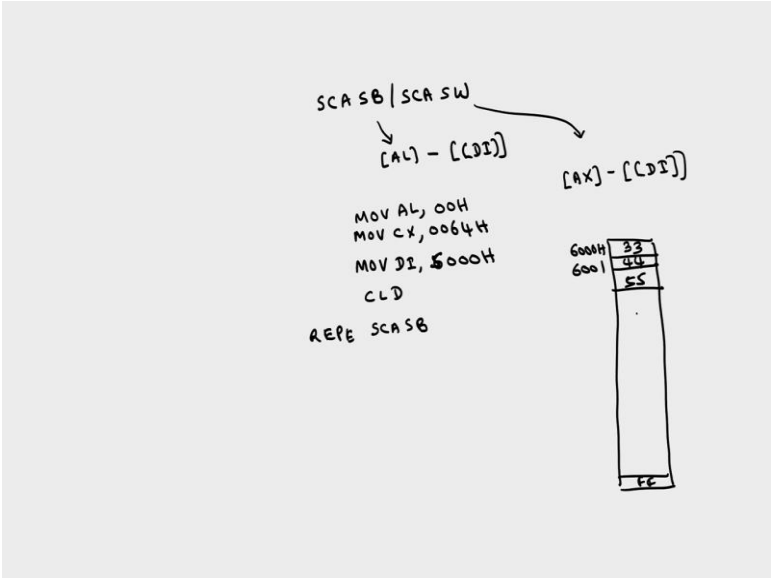
100 bytes this data should not be equal at any point if data is equal even it has not completed 64 comparisons it will quit that is additional thing in case of repeat equal.

Similarly, we can have if I have repeat not equal if I write something like repeat not equal compare string byte then this is reverse operation the operation will be continued until CX is equal to 000 and if these 2 are not equal. So, this operation will be continued if CX is equal to 0000 until CX is equal to 0000 or the contents of contents of SI is not equal to is equal to DI if this is not equal here if this is equal here ok; that means. So, this data is 34 this also should be 34 otherwise it will quit at the first I mean byte itself, if this is 54, this also should be 54 then only it will continue until otherwise it will quit that comparison, similarly if this is 99 this also should be 99 ok.

So, this will perform all the 100 comparisons if this 2 array of data's are equal otherwise. So, if at any point if these 2 are having different data's the microprocessor control will quit this string operations, ok. So, these 2 are opposite to one another. So, this will be continued if CS is not equal to 0 and SI is not equal to DI this will continue if CS is not equal to 0 SI is not equal to DI ok. So, this is about repeat there are 3 prefixes repeat, repeat equal, repeat not equal this is for the byte.

Similarly, we have for the word also the only difference is a word will be compared. So, we will take a word which is 5000 and 5001 H, 6000 and 6001 H word will be compared 16 bit data and if the direction flag is reset, then SI and A will be automatically incremented by 2, if DF is 1 then it will be incremented by 2 that is the difference between string byte and string word the everything operation is same.

(Refer Slide Time: 49:33)



So, the next I mean string instruction is scan string byte or scan string word. So, here scan is nothing, but comparison only in comparison SI and DI contents will be compared whereas, here if this is byte contents of AL will be compared with DI. This is byte and this is byte if this is word then contents of AX will be compared with DI.

The destination is by default DI, like in case of load it will load onto the AL by default source is SI and if it is byte if it is a word the destination source is by default SI and destination will be AX it similarly scan also. So, the contents of DI will take a string of data that will be compared with the AL contents, at any point if they are equal then we can quit the operation otherwise you can continue. So, like that this AL contents will be compared with DI, AX contents will be compared with DI ok.

For example if I take the same thing if I want to compare 100 decimal data with AL same instructions we will write MOV CX comma 0064 H MOV DI 6000 H, if the contents of 6000 to 6023 are this is 6000 H 33 some 44 55 some data FF say. If the contents of AL is 00 some data whatever the data you can take this has to be compared with this data string, then clear the direction flag.

So, that the contents of DI will be automatically incremented by 1, then you have to write scan string byte and you have to write repeat, repeat prefix can be used here also then what happens is first 33 will be compared with 00 ok. So, 00 minus 33 carry flag

will be set these 2 are not equal then it will be compared with 44 it will compare with 55 like that it will goes on compares ok.

So if I write repeat equal then so, if none of this data are 00 it will perform 64 comparisons that is hundred comparison decimal otherwise if at any point a 00 is found because this 0000 both are same, then it will quit this comparison operation that is the difference between repeat equal. If I write repeat not equal then it will compare all the 100 comparisons will be performed if no data here will be 00, if any data is 00, then again it will quit this operation ok.

So, this is repeat on equal if any 00 is there it will stop at that particular point if I write repeat not equal here first step itself it will quit, because this is not equal 00 and 33 are not equal it will quit ok. So, you have to load all this with 00s then only if all this 100 locations are having 00 then this our instruction will be execute a 100 times at any point if data is other than 00 it will quit the operation that is about the scan string byte ok. So, these are about these string instructions which always deal with source index and destination index and directional flag.

Thank you.