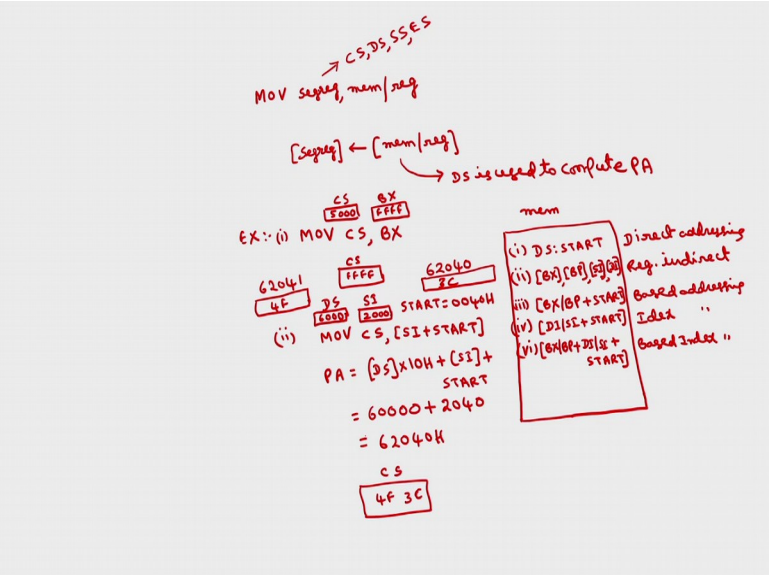
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture – 06
Data Transfer Instructions

So, in the last class we are discussing about this instruction set. In that the first group which is data transfer group; so in that again the first instruction, MOV instruction. So, we have discussed about the MOV moving of the data from register to register, register to memory, memory to register or we can transfer some immediate data to the memory or register. Then the next one is we can load something into the segment registers also.

(Refer Slide Time: 00:59)



So, you can transfer something into the segment register from memory or register. So, the operation is the contents of either memory or register is transferred to the segment register. So, this segment register can be we have 4 segment registers; code segment, data segment, stack segment, extra segment. If this is memory from memory if you want to transfer the data to the segment register.

So, this memory while calculating the memory address we have to use data segment by default to compute the physical address. If I take the examples MOV some CS comma, if I take say for example, register BX simply. So, whatever the contents of BX will be transferred; before this and the contents of CS is 5000 if BX is FFFF say.

So, after this CS will loaded with FFFF; this is example of data transfer from register to segment register, then from memory to. So, again this memory means yesterday also we have discussed there are different addressing modes. Memory can be this memory wherever the memory comes we just remember that so, wherever the memory comes that memory comes from 1 of these combinations. So, memory options are one is you can specify DS comma START in case of direct addressing mode this I have given the example in the last class.

Second way to represent this memory can be it can be contents of BX or BP, SI or DI. Third way is, this is called register indirect addressing mode, this is direct addressing mode, thus we have already discussed in the last class this is register indirect and then we have based addressing mode. So, in this will be so, BX or BP plus some START value this is called register based addressing mode.

In indexed we have this can be either SI or DI plus some offset; which is defined by the assembler; assembler direct is through assembler direct the user can define the value of START this is called index addressing mode. Next one is based index, based register as well as index register. So, this can be BX slash BP plus DI slash a SI plus START.

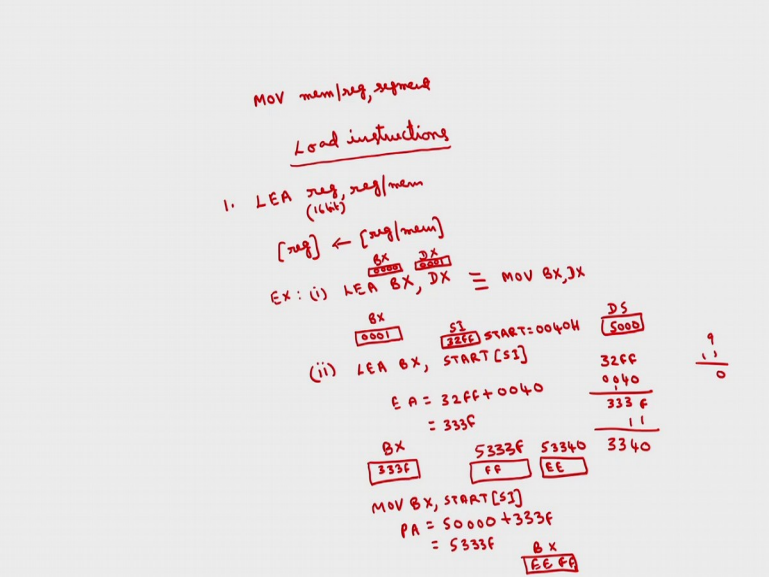
So, there are different ways to specify the memory; wherever the memory comes throughout this instruction set that memory refers to one of these combinations. Say if I take directly to the examples, MOV CS comma, the contents of sum SI plus START. So, the physical address while calculating the physical address as I have told DS register will be used. So, if the contents of DS is 6000 say; then what else is required? SI is required SI is say 2000, let START is defined as 0040 yesterday also I have taken the same value I think.

Then physical address will be as we know that. So, this DS into 10 H plus SI contents plus START. So, this will be 60000 plus 2040 this is 62040 H. So, if the contents of 60000 H here will be before the execution of this one 62040 will be 3 say C. Because this destination is a 16 bit this will read the 2 bytes, so this will be loaded into lower order then higher order will be the next address.

And if I assume that the next address 62000; so, the contents of say for example, 62041 is 4F say. Then the content of CS will be the lower order will be this 3C, higher order will be 4F; this is how you can load any segment register with the contents of any

register or contents of any of this memory locations. Similarly you can have the reverse operation. So, here we have discussed about this loading this contents into segment register from memory or register, from segment register also we can load the contents into memory or register.

(Refer Slide Time: 08:11)



MOV memory slash register from segment register. So, no need to explain this is obvious so, this will be reverse operation. So, these are the different MOV instructions that are present in 8086 so under this data transfer group. And the next instructions, that I am going to take is load instructions.

So, the first load instruction is LEA register comma, memory slash register. Load Effective Address, E A stands for Effective Address directly into any register. So, this register can be BX, BP, SI, DI ok. So, the contents of register slash memory is transferred to this register. So, this is 16 bit register; because effective address is 16 bit. So, the register source remain destination register should be 16 bit.

So, again here this memory means as I have discussed with the previous instruction so, one of this 5 or 6 combinations one of these 6 combinations sorry 5 combinations so, register can be any register. For example, if I write LEA some BX comma, some DX say. So, simply BX will be loaded with the contents of DX. Say BX is 0000 this is 0001 say. So, simply BX will be loaded with 0001; this is actually equivalent to this

instruction is equivalent to MOV BX comma DX. So, the difference here is if you have the memory location to register then there is some difference.

If we take the second example, load effective address sum BX comma if I give START SI. So, in this case if I assume that the START value is same 0040; which I have assumed in the last example. SI contents are say some 32FF say START is 0040 H effective address will be simply 32FF plus 0040; I am not calculating the physical address remember this is actually load effective address. So, I am calculating the effective address. So, what will be this so, 32FF plus 0040.

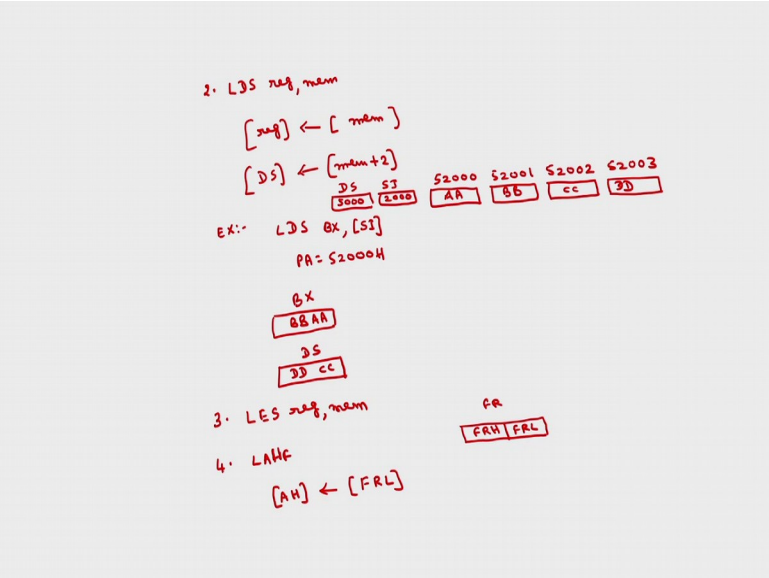
F plus 4 is F plus 1 is 10, F plus 2 is 11, F plus 3 is 12, F plus 4 is 13 so, it will be 13. Like in case of decimal 9 plus 1 will be 10, 9 plus 2 will be 11; like that here F is fifteen the maximum number F plus 1 becomes 10 so, 1333. So, simply 333 F after the execution of this instruction BX will be loaded with 333 F directly the effective address. Now, the difference between the LEA and MOV will be suppose if I write MOV BX comma same START SI in this case what happens is this will load the contents of the physical address computed from data segment register.

So, this is the effective address and physical address will be, if I assume that the data segment is some 5000, physical address will be 50000 plus 333 F. So, this will be 5333 F. So, from this location 5333 F; if you have some data and the next location what will be the next location? If I add 1 to this 1 F plus 1 becomes 10433. So, this will be 53340 if the contents of this 1 is some FF, contents of this one say EE.

So, after the execution of this one what happens is, BX will be loaded with. So, contents of the physical address it will read a word; word means from 5333 F; it will load into the lower order register and EE will be loaded into higher order register.

Now, I think you understand the difference between MOV BX comma START SI and LEA BX comma START SI. Here the effective address will be loaded whereas, here the contents of the physical address will be loaded,a word from the physical address which is computed from DS START and SI will be loaded into the BX register, this is about the LEA instruction you can rightly load the effective address.

(Refer Slide Time: 14:42)



The next load instruction is LDS Load Data Segment register with some register comma register slash memory; register comma memory. Again wherever the memory comes this is implied, the contents of register slash memory sorry this here there is no register only memory is there register comma memory.

So, this memory contents will be transferred to the register, then the memory plus 2 contents will be transferred to data segment. I will give the examples so that it will be clear. If I give the example using simply direct addressing mode or register indirect addressing mode say LDS BX comma contents of SI simply. So, if the contents of DS is 5000, contents of SI is 2000, contents of 52000, 52001, 52002, 52003.

I need here 4 locations; I will explain why 4 locations are required. So, this is AA, BB, CC, DD. So, what happen now physical address is its clear that 52000; DS plus 10 into 2 is 52000. So, this BX will be loaded with the first 2, contents of the first 2 locations so, this plus next. So, AA will be loaded here, BB will be loaded here; then DS will be loaded with the next 2 locations that is CC DD.

So, this is about this LDS register comma memory, this take the 4 consecutive locations the physical address will be computed by using. So, zeta segment into 10 plus the register content SI content here then the 4 locations. The first 2 locations will be transferred to this register which is specified in the instruction and the next 2 location will be transferred to the data segment.

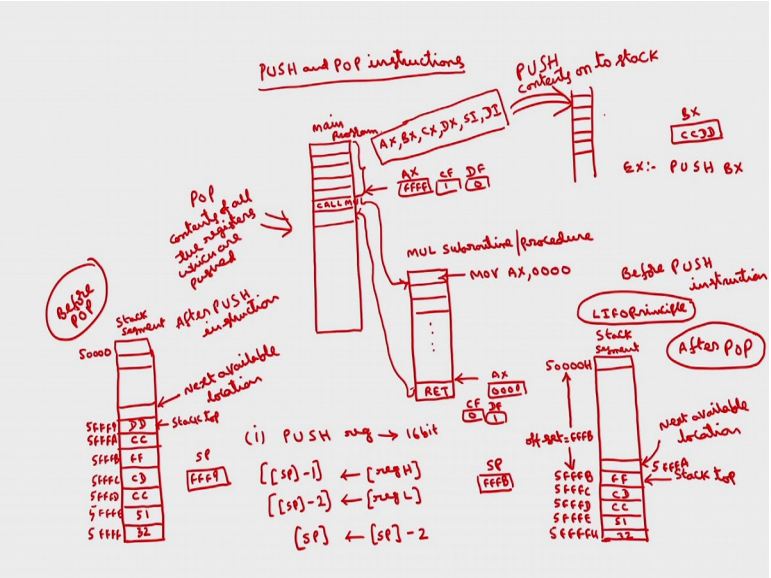
So, next instruction is similarly you can perform for LES also extra segment register comma, memory this is exactly similar to this except for that here we have extra segment. So, I am not going to give example here it is obvious. Next there is one more load instruction, which is LAHF load AH register with flag register contents of the flag, but the flag is 16 bit register AH is 8 bit.

So, what happens is AH will be loaded with here AH is the higher order 8 bits of AX register, will be loaded with lower order flag register flag register lower. So, out of this 16 bits and the flag register that I have explained in the earlier classes this is flag register this is flag register lower flag register higher ok. So, this AH will be loaded with flag register lower values even though this is AH. So, the flag register lower order 8 bits will be loaded into this one. So, these are about the 4 load instructions.

One is effective address can be loaded, second one is data segment can be loaded with some along with some register, they have extra segment along with some register and then the flag register contents. In many cases we need to check the contents of this flag register that will be loaded into AH why only this lower order? This we have discussed in this while discussing about the flag register; in the lower order we have all conditional flags.

So, the conditional flag status is required in many applications whereas, in higher order we have control flags which the programmer is going to set. So, he does not need the admin status because he is going to set that values. So, we don't need the status of those registers those flag registers ok. That is the reason why we are going to use lower order flag register. So, this is about the 4 load instructions.

(Refer Slide Time: 20:17)



So, the next instruction that comes under this data transfer group is PUSH and POP instructions they are very important instructions. So, in this PUSH and POP instructions we are going to use stack. So, I have already briefly explained these instructions while discussing about the stack pointer ok. So, as we have discussed earlier also we just repeat that the microprocessor is executing the main program. This is the main program. So, there is 1 operation which repeatedly occurs. And so, we have to that for that we will write a procedure or subroutine.

Say multiplication operation is coming several times I can write multiplication operation MUL subroutine or you can also call as procedure. So, while the microprocessor is executing these instructions suppose here somewhere call instruction is there call some type of MUL I am going to explain the call instructions also in the instruction set.

So, what the microprocessor will do is immediately this before going for this 1 this instruction pointer contents will be stored onto the stack that will be inside the call instruction no need to write any PUSH and POP instructions. So, the control has to go to here but before the control is moving to this subroutine so, what this will do is, so during this program in the main program so, the main program can be written by using the general purpose register.

So, here I might have used these AX, BX, CX, DX, SI, DI for some purpose ok. Suppose if I have this end of this one if I have some contents of AX is some FFFF. So, if I directly

call this subroutine, here if I write something expression like MOV here if I write the instructions like MOV AX comma 0000. So, what happens these contents will be last so, after the execution of this subroutine this last instruction is returned.

So, the microprocessor control after this call instruction come to this one after the written instruction it will go to the next instruction. So, till this point the AX contents was FFFF; suppose in this process at the end of this before this return instruction if AX contains 0 0 0 0; because I have used this to write some subroutine. So, while return back what will be the contents of AX? The old contents will be last now the new contents will be 0 0 0 0; so, this will affect the main program. Because I will expect that AX should be FFFF; because I have already written some instructions so, with that AX must contain FFFF, 4Fs.

But here in subroutine I am using the same AX register; because I do not have any other option I have to use only these registers to write any program whether it can be a main program or a subroutine. So, here in the subroutine if I use this register for some other purpose. So, while return back the contents of AX has been changed. So, now, what is the solution? The solution for this one is so, before you are going the microprocessor control is going to this sub routine.

So, what the multiprocessor has to do is, what the user has to do, not the microprocessor. So, what the programmer has to do is he has to save the contents of all the registers temporarily in some location which is called as stack ok. So, before calling this subroutine, what you have to do is we store these contents of all the registers regardless of whether these registers have been used in this subroutine or not you just PUSH this onto the stack this operation is called PUSH operation. PUSH these contents onto stack. Stack is a set of memory locations again and after this return instruction, while it is coming for this next instruction here you have to POP these contents.

So, at this point software this returning to this one before returning for this one here you have to POP you POP these contents into this. So, you have to POP all the contents from the same stack, all the registers which are pushed. So, that the contents of this whatever the contents this FFFF will be again restored. So, this is the process of stack operation. So, in case of I mean subroutines are procedures, we need to I mean store the data of all the registers in some temporary location which is called as stack.

So, while doing this is the PUSH operation and POP operation so, for this we have the instructions ok. So, if I take there are some again 4 PUSH instructions and 4 POP instructions. So, if I take the first PUSH instruction PUSH register. So, simply the contents of register will be saved onto the PUSH. So, this register will be 16 bit register. Now, here I am going to assume some stack locations. So, you see the stack pointer this is stack.

So, this is stack segment (Refer Time: 27:09). Let us assume that 64 kilobytes of the stack I have used. So, if the starting address is 50000H, ending address will be 5FFFFH. So, totally 64 kilobytes of the memory I have reserved for stack. So, whatever the data that we are going to store here will be placed here. So, this stack operates on last in first out principle. So, whatever the data that we are going to PUSH will enter here like this.

So, while taking back it will be the input which is the I mean data which is inputted last will be outputted first this is something like arranging the plates in a rack. So, in a rack I will first insert 1 plate above that another plate above that another plate. So, while taking back first I will take the plate which I have inserted at the end at last ok. So, in a similar way this is last in first out concept; suppose if this data is filled up to here some say this is 32, this is 51, this is CC, this is some CD, this is some FF. Let us assume that up to this point the data is already full.

So, what are these locations? This will be 5FFFE this will be 5FFFD this will be 5FFFC this will be 5FFFB. So, the data up to which the data is full is called stack top. So, in the stack segment so, the address the location up to which the data is full is called stack top; then the stack pointer holds the offset from the starting of the segment to the stack top only the offset from starting of the segment to stack top. So, what is the difference between these two addresses this offset is equal to FFFB; so, this stack pointer contains FFFB.

Now, this is next available location. So, whenever you want to PUSH the data so, this is the location where you have to store the data this is next available location. Now, so in this so, PUSH register this register must be a 16 bit register. So, what happens is here, the contents of register 16 bit register will be transferred to so, in this register again we can say that this is lower order as well as higher order, register higher H stands for higher order will be transferred to. So, what is the next location?

If this is the stack pointer what is the offset of this address? This will be 5FF this address will be 5FFFA. So, what is the offset from 5000 to 5FFA is FFFA which is SP minus 1. So, this will be transferred to SP minus 1; then the lower order register contents will be transferred to SP minus 2, then SP becomes SP minus 2 will be assigned to SP. I will give the example so, that this will be clear for example, if I write here itself if I write PUSH BX suppose if BX content is CCDD.

So, this is the contents of the stack pointer before PUSH operation, before PUSH instruction is executed. Now, what will be the contents after this PUSH instruction? This everything is same 32, 51, CC, CD, FF. So, the locations are this is 5 triple 4 F's and this is 5FFFE, 5FFFD, 5FFFC, 5FFFB. So, the stack pointer was here initally stack pointer contents was this FFFB.

Now, what happens is PUSH B if write after this PUSH instruction the starting of this 1 will be somewhere here this is stack segment. So, the initial address is 50000H after PUSH instruction what will be the status. So, according to this operation the higher order 8 bits of this BX register example I have given is PUSH BX this CC will be transferred to the contents of SP minus 1; here SP is FFFB SP minus 1 becomes FFFA .

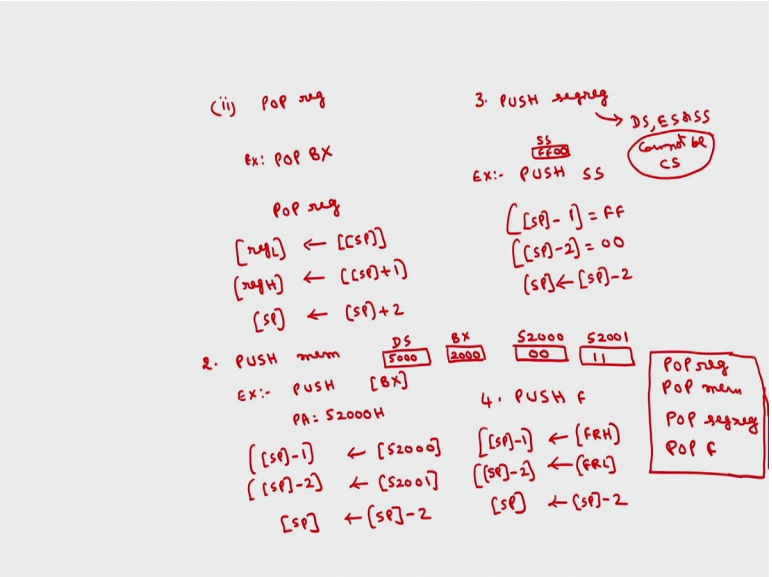
So, this is FFFA , 5FFFA and this will be 5FFF9, after 9, A will be there 10 is A. So, this CC will be transferred here then lower order contents which is DD will be transferred to SP minus 2 and what is the contents of SP before the execution of this instruction? FFFB. SP minus 2 becomes so, if I subtract 1 it will be A if I subtract 2 it will be 9. So, here this DD will be transferred.

Now, the next available location is this, now what will be the contents of the stack pointer. So, we know that stack pointer always holds the offset from the starting of the segment to stack top now this becomes stack top. So, it is clear that the difference from here to here is FFF9. So, compared to this previous value what is this? This will be SP becomes SP minus 2. So, this is what the operation that takes place in this PUSH operation ok.

So, the higher order register contents will be saved onto the SP minus 1, lower order onto SP minus 2, SP becomes SP minus 2. Now, the next available location will be this, so this is before the PUSH operation this is after PUSH operation. Now, the reverse operation; so, as I have told here so, here at the CALL after the CALL what happens is

we are going to PUSH the contents of all the registers before the PUSH. And before the return you have to call back all the register POP you have to all the registers.

(Refer Slide Time: 35:41)



So, if I write the POP instruction in a similar manner POP register; again this register is 16 bit. Now, we can guess what are the operation that takes place this is before the POP instruction; after the POP instruction what happens the contents of DD has to be transferred to the lower order register if I take the same example POP B; if I write the example of same POP BX ok.

So, before this POP instruction, let us assume that these are the contents of this. So, this DD has to be taken onto the this is original what is there in; I mean BX without before execution of the PUSH instruction is CCDD after the POP again the BX must contain CCDD. So, for that so, this DD that is stack top contents has to be transferred onto the lower order register. Then what about this CC, what is the address of this with respect to stack point? Stack pointer is here FFFF9.

So, the contents of the stack pointer has to be transferred to the lower order register and the contents of stack pointer plus 1; contents of stack pointer plus 1 has to be transferred to the BH. Then what happens? Once this DD is transferred CC is transferred now what will be the contents this stack top stack top becomes again the same old FF; the same thing and what will be stack pointer FFFB.

So, when the when compared to this FFF9, what is FFFB? So, this is equal to this plus 2 ok. Now, you can write the instructions the operations inside this POP register. So, here what are the instructions? So, in POP register the operations are the contents of SP is transferred to lower order register and the contents of contents of SP plus 1 will be transferred to higher order register.

Then SP becomes SP plus 2. So, this is clear here. So, contents of contents of SP what is the contents of contents of SP? Contents of SP is this contents of this will be DD that will be transferred to the DL. And contents of SP plus 1 is FFFA that is CC that will be transferred to the DL; once these 2 has been popped these 2 becomes empty. So, next available location is this one stack top will be this. So, if I assume that this is before PUSH and this is after PUSH and if I take this as before POP, this will be again after POP. So, the original contents of the BX will be restored.

In this way we can use the registers in both main program as well as subroutine by using the stack; this is about the first PUSH instruction which is pushed register. The second instruction PUSH instruction is PUSH memory also. So, the memory can be again 1 of the 5 options. So, what is the operation again similar to this one? So, whatever the address that we are going to compute physical address from that location the contents will be pushed on to the stack pointer the same operation.

So, this instead of register high that will be the physical address will be SP minus 1 physical address plus 1 to SP minus 2 SP from SP minus 2. If I write the same example PUSH simply BX; suppose before this DS is again in all memory computations DS will by default DS is the segment register. BX is 5000, 2000 then the physical address will be 52000 let in 52000 and 52001.

If you have some 0 0 some 1 1 say. So, here what happens is this contents of 52000 will be transferred to contents of SP minus 1 and 52001 will be transferred to contents of SP minus 2 and SP becomes SP minus 2; this is same as that register and also this is actually this is contents of.

So, because SP say for example, FFFB SP minus 1 becomes FFFA, but this is not FFFA in that inside that FFFA we are going to store this is contents of again FFFA . So, this is the contents has to be present here so, this is about PUSH memory. The third option is third PUSH instruction is PUSH segment register contents also ok. The same operation

segment register contents will be transferred, but this segment register can be either DS, ES or SS; this cannot be CS; CS is not allowed. similar operation.

Suppose if I take the example PUSH some SS if the contents of SS is FF00. Then, the higher order which is FF will be transferred to the contents of SP minus 1 these contents becomes FF contents of SP minus 2 contents becomes 00, SP becomes SP minus 2 ok. Then there is a fourth instruction which is called as PUSH flag register.

So, flag register is 16 bit register again the contents of the flag register can be also saved onto the stack because again the same concept. So, while the microprocessor is executing these programs at this point AX is this the status of the various flags say carry flag is here set direction flag is reset some status is there.

Whereas, here if I use for some other mathematic logical operations this carry flag status maybe changes at the end of this the carry flag status maybe changed to 0; direction flag maybe changed to 1. So, if I return back here what happens these old contents will be lost and new contents will be present in the flags various flags that is why. So, before you are going for the subroutine you have to save the contents of all the flag registers through this instruction PUSH F ok.

Similarly the same thing here we have flag register, lower order flag register, higher order flag register. So, initially higher will be saved to the SP minus 1 lower order to SP minus 2 SP becomes SP minus 2. So, there are four different PUSH instructions ok; pushing the contents of register, pushing the contents of any memory, PUSH the contents of segment register, PUSH the contents of flag register.

Similarly, we have four opposite operations which are POP operations. And so, I have discussed here this is POP; POP register; we can have similarly POP memory POP segment register POP flag ok. So, totally we have four POP instructions correspondingly, these are the reverse operations to PUSH. Now, operations inside this will be similar to this. Here I have explained for this POP register contents of contents of SP 2 or register lower contents of SP plus 1 to register higher SP plus 2 becomes SP that is POP register and POP memory also same.

So, whatever the memory location that we are going to compute that physical address so, here that physical address will be present here physical address plus 1 SP becomes SP

plus 2. So, in this case the lower order segment register higher order segment register, in this case lower order flag register higher order flag register ok. This is about the PUSH and POP instructions ok.

Thank you.