Microprocessors and Interfacing

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

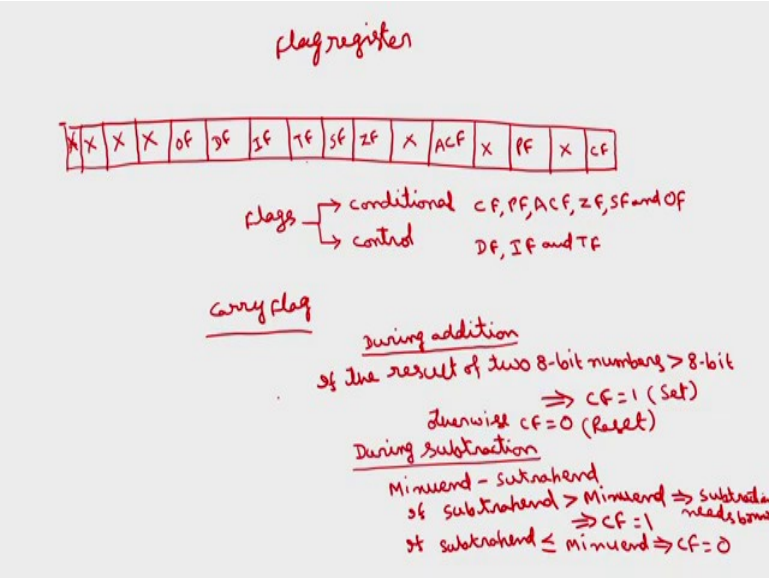
Lecture - 02
8086 Flags

So, yesterday I have discussed about the various microprocessor operations. So, in that there are three types of these operations: one is the microprocessor initiated operations such as memory read, memory write, I/O read, I/O write. So, in order to perform these operations, the microprocessor needs some address bus, data bus and some sort of control signals, which we have discussed in the last class. And the second operations microprocessor operations are internal operations, i.e., without communication with the peripheral device, it can perform some operations inside the microprocessor itself.

So, in these internal operations also there are again five sub divisions. So, the first internal data operation stores the data that is we have discussed yesterday. To store the data, the microprocessor need some registers. So, I have discussed various types of the register present in 8086: general purpose registers, base index, base registers, index registers, then segment registers. Then coming for the third operation which is internal operation which is testing for the conditions.

So, in order to test for the conditions, the hardware element that is required by the microprocessor is called flag register.

(Refer Slide Time: 01:46)



So, in 8086 there is a 16 bit flag registers. So, whereas, in case of 8085 only 8 bit flag register; whereas, in 16 bit 8086 there is a 16 bit flag register. Although the 16 bit flag register is there, but there are only 9 flags; the remaining are do not care, they are not used. So, the positions of these various flags in this flag register are; the first one is carry flag, then this position is not used, this is do not care.

Then we have parity flag, then this position is again do not care; then we have auxiliary carry flag, then this position is again do not care. Then we have zero flag, sign flag. So, these 8 bits are exactly similar to that of 8085 microprocessor. The additional flags that are presenting in 8086 are; they are trap flag TF, interrupt enable flag IF, then direction flag DF, then overflow flag. These are the four additional flags which are available in 8086 and not available in 8085. So, the remaining four are do not cares, they are not used.

Now, we will discuss the function of each flag. So, the flag is basically a flip flop. So, it will be having two states either the flag can be set or reset. So, under what condition the flag is set? Again the flags will be divided into two categories; so the flags are conditional flags and control flags.

So, out of this 9 flags there are 6 conditional flags which are carry flag, parity flag, auxiliary carry flag, zero flag, sign flag, overflow flag. So, they are the 6, I mean conditional flags; the remaining three are called direction flag, interrupt enable flag and

trap flag, they are called control flags. So, here as the name implies conditional, under some conditions only these flags will be set or reset; whereas, the control flags have to be programmed by the user, the programmer. So, he can set or reset this flags control flags, accordingly there will be some control action.

So, coming for the conditional flag; the first flag which is carry flag. So, carry flag. Carry flag is set, we have only two conditions set and reset conditions; carry flag is set if it is during addition. So, this carry flag will be affected by arithmetic operations; addition as well as subtraction. So, in during addition; if we want to add two 8 bit numbers, if the result is more than 8 bit the carry flag is set.

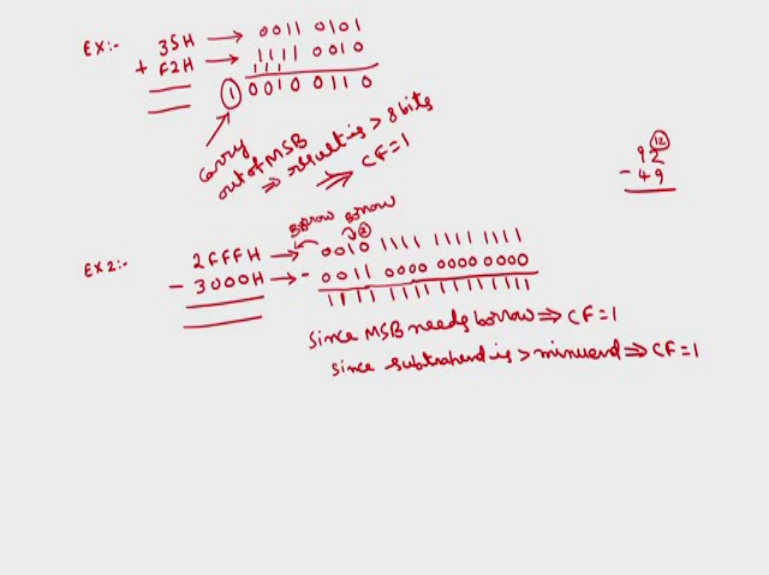
So, if the result of two 8 bit numbers is greater than 8 bit then the carry flag is set; set means 1. If the result of two 8-bit number, sum of two 8 bit numbers during addition; so sum of two 8 bit numbers is less than or equal to 8 bits only, then carry flag will be reset, otherwise carry flag is reset; this is set and this is reset.

Similarly, if want I mean perform addition of two 16 bit numbers; if the result is greater than 16 bit in that case also carry flag will be set. If it is 8-bit operation, if we add two 8 bit numbers; if the result is more than 8 bits in that case also carry flag is set. So, if we if we add two 16 bit numbers, if the result is more than 16 bit in that case also the carry flag will be set; this is during addition.

Whereas during subtraction, if MSB bit needs to borrow or otherwise if I take; so subtraction will be normally performed by minuend minus subtrahend. If subtrahend is greater than minuend; then in that case the carry flag will be set. So, there is no borrow flag. So, even in case of borrow also carry flag will be set, is greater than minuend implies carry flag is set; that means, this needs this subtraction needs. So, this is a case imply subtraction needs borrow.

So, in that case carry flag is set; if subtrahend is less than or equal to minuend, then carry flag will be reset. So, this is the case of whether this 8-bit number or 16 bit number. To perform the subtraction of two 8-bit number subtraction of two 16 numbers; if subtrahend is less than minuend, then the carry flag will be set; if subtrahend is less than or equal to minuend, then carry flag will be reset.

(Refer Slide Time: 09:29)



So, you can take some examples. Suppose if you want to perform 35 H plus F2 H. So, if we take the corresponding binary 35 H will be 3; what do we mean 3 bit, I mean 4-bit hexadecimal equivalent of 35 H; 3 will be 0 0 1 1; 5 will be 0 1 0 1 and F 2; F is 15 which is 1 1 1 1 and 2 is 0 0 1 0. If you want to perform this addition, this is 8 bit addition. So, if I add 1 plus 0 is 1; 0 plus 1 is 1; 1 plus 0 is 1; 0 plus 0 is 0. 1 plus 1 will be 1 carry sum is 0; 1 plus 1 plus 1 is 1 carry sum is also 1; 1 plus 1 is 1 0; 1 plus 1 plus 1 is 0 1.

So, there is a carry out of MSB bit. So, the result is greater than 8 bits. This implies carry flag will be set. If I take another example for 16 bit; if I take say subtraction 2FFF H minus 3000 H. So, 2FFF H will be the 16 bit equivalent of this one is 0 0 1 0, 1 1 1 1, 1 1 1 1, 1 1 1 1 3000 H will be 0 0 1 1, 0 0 0 0, 0 0 0 0, 0 0 0 0; you have to subtract this, binary subtraction.

So, 1 minus 0 is 1 1 1 1, 1 1 1 1, 1 1 1 1; 0 minus 1. How to perform 0 minus 1? So, 0 minus will normally people will think that this is 0; so, but here this requires borrow, it takes a borrow from this bit. So, if it takes the borrow, then the decimal equivalent of that value will be 2; because in case of a decimal number if you want to subtract 92 minus 49.

If you take a borrow from here, this become 12; means 10 will be added, 10 is the base of the decimal number system. A 10 will be added to these two that is why it becomes

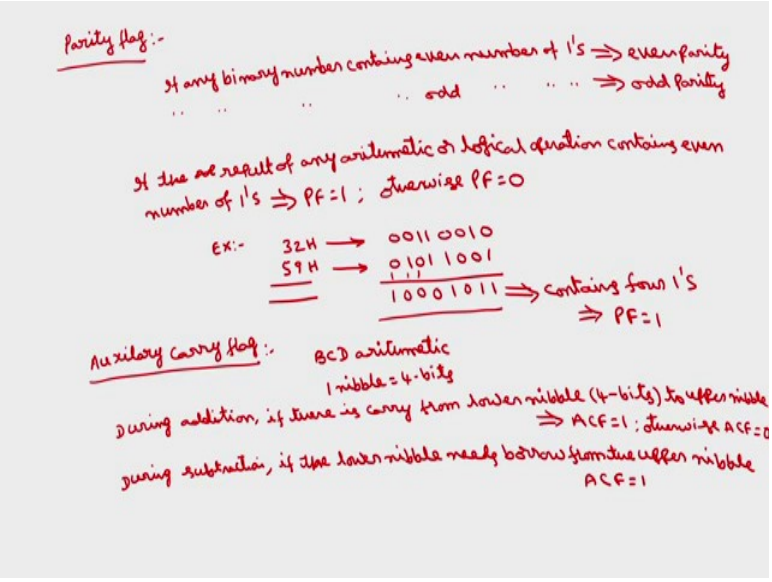
12; whereas, here if I take the borrow, the base of the binary number system is 2. So, 2 will be added. So, this is 0 is there, 2 will be added. So, 2 minus 1 becomes 1.

Similarly, now here this one has been already given to borrow to this one. So, this becomes 0. So, it has 0 minus 1 to perform this we require a borrow from this stage. So, this becomes 2, 2 minus 1 is 1. So, this is actually 0 minus 0; but this has given on borrowed to this previous bit. So, to pay that borrow again it has to take the borrow. So, if it takes the borrow, it becomes 2. So, out of these two; one has been given to the previous stage, the remaining 1 minus 0 becomes 1.

Similarly, the case here will be, this has to take borrow from here; this will be 1, if I take the borrow. Because the MSB bits requires borrow, this is more significant bit; carry flag will be set. Now as I have told the second one, this number 3000 H this subtrahend is greater than minuend. So, the carry flag is set; then the other way to check this one is, since subtrahend is greater than minuend implies carry flag is set.

So, you had seen about this carry flag; so during the addition as well as during the subtraction. So, these are the conditions for the carry flag to be set; then coming for the second flag, second conditional flag, so which is parity flag.

(Refer Slide Time: 14:30)



So, parity flag will be useful, so in the data communication systems. So, where if you want to correct detect and correct the errors, so there we need the parity bits, ok. So, it is

a different thing. So, the applications of this one is parity, I mean generation checking will be in error detection and correction. Parity is a bit which will be added to the message bits, so that the entire message will becomes either even parity or odd parity; accordingly we will check it with the receiver, ok. So, we have some hamming codes; and there is a course on this coding theory.

So, now coming for the parity, so if the binary number contains even number of one's then it is said to be having even parity; if any binary number contains even number of 1's, it is said to be having even parity. On the other hand, if it contains odd number of 1's as the name implies, implies odd parity. Now, condition for this parity flag to be set is; if the result of any arithmetical logical operation contains even number of 1's or even parity then the parity flag will be set.

If the result of any arithmetical logical operation contains even number of 1's implies parity flag is set; otherwise parity flag is reset. So, if I take this some example here also, this is regardless of 8 bit number or 16 bit number. If I want to add 32 H plus 59 H, so this 32 will be 0 0 1 1, 0 0 1 0; 5 9 will be 0 1 0 1, 1 0 0 1. So, if we add this 1 1 0 1; 1 plus 1 is 1 0; 1 plus 1 is 1 0; 1 plus 1 is 1 0 1. So, how many number of 1's are there? This contains four 1's. So, implies parity flag is set because four is even number, it contains even number of a; so parity flag is set, this is a simple flag.

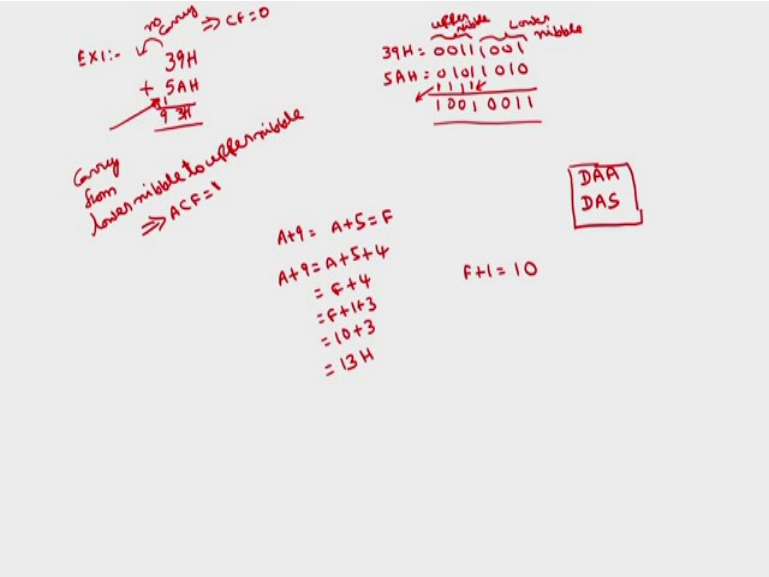
Then coming for the third conditional flag; so which is auxiliary carry flag. One is simple carry flag, this one is auxiliary carry flag. So, normally this auxiliary carry flag will be used in BCD arithmetic and this is BCD addition and subtraction. So, in BCD addition and subtraction; so the lower order 4 bits will be important whether we are going to perform packed one bit one digit BCD addition or two digit BCD addition; one BCD digit means 4 bits, ok.

So, the lower order 4 bits will be important. So, if the lower order 4 bit; if there is a carry from MSB carry flag will be set, as we have discussed in the previous slides, ok. So, if there is a carry from lower nibble to upper nibble, nibble is called a 4 bit; 1 nibble means 4 bits.

So, during the addition if there is a carry from lower nibble to lower 4 bits to upper nibble implies auxiliary carry flag will be set. If there is no carry out of lower nibble to upper nibble; otherwise reset, this is during the addition.

Similarly, during subtraction; if the lower nibble needs borrow from the upper nibble. So, in this case also auxiliary carry flag will be set otherwise reset. So, there is a carry from lower nibble to upper nibble or the lower nibble needs a borrow from the upper nibble; so that the regardless of whether the number is 8 bit or 16 bit, we will just check only the last 4 bits only, last nibble.

(Refer Slide Time: 21:08)



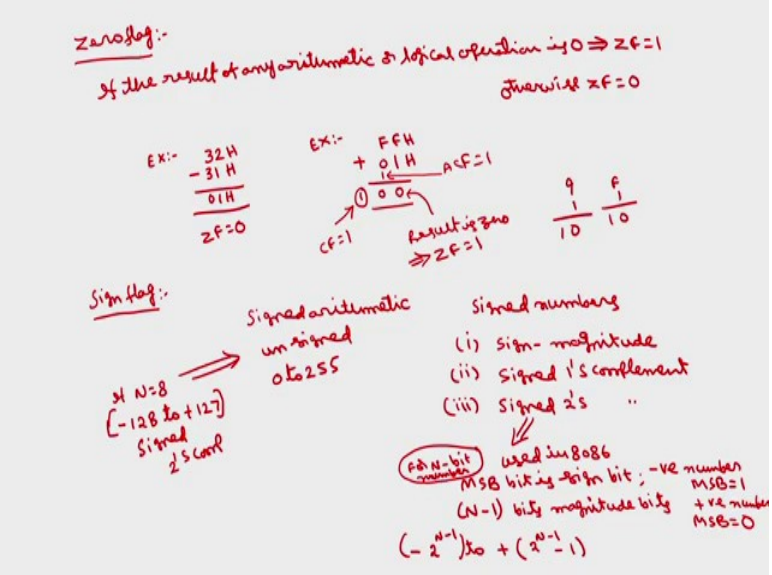
So, example of this auxiliary flag will be I will take for the sake of simplicity just 4 bit, I mean say 8 bit numbers; take 39 H plus 5A H. So, you can perform directly by using hexadecimal addition also, if you want to say for example, perform hexadecimal addition 9 plus A; so A is 10, 9 is A plus 10 will be 19. So, what will be the hexadecimal equivalent of 19 decimal equivalent? So, 9 plus A plus 1 will be B C D E F after that 1 0 1 1 1 2 1 3, ok.

So, A has to be added to, A if I add A plus 9 is nothing, but A plus 5 will be F. So, F plus 4 this will be equivalent to F; so A plus 9 means you can write this one as A plus 9 is A plus 5 plus 4. So, this is equal to A plus 5 is F plus 4; F plus 1 becomes 1 0, F plus 1 becomes 1 0. So, this will be F plus 1 plus 3, this is equal to 1 0 plus 3 which is equal to 13 H, ok. This will be 1, this will be 3 and this will be simply 9, 93 H. So, because they carry from lower nibble to upper nibble, this is the carry from lower nibble to upper nibble; this is lower nibble 9 and A, implies auxiliary carry flag is set.

We can also check that there is no carry out of the MSB. So, it implies carry flag is reset. Or you can do this using binary also, 39 H will be 0 0 1 1 1 0 0 1; 5 A will be 0 1 0 1 1 0 1 0. If you perform the addition; this is lower nibble, this is upper nibble. 1 plus 1 is 1; 0 plus 1 is 1; 0 plus 0 is 0; 1 plus 0 there will be carry, so 1 0. 1 plus 1 plus 1 is 1 1; 1 plus 1 is 1 0 this is 1, this is the result. So, this is same as this 9 3, ok. So, you can observe that there is a carry from here to here; from here to here there is no carry. So, because of this carry, auxiliary carry flag will be set; because there is no carry out of this, carry flag will be reset. So, you see the difference between carry flag and auxiliary carry flag.

So, this type of this auxiliary carry flag will be used in as I have told BCD arithmetic; there are two instructions like decimal adjust accumulator DAA well while discussing about the instruction we will see, this I mean auxiliary carry flag will be used in this instruction. Similarly we have decimal adjust after subtraction. So, there are two instructions. So, during this instructions that auxiliary carry flag will be discussed, that will be used by these two instructions.

(Refer Slide Time: 25:23)



Coming for the fourth conditional flag; so which is zero flag, this is a simple one as the name implies. If the result of any arithmetical or logical operation is regardless of this 8 bit operation or 16 bit operation; if the result of any arithmetical or logical operation is 0 plus zero flag is set, otherwise zero flag is reset. If I take the example say. So, if I take 32 H minus 31 H say, the result is 1 0 0 1 H. So, zero flag will be reset.

So, if I take the second example; if I want to perform F F H plus 0 1 H now what happens, you can do it in hexadecimal or as well as binary also. So, F plus 1 as I have told 1 0; because in decimal number system 9 is the maximum number, 9 plus 1 will be 1 0. Similarly here also F is the maximum number, F plus 1 becomes 1 0; and F plus 1 is again 1 0. So, there is a carry out of this one. So, carry flag will be set and from F plus 1 these are also there is one carry from lower nibble to upper nibble also.

So, there is a carry from upper nibble to lower nibble also. Here auxiliary carry flag also set, carry flag is also set; but the result is you have to neglect the carry. So, because you are adding two 8 bit numbers, you have to see only the lower order 8 bits. So, lower order 8 bits result is 0, this result is 0; result of this addition is 0 implies zero flag is set. There will be carry, but the result 8 bit result, lower order 8 bit result is 0. So, zero flag is set.

Similarly, the next conditional flag is signed flag; as the name implies the sign flag will be used in signed arithmetic. Means, if you want to represent; we know that the computer cannot understand plus and minus, so computer understand only the language consisting of zeros and ones. So, if you want to give the plus sum number minus sum number, you have to represent plus and minus also with in terms of zeros and ones.

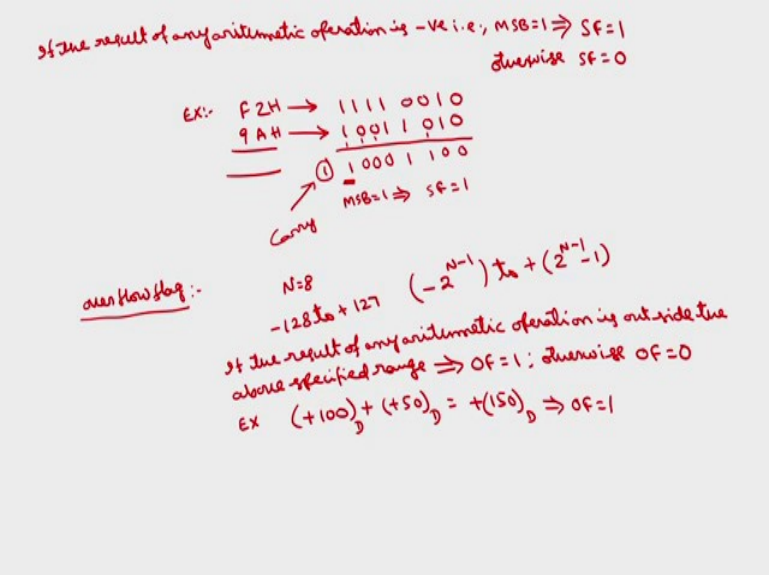
There are three ways to represent the signed numbers; signed numbers can be represented in three ways. So, we can have one signed magnitude form; signed 1's, signed 2's. Signed 1's complement form, signed 2's complement form; but in 8086 the signed 2's complement will be used, because there are some advantages of sign 2's complement representation. If you add a group of numbers; so if the final result is within the limit, then intermediate overflows you can neglect.

So, in signed 2's complement numbers; suppose if you have N number, N bit number. So, if it takes a 8 bit number or 16 bit number in general N bit number; the first bit MSB bit, MSB bit will be used for side bit. Then N minus 1 bits for N bit number; MSB bit is sign bit, N minus 1 bits will be magnitude bits. So, as the result of that the range of these 2's complement numbers that can be represented for N bit number is, it varies from minus value of 2 raised to the power of N minus 1 to plus value will be plus value of 2 raised to the power of N minus 1 minus 1.

For example if I take N is equal to, if N is equal to 8. So, this range becomes minus 128 to plus 127. So, even though we have 8 bits, this is the range of the numbers that we can represent in sign 2's complement representation. If it is unsigned number, so 8 bit means you can represent from 0 to 255; for 8 bit if N is equal to 8 in case of unsigned numbers. If I use all the bits for magnitude only; then the range will be 0 to 255.

Whereas, if I use signed representation, signed 2's complement representation, the range of numbers only; because the first bit I am using for this sign bit, and remaining N minus 1 bits we are using for the magnitude bits. So, the sign bit is again here. So, for negative numbers MSB is 1; for positive numbers MSB is 0. So, this is the range of 8 bit numbers in sign 2's complement representation. Now, here the sign flag, if the result of any arithmetic operation is negative number means MSB bit is 1; then the sign flag will be set, otherwise sign flag is reset.

(Refer Slide Time: 32:20)



So, the condition for the sign flag will be set is, if result of any arithmetic operation is negative that is MSB is 1; whether the number can be 8 bit number or 16 bit number implies sign flag is set, otherwise sign flag is reset. So, if you take the example here also; if you want to add F 2 plus 9 A H. So, F 2 is 1 1 1 1; 2 is 0 0 1 0; 9 is 1 0 0 1; 2 is 0 0 1 0 sorry this is 10, 1 0 1 0. So, if we add these 0 1 plus 1 is 1 0; 1 1 1 plus 1 is 1 0; 1 plus 1 is 1 0; 1 plus 1 is 1 0; 1 plus 1 is 1 1 this is carry. So, carry flag will be set.

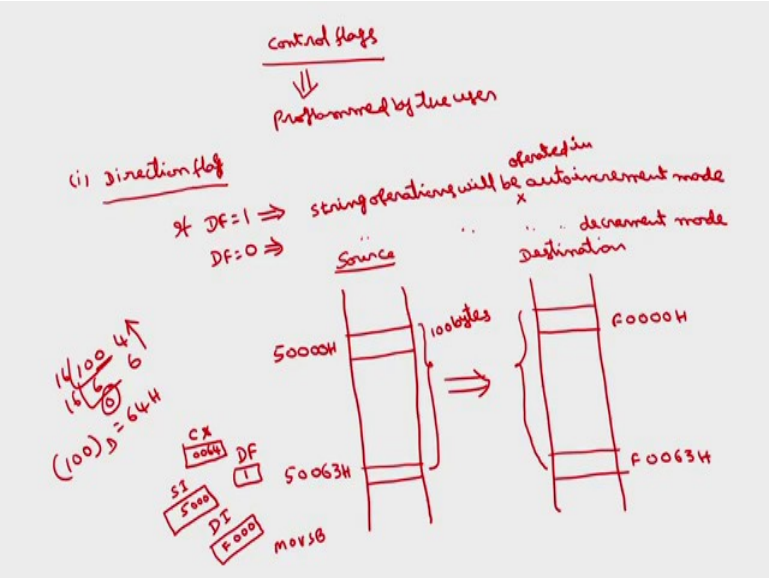
Now, neglecting the carry; so this will be the MSB bit, this is the MSB bit. Because MSB bit is 1; the sign flag will be set, this represent the number is negative. MSB 1 implies sign flag is set. So, if this bit is 0, sign flag will be 0; simply this MSB bit will be copied onto SF. If this is 1, SF is 1; if this is 0, SF is 0. So, in this case if I take the other flags also carry flag will be set, because there is a carry and from here to here there is no carry from lower nibble to upper nibble, auxiliary carry flag will be reset; result is nonzero, zero flag is reset. So, these are the different flag conditions.

Now, coming for the last conditional flag, which is overflow flag. So, if the result of any arithmetic operation is outside the range of the signed 2's complement number, as I have given the range for the 8 bit numbers. So, if the result of arithmetic operation is outside the range; if it is N is equal to 8, I have given minus 128 to plus 127. In general this is 2 raise to the power of N minus 1 minus of this to plus of 2 raise to the power of N minus 1 minus 1. If the result of any arithmetic operation is outside this range, then overflow flag will be set; implies overflow flag is set, otherwise reset.

If I take say for example, if I want to add plus 100 plus 50; decimal say for example, this is example is decimal, I am saying that this is decimal. So, what will be result plus 150; but what is the range if it is 8 bit number assuming that N is equal to 8 here, because to represent the 100 we require 8 bits. So, N is equal to 8, the range should be this; but this is outside the range, so implies overflow flag is set. So, if the result of any arithmetic operation is outside this range, this range in general; so it this is can be, N can be 16 bit or 8 bit or any number of the bits.

So, if the result is outside this range, then the overflow flag is set; otherwise overflow flag will be reset. So, these are the conditional flags and there are some three control flags.

(Refer Slide Time: 37:11)



So, the control flags will be set by programmed by the user; the flags can be programmed, we can set or reset this three flags. There are some instruction to set and reset the control flags. So, the first control flag is direction flag. If direction flag is set to 1 by programmer, there is a instruction setting the directional flag. If direction flag is 1; then the string operations will be, this will be used in string operations auto increment will be operated in auto increment mode, I will explain what is this.

So, if directional flag is reset, string operations will be operated in auto decrement mode. So, string operations basically if you want to transfer a string of data from one set of the locations to another set of locations; this is the source and this is destination. So, I want to transfer 100 data from source to destination; say starting address of the sources is say some 50000 H to 100 data. If you want to transfer 100 data; 100 bytes decimal 100 bytes to here block transfer to this.

So, if the 100 bytes decimal 100; if the starting address is 50000, what will be the ending address? So, this 1000 bytes means 1000; what is the hexadecimal equivalent of 1000? 16x6 are 96, 4 is the remainder, 16 zeros 6 is the remainder. So, you have to stop here. So, 64 H; 100 decimal is equivalent to 64 H. But you are starting from 0; so total 64 H means 0 means the ending will be 6 3; 5 0 0 63 H, because you are starting with 0.

So, I want to transfer this to say some locations F 0 0 0 0 H to F 0 0 6 3 H. So, this data I want to transfer here; 50000 one data to F 0 0 0 0 1, so one up to 50000 6 3 H to F 0 0 0

6 3 H. I want to simultaneously transfer 100 bytes of the data; these steps of operation are called string operations. So, in the string operation what we will do is; so this is in case of 8086 automatic. So, later while discussing about these instructions and programs, we will discuss this. So, there will be a counter register. So, normally CX will acts as a counter register. So, in this you load the number of bytes to be transferred, so 0 0 6 4; and the starting address of the, this you have to represent in the source index.

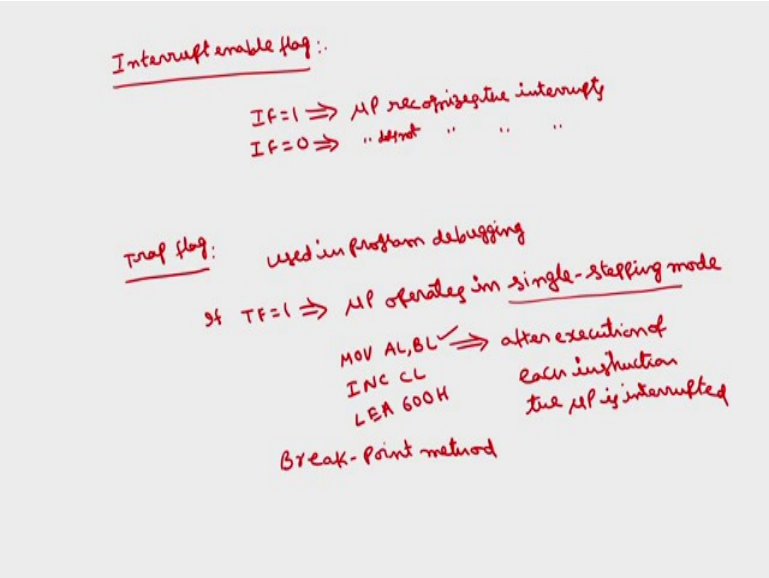
So, we know that in the last class also I have described; so all the registers have 16 bit registers, but physical address are 20 bit. So, all these registers will store only the starting 16 bit of that particular address. So, here this is 50000, but the starting 16 bits will be 5000. So, source index will be having 5000. This is another application of the source index register that I have discussed in the last class; then destination index, the starting address of the destination, the first 16 bits, F 0 0 0.

So, we see that what that we have to give, and you give simply as simple instruction, move string byte; I will discuss all the instructions later, there is a instruction like MOVE string byte. Then what happens is, automatically the hundred bytes will be transferred, ok. But in which direction, this is the starting address I have given, this is the starting address, the total number of bytes I have given as 64 which is 100 bytes; but after transferring the first step, whether this has to transfer from this to I mean bottom to top or top to bottom that will be decided by the directional flag.

So, if I set the directional flag to 1; if directional flag is 1, then this address will be automatically incremented, SI and DI will be automatically incremented. So, this transfer will be from top to bottom; if the directional flag is set to 0, the transfer will be from bottom to top. So, this will be first transfer, then 50000; the one address which is above this one that will be transfer above that, that will be transfer like that total 100 bytes will be transferred.

So, this is going to decide the direction in which the string operations will be operated. So, if direction flag is set. So, these string operations will be auto increment; means SI and DI will be automatically incremented after each byte transfer. If direction flag is reset, then after each byte transfer the contents of SI and DI will be automatically decremented that is about this direction flag.

(Refer Slide Time: 44:10)



And the second control flag is interrupt enable flag. As the name implies if interrupt enable flag is set; so the microprocessor enables or recognizes all the interrupts. So, external device can interrupt the microprocessor through interrupts. If interrupt enable flag is 0, microprocessor does not recognize. See even though if external device want to interrupt the microprocessor; that is not possible, if interrupt enable flag is 0.

So, normally this interrupt enable flag will be reset, if the microprocessor want to perform any emergency operation. So, in that case; so it can set the interrupt flag, so that the external device will not disturb that microprocessor. So, if interrupt enable flag is 1, then microprocessor does not disturb interrupts. There are some different types of the interrupts in 8086 that we will discuss later. So, you see about interrupt enable flag. And the last flag is the control flag, third control flag; so is trap flag.

This flag is basically used for debugging a program; debugging means, checking for the errors. So, after writing the big program; so we have to check the errors in the program. So, if trap flag is set to 1 by the programmer; the microprocessor enters into single stepping mode, microprocessor operates in single stepping mode. And that is after each instruction is executed, the microprocessor basically will be interrupted. So, at the interrupt service subroutine, you can write the program to display the contents of the registers and memory locations.

So, after each instruction; say for example, if you have something like MOV AL, BL. INC CL some sort of instructions, I will describe the instructions later; LEA 600 H, some sort of the instructions. If I set the trap flag to 1, first it will execute the first instruction; after this instruction the microprocessor will be disturbed interrupted, after execution of each instruction the microprocessor is interrupted.

So, we whenever the microprocessor is interrupted what happens is, it will stop the execution of the main program and it will go to some subroutine which is called interrupt service subroutine; I will discuss that in detail in upcoming lectures. So, in interrupts service operating you can write a program to display the contents of the memory and registers.

So, that after this I can check what are the contents of the registers, what are the contents of the memory locations; then I can check whether this particular instruction is correct or not. Again if this instruction is correct; so you go into another instruction, second instruction, again you display the contents. So, like that after each instruction if you check this contents of the memory location and registers; because we know these results, so what will be the contents of registers.

So, if you get some result wrong results in the registers, then we can say that this particular instruction there is a mistake in this particular instruction; but the drawback of the single stepping mode is, it takes lot of time. If you have 1000 instructions, so after each instruction it will be interrupted; means total it will be interrupted 1000 times ok, it will take a lot of time.

One is executing the entire program once; so the problem is, it is very difficult to find out. So, the second way is single stepping; after each instruction you have to check the contents of the registers and I mean memory locations. So, this is time consuming process, there is one method which is in between to these two; instead of executing the entire program ones are executing one by one instruction, there is one called break point method.

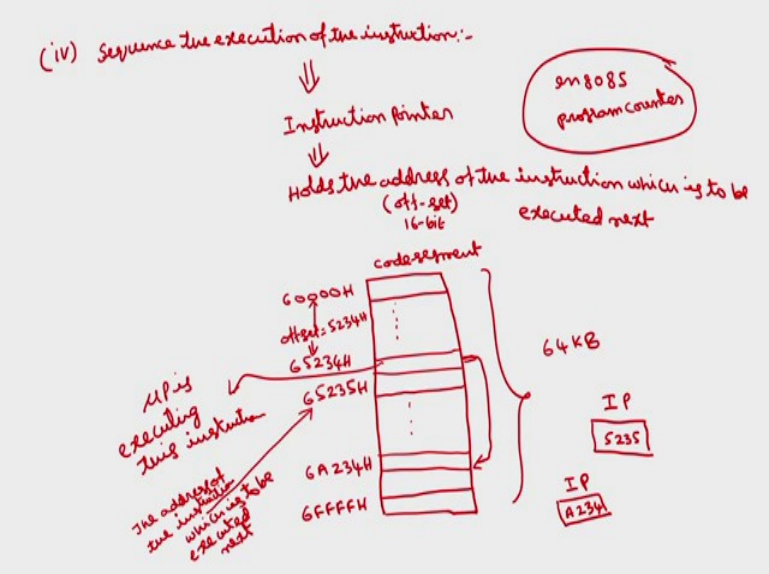
So, in this what happens is? So, the programmer can write after 10 instructions say I want to check the contents of registers and memory; after another 10 instruction, after another 10 instructions. So, we can increase the speed up and compare to the single stepping and also it is easier to check 10 instruction is not many. So, you can check in

which instruction the error is there. So, it is a break point method. So, this is single stepping.

So, in case of if trap flag is set, the microprocessor operates in single stepping mode; otherwise it will execute the entire program once. For breakpoint also there will be some directives, using assembler directives you can set this breakpoint also. So, it is all about all the, I mean flags which are available in the 8086 microprocessor; you see the third operation we are discussing about the internal data operations.

So, the first one is stored the 8 bit of the data for which we need various registers; second one is, so it will perform arithmetic logical operations, we need a single ALU. Then the third one is testing for the conditions. So, in testing for conditions, the microprocessor needs the hardware element called flag register. So, I have discussed about 8086 different flags.

(Refer Slide Time: 50:21)



Then the fourth operation; so the fourth internal data operations is, sequence the execution of the instructions. After a particular instruction is executed, which instructions needs to be executed next. So, in order I mean sequence execution of the instruction; the hardware component that is required by the microprocessor is called instruction pointer. Instruction pointer is used for sequencing the execution of the instruction. So, this in 8086 called as instruction pointer; whereas in 8085, this is called as program counter. The same purpose, but the name of that one is program counter.

So, instruction pointer always holds the address of the instruction which is to be executed next; but the address is a not a physical address, is a logical address or offset. Instruction pointer holds the address of the instruction; this address means, this is actually effective address or you can call as offset, offset address 16 bit not 20 bit, because the register is 16-bit length. So, address of the instruction which is to be executed next.

For example, if I take the code segment as we know that, code segment will be used for storing the programs, this is code segment. So, the starting of the code segment is say some 60000 H; let us take that code segment can be a 64 kilobyte, maximum of 64 kilobytes as we have discussed in the last class. Suppose the microprocessor executing somewhere here this instruction, the instruction that is stored in this location.

Suppose the address of this one is 65234 H; then from here to here what is the offset, from here to here what is the offset? From here to here offset is difference 5234 H. So, what is the address of the next instruction; if I assume that this is single byte instruction, this instruction is single byte instruction, the address of the next instruction to be executed next will be 65235 H. This is the address of physical address of the instruction to be executed next; the instruction which is to be executed next.

Now, what is the offset from here to here is 5235; then the instruction pointer points to that offset address 5235. So, the instruction pointer always holds the address which is offset of the instruction which is to be executed next. If the microprocessor executing this instruction assuming that single byte instruction; the next address will be 1 plus, this address plus 1 which is 5235 H.

So, the offset from 60000 H is 5235; that is 16 bit offset address will be pointed by instruction pointer. So, instruction pointer always holds the address of the instruction which is to be executed next; in that way we can sequence the execution of the instruction. Suppose, if I want after execution of this instruction, I want to jump here; here whose I mean offset is something like 6A234 H.

After execution of this instruction if I want to jump to this instruction; if I want to skip all these instructions what you have to do is, we just simply load instruction pointer with the address of the offset of the instruction which used to be executed next which is A234 H. Then the program will be automatically, program control will be automatically transfer to this; it will skip all these internal instructions, that is where you can sequence

the execution of the instructions, this here instruction pointer. So, the last I mean operation which is stored the data in memory which is called stack that I will discuss in the next lecture.

Thank you.