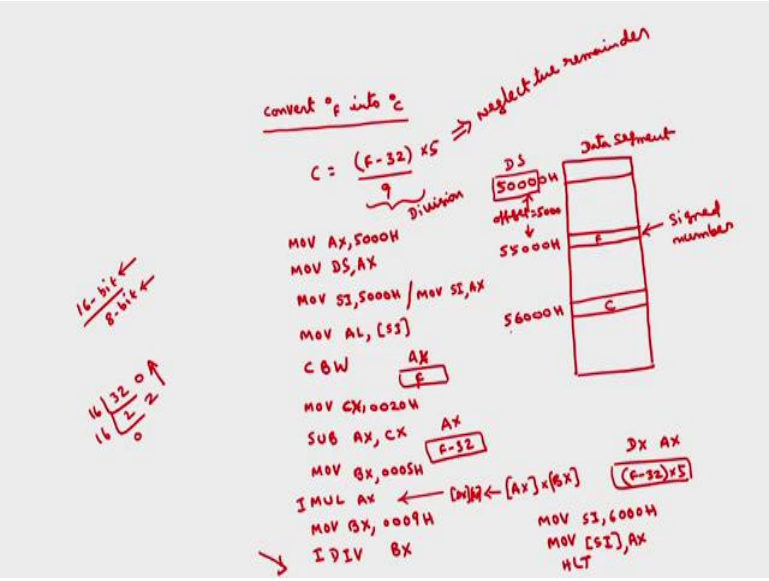
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture – 14
Programs on Subroutines

So, again in the next class we have discussed about some programs. So, in this class we will discuss some more programs.

(Refer Slide Time: 00:41)



So, the next program is conversion of Fahrenheit into centigrade for that you have to write the program. So, we know that the conversion formula is centigrade is equal to F minus 32 into 5 by 9. So, given F value you have to find out C for that you have to write the assembly language program using 8086.

So, you assume the appropriate locations in data segment. Let the starting of this one is 50000H and with a offset of 5000H the value of Fahrenheit is stored and the stored the result centigrade in location with a offset of 6000H ok. So, the program will be the first two instructions are MOV AX comma 5000 MOV DS comma AX. Then you take this Fahrenheit value which is stored in a 55000 location onto AL register.

So, you have to find this offset with some register say SI comma 5000H; here one of the advantage of assuming these locations is instead of this MOV SI comma 5000 you can

also write MOV SI comma AX also; because the data segment register also points to 5000H and the offset is also 5000H. So, in this particular example both are matching. So, you can directly take this AX contents onto SI this will save 1 byte. So, I am taking this into AL MOV AL comma contents of SI.

Now AL contains the Fahrenheit value. So, you have to subtract this 32 from Fahrenheit value ok. So, in order to subtract this, before subtraction of this one, you have to perform the division also later. This is division. So, we have division instruction directly, but the division instruction will be for 16 bit by 8-bit and 32 bit by 16 bit, but here this Fahrenheit value is only 8-bit only. So, the numerator should be the 16 bit denominator can be 8-bit.

So, before subtracting this 32 and performing the subtraction what I will do is, I will convert byte into word. This is sign extension. Now, AL will be having word AL. The sign bit of AL will be copied onto AH and this becomes AX. AX will contain now F value. This I have explained in the instruction set, convert byte into word, simply the sign bit of AL will be copied onto AH. Now, the question is so, why you are taking the sign bit? Here because the Fahrenheit can be negative value also, that is why we have to use sign numbers here we are assuming that F and C are signed numbers.

So, in case of signed multiplication or signed division so, you how to extend the sign if you want to perform 16 bit by 8-bit division. So, we see CBW. Now, AX will be having Fahrenheit value 16 bit equivalent of this 8-bit value then you have to subtract 32; we take some CL or somethinG32H. The problem is here 32 is decimal ok, but we have to subtract hexadecimal value. So, this will not be 32, what is the hexadecimal equivalent of 32? Because, the microprocessor will perform the hexadecimal operations; but this 32 is decimal value ok. So, 32 decimal is equivalent to what is hexadecimal? 16 twos are 16 twos s are 0 is the remainder, 16 0s, 2 is the remainder 20 H. So, you have take 20 H here. This is the one of the important point here.

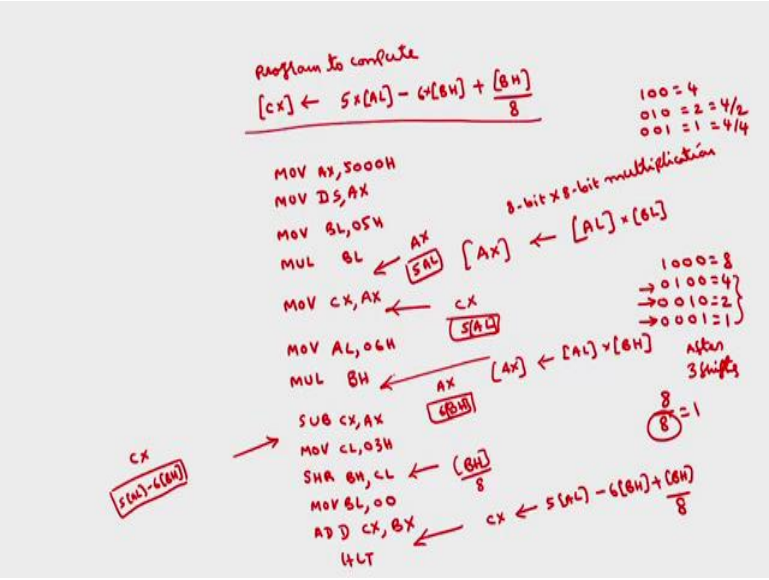
Then we have to multiply with 5. So, we take MOV 5 is actually this whether this hexadecimal or decimal both are same. Similarly, 9 subtract AX comma SUB AX comma you can take CX because the source is 16 bit we can take 0020H and subtract AX comma CX. Now, in AX what will be there? A 16 bit equivalent of F minus 32 ok. Now, this we have to multiply with 5. So, you can multiply either 16 bit by 16 bit

number or 8-bit by 8-bit number. So, you take this 5 MOV; 5 anyhow this is positive number. There is no need of sign extension. I will take these directly into some registers say BX comma 0005H.

I will perform MUL I MUL because numbers are signed numbers I MUL I MUL AX. So, what is the operation here? AX and BX will be multiplied and the result is stored in DX AX. Now, in DX AX we will be havinG32 bit result of this 16 bit into 5. Now, you have to divide this with 9 also you have take 16 bit data. So, you take MOV BX comma 0009H you may get some remainder also. So, I am going to neglect the remainder. So, I am taking only integer value of this, I am not considering the fractional part. So, then IDIV this is 32 bit number and this is 16 bit number IDIV BX. So, this will perform DX AX by BX.

This DX AX was F minus 32 into this one. DX AX will be havinG32 bit equivalent of F minus 32 into 5 that I am going to divide with BX; BX is having 9 now. So, after this I division the remainder will be there in DX that I am not interested and quotient is there in AX that I am going to store into this location. For that I am writing MOV SI or BX any value SI you can take SI comma 6000H MOV. So, this remainder we are going to neglect, quotient will be there in AX ok. So, that I am going to store in SI. This is the program which converts the Fahrenheit temperature into centigrade temperature J direct formula computation.

(Refer Slide Time: 09:50)



So, I will discuss the next program will be suppose if I want to compute in CX register. I want to store say 5 into AL; AL contents minus some 6 into BH contents plus contents of BH by 8, basically this is multiplication. So, we will be given AL value BH value we have registers inside the 8086 is BH. So, you take those values; you multiply AL with 5, multiply BH with 6 and divide BH with 8. So, you add this two 5 AL subtract, 6 BH and you store that result in CX ok. Here no need to assume any locations because already AL data BH data is available in the microprocessor.

So, I am assuming the same data segment contents with 5000H MOV AX comma 5000 MOV DS comma AX then you compute this step by step 5 into AL ok. So, you have to take 5 into some register say MOV BL comma 05H then let us assume that these numbers are? Unsigned numbers MUL BL; AL is already there ok. So, in MUL BL what happens? MUL BL is 8-bit 8 bit multiplication. In case of 8-bit by 8-bit multiplication hence you have discussed one of the operand will be in AL second operand can be either register or memory here I have taken register BL, the result is stored in AX. This is the 8-bit by 8-bit multiplication.

So, I have taken this 5 into BL AL already some data is there I have multiplied with BL. Now, in AX this 5 into AL will be there. So, after this in AX 5 AL will be there. I am taking this into some register CX, AX. So, in CX also I am storing same 5 AL value. Why because, I want to use this AX in the other operations also because in multiplications and divisions this AL and AX will be used by default, that is why I want to store this temporary value intermediate value 5 AL into some registers CX ok.

Now, you have to perform BH into 6. So, for that you take 6 MOV AL comma 06H or you can directly take AX because you have to multiply only AL is enough sorry, AL comma 06H then MUL BH. So, what will be the operation here? This will be contents of AL into contents of BH. So, contents of BH as it is whatever the data available in BH that will be taken and contents of AL is 6. So, this will store the result in AX. So, after this what will be the contents of AX here? After this operation is 6 into BH. 5 AL is present in a CX register and 6 into BH is present in AX register.

Now, you have to subtract one from the other ok. For that we have to subtract this 6 BH which is there in AX has to be subtracted from CX; CX comma AX. So, here CX contains now, 5 into AL minus 6 into BH ok. Then we have to perform BH by 8 ok. So,

this division by 8 division by any power of 2 can be performed by using simply shift, no need of using the division operation. So, here this is basically we have to shift by 3 bits to the right or left? To the right, ok. For example, if I take say for example, 100 is 4, if I shift this to the right by one bit position what happens this is 010 which is 2 this is equal to 4 by 2.

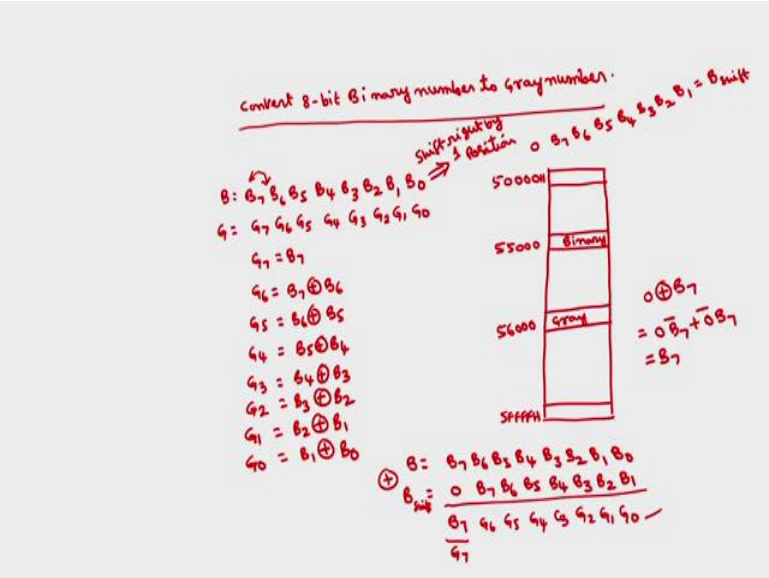
If I shift one more bit 001 this is 1 which is 4 by 4 ok. So, given a number if I shift to the right by one bit position then you are getting half of that value 1 divided by 2. If I shift by one more you will get divided by another 2 which is total 4. So, if I take 4 bit number for example, 1000 which is equal to 8; if I shift to the right by 1 bit position this will be 0100 which is 4, one more bit position 2, one more bit 1. So, after 3 shifts after 3 shifts this value became 1 which is equal to 8 by 8 which is equal to 1. So, this is division by 8 can be performed by just shifting to the right by 3 bit position.

So, in order to shift by 3 bit position, we have to add 0s here this is important. So, in that case you can use MOV CL comma 03H the number of bits to be shifted. If it is single bit shift I can specify in the instruction itself whereas, if you want to shift by more than one bits we have to take into CL by default as I have discussed in the instruction set. Then we have to shift which bits has to be shifted BH contents SHR shift right BH comma CL.

So, this operation we will shift the contents of the BH by CL number of times CL we have 3. So, after this what happens is and the result is stored in BH only. So, BH by 8 will be performed here. This we have to be we have to add this to the previous value then add where is that remaining result CX. So, because this is unsigned numbers I will take MOV BL comma 00 so that this also will become a 16 bit number. Then add with add with CX comma BX then halt. What will be there in CX? The previous value of the CX is 5 AL minus 6 BH, now the value is BH minus 8 so, this entire this value will be the 5 at CX contains now 5 AL minus 6 BH plus BH by 8. So, this is how we can write the program ok.

So, this basically involves multiplications and divisions. But, division we are not going to perform using division instruction because this is a power of 2 we are going to implement by using shift operation. This is program to compute this.

(Refer Slide Time: 19:03)



The next program is conversion of the binary into gray convert 8-bit binary number into gray number. We have to write the program for this conversion.

Let us assume appropriate memory locations say in 55000H binary number is there 8-bit binary number and we store in 56000H the corresponding gray number starting is 50000H ending is 5FFFFH. Now, how to convert the binary number into gray number? So, what is the general procedure if binary number is this is B7 B6 B5 B4 B3 B2 B1 B0; gray number is G7 G6 G5 G4 G3 G2 G1 G0. So, we know that you might have studied in your digital logic; G7 is equal to B7 and G6 can be obtained by exclusive OR exclusive ORing B7 and B6.

Similarly, G5 is exclusive OR between B6 and B5 and so, on. B5 and B4, G3 is B4 B3, G2 is B3 B2, G1 is B2 B1 G0 is B1 B0. So, we have to keep in a loop and we have to do 8 times the exclusive your operation; instead of that the logic that we are going to use here is, but by somehow if I shift this given binary number to the write by 1-bit position ok. So, that this B0 will comes into this MSB position. So, B if I want to shift by right 1 bit position so, what will be the result?

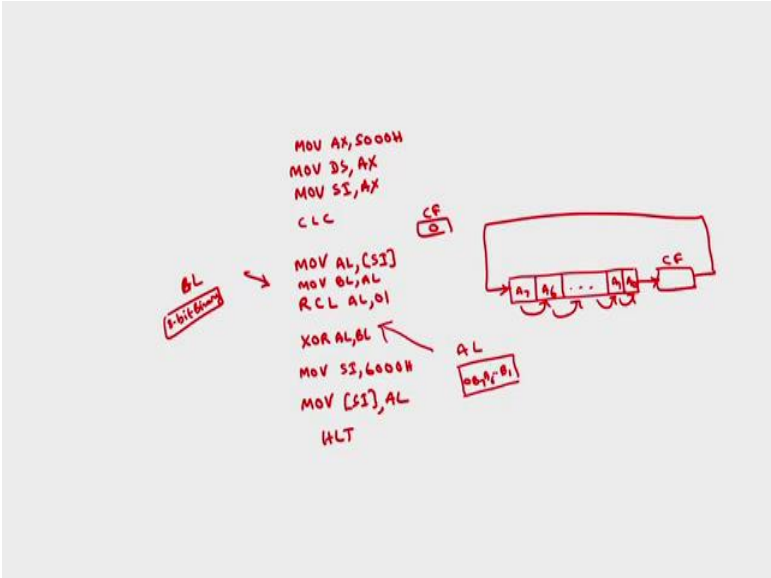
There is one instruction where this MSBs will be loaded with 0 if I reset the carry flag. So, this bit becomes 0 if the carry flag is reset then B7 B6 B5 B4 B3 B2 B1. So, B0 will enter into the carry flag will comes onto the MSB. So, this will be after shifted version;

this number if I exclusive OR with G then what will be this result B itself? You call this one as shifted version as this is equal to some B shifted version.

Now, here I will write here B is B7 B6 B5 B4 B3 B2 B1 B0 and B shift is equal to 0 B7 B6 B5 B4 B3 B2 B1. If I perform the exclusive OR operation between B and shifted version of the B, 1-bit shifted version of the B? What will be result? 0 if a exclusive OR with B7. So, what will be this? B7 itself; because 0 is exclusive OR with B7 this is nothing but 0 into B7 bar plus 0 bar into B7 which is equal to B7 itself. So, this will be B7 itself. So, what will be this bit B6 exclusive or B7 this B7 itself is G7. You can also call this one as B7 or even G7 also both are same.

Then what is a exclusive OR bit B6 and B7 which is G6 and this is G5 this is G4 G3 G2 G1 G0. So, we will get the result. The basic logic here will be you take the given binary number you rotate right by 1-bit position through carry, before that you have to clear the carry, then you exclusive OR the given number and shifted version you will get the gray result the program will be.

(Refer Slide Time: 25:08)



So, the first two instructions are same MOV AX comma 5000 MOV DS comma AX; even you can call MOV SI comma AX also because here offset as well as data segmental are same. Now, assume the location such that both are same. So, instead of MOV the address we can just directly transfer the AX content to SI also. Then CLC this is important carry flag I am going to reset then you take this onto AL because so I want

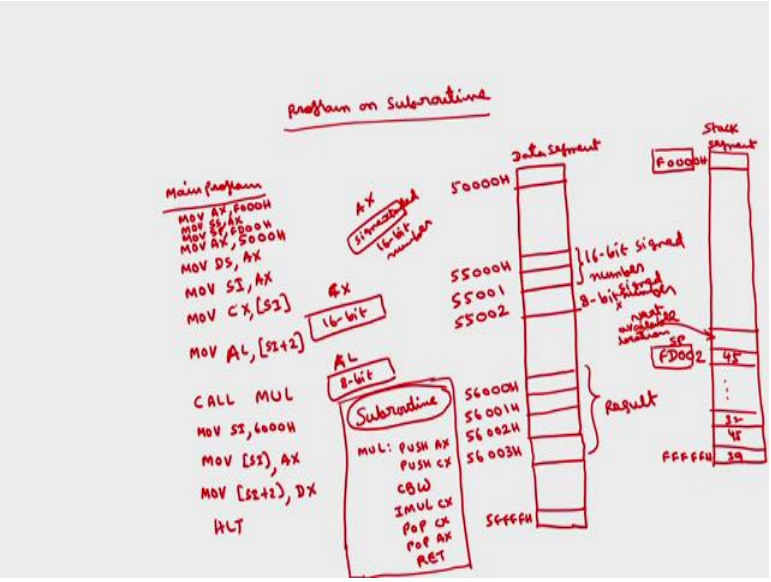
rotate this contents MOV AL comma contents of SI. So, that in AL after this AL contains the give 8-bit binary number.

This we have to rotate to the right by 1-bit position; RCL AL comma 01. So, that what happens is this 8-bits this is A7 A6 so on up to A1 A0 and here carry flag will be there. So, this will rotate here, this will go here, this will go here, this will go here, this will go to carry and carry will go to this one. So, as a result of that what happens is whatever the carry flag status will come to the MSB bit; because here I have cleared the carry. So, after this RCL operation AL before this AL contains 8-bit number after this AL contains 0. So, in the previous example 0 comma B7 to B1 B6 so on up to B1. This is shifted version then we have to just do the exclusive OR operation between these two.

So, this we can take this AL into you store here because if I rotate these the previous AL contents will be lost. So, we can move this contents onto some BL register. So, that BL will contains the original 8-bit register, 8-bit binary number; AL will contain shifted version of this then exclusive OR between AL comma BL. Then whatever the result in AL will be the gray equivalent of given binary number that you have to store MOV SI comma 6000H MOV AL comma contents of SI. So, this original number AL I have taken in to BL or otherwise you can directly take this AL contents and then you can rotate this by 1-bit position ok. So, this is the program for conversion of 8-bit binary number into gray number.

So, I will discuss one more program on a sub routines after that we will go to the next topic.

(Refer Slide Time: 28:49)



Program on subroutine, simply this program will multiply one 8-bit number stored in some location with another 16 bit number. This is data segment same address say 50000H and let us assume that in 55000H one 16 bit number is stored; 50000, 55001 and then 55002 the second 8-bit number is stored.

You multiply these two and you store the result, you store the result in some location you store in four consecutive locations this is the end of the segment from 56000H, 56001H, 56002H, 56003H to store the result; to store the MSB of 16 bit here so, starting from LSB to MSB. So, these are the requirements, but for this you can do it without subroutine also. So, you do everything here in the main program and multiplication you write a subroutine separately you write the subroutine for the multiplication. So, I am writing first main program.

So, in the main program you do the initialization and storing the result, but the actual computation of the multiplication you just perform in a subroutine. So, MOV I will come to some here I will store something MOV AX comma 5000H MOV DS comma AX; then first 16 bit number is present in with location of offset of 5000 you store that offset into SI because here both are same I am taking AX content itself in SI; then the MOV AX comma contents of SI so that AX will contain the 16 bit number. So, they are both are signed numbers.

Now, 8-bit number is there in next location. So, you have to take this on to some other register say BL or CL it is up to you or whatever it may be CL comma contents of SI plus 2. SI was 5000 SI plus 2 is 55002 where the 8-bit sign number is present this I am taking in CL; CL we have 8-bit number. This has to be multiplied but, in the subroutine, I do not want to multiply here. So, I want to multiply in the subroutine. So, for that because this 16 bit by 16 bit you can multiply or now in 8-bit by 8-bit one 16 bit by 1 bit multiplication there is no direct instruction in the 8086. So, we have to do the sign extension of this CL.

So, for that we can store this we can revise this. In fact, this I will store in CX otherwise instead of AX I will store in CX because for the sign extension I want this has to be stored in AL. So, this I will store in CX; in the CX I am storing 16 bit number and this I am storing in AL; here I am storing 8-bit number, then I am calling multiplication subroutine; call multiplication subroutine. So, I will write the multiplication subroutine somewhere.

So, before writing this sub routine so, I will discuss what is required at the main program. So, whenever you want to call a subroutine is you have to the initialize the stack segment and stack register. So, here we have this is data segment, similarly you have to define stack segment also; this is stack segment. So, whenever you want to use this sub routines, we have to define this stack segment also. Let the starting address of stack segment is some say F0000H.

So, ending address will be 64 kilobytes all Fs. This is in fact, the last 64 kilobytes of the entire one megabyte of the memory ok. So, in this we have to define the stack pointer till which point the stack is full I will assume that, up to here say stack is full. So, if this is FD00 some 2. So, up to this stack is full. So, we have some data 32, 45, 39, 45. So, this stack pointer always holds the address of the stack top which is the offset and this is next available location. So this will be stored in stack pointer.

So, with this assumption what you have to do is so, you have to initialize the stack segment register with F000 the upper 16 bit of the starting of that particular segment. So, for that again the same thing so, MOV AX comma the starting address is F000, then MOV on to stack segment register comma AX because we cannot take directly any address onto segment registers. Then you have to initialize the stack pointer with FD00

stack pointer you can take the direct address. MOV stack pointer with for this assumption FD00H, then we can write the call instruction because. So, in the call instruction automatically what happens is the instruction pointer contents will be pushed onto stack for that you have to define the stack pointer and stack segment.

Now, what will be the instruction set multiplication sub-routine? This is MUL subroutine say from here to here MUL subroutine. So, what are the instructions required at the MUL subroutine? Because here in the main program we are using AX register and CX register. So, we have to first push these registers; because here also we have to use the same registers for the operation of the sub-routine also. Then, what is to be done here, just only simply multiplication.

So, the contents of CX has to be multiplied with contents of AL, but the problem is here AL is having only 8 bit. So, we can perform only 16 bit by 16 bit multiplication or 8-bit by 8-bit multiplication, but these two are sizes not same. So, in order to convert this 8-bit AL contents into 16 bit we will write CBW so, that in AL what will be there? The sign bit will be copied. So, then AX will be having this 16 bit number which is sign extension of 8-bit number sign extended 16 bit number.

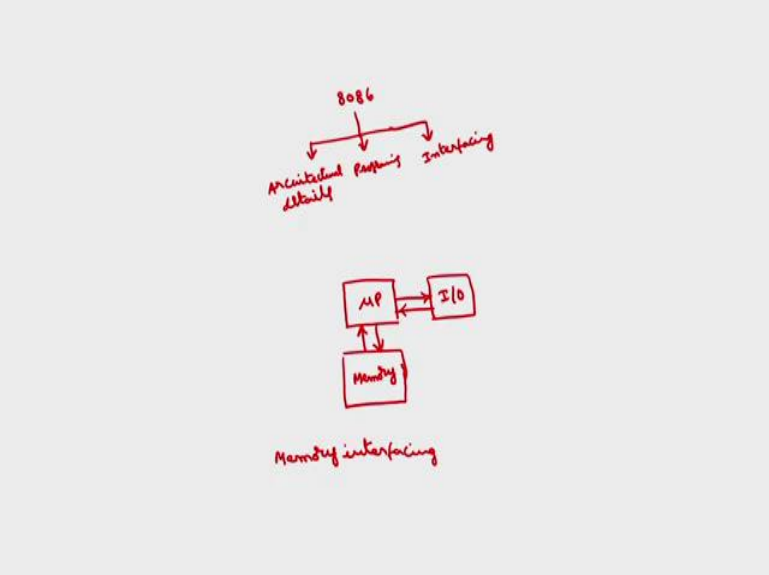
Now, I can perform the multiplication IMUL. So, one of this operand has to be taken in AX in case of 16 bit by 16 bit multiplication; in 8-bit by 8-bit multiplication one of the operand has to be taken into AL. So, here I have taken one of the operand in AX another operand in CX simply we have to write I MUL CX so that the result is stored in DX and AX that you have to store in 56000 to 56003. So, now, the operation is over only. So, this is just to explain this subroutine concept I have written this program, but the task in the subroutine is only just simply multiplication.

Then, before returning to the main program so, whatever the contents is has in pushed now it will be popped it in a reverse manner, POP CX because the stack is last in first out register. So, POP AX then return, these are the set of the instructions which basically multiplies. So, before the multiplication because we want only either 16 bit multiplication or 8-bit multiplication we have to make both the operands same size. So, here one operand was 8-bit so, we converted into 16 bit by using CBW then we return back. So, here only is in the main program what is left now is only storing the result.

So, here I am storing the result MOV SI comma 6000 because I want to store the result in the memory location with the offset starting from 6000 to 6003. So, here the result will be there after I MUL CX in DX and AX. So, I am going to store the LSB result in 56000 MOV SI comma AX MOV SI plus 2 comma DX halt. So, this 16 bit result will be stored in this four locations 56000 to 56003. This is about simple multiplication of 8-bit number by 16 bit number, but using the subroutine concept. So, these are some programs that we have discussed in the last two classes.

Now, I will discuss some more programs while discussing about the IO interfacing if want to interface I mean a input device such as keyboards, 7 segment displays, we need some programming also there ok, we need some hardware as well as some programming ok. So, there we will discuss some more programs, now I will stopp the programs here. I will go to the next topic which is the third part of this course. So, this course is basically the architecture instruction set and the programming we can call as programming itself, for programs we required the instructions ok. And, the third party is interfacing.

(Refer Slide Time: 42:04)



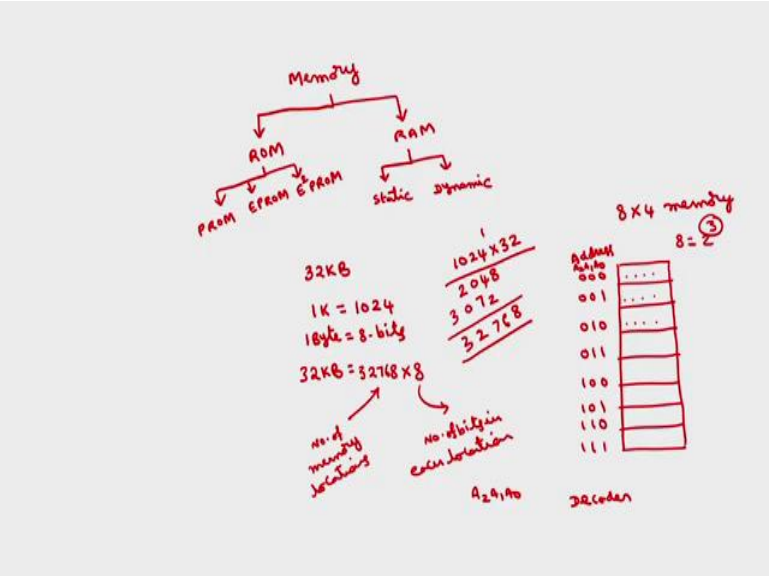
So I have told that this course we have basically three modules architectural details, including pin diagrams – this we have discussed. Then we have discussed the programming – programming in sense we require the instruction set the third part is interfacing. So, we have in the interfacing, we need some hardware as well as in some

cases to interface some IO devices we require some programming also. So, I will discuss some applications also here.

So, the first interfacing as I have told the microcomputer will be having microprocessor then IO memory this is microcontroller. This is the block diagram of a microcomputer. So, in that microprocessor will acts as CPU and then memory and IO. So, we have to connect how to connect the memory and IO to the microprocessor? Without this memory and IO interfacing the microprocessor will not perform any operation. So, microprocessor has to be connected to the memory as well as IO. So, how to connect memory to the microprocessor, how to connect IOs to the microprocessor? This is what is called interfacing how to interface the memory to the microprocessor how to interface IOs to the microprocessor.

First I will discuss the memory interfacing. Before going for the memory interfacing I will review some details of the memory; because the memory is required here memory details are required to interface the memory to the microprocessor.

(Refer Slide Time: 44:18)



So, in memory, basically there are two types of the memories – Read Only Memory Random Access Memory. Again, in this read only memory there are different types like PROM. So, we have here again in the ROM we have programmable read only memory, erasable programmable read only memory, electrically erasable E square PROM. Similarly, in RAM also we have static and dynamic.

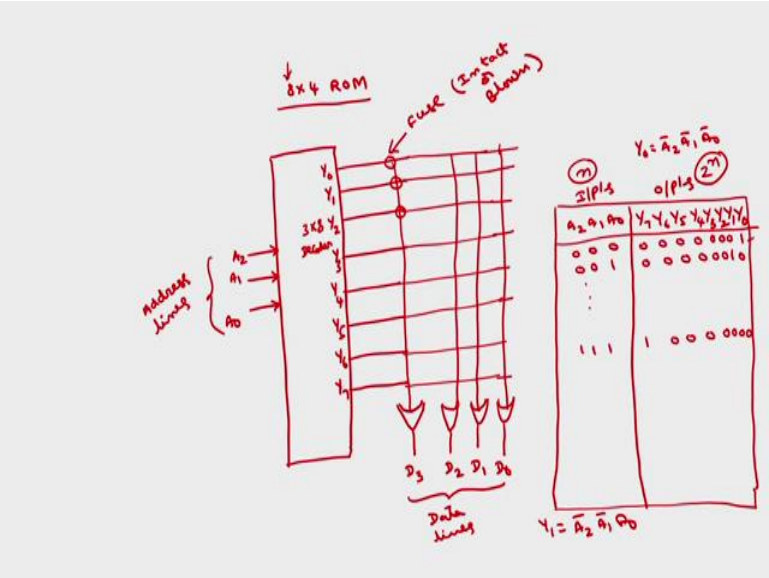
So, before going for the actual this what is ROM and what is RAM? So, how does the memory will be represented? So, normally we will call the memory as some 32 kilo byte. What is the meaning of this 32 kilo bytes? Is or simply if I take this is equivalent to as I have told in the earlier classes 1 kilo in microprocessor terminologies 1024 and 1 byte is 8 bits. So, the meaning of 32 kilo bytes is equal to this is 32 into 1024, 1024 into 32 this is equivalent to 32768. So, this is 32768 is the first one into 8. So, this represents the number of locations number of memory locations and this represents number of bits in each location. This is the meaning of a memory.

So, simply if I take a small memory like say 8 by 4 ROM any memory it can be either ROM or RAM I will call as memory 8 by 4 memory what does it mean? So, it means we have eight locations 1 2 3 4 5 6 7 8 and each location is capable of storing 4-bits of information each bit can be 0 or 1. So, there are total 8 locations. If I number this one as 8 locations, to number the 8 locations how many bits are required? So, this 8 we have to represent as a power of 2 2 cube. So, 3 bits are required to address the each location. So, normally all the memories will be having the number of locations is a power of 2.

So, here I call this one as 000, 001 this we can call as a address. The address of the first location will be 000, then the second one is 001, 010, 011, 100, 101, 110 and 111. So, in case of read only memory we can only read the data at the time of manufacturing itself some data will be stored into the ROM. So, whatever that may be so, the first block here is we have three address lines, but we have 8 locations.

So, how to choose for each combination of this three lines, if we call in general this one as A2 A1 A0. So, we have three address lines A2 A1 A0 total we have 8 combination. For each combination one of the memory location is to be selected this operation will be performed by using a digital circuit called decoder. So, basically the first block of any memory is a decoder. So, here if we want 8 by 4 memory, then we require 3 by 8 decoder.

(Refer Slide Time: 49:19)



If I take this structure of this one 8 by 4 memory, if I take say read only memory directly so, the first block will be 3 to 8 decoder. So, with three inputs A2 A1 A0 and eight outputs Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7. So, what are the Boolean expressions for this outputs which we have studied in digital logic? Basically, Y0 is if I use the positive logic basically we are having Y0 bar Y1 bar because NAND gates are universal gates normally it is NAND gates; otherwise what is the expression for Y0 is complements of all the three A2 bar A1 bar A0 bar.

Basically the decoder black diagram will be so, if I take this A2 A1 A0 the truth table of the decoder and output this is inputs and outputs; truth table represents relation between the inputs and outputs. Y7 Y6 Y5 so on up to Y4 Y3 Y2 Y1 Y0. So, decoder is a combinational circuit with any inputs and 2 raised to the power of n output maximum of 2 raised to the power of n outputs. In some cases they can be less than 2 raised to the power of n outputs also, like incase of BCDs 7 segment display BCD is 4 bits 7 segment display is as the name implies 7 bits so, which is less than 2 raise to the power of 4.

But, in general the decoder will be having n inputs and a maximum of 2 raised to the power of n outputs. For each combination of the inputs one of the output is selected. Here 000 only Y0 will be selected 001 only Y1 is selected. Similarly, for 111 only Y7 is selected. So, if you want to write the Boolean expression for this one Y0 is 1 here. If I take this Y0 column so, only one mean time is there. So, the relation is Y0 is equal to A2

bar A1 bar A0 bar and if I take the Y1 A2 is 0, A1 is 0, A0 is 1. So, this is basically Y1 will be A2 bar A1 bar A0 because all these details are required to interface the memory to the micro processor.

So, inside this decoder we will be having 8; 3-input AND gates and sometimes you can have some enable signal also. So, suppose if you want to disable this entire decoder itself; here there is no option at any time only one of the output is selected. So, for that what I will do is so, I will make one signal such that output will be 0, none of the output will be selected that we can call as enable signal. That I will discuss later. So, we see about basically 3 by 8 decoder.

Now, in order to store 4 bits what I will do is so, I will be having so many OR gates in case of ROM the first 8 represents this is the output of the decoder accordingly it can find out the number of input the first block is a decoder and second block is an array of OR gates. So, the number of OR gates will be equal to the number of bit that is to be stored in each location here we have 8 by 4 means 4 OR gates will be there. So, in each location I want 4 data lines I will call this one as D3, D2, D1, D0; these are address lines, these are data lines. So, any memory will have some address lines and some data lines.

Similarly, microprocessors also having some address line data lines. So, in order to interface we have to basically connect the address lines and data lines together and we have to select that particular memory using some chip select signal; data lines are data bus. Now, whatever the number of outputs of the decoder so, that we have to give as input for this all the OR gates. So, all the OR gates are 8 input OR gates. Even though I have represented a single line this is in fact, all the 8 inputs are connected to this particular line.

Now, depends upon the data that is to be stored in each location. So, here all this will be connected through the fuses. Here we will be having fuses. At each inter section we have fuses. If fuse is intact, then that particular line will be connected to the input of OR gate; if fuse is blown then there will be no connection. So, fuse can be either intact or blown. So, depends upon the data that is to be stored here we will either intact the fuse or will blown the fuse. So, how to store the data in each location that I will discuss in the next class.

Thank you.