Digital Circuits

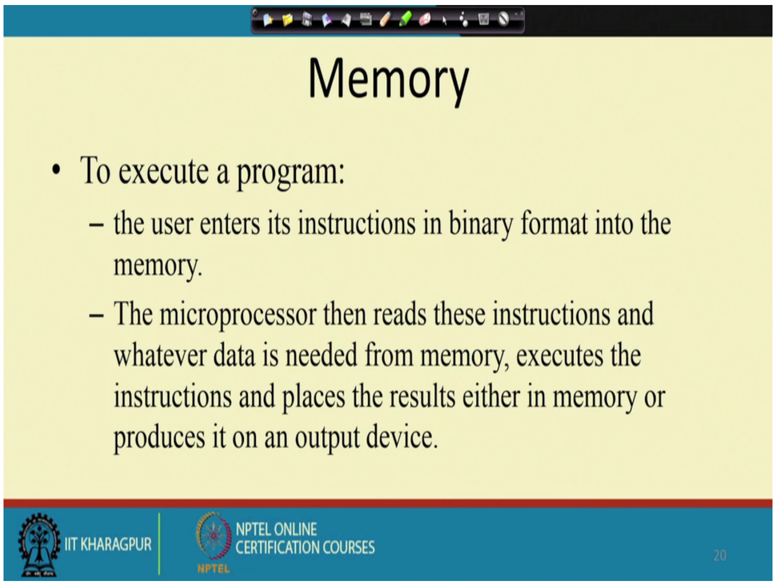
Prof. Shantanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 50
8085 Microprocessor (Contd.)

We have seen that in a computer system or in microprocessor based system, the microprocessor talks to the memory to get instructions to be executed, sometimes, it also gests the data from the memory or from the input output devices.

Now, as far as the user is concerned if the user wants to get some program executed.

(Refer Slide Time: 00:42)



Memory

- To execute a program:
  - the user enters its instructions in binary format into the memory.
  - The microprocessor then reads these instructions and whatever data is needed from memory, executes the instructions and places the results either in memory or produces it on an output device.

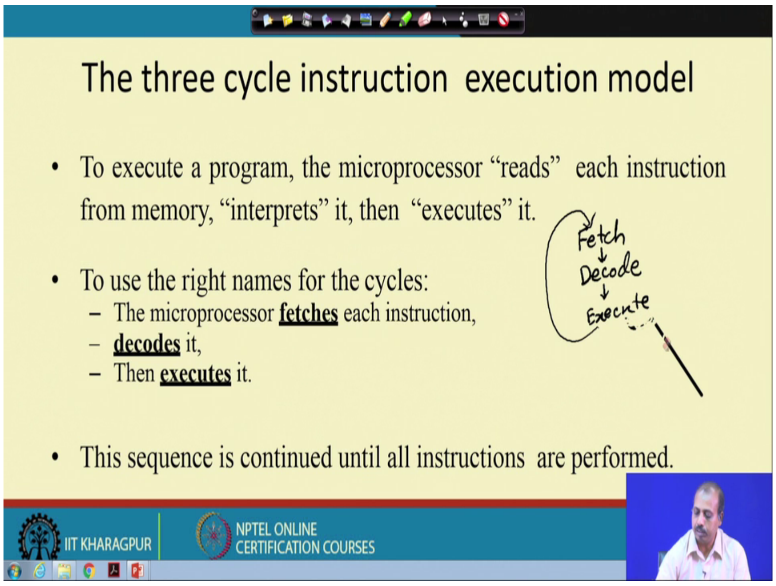
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
20

So, the user has to enter the instructions that we have that the program has to in some binary format into the memory, the memory ultimately stores only binary information. So, the program that we have so that has to be coded into binary and this binary program has to be loaded into memory.

Now, when the microprocessor is reset; so, there is a set address from which it starts in case of 8085 that address happens to be 0. So, it accesses location 0 and reads the first instruction from there executes the first instruction; they it goes to the next instruction. So, that way it continues.

So, as you can understand that any computer system for starting at this 0 should have some monitor program or some operating system program. So, which will in turn transfer the control to the user program and user after the user program is over, then the control will be transferred back to the operating system. That is how the whole system works, but as far as the microprocessor is concerned, it reads instruction from the memory and if the data is needed from memory.

(Refer Slide Time: 02:03)



The three cycle instruction execution model

- To execute a program, the microprocessor "reads" each instruction from memory, "interprets" it, then "executes" it.
- To use the right names for the cycles:
  - The microprocessor **fetches** each instruction,
  - **decodes** it,
  - Then **executes** it.
- This sequence is continued until all instructions are performed.

So, it will read the data also from the memory; execute the instruction and then it will place the result back onto memory or onto the output device as the situation may be. Now, in this execution process, you can understand that there are 3 stages, the first part of it when it read the microprocessor reads the instruction from memory so, that is known as the fetch operation. So, it is fetching the instruction from memory. So, it the processor does not know whether it is an instruction or data or whether it is meaningful or not, but it will interpret that for the first access will give it the instruction. So, so that will be that that is called the fetch operation.

Now, after getting the first word from the memory; so, it may so happen that this processor will understand that this is the processor will understand that it needs some more part of the instruction or may be more data from the memory. So, for that purpose it has to decode the instruction.

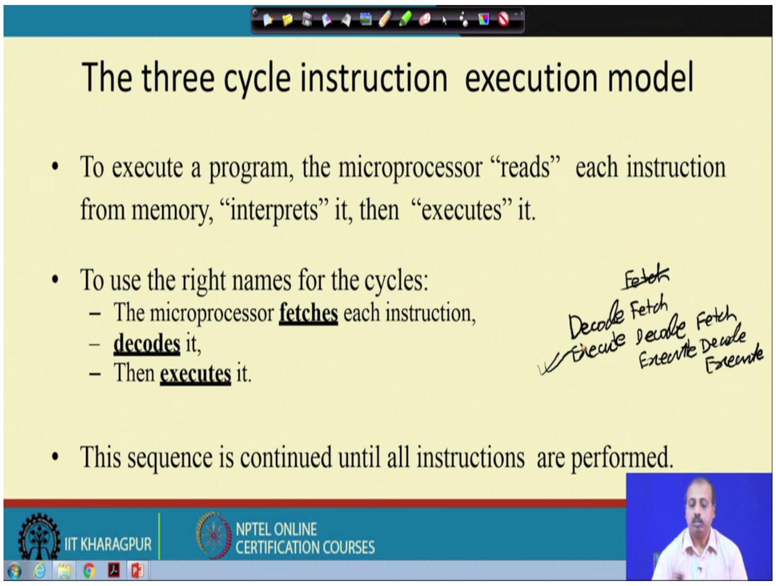
So, after decoding the processor understands what exactly has to be done whether it needs data from memory or not, whether it need some data from some input output port or not I O device, etcetera and then a after getting those data it will execute it in the execution process may be some operation followed by storage of data; so, this entire thing comes as the execution.

So, you can say that the way the processor works is first it performs the fetch operation, first, it perform the fetch operation, then after fetching the instruction the processor enters into a phase which is known as the decode phase. So, from fetch, it comes to the decode phase and after decoding the instruction, it goes into execution of the instruction. So, this is the execute phase.

So, these are the 3 phases that we have in instruction execution fetch decode and execute. So, after the executive over; so, processor is done with the current instruction. So, you should get the next instruction from memory. So, what it does it now goes back to the it now goes back to the previous next fetch operation to get the instruction from the next and to get the next instruction and execute it.

So, this way; this fetch decode execute it forms a cycle. So, if we look into any processor. So, it will have this fetch decode execute cycles ok. So, that the that is the that they are actually in some sense we can say the that they are pipeline. So, fetch followed by decode followed by execute. So, in 8085; so, there is no overlapping of these stages, but if you look into later processors, you will find that there is an overlapping of these stages when the first instruction is fetched. So, no other instruction can be fetched at that time after that the first instruction goes to the decode stage and at that time the next instruction may be fetched.

(Refer Slide Time: 05:07)



The three cycle instruction execution model

- To execute a program, the microprocessor "reads" each instruction from memory, "interprets" it, then "executes" it.
- To use the right names for the cycles:
  - The microprocessor **fetches** each instruction,
  - **decodes** it,
  - Then **executes** it.
- This sequence is continued until all instructions are performed.

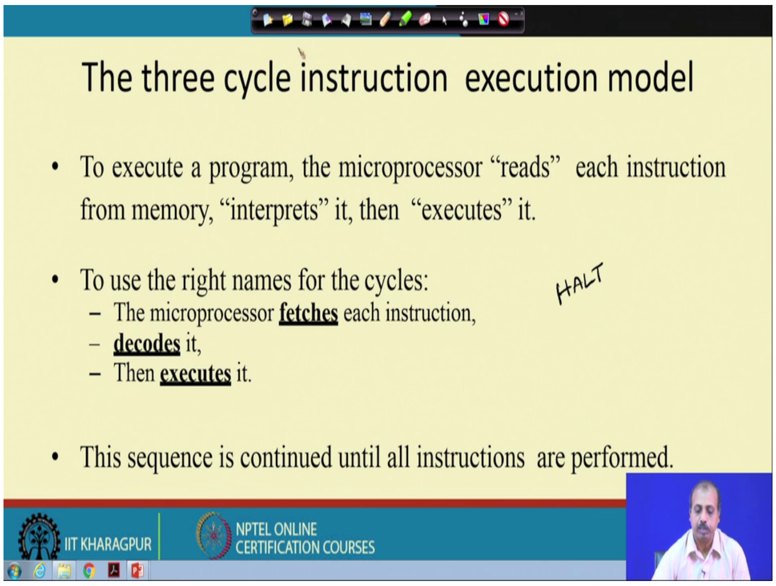
Then when the instruction the first instruction goes into the decode stage the first instruction goes into the decode stage the previously phased instruction. So, so, the when the first instruction the decode stage the next instruction can be in the fetch stage and when this first instruction goes to the execute phase; when it goes to the execute phase the next instruction.

ah Next instruction goes to the decode phase. So, this goes to the decode phase then this instruction goes to the second instruction goes to the execute phase and in the meantime the third instruction when the first you say second instruction was in the decode stage. So, first instruction the third instruction was being fetched, then the third instruction goes into the decode stage, then it goes to the execute stage.

So, that way there is an overlapping between this fetch decode execute. So, if you look into say this particular cycle, the first instruction will be executing second instruction is in decode stage and third instruction is in the fetch stage. So, this is also known as fetch decode execute pipelining it is common in many of the processors.

Now,; so, this sequence of this fetch decode and execute it continues still all instructions are done. So, how do you know all instructions are done like any processor it will have an instruction.

(Refer Slide Time: 06:35)



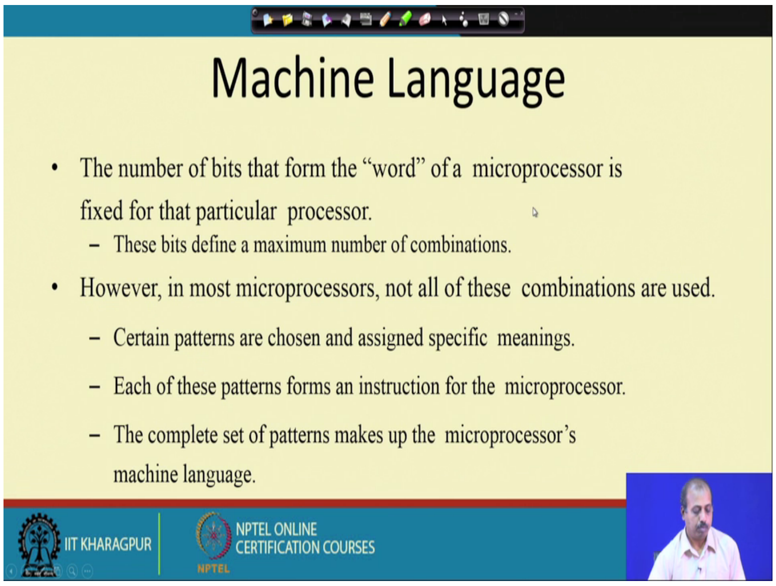
The three cycle instruction execution model

- To execute a program, the microprocessor "reads" each instruction from memory, "interprets" it, then "executes" it.
- To use the right names for the cycles:
  - The microprocessor **fetches** each instruction,
  - **decodes** it,
  - Then **executes** it.
- This sequence is continued until all instructions are performed.

So, which tells the processor to halt; so, there is an instruction like halt. So, when this halt instruction is executed then the processor halts it is executed. So, all the signals are in I can say inactive stage and. So, to take the processor out of this halt stage special mechanism knows a known as interrupt has to be used and when this interrupt is reset, then the processor comes back to wake up mode and it will again start executing from some defined address ok.

So, this is the sequence of operation that happens in the execution of instructions.

(Refer Slide Time: 07:14)



Machine Language

- The number of bits that form the "word" of a microprocessor is fixed for that particular processor.
  - These bits define a maximum number of combinations.
- However, in most microprocessors, not all of these combinations are used.
  - Certain patterns are chosen and assigned specific meanings.
  - Each of these patterns forms an instruction for the microprocessor.
  - The complete set of patterns makes up the microprocessor's machine language.

Next we will see how this how this program can be written like binary program that we have talked about. So, how this can be developed; so, we will introduce that terminology called machine language. So, we are familiar with many programming languages like say C, C++, Java, etcetera, but these languages they are for to some extent human understanding; so, as a human being. So, as a if you read a C program you understand the meaning of that.

But a machine or a processor does not understand a program in that language see it only understands the binary 0s and 1s ok. So, when you interpret the program that you have written in a high level language into a language it is understandable by the machine; so, this is called a machine language. So, in a in case of a microprocessor the number of bits that form the word of a microprocessor is fixed for that particular processor. Word means in one access how many bits it get or when it is doing some operations some arithmetic logic operation. So, what is the minimum number of bits on which it is operating.

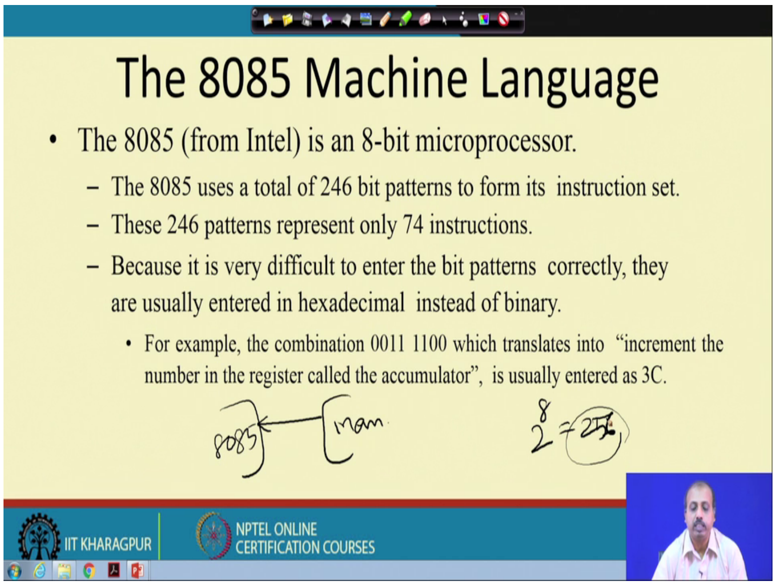
In case of 8085; so, we will say that it is 8 bit processor because it all the operations are 8 bit operations. So, if I say that my word size is say 8 bit; that means, when you get the first instruction when the processor is in fetch stage, it gets the first instruction and then the that 8 bit pattern it will depute what is the instruction. So naturally; so, this; so, with 8 bit, I can have at mot to power 8 that is 256 combination. So, these bits will define a maximum number of combinations that are possible for the machine instructions.

However, in most microprocessors all this not all these combinations are used a 256 is a very large number, but as far as this simple processors like 8085 is concerned; however, for complex processors that that number may not be that significant; however, for complex processors the word size is also large. So, it is 16 bit; so, that way your instructions of the number of possible combinations is also very high.

So, when the processor fetches a word from the memory. So, its type will determine the meaning of the state meaning of that word; so, certain pair. So, out of this all combinations that are possible all combinations are not used and it is up to the designer of the processor to choose certain pattern and assign themselves specific meaning.

Each of this patterns will form an instruction for the processor. So,. suppose I have got 256 patterns; now as a designer of a processor, I see that my processor will be executing 100 different instructions. So, I will be satisfied with 100 different codes. So, rest of the codes are do not care for me because the processor will simply ignore those codes; so, this patterns they will. So, each of these 100 words; so, they are actually one instruction for the microprocessor. So, this complete set of patterns it will make the machine language of the microprocessor. So, this is called the machine language. So, the language it is understood by the machine.

(Refer Slide Time: 10:39)



The 8085 Machine Language

- The 8085 (from Intel) is an 8-bit microprocessor.
  - The 8085 uses a total of 246 bit patterns to form its instruction set.
  - These 246 patterns represent only 74 instructions.
  - Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.
    - For example, the combination 0011 1100 which translates into "increment the number in the register called the accumulator", is usually entered as 3C.

Let us look into 8085 machine language. So, in as we have said that 8085 is an 8 bit microprocessor; so, 8 bit microprocessor as I have said that it is the external data bus side is 8 bit and internally when it is doing the operations. So, all the operations are on 8 bit data; so, with 8 bit. So, we can have 2 to the power 8 that is 256 different combinations are possible ok. So, when this processor if this is 8085 and this is your memory then as I said that if the at the very very beginning, it does a fetch operation and gets 1 byte from the 1 word that is equal to 8 bit from the memory.

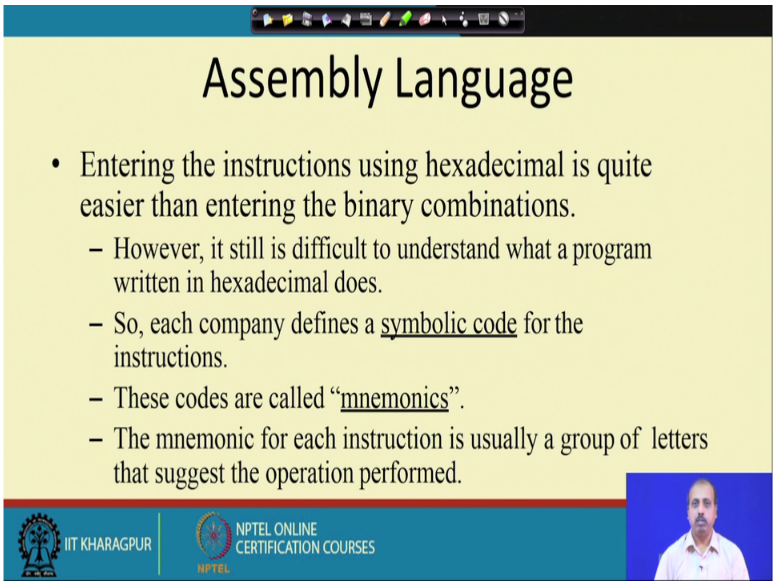
Now, this with is 8 bit I can have 256 different patterns possible, but in case of 8085 out of this 256 patterns only 246 patterns are used. So, remaining 10 patterns are not used by the processor. So, it uses a total of 246 different bit patterns to form a instruction set and these 246 patterns, they represent 74 instructions.

So, 74 types of instructions; so, we will see that within the same instruction they are may be variants. So, that is why for 74 instructions, we will need 246 different bit patterns. So, it is very difficult now as a user of the microprocessor systems. So, if I am asked to write the program in this binary number system the binary pair bit pattern, then it is very cumbersome ok.

So, normally we can use some hexadecimal instead of binary, for example, if the bit pattern that I want to have that I have to enter is like this 0 0 1 1 1 1 0 0 which for in case of 8085 this corresponds to the instruction that implement the number of number in the register called accumulator. So, this 8085 has got a special register accumulator and this particular bit pattern to the processor, it we will mean that it asks for incrementing the accumulator register. So, we will see in detail after for some time.

So, instead of entering the data as 0 0 1 1 1 1 0 0; so, we can break it into hexadecimal digits and this is 3 and this is C; so, 3 C. So, if we have got a hexadecimal keypad; so, we can enter this 2 digits very very easily 3 C ok. So, this way it can be done.

(Refer Slide Time: 13:18)



Assembly Language

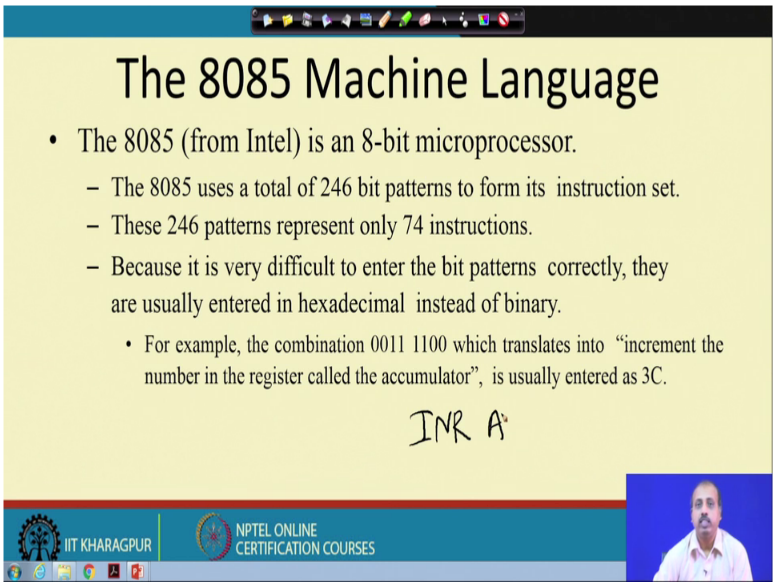
- Entering the instructions using hexadecimal is quite easier than entering the binary combinations.
  - However, it still is difficult to understand what a program written in hexadecimal does.
  - So, each company defines a symbolic code for the instructions.
  - These codes are called "mnemonics".
  - The mnemonic for each instruction is usually a group of letters that suggest the operation performed.

Now, this hexadecimal type of entry so that is also not very much comfortable for the users because then if the. So, the as so, if you if you look into a program that is written in this hexadecimal codes then naturally it looks very crafty. So, as a reader of the program, we it we do not understand the meaning until and unless for each instruction we look into the corresponding meaning from the manual ok. So, the so, we need to bring some amount of readability of the program even in this machine language version ok.

So, that gives rise to another level of language which is known as the assembly language. So, entering instruction in hexadecimal easier than entering in binary combination; however, it is the it is still difficult to understand what a program written in hexadecimal does because the as I said that I need to interpret each and every hexadecimal number that I have entered.

So, what has been done is that different companies they have define a symbolic code for the instruction.

(Refer Slide Time: 14:30)



The 8085 Machine Language

- The 8085 (from Intel) is an 8-bit microprocessor.
  - The 8085 uses a total of 246 bit patterns to form its instruction set.
  - These 246 patterns represent only 74 instructions.
  - Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.
    - For example, the combination 0011 1100 which translates into "increment the number in the register called the accumulator", is usually entered as 3C.

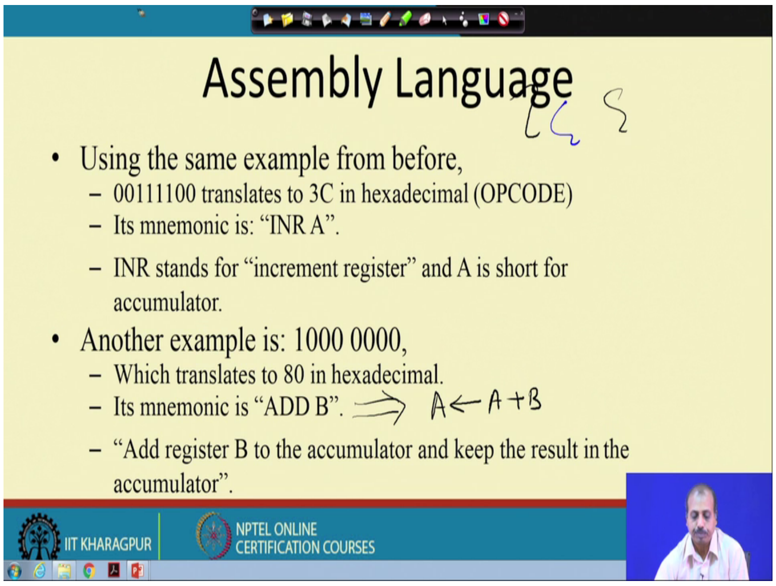
INR A

So, as I said that previously we have seen this instruction that implement the number of a number in the register called accumulator. So, instead of writing this big statement it may be simply implement is implement may be written as say INR. So, it may be simply INR and this accumulator is A. So, this is INR A; so, by this by these testes, we understand that it is asking for implementing the register A.

So, if here you see. So, this is; so, it is easier to say instead of hexadecimal number. So, we can have some symbolic code for the instruction and these codes are called mnemonics. For example, that INR A the INR part is known as the mnemonic and A is the operant on which that mnemonic works. So, as a designer of the processor; so, we can think about different mnemonics that are possible that we will with that we are going to use and using those mnemonics. So, we can with as a user of the system, we will know that those mnemonics and right programs using those mnemonics.

A mnemonic for each instruction is usually a group of letters that suggest the operation to be performed. So, this is a as I said that this is INR. So, it is a combination of in and R almost all the processors will have some mnemonic called mov mov. So, these are quite common. So, we will see as we proceed into the assembly language of 8085.

(Refer Slide Time: 16:00)



Assembly Language

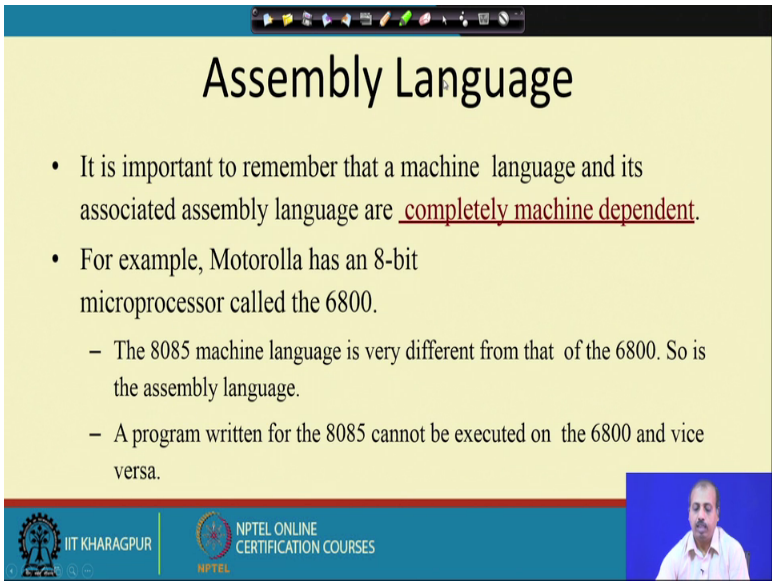
- Using the same example from before,
  - 00111100 translates to 3C in hexadecimal (OPCODE)
  - Its mnemonic is: "INR A".
  - INR stands for "increment register" and A is short for accumulator.
- Another example is: 1000 0000,
  - Which translates to 80 in hexadecimal.
  - Its mnemonic is "ADD B". $\Rightarrow A \leftarrow A + B$
  - "Add register B to the accumulator and keep the result in the accumulator".

So, as I said that this 0 0 1 1 1 1 0 0. So, this is 3 C in hexadecimal. So, this hexadecimal code is called the OPCODE or the operation code. So, this actually tells which operation has to be done; so, that is why it is called OPCODE. So, this bit pattern or this hexadecimal code; so, this is called OPCODE and the mnemonic is INR A where INR is the actual mnemonic part and A is the operand ok. So, A is the short hand for accumulator. So, instead of INR A; so, it may be some other register also. So, that is the incrementing the that particular register. So, INR part is the mnemonic and this A is the operand.

Take another example, suppose we have a bit pattern 1 0 0 0 0 0 0 0. So, this is A 8 0 in hexadecimal and the if you look into the manual of 8085, you will find that this particular code corresponds to the mnemonic add B ok. So, the where add is actually the mnemonic and B is the operand. So, this whole thing is an instruction in the assembly language. So, what it what it does this is an this is an instruction that tells the 8085 microprocessor that the user wants to add the register B to the accumulator and keep the result in the accumulator only.

So, this is what the user wants is something like this the accumulator a should get the content of register b added to its A gets A plus B. So, so, this is the there. So, this why the assembly language programs they actually help in understanding the meaning of the machine language program to the users not to the processors.

(Refer Slide Time: 17:40)



Assembly Language

- It is important to remember that a machine language and its associated assembly language are completely machine dependent.
- For example, Motorolla has an 8-bit microprocessor called the 6800.
  - The 8085 machine language is very different from that of the 6800. So is the assembly language.
  - A program written for the 8085 cannot be executed on the 6800 and vice versa.

So, processor is happy with the machine language code in some binary format, but for human understanding. So, this is in assembly language. So, it is important to remember that a machine language and its associated assembly language are completely machine dependent.

So, if you learn the machine language if you if you learn the assembly language of one processor it is not going to be same as the assembly language or another processor. So, because this is done by the processor designers and they do it at their at their own effort you can say or at their own wish ok. So, it is it is very much unlikely that the two processors will have exactly one exactly same assembly language that there will definitely, we some difference as because the assembly language is very close to the actual hardware. So, since the actual hardware between 2 processors are varying. So, this assembly language is also going to vary.

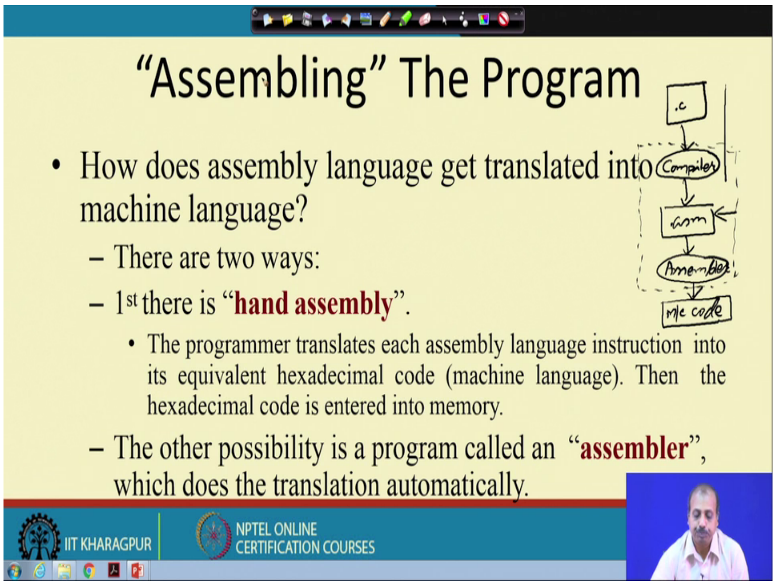
So, that is why; so, normally we learn programming in high level language and then use some tool by which the program is translated onto say machine language or it can also be translated to assembly language. So, if you look into say any C compiler normally this C compiler, they do have an option by which it can tail it to keep the assembly version of the translated program. So, it actually from your high level language program it generates the assembly language program and from the assembly language program, it generates the machine code.

So, you can tell the processor we can tell the compiler to keep the assembly language code available as a separate file. So, so, they; so, as a as a if we are expert in the assembly language of that processor, then we can look into that code and possibly try to optimize that code further.

So, if you look into say Motorollas 8 8 bit processor 6800. So, this is also 8085 is from INTEL and this 6800 is from Motorolla. So, this 8085s assembly language is machine language is very different from that of 6800 and. So, is the assembly language. So, the are different. So, program written for 8085 cannot be executed on 6800 or vice versa. So, that is true.

So, it is so, the target has to be specified like if you are installing any compiler, you must be knowing that it asks for the target processor for which the code has to be generated. So, this is because of this reason that the machine language of 2 different processors are totally different. So, something targeted to one processor will not work with the other processor.

(Refer Slide Time: 20:33)


"Assembling" The Program

- How does assembly language get translated into machine language?
  - There are two ways:
  - 1st there is "**hand assembly**".
    - The programmer translates each assembly language instruction into its equivalent hexadecimal code (machine language). Then the hexadecimal code is entered into memory.
  - The other possibility is a program called an "**assembler**", which does the translation automatically.

Now, as I said that from the assembly language program is for user understanding. So, machine or the processor does not understand it. So, what we have to do is we need to translate this assembly language program into machine language program. So, how do we do it? First we can do an hand assembly. So, we look into the manual of the processor and then for every instruction that we have in the assembly language. So, we just see; what is the corresponding instruction we corresponding machine language code.

So, programmer will translate each assembly language instruction into its equivalent hexadecimal code. And then this hexadecimal code values will be entered by some keypad and ultimately the when it is entered that software which is getting the data from the keyboard will finally, store the corresponding binary pattern into the memory.

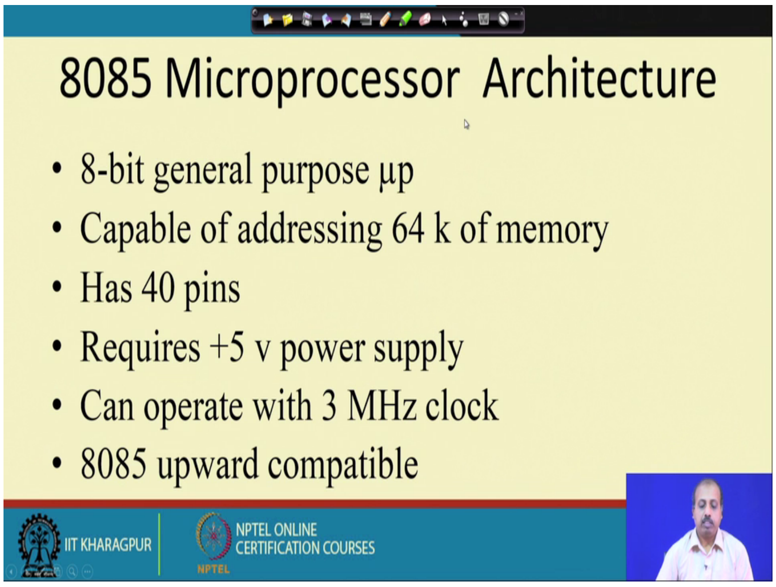
So, this is the hand assembly process for; so, for simple processors like 8085, it is possible that we do an hand assembly and get the program get the assembly language program translated into machine language. So other option is that we have an assembler; so, assembler is a tool that can translate assembly language program into machine language program. So, this assembler can be used that will translate the assembly language code into machine language.

Now, as I was telling; so, if I have got some high level language program say I have got A C program dot C program. So, it passes through a compiler now this compiler it generate some assembly language program. So, dot let us say the extension is dot asm and then this assembly language program it passes through an assembler it passes through an assembler and it generates the machine code. So, this is the machine code.

So, now if you if you can write directly in the assembly language. So, you do not need this part. So, you can start straight way at this point and then you can generate the you can run the assembler and get the machine code. So, normally when we are doing this translation this C compiler. So, it the it integrates the assembly with it. So, we see this whole pla whole thing as a as the compilation process see this whole thing as a compilation process that is why you do not see the assembler code.

However you can you can tell the compiler to keep the assembly file dot asm files for your view viewing and doing some further optimizations. So, so that is the so, that is done by the automated tool called assembler.

(Refer Slide Time: 23:43)


8085 Microprocessor Architecture

- 8-bit general purpose µp
- Capable of addressing 64 k of memory
- Has 40 pins
- Requires +5 v power supply
- Can operate with 3 MHz clock
- 8085 upward compatible

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

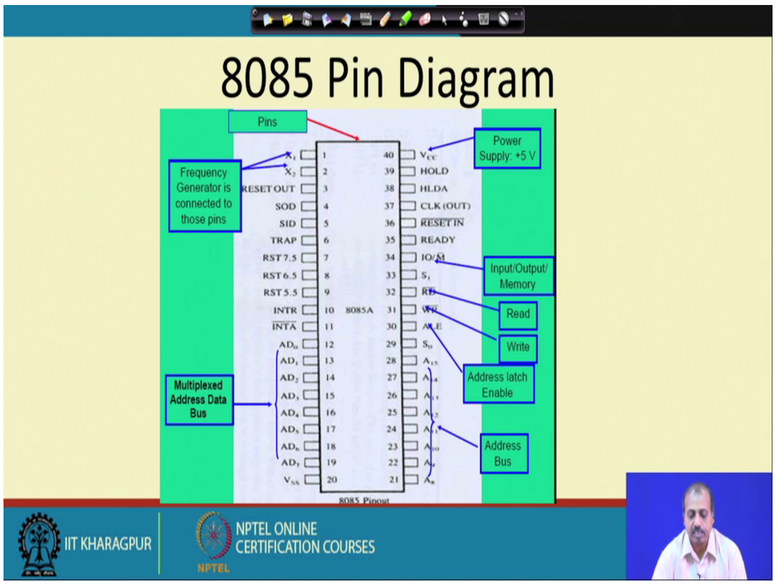
Next so, next we will be looking into this 8085 microprocessor architecture. So, this assemblers are definitely, it is targeted to different processors. So, if you are looking for 8085 will target it to 8085 otherwise you can tell that it is for other processor. So, it will do it accordingly.

Now, in case of 8085 as I said that it is an 8 bit general purpose microprocessor. So, general purpose microprocessor means it is it is not doing some special operation, for example, if you are doing some signal processing tasks. So, there are digital signal processors that are used for that then for graphics processing we have got some special graphics processors. So, like that, but 8085 has a processor. So, it is a general purpose processor. So, general; so, it can do the basic arithmetic operation logic operations like that. So, it is not specific for a particular application.

It is capable of addressing 64 kilobyte of memory. So, this is. So, total memory space that it can address is 64 kilowatt; so, with 8085 system. So, m you can connect memory up to 64 K and if you want to connect it is not that you cannot connect beyond this, but for that purpose. So, we have to take some extra hardware for the connection, but up to 64 K. So, it can be directly interfaced with 8085.

So, as since 8085 comes as an IC chip. So, it has the chip has got forty pins in it the power supply requirement is plus five volt. So, the clock frequency it can operate is with 3 megahertz and it is upward compatible. So, upward compatible means that if you have got some older versions of some processors then this is. So, that that you can just check up that chip and put this chip into the into the system. So, that is there. So, that is upward compatibility is maintained.

(Refer Slide Time: 25:44)



8085 Pin Diagram

So, as far as the 8085 pin diagram is concerned. So, a here it is like this. So, it is a forty pin chip as I have said. So, pin numbers are 1 to 40 out of that. So, pin numbers one and two. So, they are called X 1 and X 2. So, this is a frequency generator is connected to those pins. So, the so, normally what we do is that we connect a crystal between this 1 and 2 and there that crystal frequency will determine the frequency at with this 8085 will work. So, that the frequency generated is connected to this 1 and 2

Then another important client is VCC. So, this is the power supply plus five volt and there will be there is a ground line there is a ground line. So, which will be giving as the ground. So, this VSS this is the ground line.

So, other important pins; if you want to see, then it has got this address bus. So, this is a 8 to a 15. So, this is the higher. So, this is this is a higher order address bus and these pins a D a D is 0 to 7. So, a D 0 to 7. So, that will constitute the lower order address bus. So, as I said that 8085 memory space that it can address is 64 K. So, for 64 K, we need 16 bit bus ok. So, 16 k because 2 power 16 is 64 K. So, 16 bit is obtained like this. So, here I have got a bits a 15 to A 8 and here I have got pins A 7 to A 0 ok. So, A A A 7 to A 7 is this one pin number 19 and A 0 is this one pin 12.

And we have got say 8 it is an 8 bit processor. So, whenever it accesses data from outside world. So, it is in terms of 8 bits. So, that 8 bit data bus is the this D 0 to D 7. So, this is D 0. So, this is D 7. So, this address the 0 and data 0. So, these 2 pins are same similarly address 1 and data 1. So, these 2 pins are same; so, this is called; so, we will we will see that to reduce the number of pin count. So, it has been done like this otherwise what would have happened is that the 16 bit address line. So, that is 16 pin and 8 bit data line that is 8 pin. So, 16 plus 8; 24 pins would have already been done just to create this address and data line. So, we do we will not have much other pins left for other operations ok.

So, to just; so, the designers they have done some multiplexing of this address and data buses. So, that the number of pins required is less. Now there are some additional pins that we will see slowly as you proceed through this course, but some important things like this is the read bar signal. So, read bar signal. So, this is activated when the processor wants to read something from the memory.

Similarly, write bar. So, this is activated when the processor wants to write something onto the memory then there is S 0, S 1. So, this means. So, that gives me the status line. So, there are actually 3 piece the IO M bar S 1 and S 0. So, they tell what the processor is doing at present. So, if you just want to know probe the processor and see; what is it doing. So, this IO M bar S 0 and S 1; so, this will tell you some and this will give you some indication about what the processor is doing now.

Apart from that; so, there are some interrupt lines INTR and this line TRAP, RST, this 5.5; 6.5; 7.5. So, they are actually some facility by which you can interrupt the normal operation of the processor and tell it to do something special ok. So, this is that this is for that interrupt. So, we have got this another important pin that we have is through this SID and SOD. So, this is serial input data and serial output data. So, if you want to transfer transmit some bits to serially from this 8085. So, you can do it where this SID SOD pins ok.

So, as we proceed further, we will be looking into more and more details of this individual pins. So, like say, we will see we will we will be actually viewing these pins in terms of functional groups that is the functions that they are doing and we will try to group them together.