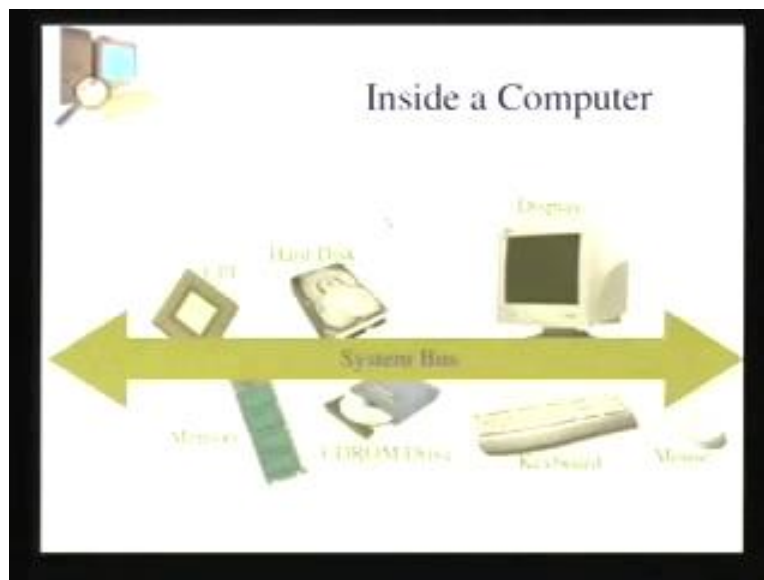


**Introduction to Problem Solving and Programming**  
**Prof. Deepak Gupta**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Kanpur**

**Lecture – 3**

In the previous lecture, we have familiarize that (( )) files and directory, how to create files and how to added the content and so on and so forth. I hope that you have also tried these things of three lectures and now comfortable using the (( )). In this lecture, we will look at the basic components of a computer and try to understand how it is work. At the very low level, a computer consist of a very large number of complex electronics circuit, but we are interested in this lecture, in a more abstract higher level view of the computer.

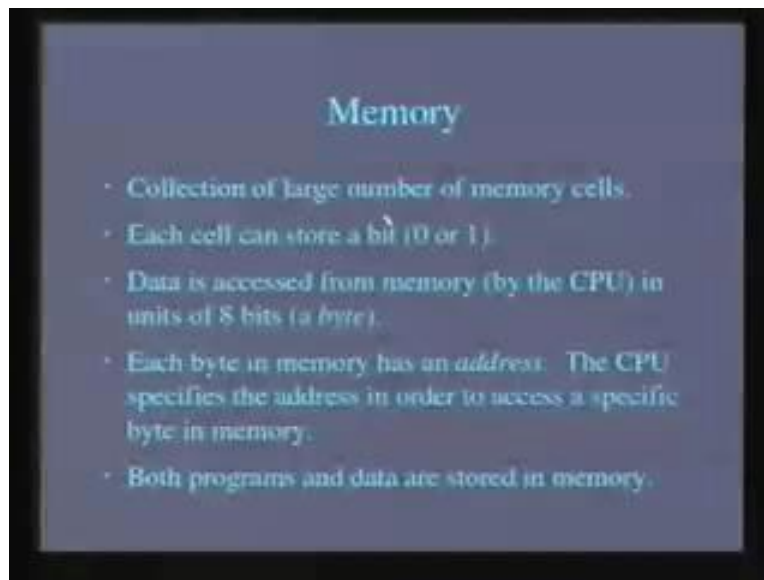
(Refer Slide Time: 01:06)



So let's look at what is the main component of a computer at this high level. The first component is a system bus, which connect all the other medium components of the computer. You can think of this bus as ah the nervous system of the computer which transmit signal from one place to the other. The two most important components are the c p u and the memory. The CPU c p u stands for Central Processing Unit, and it is disk circuit execute and understand all programs at the computer execute. The memory stores all data that is needed to run this program. We will look at some details of the memory and the CPU ah shortly.

Apart from the CPU and memory, the computer also has a number of input, output devices. Some of which we have already come across in the previous lecture, for example, we already familiar with the keyboard, the mouse and the monitor. Here are to more I O devices namely the hard disk and the CD ROM drive, both of these are mass storage media that is we can store large volumes of data for a long time. These are in contrast with memory, the memory is ah much smaller in size as compared to the hard disk and CD ROM drive, but is much quicker to access for the central processing unit. Also and important different between memory and hard disk and CD ROM drive is that the data in memory is volatile. What it means is that if the power switched off, the data is (( )). Whereas the data stored in the hard disk and the CD ROM drive remains even if the powers to the computer is switched off. So all the files and directory is that we have talked about in the last lecture are actually stored on the hard disk and the CD ROM drive.

(Refer Slide Time: 02:57)

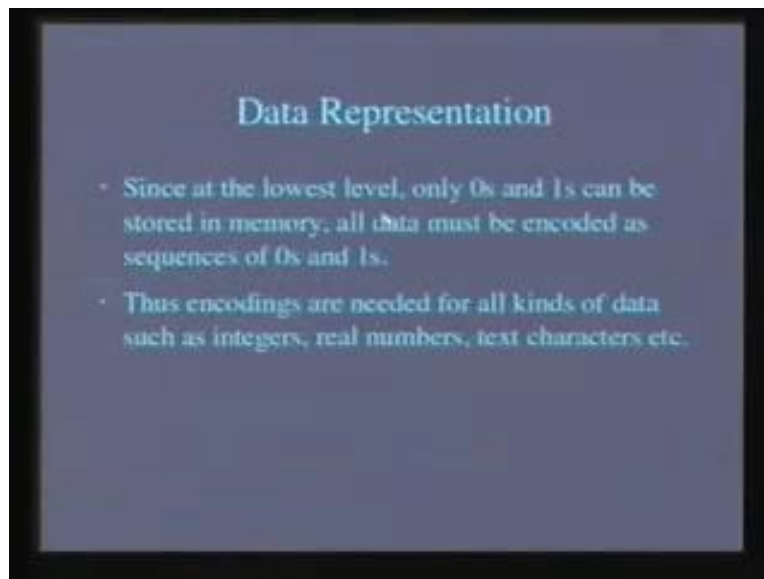


Let's first discuss the memory in some details. Memory is essentially a correct collection of large number of memory cells and each cell can store what is known as a bit. A bit is a binary digit which means that the value is either 0 or 1, and the memory of the computer is nothing but, a collection of large number of memory cells, each of which can store just one bit that is either 0 or 1. And the reason for this storing bit is that we current electronic technology is transistor gate, and transistor can be in two state either on or off. And the two states essentially represent the number 0 or 1.

For efficiency, data usually accessed from the memory in terms of bytes that is group of eight bits, which is known as a byte is accessed at one time. Data may also be accessed from memory in larger units for example two bytes or four bytes or some computers even 8 bytes. Now you can see that the memory consists of a large number of bytes, so for example, might be heard the term there is computer has 256 mega bytes of memory. What that means is that in this computer the number of bytes in the memory is 256 into 2 to the power 20, roughly taking this is 256 million bytes.

So when the CPU need to access a particular byte of data from the memory, it has to specify to the memory circuit, somehow which byte out of this 256 million bytes it wants to access; and for this reason, every bytes has a distinct address. So the address is of this byte who starts from 0, 1, 2 and go up to 256 million or so. It is important to note that both programs as well as the data for the program are stored in memory.

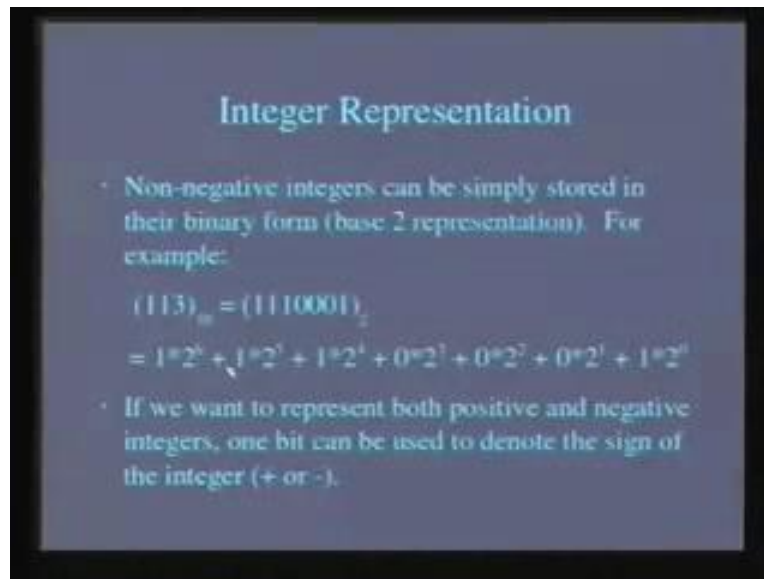
(Refer Slide Time: 04:52)



As you can see that since at the lowest level we can only store zeros and ones in memory, it is necessary that all the kinds of data that we are interested in dealing with or represented as sequences of set zero and one. So we need encoding all kinds of data such as integers, real numbers, characters and so on and so forth. In this lecture, we only look at ah coding use for

integers, and we will not talk about encoding use for real numbers and text characters and so on. We will talk about them in subsequent lecture.

(Refer Slide Time: 05:31)



### Integer Representation

- Non-negative integers can be simply stored in their binary form (base 2 representation). For example:  
 $(113)_{10} = (1110001)_2$   
 $= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
- If we want to represent both positive and negative integers, one bit can be used to denote the sign of the integer (+ or -).

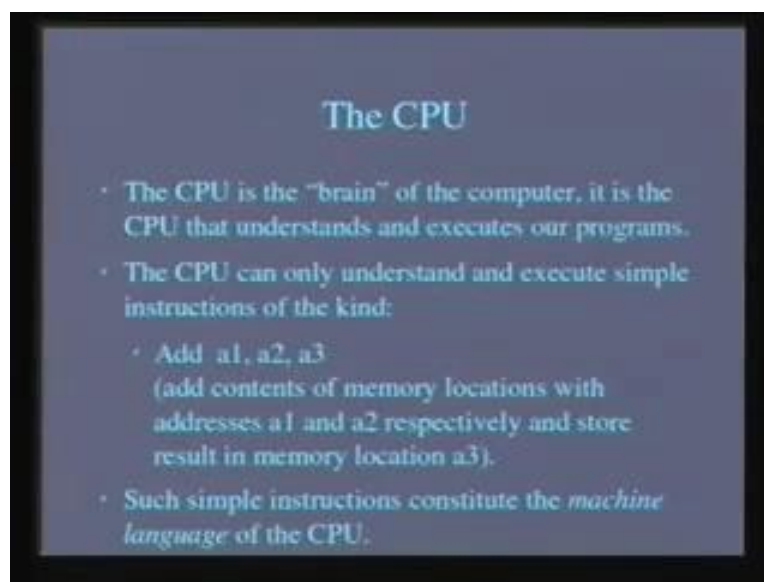
Let's look at how integers are represented inside the computer. So we are dealing only with non-negative integers. As you know, any integer can be converted to its binary representation, that is the base two representation. With them becomes just the sequence of zeros and 1. For example, hundred thirteen, decimal number 113, when represented in binary is 1 1 10 0 0 1. And you can verify this, because as you can see this binary in decimal is equivalent to 1 into 2 to the power 6 plus 1 into 2 to the power 5 plus etcetera up to 1 into 2 to the power 0. This is what the binary system, is a positional system. And this entire expression, evaluate the number hundred thirteen.

But what if we want to represent both the positive as well as negative numbers, one possible way is that out of the number of bits available to us, we use one of the bits to store or represent the sign of the number, that is plus or minus. So for example, we can have the convention that the left most bit is 1 that denotes the negative number; and if it 0, it denotes positive number and rest of the bits give the magnitude of the number. Given this information, you can observe some facts, if we are using eight bits to represent an integer and let us say we are representing only non-negative integers. Then the largest integer that we can so would be 255. Why is that because the largest binary number in eight bits, we can have is 11111111. And which in decimal is equal to

255. And if you are storing both positive and negative numbers, then we have to use one bit for the sign, which means you have only seven bits for the magnitude.

And in this case, maximum positive number that will be able to store will be 127, which is two to the power seven minus one. And as you can see, if you just choose 1 byte for storing an integer, we cannot store a very large range of integer. And therefore, actually the number of bytes are actually used to store a single integer. Typically most computer would use a group of conjugative four bytes in memory to store an integer. So 4 bytes means 32 bits because each byte is 8 bits, which means that in storing both positive and negative numbers, one bit is used for magnitude and which means in turn that the largest positive number that we can store is two to the power 31 minus 1.

(Refer Slide Time: 08:26)

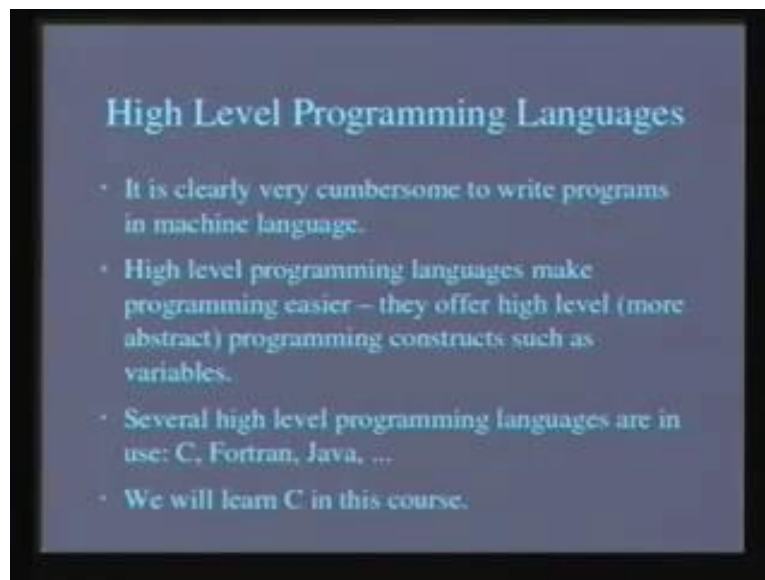


Let's now come to the CPU. The CPU as I said the brain of the computer, it is the component that understands and executes our programs. However, it can only understand and execute only very simple instructions of the kind, add a one, a two, a three. In this example, what this instruction represents is an instruction to the computer to add the content of the memory location, whose addresses are even (( )) respectively. Add these two values, and store the result in the third memory location, whose address is a three. So instructions of this kind, (( )) are the machine language of the CPU and the CPU can only understand and execute our program which are written in this

machine language. So you can imagine, writing reasonably complicated program in such a language, is an extremely conversion part.

For example, try to imagine what is the program for computing the factorial in ah machine language would look like. We would have to worry about what addresses to assigned to various variable and we would not be able to such as use and assignments and so on and so forth. And therefore, we require some tools that the task of programming is somewhat easier.

(Refer Slide Time: 09:55)

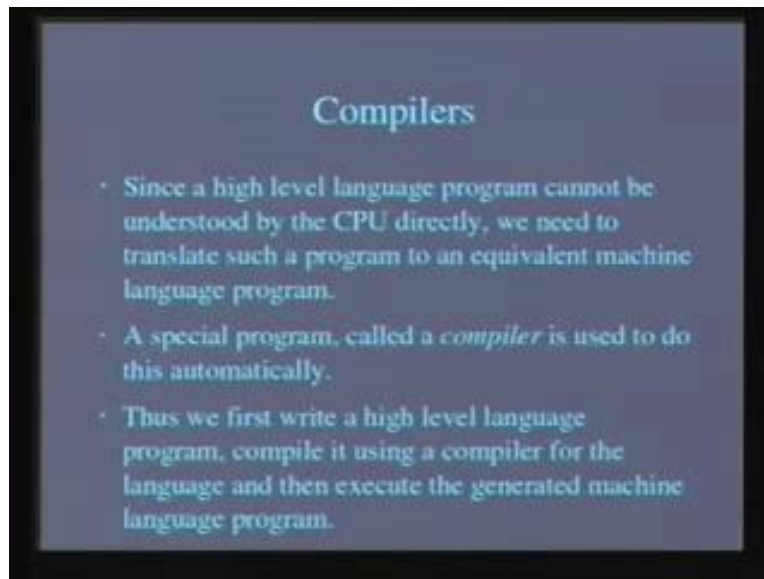


And that is role of what are known as high level programming languages. High level programming languages make programs easier, because it offers higher or more abstract programming concept such as variables. We have already seen what this variables are, there are other high level obstruct, construct the programming languages offered that will see in this course. There are large number of high level programming languages which have been used over a number of years, for example, C, Fortran, Java and so on and so forth. In this particular course, we will learn the programming language C, but we know that the focus of this course is on learning the concept of programming.

In type programming language is only a vehicle is actually to implement concept in practice. So even though we are learning C in this course, you might actually be required to write a program later on in your professional carrier, in other programming languages possibly Fortran or Java.

But having to learn the concept of programming, you will find that it will not be very difficult task to (( )) from C to some other programming languages.

(Refer Slide Time: 11:20)



Now the problem is that the programs determine the high level programming languages cannot be directly understood by the machine, and therefore we need some tools using which we actually run this program. And that's tools for compiler come in. So a compiler essentially translates, a program in a high level language such as C to an equivalent machine language program for a particular kind of a CPU. So the compiler as you can see is (( )) to both the high level language translate from, and the kind of CPU the machine language translate the program too. So this program is called a compiler automatically performs the translation from the high level language to the machine language of the computer.

And therefore, when we write a program, we write it a high level language, in this course, we will write all are in program C, then we will compile it using the compilers for this language. In this course, we will C compiler. And then we will execute the generated machine language program. Compiler will automatically translate the C program into the machine language program for our computer. And this machine language program that we will be able to execute directly on the computer, and which would have same effect that what we intended in a C program.

So let's now see what the factorial program written in C looks like. So I have already written this program, and let's identify this (( )) the C program for computing factorial. And this is what it looks like. In this lecture, we will not go into details of the C syntax, but you can still note that this program in C, looks very similar to the algorithm that we had written in formally. So here we are saying that while n is greater than one, we have to repeat these two statements. Note that the assignment operation in C is denoted by an equal sign, so this particular statement, results is equal to results are n, so actually the sign result are n, equal symbol here denotes not equality, but assignment.

So therefore, saying that the results are equal to result char n that makes n, doesn't mean that n is necessarily one. And similarly in the next line, we are not stating or (( )) that n is equal to n minus one, which is of course a term we are saying that the new value of n becomes the old value of n minus one. Here we are initializing results to one. Here we are reading the value of n. We will see the details of all these statements later on in this course; and here, we are printing the value of result.

Let us now actually compile this program, and try to run it. To compile this program, we need to use the C compiler, and the command for the C compiler is `c c`. Choose the arguments that I have given to the `c c` commands are first of all `minus o`; `minus o` says that the equivalent machine language program which is generated for this C program. So go in the file, which is named after `minus o`. So what this is saying is that the generated machine language program for computing factorial should be stored in a file called `fact`. And `fact dot c` is the name of the file, which is contain the C program for computing the factorial. It is the convention that the names of all files containing C program and `n dot c`. So let's this command by written in C.

We didn't get any error messages, which means that the compiler accepted our program. And now you can check that the file called `fact` has been generated. And this file actually contains the machine language program, which is equivalent to the `c` program that we wrote for computing factorial. Let's now execute this program. So since this is in the current directory, the relative path of this file is `dot slash fact`. And execute this program, I just need to give a command `dot slash fact`. And when I get enter, actually nothing happen, and this is so because the program is waiting for a (( )) number, whom factorial is to be computed. Let's enter some number, let us say



5, enter the return key, and the result which is 120 is printed. Let's try with some more numbers. This is the factorial of fourteen.

Let's said to compute the factorial of 20 and you see that the output is an negative number, which is obviously wrong, and this is happen because the factorial of 20 is too large we stored an integer. In this machine, the representation of an integer using four bytes that means 32 bits and since both positive and negative integers are represented that means largest positive number, we can represent in this notation is  $2^{31} - 1$ . And the factorial of 20 has actually happen much larger than this number, that is  $2^{31} - 1$ . And therefore, we are getting some concentric results.

In this lecture, we have looked at the basic components of a computer, and how to interacts with each other. We also saw the notions of machine language and high level languages and the role of compiler. And finally we saw first C program, and we saw how to compile and execute this program. From the next lecture onwards, we start looking at the C language in more details, and start developing program which are somewhat more complex than the simple C program that we are seen so far.