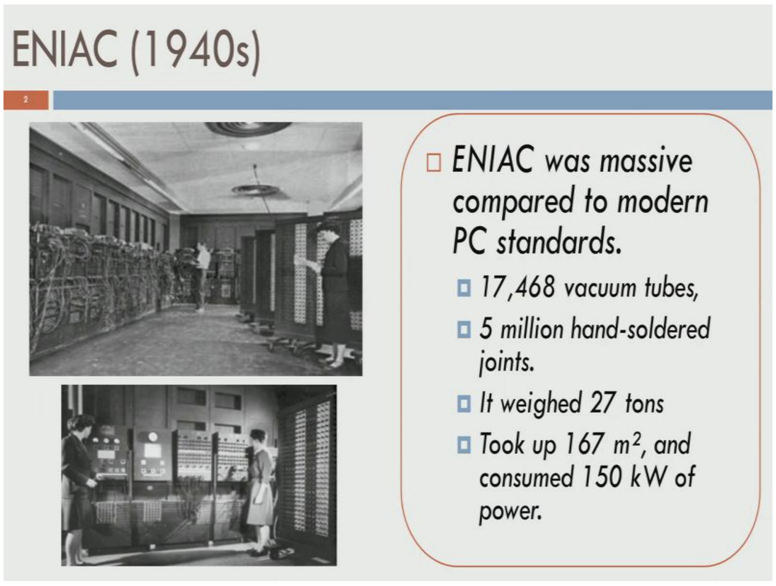
Programming, Data Structures and Algorithms

Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute Technology, Madras

Module – 01
Lecture – 01
Introduction to Computers

Welcome to this online course on programming. I am Shankar Balachandran from the Computer Science Engineering Department at IIT, Madras. This course is designed in such a way that it can be spread across 5 weeks and you are going to have about 2 lectures every week. So, total of 10 lectures and this course is also designed in such a way that you can take this in 15 minutes chunks called modules and you would be given a small practice test, right after every module. So, let us jump right into the course.

(Refer Slide Time: 00:50)



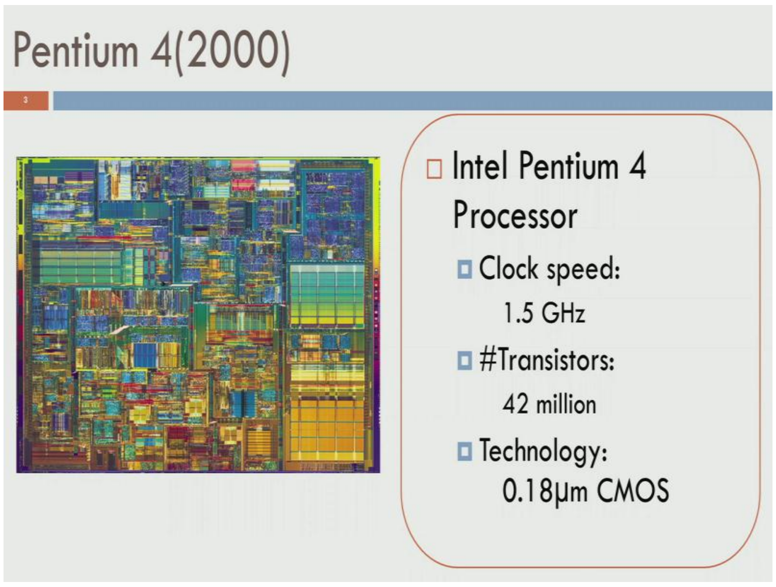
ENIAC (1940s)

2

- ENIAC was massive compared to modern PC standards.
  - 17,468 vacuum tubes,
  - 5 million hand-soldered joints.
  - It weighed 27 tons
  - Took up 167 m^2, and consumed 150 kW of power.

So, before getting into programming we need to understand little bit about what computers are about. What you are seeing in this picture here is ENIAC the, so called first digital computer, which was built in the 1940s. And you can see how huge it is, it is almost like a house in size. It was massive compared to the modern personal computer standards. By no means ENIAC can be called the personal computer. It had 17000 vacuum tubes, 5 million hand solder joints, it weighed quite a bit and it consumed 150 kilo watts of power; no way this would be a personal computer.

(Refer Slide Time: 01:31)

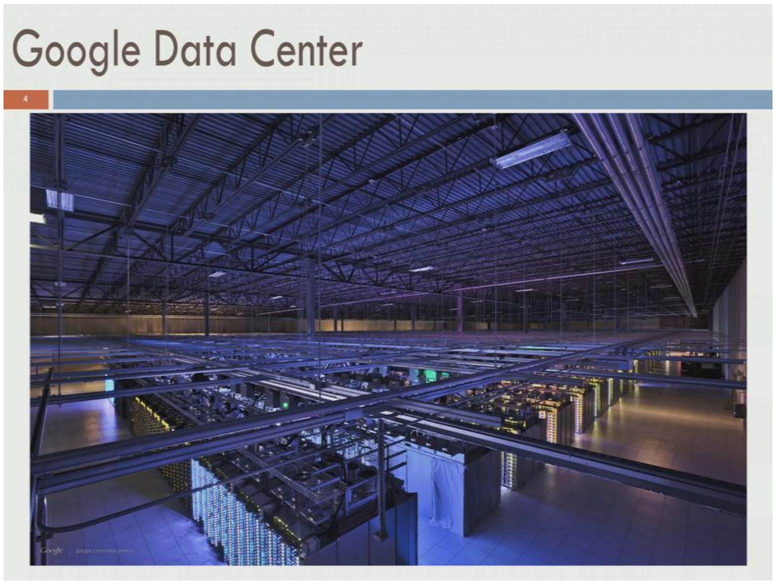


Pentium 4(2000)

- Intel Pentium 4 Processor
  - Clock speed: 1.5 GHz
  - #Transistors: 42 million
  - Technology: 0.18µm CMOS

However, June 2, 2000 what you are seeing on this picture is a micro photograph of Pentium 4 and it was designed in the year 2000 and deployed in the market, it could run at 1.5 gigahertz. That means, it can do 1.5 billion operations per second, it had 42 million transistors as opposed to this puny ENIAC. And it was build in this technology called 0.18 micron technology, which means the transistors and the gates that were used to design as small as 0.18 micron. So, in the 40 or 50 years computers have moved quite far away from the first notion of digital computer.

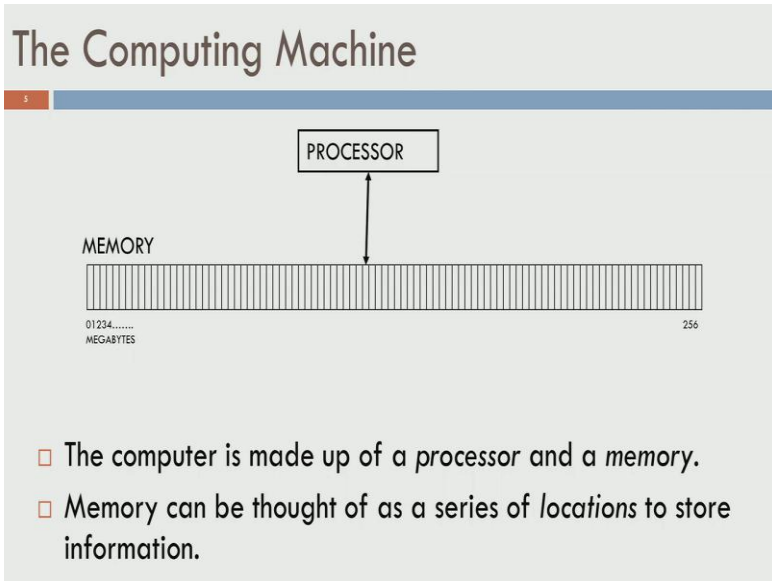
(Refer Slide Time: 02:15)



Google Data Center

In this picture you can see Google's data center. It is one of the data centers that they have across the world. And what you are seeing is racks and racks of machines, crunching data, running algorithms and running programs, various kinds of software that we use on a daily basis. And this is just one of the data centers and this requires enormous organization of the equipment, power systems, cooling systems and so forth.

(Refer Slide Time: 02:49)



The Computing Machine

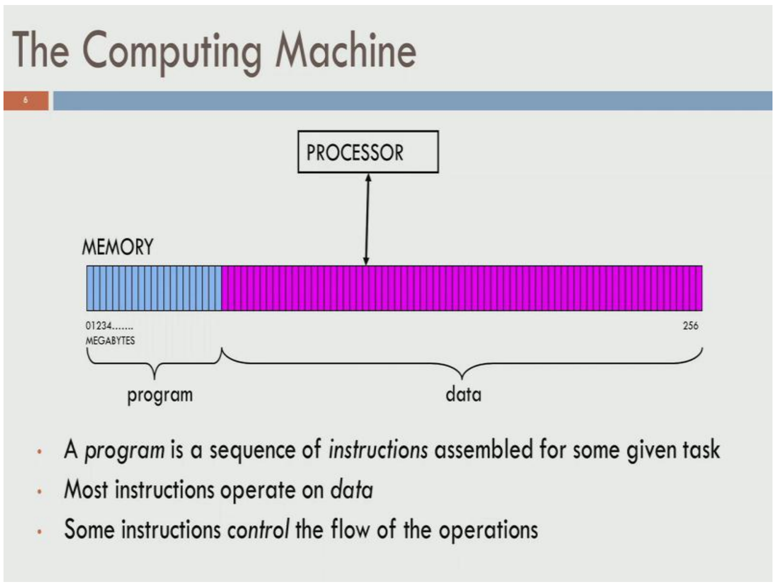
5
PROCESSOR
MEMORY
01234......
MEGABYTES
256

- The computer is made up of a *processor* and a *memory*.
- Memory can be thought of as a series of *locations* to store information.

But, all of this is basically a computing machine, all we see ENIAC, Pentium 4 all these data centers are all build of what are called computing machines and the basic abstraction is what we see here. We have a processor and we have memory and this is what makes any computer, you have processor and memory. You can think of the memory as a series of locations to store information.

Let us say for example, we have 256 megabytes of RAM and you too would be laid out in some order and you can address them as location 0, location 1 so, on up to 256 megabytes, just like how you houses would be numbered in a, if they were all lined up in a line and processor is the heart or the brain of the computing system.

(Refer Slide Time: 03:43)

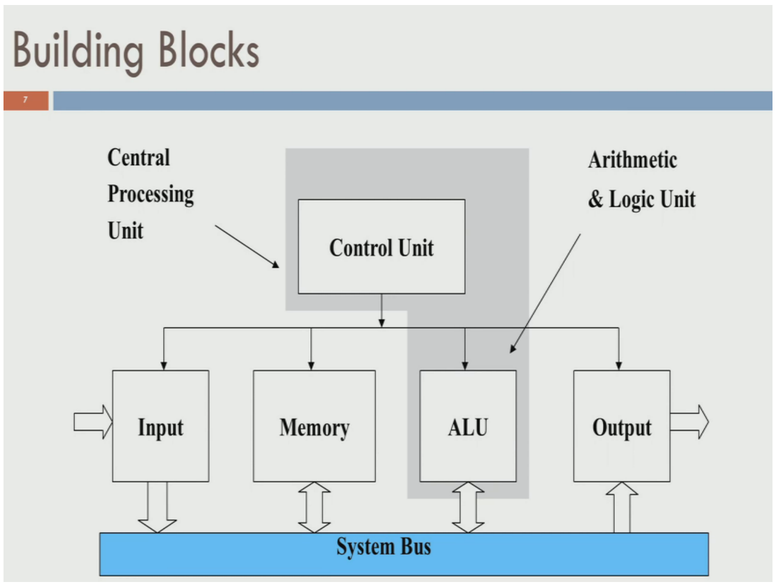

The Computing Machine

- A *program* is a sequence of *instructions* assembled for some given task
- Most instructions operate on *data*
- Some instructions *control* the flow of the operations

So, typically memory is divided into two portions, there is some portion dedicated for programs and some portion dedicated for data. A program is essentially a sequence of instructions assembled to do some task. So, it could be again it could be a piece of software that you write for this course, it could be a search engine, it could be browser, it could be anything. And most of these instructions actually operate on data and the data is something that you store in the memory as well.

There are instructions which could also control the flow of operations, it is not that all the programs have what is called a straight line sequence, they do task a, task b, task c and so on up to end. Based on the conditions that come through some branch operations could happen and because of which control could change. We wiill see these in more details later anyway.

(Refer Slide Time: 04:43)

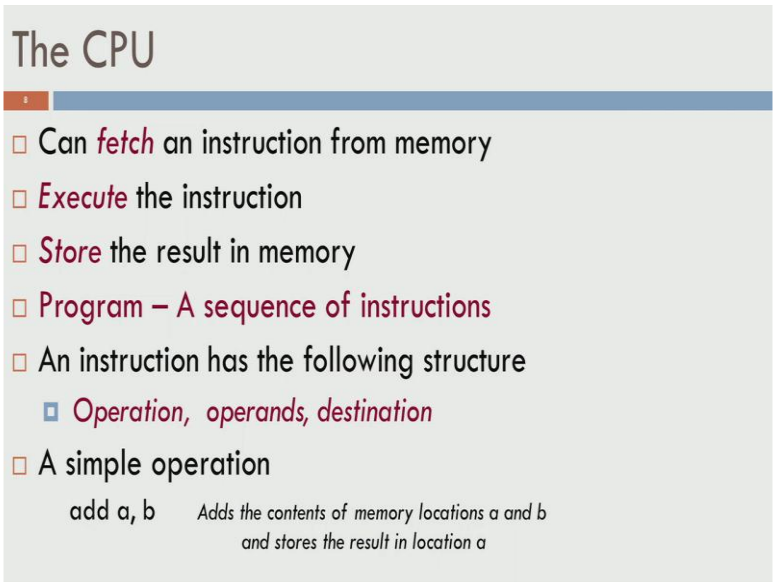


Building Blocks

But, the basic set up behind a processor is given in this picture here. We have an input system - the input system could be a key board, a mouse or any other device; you have an output system - this could be a monitor, some gears that are shifting, it depends on the computer that you are building. And most systems have a reasonable amount of memory, nowadays you probably have 2 gigabytes or 4 gigabytes of RAM on your desktops and laptops.

And then there is this central processing unit. The central processing unit consists of two things: one is call the control unit, the other one is call the ALU or the Arithmetic and Logic Unit. Arithmetic and Logic unit is, it consists of various circuitry; it can do things like, additions, subtractions, comparisons and so on. And control unit in some sense is the over all master. So, it controls what happens in each of these units and how data gets processed in each of these units and when data moves in, when data moves out and so on.

(Refer Slide Time: 05:49)



The CPU

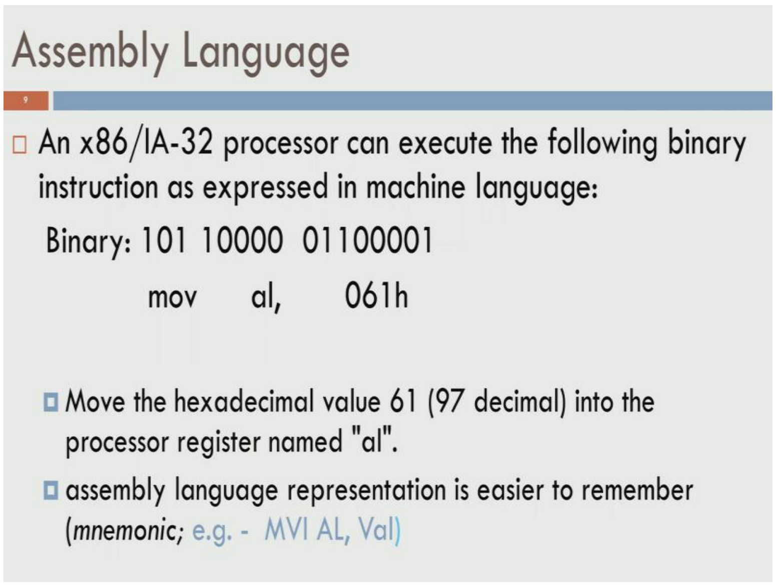
- Can *fetch* an instruction from memory
- *Execute* the instruction
- *Store* the result in memory
- Program – A sequence of instructions
- An instruction has the following structure
  - *Operation, operands, destination*
- A simple operation

  add a, b *Adds the contents of memory locations a and b and stores the result in location a*

So, let us look at the basic operations of a CPU. CPU can fetch an instruction from memory. It can execute the instruction based on whatever instruction is given to it. It can actually execute it. This could be addition, subtraction, multiplication, comparison, it could be anything. It can also store the result back in memory. So, when you write a program it is going to be translated into sequence of instructions.

And a basic machine instruction would have this following setup. You have an operation, you have all the operands or the data on which the operation is going to be done. And where is the result going to be stored, also call the destination. A simple operation of could be this kind add a comma b, it adds the contents of memory locations a and b and it could be storing the result back in a itself.

(Refer Slide Time: 06:41)


Assembly Language

- An x86/IA-32 processor can execute the following binary instruction as expressed in machine language:

  Binary: 101 10000 01100001

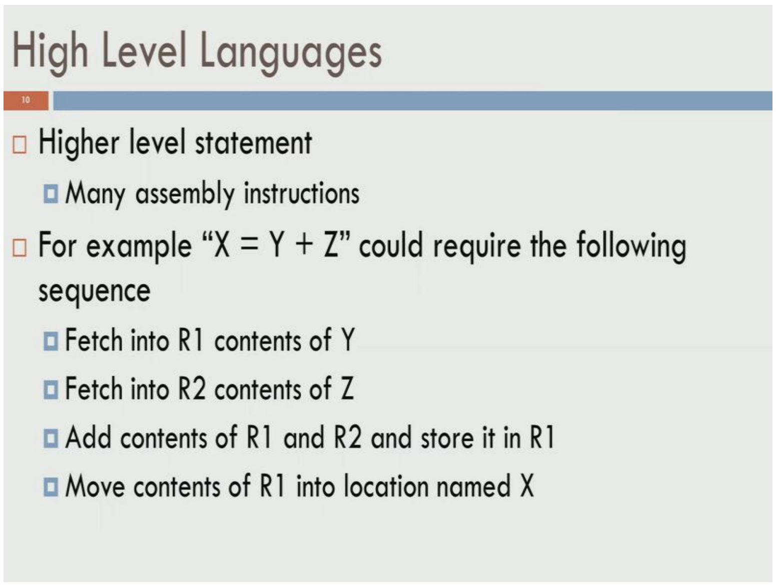
  mov al, 061h

  - Move the hexadecimal value 61 (97 decimal) into the processor register named "al".
  - assembly language representation is easier to remember (*mnemonic;* e.g. - MVI AL, Val)

So, sometimes you may somewhere down the line, you may learn a language called the assembly language. And we are seeing some small example here: an x86 Intel 32 processor can execute the following binary instruction. So, you have 1011 four 0s 0 1 1 four 0s and a 1, this is the binary code for moving 61 in hexadecimal to an internal memory called a register of the name al. So, al is a register which is an internal memory inside the CPU.

So, the meaning of this instruction is move number 61 in hexadecimal to al. And this is something that you and I may not know, you and I may not even understand if you are just given the binary bits. However, for the CPU everything is translated into these bits, operating in these bite at this level is very, very hard for us humans. Therefore, we use, so called high level languages. But, sometimes you also have this intermediate language called the assembly language, which is human understandable, but at the same time it is more detailed. For example, we have MVI AL comma value. So, this could mean move the value to al and such instructions are called mnemonics. So, mnemonics are essentially easy to remember and this binary sequence, we could write them as MVI AL comma Val, this could be given to an assembler which would translate that in to this binary code. But, even operating at this assembly level is quite hard, this does not capture the kind of problems that we want to solve directly, they get too detail.

(Refer Slide Time: 08:30)



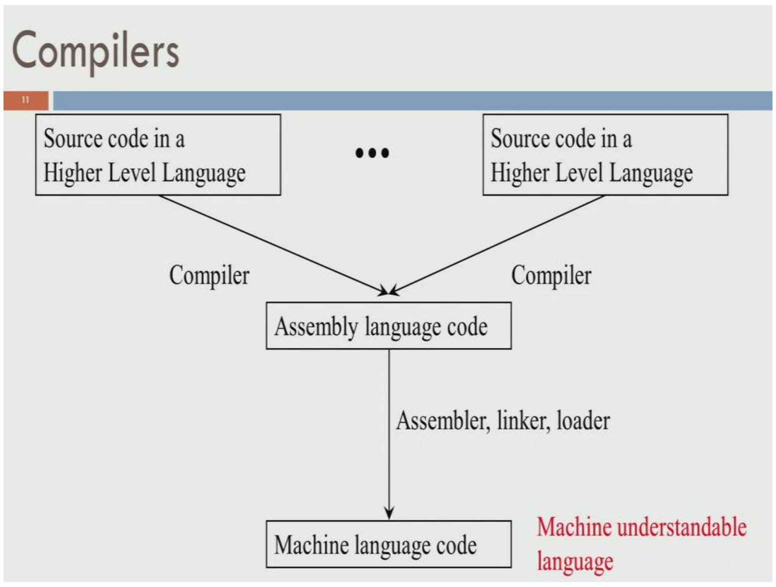
High Level Languages

- Higher level statement
  - Many assembly instructions
- For example "X = Y + Z" could require the following sequence
  - Fetch into R1 contents of Y
  - Fetch into R2 contents of Z
  - Add contents of R1 and R2 and store it in R1
  - Move contents of R1 into location named X

Instants we use, so call high level languages and a single high level statement could have more than one assembly instruction in it. So, let us take a small example, let say I want to add Z, Y and the result has to be stored in X. So, I would right this as X equals Y plus Z and it could recover the following sequence of operations. You may have to fetch Y from the memory, stored it in an internal memory location or a register called R1, you fetch Z also from the memory, you store it in another register called R2. And the ALU would then add R1 and R2 store the result back in R1 itself and it may require a move from R1 to the memory location name x.

So, a single operation X equals Y plus Z has resulted in four smaller operations and these operations could be written in assembly language if you would like to. However, this becomes too tedious. So, we would like to operate at a level which is much higher, then what the processor can understand and something that easier for the human beings to program with. And that is how the evolution of high level languages started. And in this course, we are going to learn one such high level language namely c.

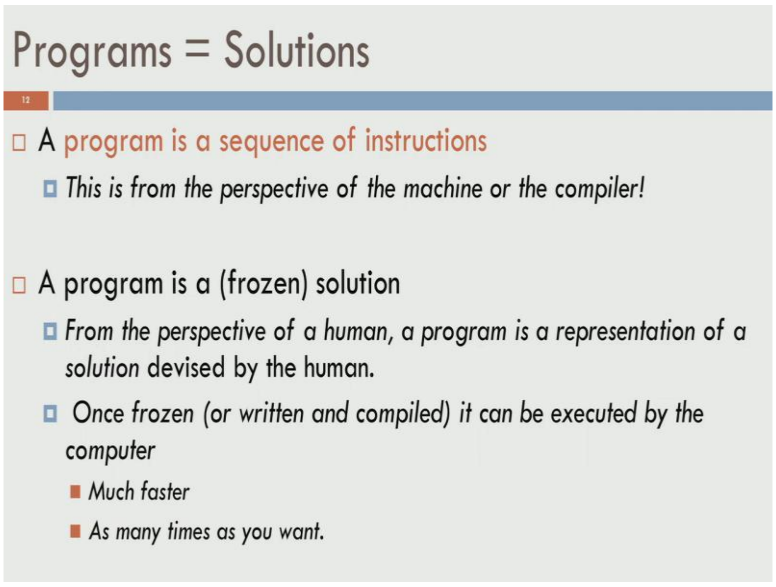
(Refer Slide Time: 09:55)


Compilers
11
Source code in a Higher Level Language
•••
Source code in a Higher Level Language
Compiler
Compiler
Assembly language code
Assembler, linker, loader
Machine language code
Machine understandable language

Once you have a program written in high level, you need a set of tools to convert them into the machine level. And typically the programs that we write are called source code. They are called source because, that capture the intention of the programmer and a set of tools will convert the source code into machine code, you may have more than one such source program available. And you give this to a tool called a compiler, compiler is essentially a piece of software, which can convert this high level code into assembly code.

And in turn an assembler can take this assembly code and generate machine code, at run time you have a linker and loader which will actually execute the program. Even if we don’t understand the details now, slowly and steadily we will build up an understanding towards all these ideas, we will also see software demonstrated along the way. So, as of now just remember that, you have a high level program which gets converted by a compiler into a assembly code. And assembly code downwards is taken care by assembler, linker and loader what we bother about is at the high level namely the high level language.

(Refer Slide Time: 11:19)


Programs = Solutions

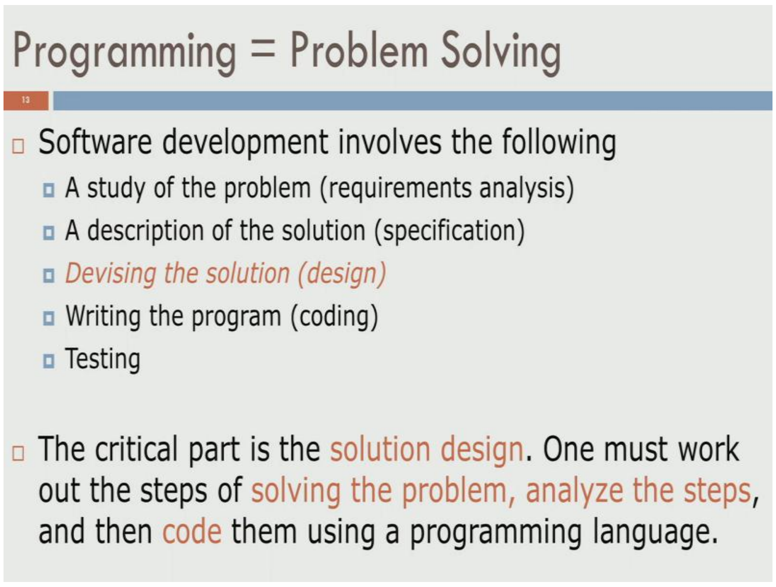
- A program is a sequence of instructions
  - *This is from the perspective of the machine or the compiler!*
- A program is a (frozen) solution
  - *From the perspective of a human, a program is a representation of a solution devised by the human.*
  - *Once frozen (or written and compiled) it can be executed by the computer*
    - *Much faster*
    - *As many times as you want.*

So, when we write programs, we are actually looking at solutions, we are trying to solve things. From the CPU perspective a program is nothing but, a sequence of instructions, you have instruction 1, 2, 3, 4, instruction 5 could be a condition based on whatever the result of instruction 5 is, you may execute instruction 6 for a instruction 10 and so on.

However, this is not the way we think about solving a problem. For us a program is a solution to a problem and sometimes it is an frozen solution. By frozen what we mean is we have written the program, it is already compiled and it is ready to execute. And at this time, the solution is frozen, you cannot change the solution, unless you go and change the program and compile it once more.

So, from the perspective of a human being a program is the representation of a solution devised by a human being. And the nice thing about program is that, it can be compiled and store and it is ready for execution from there on. You can also distribute the programs to others for them to read understand or even comment and change, you could distribute the machine version or machine code for others to execute, but, not be able to see the program. You can do lot of things and you write the program once, you can run the program as many times as you want.

(Refer Slide Time: 12:43)


Programming = Problem Solving

13

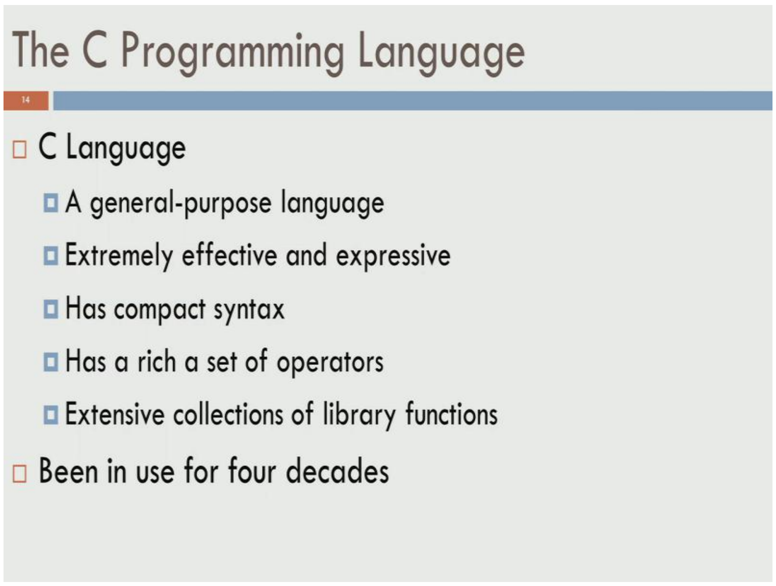
- Software development involves the following
  - A study of the problem (requirements analysis)
  - A description of the solution (specification)
  - *Devising the solution (design)*
  - Writing the program (coding)
  - Testing
- The critical part is the solution design. One must work out the steps of solving the problem, analyze the steps, and then code them using a programming language.

Let us get into what programming is about, a program is essentially a piece of software that you write and programming; however, is problem solving. So, this is what we really want to do. So, in this course we will learn how to solve problems, but in the process we will also learn this language called c, with which we can take problems, break them down and write them using a program use tools to compile and run them, we will learn this whole cycle.

So, any software development process starts with understanding the problem, you should first of all understand what the problem is, typically the problem is stated in English. So, that is called requirements analysis, from there you get a precise specification of the problem, usually mathematically specified and you device the solution. So, given a problem you go and device the solution or design the solution. And once you have the design for the problem, then you go and write it in the program, you could write it in any programming language, this is also called coding. And finally, once the coding is done we have to tested before we deploy it. So, we start with requirements analysis and get the specifications out, design a solution for the problem go and modulate and program it using a language compile it, run it and test it before it is ready for use.

However, the most crucial part of this whole process is actually the solution design. So, you have to solve the problem and analyze the steps and ensure that this problem is captured properly and you have a solution, that is indeed correct, any ambiguity in the specification can result in program, that does not behave as expected. So, you have to be careful about understanding this specification. You should also understand the nuances of the programming language itself. So, that the intent is captured carefully and you use a programming language for that, finally you test it.

(Refer Slide Time: 14:50)



The C Programming Language

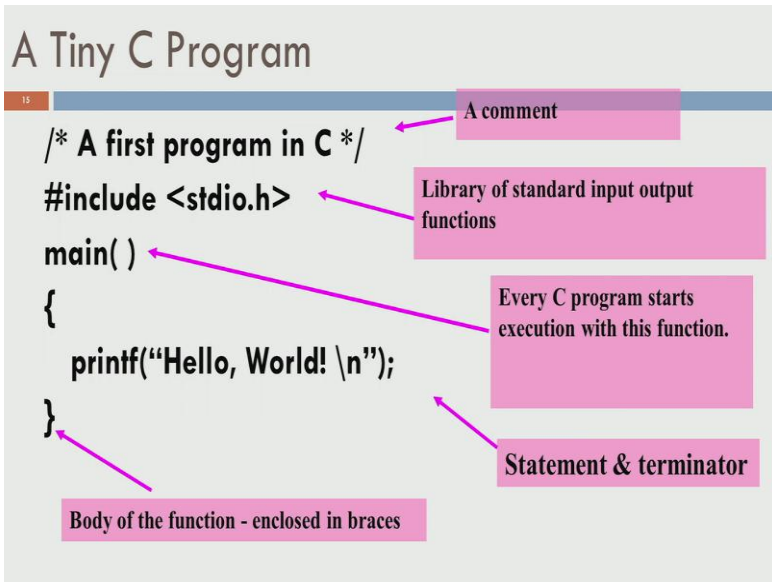
14

- C Language
  - A general-purpose language
  - Extremely effective and expressive
  - Has compact syntax
  - Has a rich a set of operators
  - Extensive collections of library functions
- Been in use for four decades

So, as I mentioned earlier we are going to use this language called C, C is a very old language it is been therefore, almost four decades now very widely used in industry even now. So, there are other languages like C plus plus and Java and so on. However, learning C well and understanding C thoroughly is basic requirement in the industry even now. C is a general purpose language, it is not for any specialized purpose, unlike a language like a HTML and so on.

It is extremely effective and expressive, you will see this in a short while, it has a fairly compact syntax. The language is not too big, it has a rich set of operators, pretty much every arithmetic logic operation that you think about is already available as an operator and C also has an extensive collection of libraries. So, you do not have a really write every single thing down, you can call libraries, library functions whenever you need them and C comes with a rich set of libraries and that is one of the basic reasons why we pick c.

(Refer Slide Time: 15:57)



A Tiny C Program

```
/* A first program in C */
#include <stdio.h>
main( )
{
   printf("Hello, World! \n");
}
```

15
A comment
Library of standard input output functions
Every C program starts execution with this function.
Statement & terminator
Body of the function - enclosed in braces

Let us look at a tiny C program. So, you may not understand it write away, but it is really tiny, it has only 6 line of code. The first line is actually comment, it is says a first program in C and the next line says hash include stdio dot h, there is this so called function by name main and there is exactly one statement inside this main. So, watch the mouse pointer to have a function called main and within the braces we have a single statement call printf and printf seems to be taking this hello world.

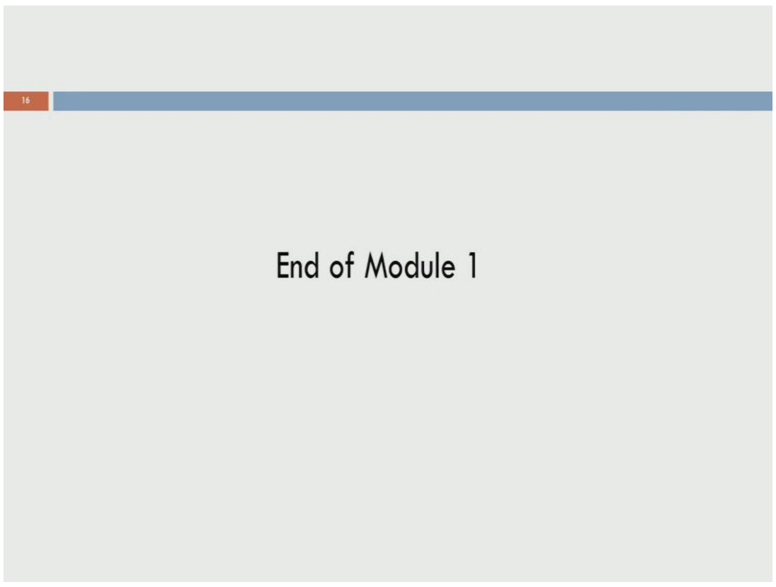
So, this is been a custom for while now, almost every programming language book that you go to will teach you how to printf hello world. So, let us see are let us break down what this program is about, as I mentioned the first line is just a comment, is for us to understand it does not really result in anything that is executable. So, the program has it, but the machine code will not have these comments, it is not an instruction for the CPU to do anything, it just for us to understand.

And let us look the next line, it is says hash include, it means include something which is already built. And I mention earlier that C has several library calls that you can make and stdio dot h is one such library with which you can do standard input and output. So, if I want type something on a keyboard and if I want see something on a screen, that is what you get from stdio dot h.

Every C program will have something called a main function. And the very first instruction, that is executed will be from main and within this body of these curly braces, you have a single statement here call printf. You can see that, this printf is a function we will see what a function is a more detail later, it is seems to a taken argument or a parameter. So, it can say printf hello world and it will printf hello world on the screen.

You can say print hello you are name will print hello and you are name on the screen and so on. But, every statement is terminated with the semicolon and the body of the function is usually within the braces. So, this is a fairly simple program, if we compile it and run it will print hello world on the screen.

(Refer Slide Time: 18:38)



16

End of Module 1

So, with this we are at the end of module 1, in the next module we will see how… we will take up a small problem and we will see how to solve the problem and we will also see how to write it as a program.