Computer Programming (NCSV101) [Winter 2025-26]

Slides 1.1

Introduction to Computing

SAURABH SRIVASTAVA

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT (ISM) DHANBAD

You know this device well…

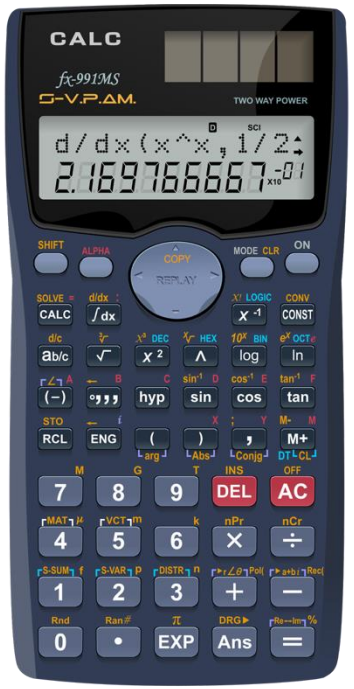
What does it do?

You know this device well…

What does it do?

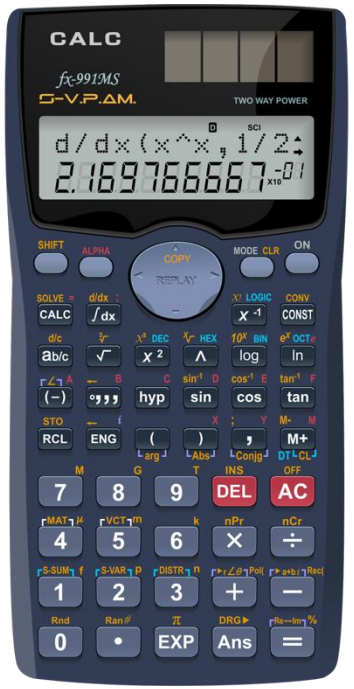
Computations – Addition, Subtraction, Multiplication, Division… and maybe a few more !!

You will use this one soon as well !!



This one basically does the same

You will use this one soon as well !!



This one basically does the same

Just that it can do way more than just basic arithmetic

… and what about these?

They can actually do far more than just simple computations

… and what about these?

They can actually do far more than just simple computations

You can use them for performing “higher level” tasks – which are closer to human lifestyle

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?

How do these devices do what they do?

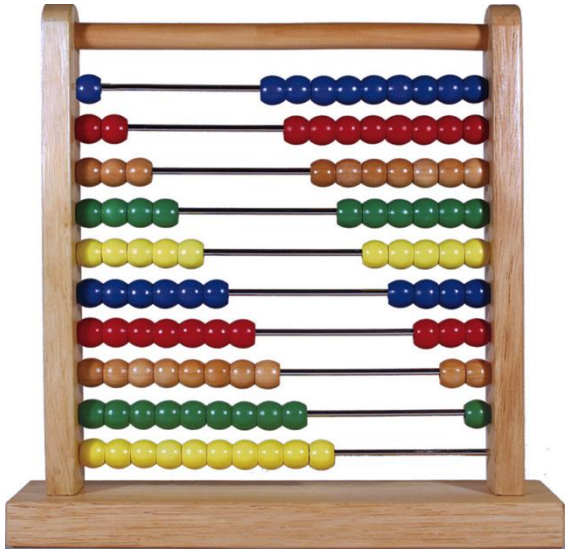
Calculator

- Let us take the example of addition – how can we add two numbers?
- Before that, let us ask the question - how did WE learn to add?

How do these devices do what they do?

Calculator

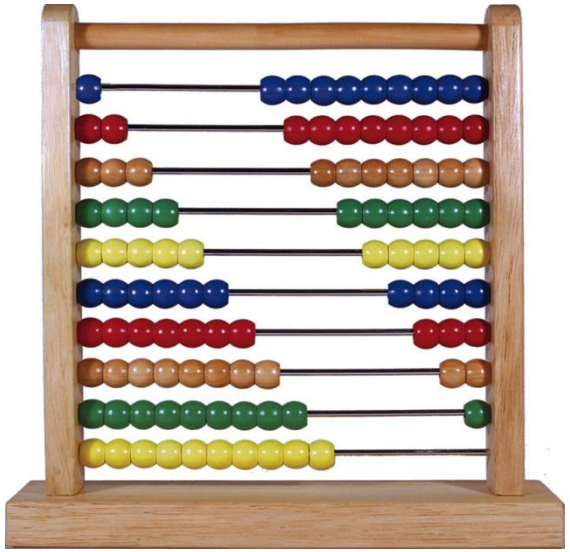
- Let us take the example of addition – how can we add two numbers?
- Before that, let us ask the question - how did WE learn to add?
- Probably through some device like this



How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- Before that, let us ask the question - how did WE learn to add?
- Probably through some device like this

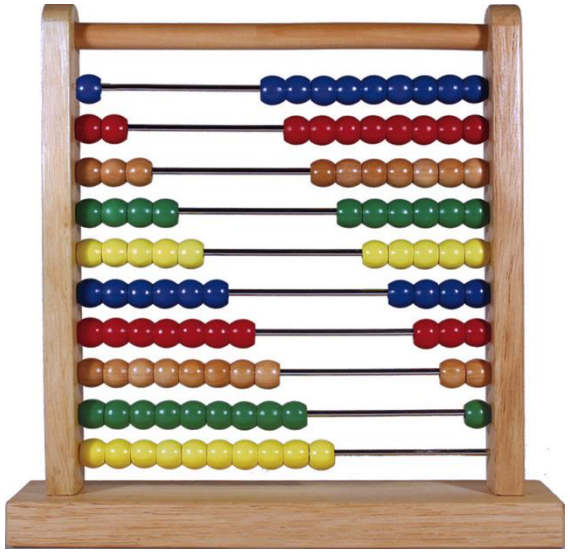


Essentially, we only learnt addition of two digits

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- Before that, let us ask the question - how did WE learn to add?
- Probably through some device like this



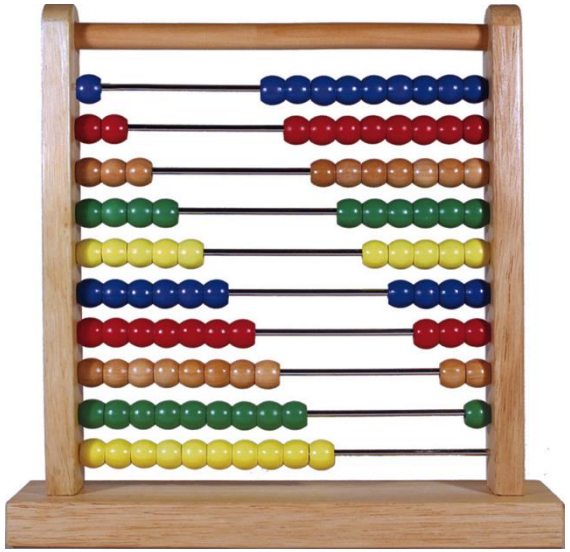
Essentially, we only learnt addition of two digits

…and we learnt a *Number System*, where the position of a digit (ones, tens, hundreds etc.) changed its interpretation

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- Before that, let us ask the question - how did WE learn to add?
- Probably through some device like this



Essentially, we only learnt addition of two digits

…and we learnt a *Number System*, where the position of a digit (ones, tens, hundreds etc.) changed its interpretation

We then started adding *digits in corresponding* positions, and learnt the concept of a *carry-over* to the next position

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- We also studied that the Number System that we use in day-to-day life *uses the base 10*

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- We also studied that the Number System that we use in day-to-day life *uses the base 10*
- We can have Number Systems based on any other number as well, e.g., the Binary System *uses base 2*

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- We also studied that the Number System that we use in day-to-day life *uses the base 10*
- We can have Number Systems based on any other number as well, e.g., the Binary System *uses base 2*

So, what is so sacred about base 2?

- It is important because the world of computing devices use base 2 at the core

How do these devices do what they do?

Calculator

- Let us take the example of addition – how can we add two numbers?
- We also studied that the Number System that we use in day-to-day life *uses the base 10*
- We can have Number Systems based on any other number as well, e.g., the Binary System *uses base 2*

So, what is so sacred about base 2?

- It is important because the world of computing devices use base 2 at the core

But why base 2? Why can't they work with base 10?

- It is because electronic circuits are mostly made up of *transistors*
- A transistor can either be *on* or *off*; based on its state, we can do a number of tasks !!
- Since the Binary System only has two digits – *0* and *1* – it is natural to use that for all computing devices

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

In short, the basics of Boolean Algebra can be summarized as follows:

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

In short, the basics of Boolean Algebra can be summarized as follows:

- The variables in Boolean Algebra can only take one of the two values – 0 or 1

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

In short, the basics of Boolean Algebra can be summarized as follows:

- The variables in Boolean Algebra can only take one of the two values – 0 or 1
- The *AND* function is defined over two or more Boolean variables as
  - `The AND function has a value 1, if and only if, all variables have the value 1`
  - `Otherwise, the function has a value 0`

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

In short, the basics of Boolean Algebra can be summarized as follows:

- The variables in Boolean Algebra can only take one of the two values – 0 or 1
- The *AND* function is defined over two or more Boolean variables as
  - `The AND function has a value 1, if and only if, all variables have the value 1`
  - `Otherwise, the function has a value 0`
- The *OR* function is defined over two or more Boolean variables as
  - `The OR function has a value 0, if and only if, all variables have the value 0`
  - `Otherwise, the function has a value 1`

Logical Operations

Using the transistors, we can perform arithmetic and “logical” operations

- The logical arithmetic is called *Boolean Algebra* which is the basis of the *Switching Theory*

In short, the basics of Boolean Algebra can be summarized as follows:

- The variables in Boolean Algebra can only take one of the two values – 0 or 1
- The *AND* function is defined over two or more Boolean variables as
  - The AND function has a value 1, if and only if, all variables have the value 1
  - Otherwise, the function has a value 0
- The *OR* function is defined over two or more Boolean variables as
  - The OR function has a value 0, if and only if, all variables have the value 0
  - Otherwise, the function has a value 1
- The *NOT* function is defined over one Boolean variable as
  - The NOT function has a value 1, if the variable has the value 0
  - Otherwise, the function has a value 0 (i.e., when the variable has the value 1)

The “Central Processing Unit”

At the heart of any sophisticated computing device, is an **Arithmetic-Logic Unit** or **ALU**

It performs the arithmetic and logical operations which are the basis of all complex computations

Computers also have another unit which can be considered as the *boss* of ALU :-P

- It controls the use of ALU, i.e., which Arithmetic and Logic operation is performed at what time
- The unit is called the **Control Unit** or **CU** (can there be a name less obvious than this? :-D)

The “Central Processing Unit”

At the heart of any sophisticated computing device, is an **Arithmetic-Logic Unit** or **ALU**

It performs the arithmetic and logical operations which are the basis of all complex computations

Computers also have another unit which can be considered as the *boss* of ALU :-P

- It controls the use of ALU, i.e., which Arithmetic and Logic operation is performed at what time
- The unit is called the **Control Unit** or **CU** (can there be a name less obvious than this? :-D)

The “Central Processing Unit”

At the heart of any sophisticated computing device, is an **Arithmetic-Logic Unit** or **ALU**

It performs the arithmetic and logical operations which are the basis of all complex computations

Computers also have another unit which can be considered as the *boss* of ALU :-P

- It controls the use of ALU, i.e., which Arithmetic and Logic operation is performed at what time
- The unit is called the **Control Unit** or **CU** (can there be a name less obvious than this? :-D)

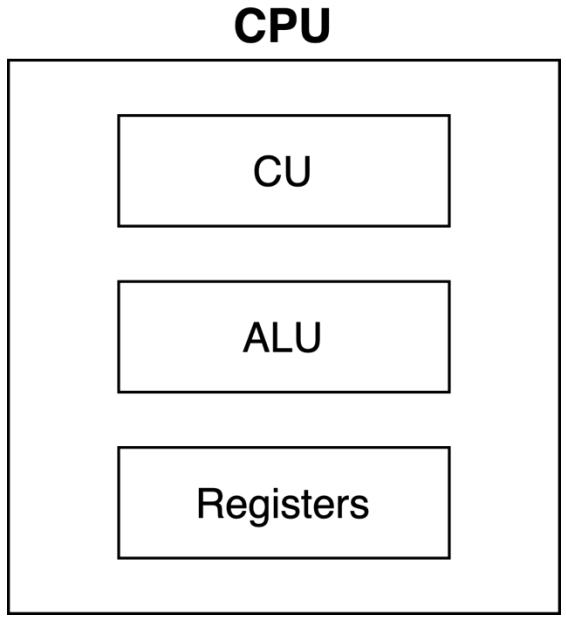
To store intermediate state of variables of a complex computation, some “memory” is also needed

- Known as **Registers**, these elements are capable of holding on to some 0s and 1s, till we keep them powered
- Each 0 or 1 is called a *bit* and a set of eight 0s or 1s is called a *byte*

The ALU, CU and the Registers, together constitute the brain of a computer

- We call this brain the **Central Processing Unit** or **CPU**

The block diagram of a CPU



Just what we discussed !!

von Neumann Architecture

We can now try to imagine the block diagram of a basic computer

It will certainly have a CPU

You will need a way to send some “inputs” to the CPU – telling it what computations to do

The CPU must “output” the results of the computation somewhere on completion

von Neumann Architecture

We can now try to imagine the block diagram of a basic computer

It will certainly have a CPU

You will need a way to send some “inputs” to the CPU – telling it what computations to do

The CPU must “output” the results of the computation somewhere on completion

It is not a bad idea to add “some more memory”, especially for repeated computation

- You can store the computation details in the memory, not having to input them all the time
- The CPU too, can store the results in the memory, in case the output “device” is busy

von Neumann Architecture

We can now try to imagine the block diagram of a basic computer

It will certainly have a CPU

You will need a way to send some “inputs” to the CPU – telling it what computations to do

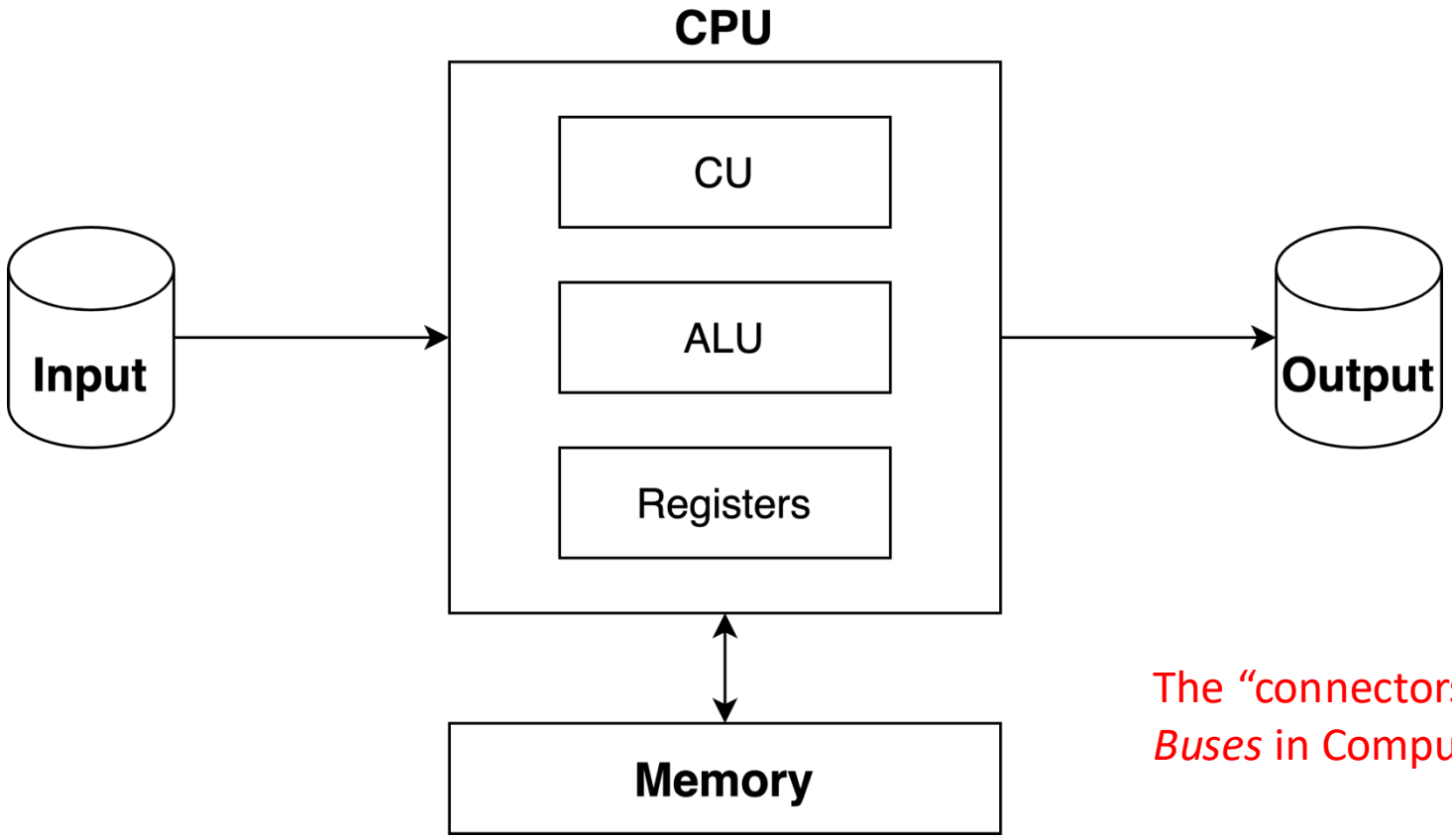
The CPU must “output” the results of the computation somewhere on completion

It is not a bad idea to add “some more memory”, especially for repeated computation

- You can store the computation details in the memory, not having to input them all the time
- The CPU too, can store the results in the memory, in case the output “device” is busy

This architecture of a computer, was originally called the **von Neumann Architecture**

von Neumann Architecture



The “connectors” here are also called *Buses* in Computer Science terminology

von Neumann Architecture

We can now try to imagine the block diagram of a basic computer

It will certainly have a CPU

You will need a way to send some “inputs” to the CPU – telling it what computations to do

The CPU must “output” the results of the computation somewhere on completion

It is not a bad idea to add “some more memory”, especially for repeated computation

- You can store the computation details in the memory, not having to input them all the time
- The CPU too, can store the results in the memory, in case the output “device” is busy

This architecture of a computer, was originally called the **von Neumann Architecture**

- It is named after the scientist who first proposed it in 1945 – John von Neumann
- While the modern computers are not based on this, it was very close to the initial computers

Stored-program Computer

Imagine a computer that is built over the von Neuman Architecture

What this computer does is something like this

- It uses the Memory unit to “store some instructions” – essentially a sequence of computations
- It uses the same Memory to “contain some data” – essentially the data or operands for computations
- The Control Unit interprets the instructions stored in the memory
- Based on the type of computation, it fetches the expected data from the memory as well
- It instructs the ALU to perform the computations, and stores the results back in the Memory

Stored-program Computer

Imagine a computer that is built over the von Neuman Architecture

What this computer does is something like this

- It uses the Memory unit to “store some instructions” – essentially a sequence of computations
- It uses the same Memory to “contain some data” – essentially the data or operands for computations
- The Control Unit interprets the instructions stored in the memory
- Based on the type of computation, it fetches the expected data from the memory as well
- It instructs the ALU to perform the computations, and stores the results back in the Memory

For now, assume that “there is a mechanism” to perform input and output, to and from the Memory

- This type of a computer, is known as the *Stored-program Computer*

Stored-program Computer

Imagine a computer that is built over the von Neuman Architecture

What this computer does is something like this

- It uses the Memory unit to “store some instructions” – essentially a sequence of computations
- It uses the same Memory to “contain some data” – essentially the data or operands for computations
- The Control Unit interprets the instructions stored in the memory
- Based on the type of computation, it fetches the expected data from the memory as well
- It instructs the ALU to perform the computations, and stores the results back in the Memory

For now, assume that “there is a mechanism” to perform input and output, to and from the Memory

- This type of a computer, is known as the *Stored-program Computer*

While we have evolved to more complex computers, the core principles of computing are the same, i.e., there is a Memory where data and instructions reside, and the CPU executes them in a particular order

Homework !! (life is cruel, you know :-P)

Read about the criticism of von Neumann Architecture

- To be precise, read about von Neumann bottleneck
- Start here:
  https://en.wikipedia.org/wiki/Von_Neumann_architecture#Von_Neumann_bottleneck

The definition of Stored-program Computer is somewhat a grey area

- This is why we did not discuss a proper definition of the term
- The Wikipedia page for Stored-program Computer talks about some *Universal Turing Machine*
- You'll formally study about Turing Machines in a course on Theory of Computation
- However, it is not a bad idea to try and understand the concept (it is fine even if you don't get much)
  The relevant Wikipedia pages are:
  https://en.wikipedia.org/wiki/Stored-program_computer
  https://en.wikipedia.org/wiki/Universal_Turing_machine