Introduction to Computer Architecture (ACOL 216)

Computer Architecture

Tanmoy Chakraborty
IIT Delhi Abu Dhabi

Logistics

ACOL 216: 4 credits, 3-1-0

Regular lecture slot: Tue, Thu, Fri: 10-10:50 am
Venue: M4 Classroom 6

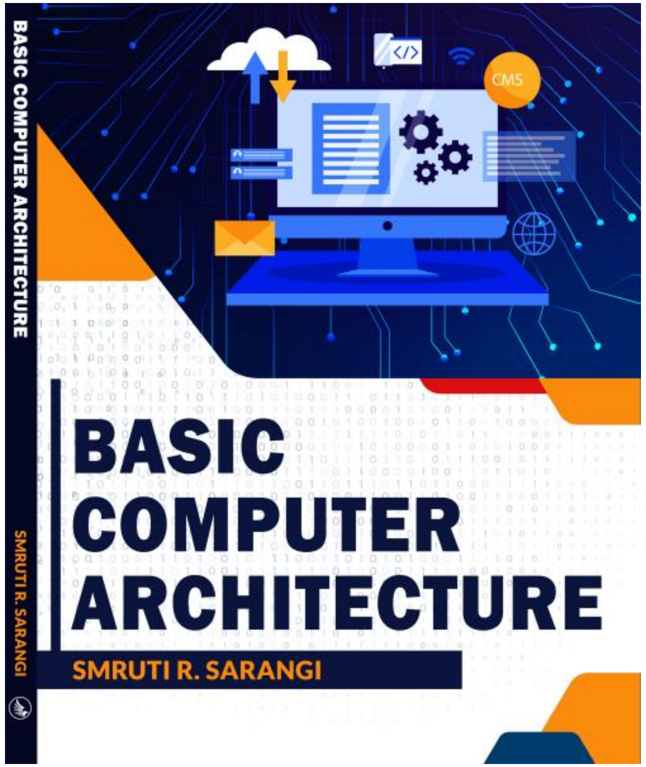
Tutorial: Mon, Thu: 11-11:50 am
Venue: M2-2-009 (Mon), M2-2-007 (Thu)

TAs: Anwoy Chatterjee, XXX

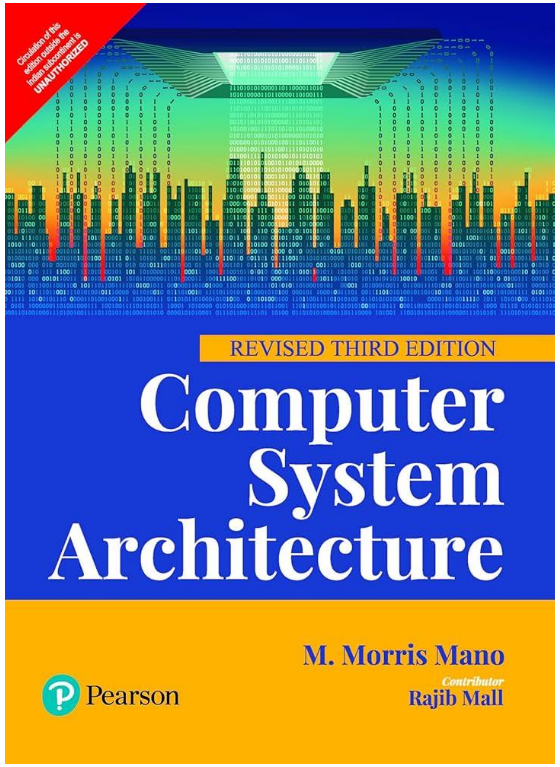
Course website: https://lcs2.in/acol216

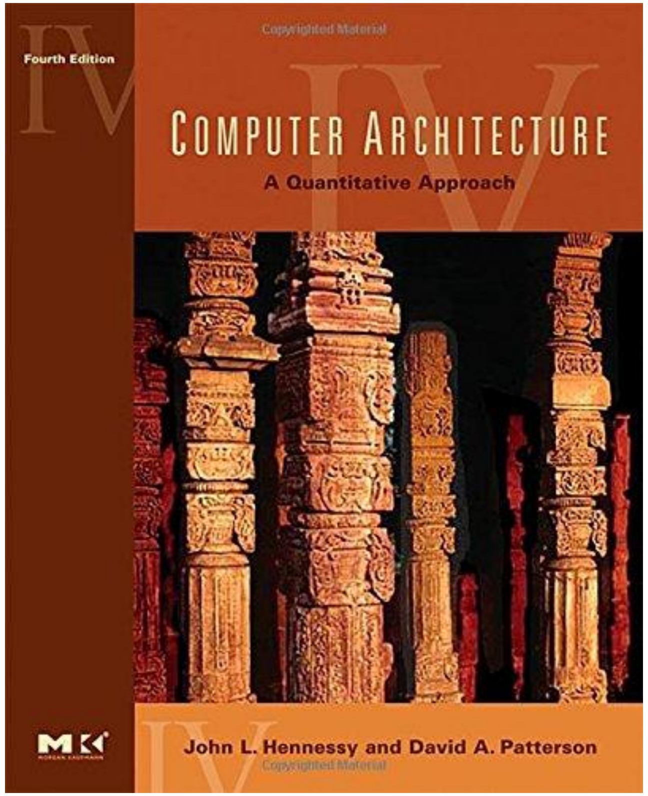
No Tutorial until I announce it

Textbooks



Slides are primarily adopted from this book





Tentative Syllabus

Introduction
Language of Bits
Assembly Languages
Digital Logic
Computer Arithmetic
Processor Design
Memory System

Tentative Grading Scheme

- Assignment 1 (10 marks) – Before Minor
- Assignment 2 (10 marks) – After Minor
- Three Quizzes (20 marks) – One before minor, two after minor (best 2 out of 3)
- Minor (30 marks)
- Major (30 marks)
- Attendance: 75%
- Audit: 40% with 75% attendance

What is Computer Architecture ?

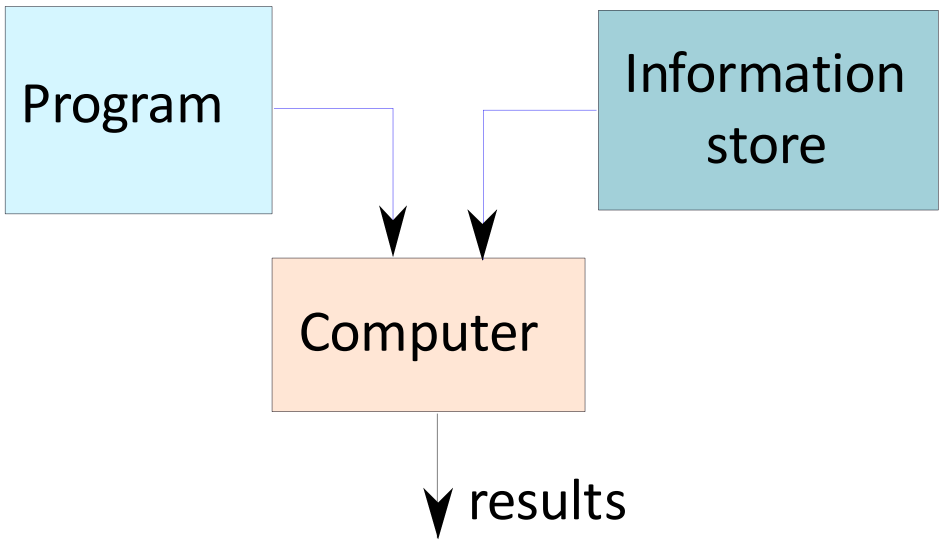
Answer : It is the study of computers ?

* Computer Architecture
  * The view of a computer as presented to software designers
* Computer Organization
  * The actual implementation of a computer in hardware.

What is a Computer ?

A computer is a general purpose device that can be **programmed** to **process** information, and yield **meaningful** results.

How does it work ?

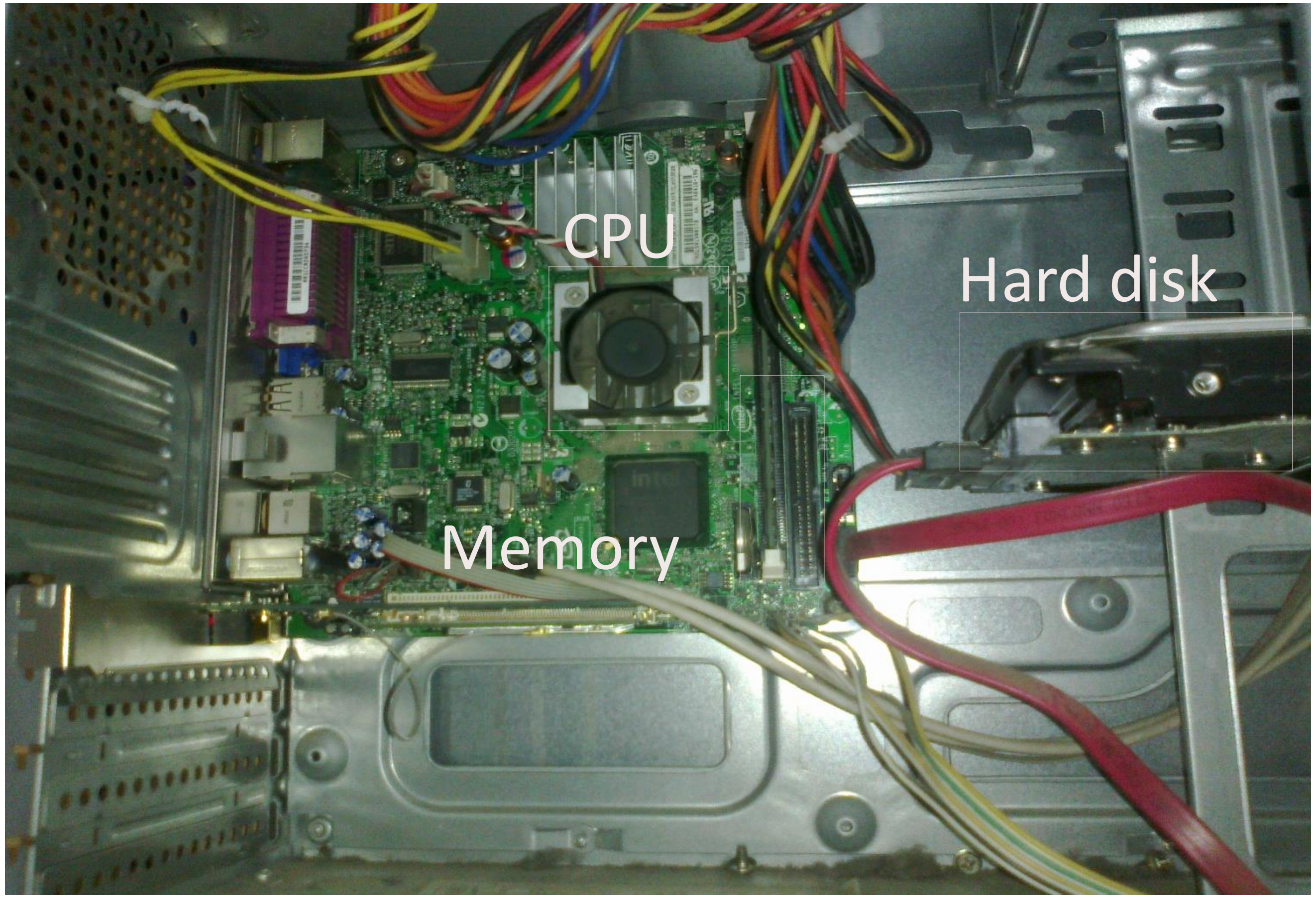


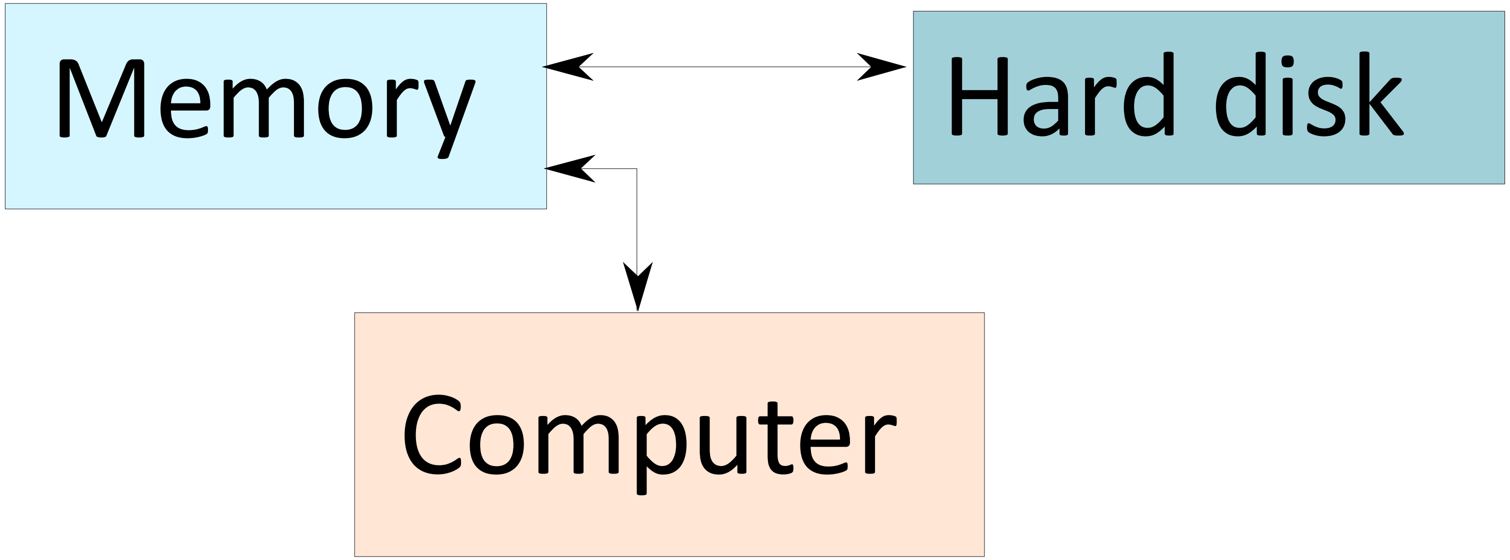
- Program – List of instructions given to the computer
- Information store – data, images, files, videos
- Computer – Process the information store according to the instructions in the program

What does a computer look like ?

* Let us take the lid off a desktop computer

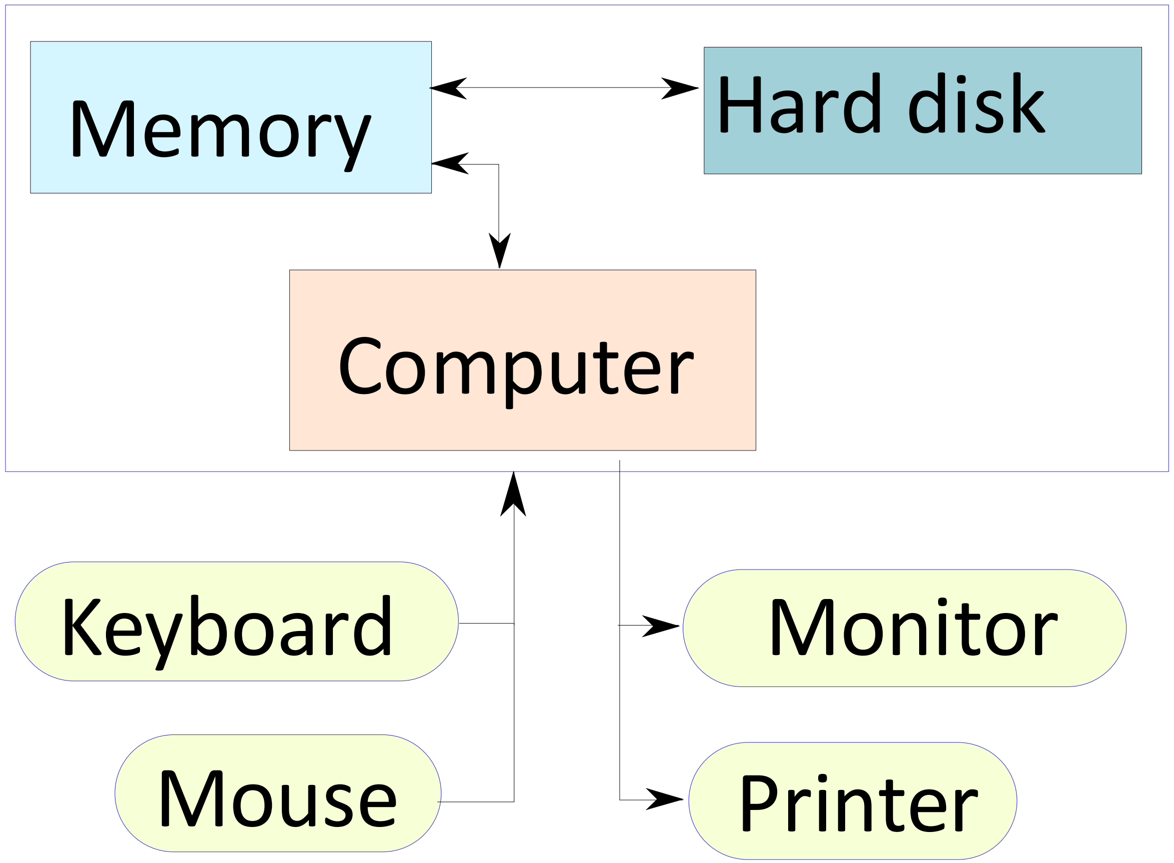
- FPU (Floating Point Unit)
- ISP (Image Signal Processor)
- Microcontrollers (MCU)
- Network Processor / NIC Processor
- Storage Controllers



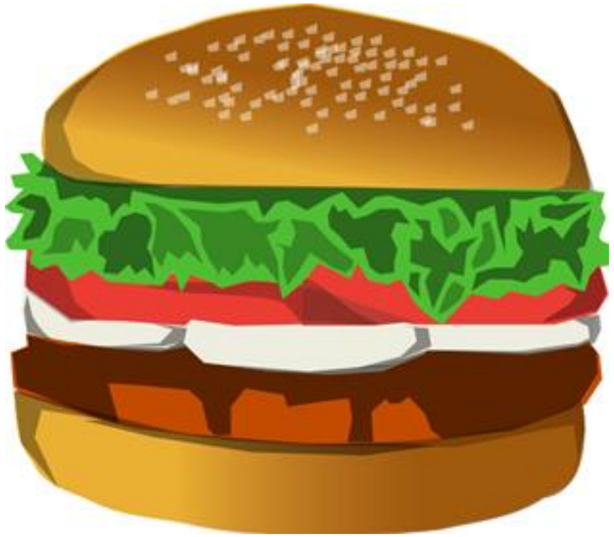


* Memory – Stores programs and data. Gets destroyed when the computer is powered off
* Hard disk – stores programs/data permanently

Let us make it a full system …

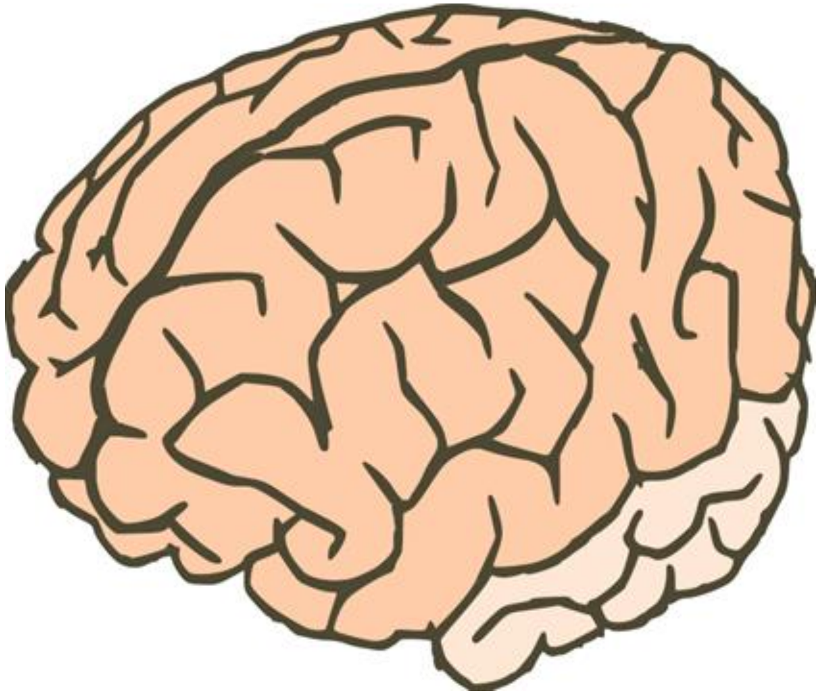


Food for Thought...



* What is the most intelligent computer ?

Answer …



- Our brilliant brains

How does an Electronic Computer Differ from our Brain ?

Feature	Computer	Our Brilliant Brain
Intelligence	Dumb	Intelligent
Speed of basic calculations	Ultra-fast	Slow
Can get tired	Never	After sometime
Can get bored	Never	Almost always

* Computers are ultra-fast and ultra-dumb

How to Instruct a Computer ?

Program $\xrightarrow{\text{compile}}$ Executable $\xrightarrow{\text{execute}}$ Output

* Write a program in a high level language – C, C++, Java
* **Compile** it into a format that the computer understands
* Execute the program

What Can a Computer Understand ?

- Computer can clearly NOT understand instructions of the form
  - Multiply two matrices
  - Compute the determinant of a matrix
  - Find the shortest path between Mumbai and Delhi
- They understand :
  - Add $a + b$ to get c
  - Multiply $a * b$ to get c

The Language of Instructions

* Humans can understand
  * Complicated sentences
    * English, French, Spanish
* Computers can understand
  * Very simple instructions

The semantics of all the instructions supported by a processor is known as its instruction set architecture (ISA). This includes the semantics of the instructions themselves, along with their operands, and interfaces with peripheral devices.

Features of an ISA

- Example of instructions in an ISA
  - Arithmetic instructions : add, sub, mul, div
  - Logical instructions : and, or, not
  - Data transfer/movement instructions
- Complete
  - It should be able to implement all the programs that users may write.

Features of an ISA – II

* Concise
  * The instruction set should have a limited size. Typically an ISA contains 32-1000 instructions.
* Generic
  * Instructions should not be too specialized, e.g. add14 (adds a number with 14) instruction is too specialized
* Simple
  * Should not be very complicated.

Designing an ISA

* Important questions that need to be answered :
  * How many instructions should we have ?
  * What should they do ?
  * How complicated should they be ?

Two different paradigms : RISC and CISC

RISC
(Reduced Instruction Set Computer)

CISC
(Complex Instruction Set Computer)

RISC vs CISC

A reduced instruction set computer (RISC) implements simple instructions that have a simple and regular structure. The number of instructions is typically a small number (64 to 128). Examples: ARM, IBM PowerPC, HP PA-RISC

A complex instruction set computer (CISC) implements complex instructions that are highly irregular, take multiple operands, and implement complex functionalities. Secondly, the number of instructions is large (typically 500+). Examples: Intel x86, AMD, VAX

CISC

Who developed it?
CISC emerged from **early computer architects in industry**, most notably at **IBM** and later **Intel.**

Key contributors:
- IBM engineering teams
- Intel processor architects

When?

- **1960s–1970s**

Why CISC?
- Early computers had **very limited memory**
- Goal: **reduce program size** by packing more work into a single instruction
- Instructions were often **complex, multi-cycle, and variable-length**

RISC

Who developed it?

RISC was pioneered in **academic and industrial research labs**, led by:

- John Cocke at IBM
- David Patterson at University of California, Berkeley
- John L. Hennessy at Stanford University

When?

Late 1970s – early 1980s

Why RISC?

- Empirical studies showed most programs use **simple instructions**
- Simpler instructions allow:
    - Pipelining
    - Higher clock speeds
    - Better compiler optimization

Summary Uptil Now ...

* Computers are dumb yet ultra-fast machines.
* Instructions are basic rudimentary commands used to communicate with the processor. A computer can execute billions of instructions per second.
* The compiler transforms a user program written in a high level language such as C to a program consisting of basic machine instructions.
* The instruction set architecture(ISA) refers to the semantics of all the instructions supported by a processor.
* The instruction set needs to be complete. It is desirable if it is also concise, generic, and simple.

Completeness of an ISA

How can we ensure that an ISA is complete ?

Turing Machine

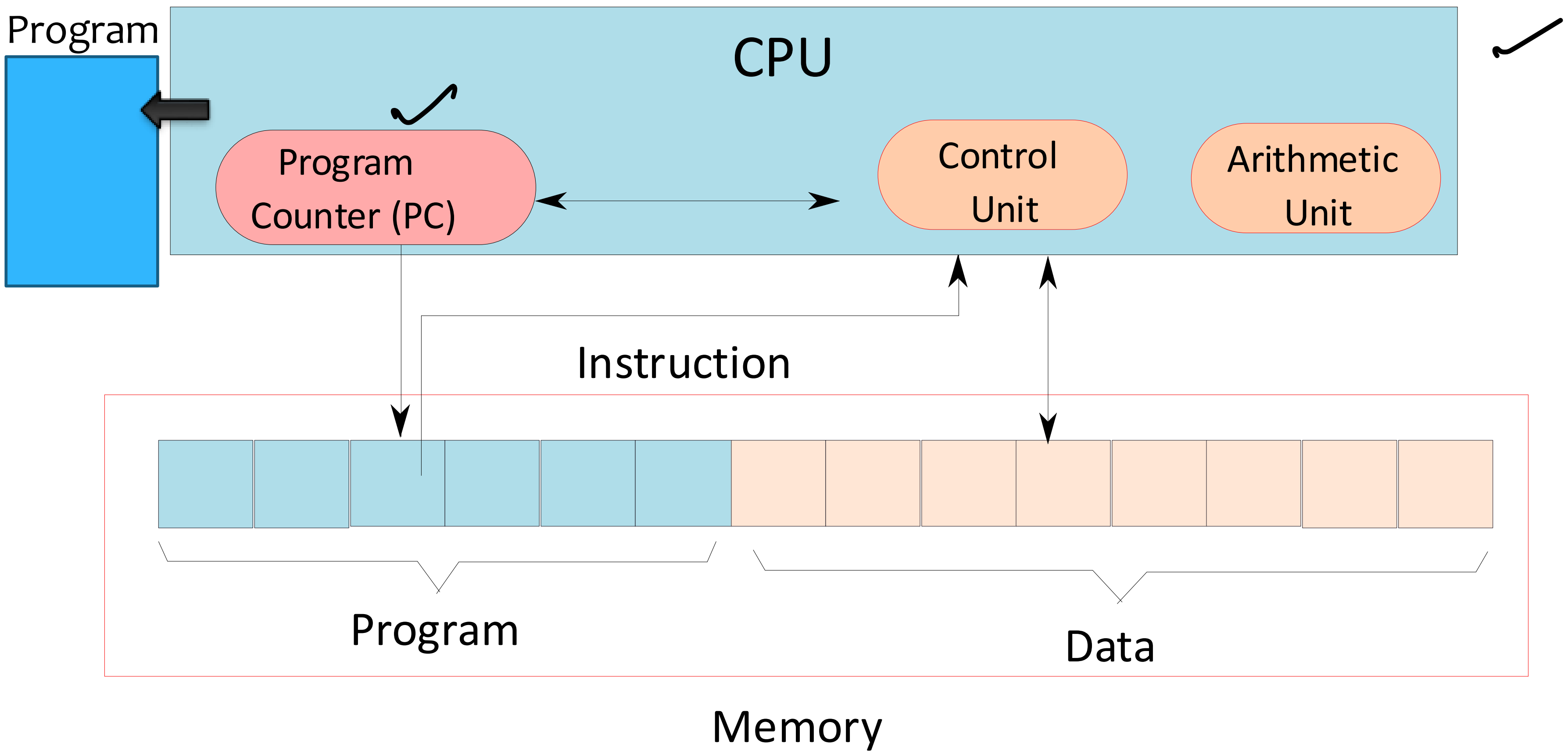
* Complete means :
  * Can implement all types of programs
  * For example, if we just have add instructions, we cannot subtract (NOT Complete)

Completeness of an ISA – II

How to ensure that we have just enough instructions such that we can implement every possible program that we might want to write ?

More related to ToC

Computer Inspired from the Turing Machine



Elements of a Computer

* Memory (array of bytes) contains
  * The program, which is a sequence of instructions
  * The program data → variables, and constants
* The program counter(PC) points to an instruction in a program
  * After executing an instruction, it points to the next instruction by default
  * A branch instruction makes the PC point to another instruction (not in sequence)
* CPU (Central Processing Unit) contains the
  * Program counter, instruction execution units

Let us now design an ISA …

- Single Instruction ISA
  - sbn – subtract and branch if negative
- Add (a + b) (assume temp = 0)

```
1: sbn temp, b, 2
2: sbn a, temp, exit
```

temp = temp - b
= -b

a = a - temp
= a - (-b)
= a + b

sbn a, b, <line no>
a = a - b
if (a < 0)
goto <line no>

ISA Completeness

Sbn a, b, < line no>

a = a - b
if (a < 0)
goto line no.

1. Add a, b, c
2. Sub b, a, 2.
3. Sbn a, b, 1
4. Mul b, a, 2

Add a, b

Add number from 1 - 10.

Single Instruction ISA - II

* Add the numbers – 1 … 10

```
Initialization:
      one  = 1
      index = 10
      sum = 0
```

```
1: sbn temp, temp, 2        // temp = 0
2: sbn temp, index, 3       // temp=-10
3: sbn sum, temp, 4         // sum += index
4: sbn index, one, exit     // index -= 1
5: sbn temp, temp, 6        // temp = 0
6: sbn temp, one, 1         // (0 - 1 < 0), hence goto 1
```

exit

temp = 0

temp = temp – index = –10

sum = sum – temp = 0 + 10 = 10

index = index – one = 10 – 1 = 9

temp = temp – temp = 0

temp = temp – one

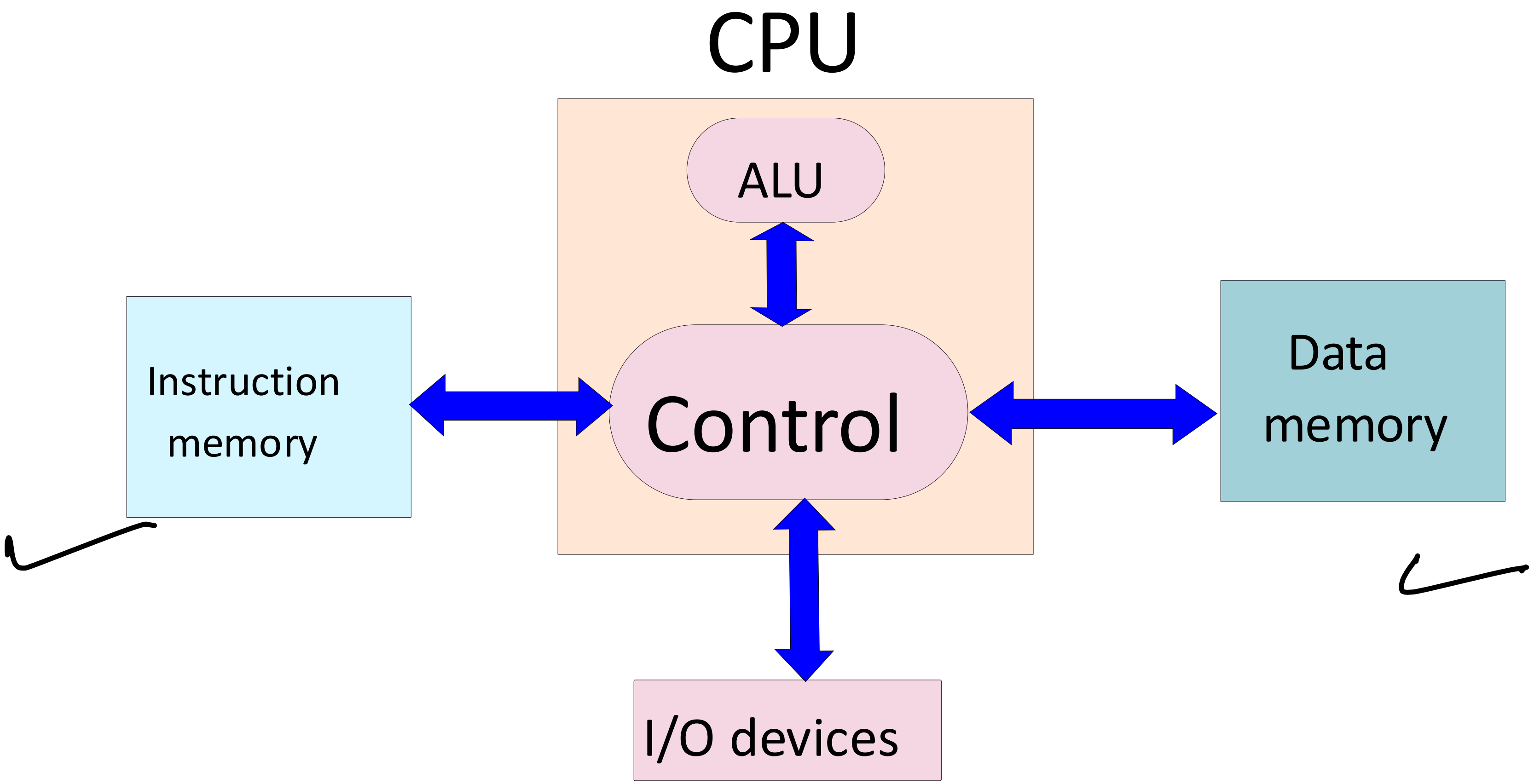
= 0 – 1 = –1

Multiple Instruction ISA

* Arithmetic Instructions
  * add, subtract, multiply, divide
* Logical Instructions
  * or, and, not
* Move instructions
  * Transfer values between memory locations
* Branch instructions
  * Move to a new program location, based on the values of some memory locations

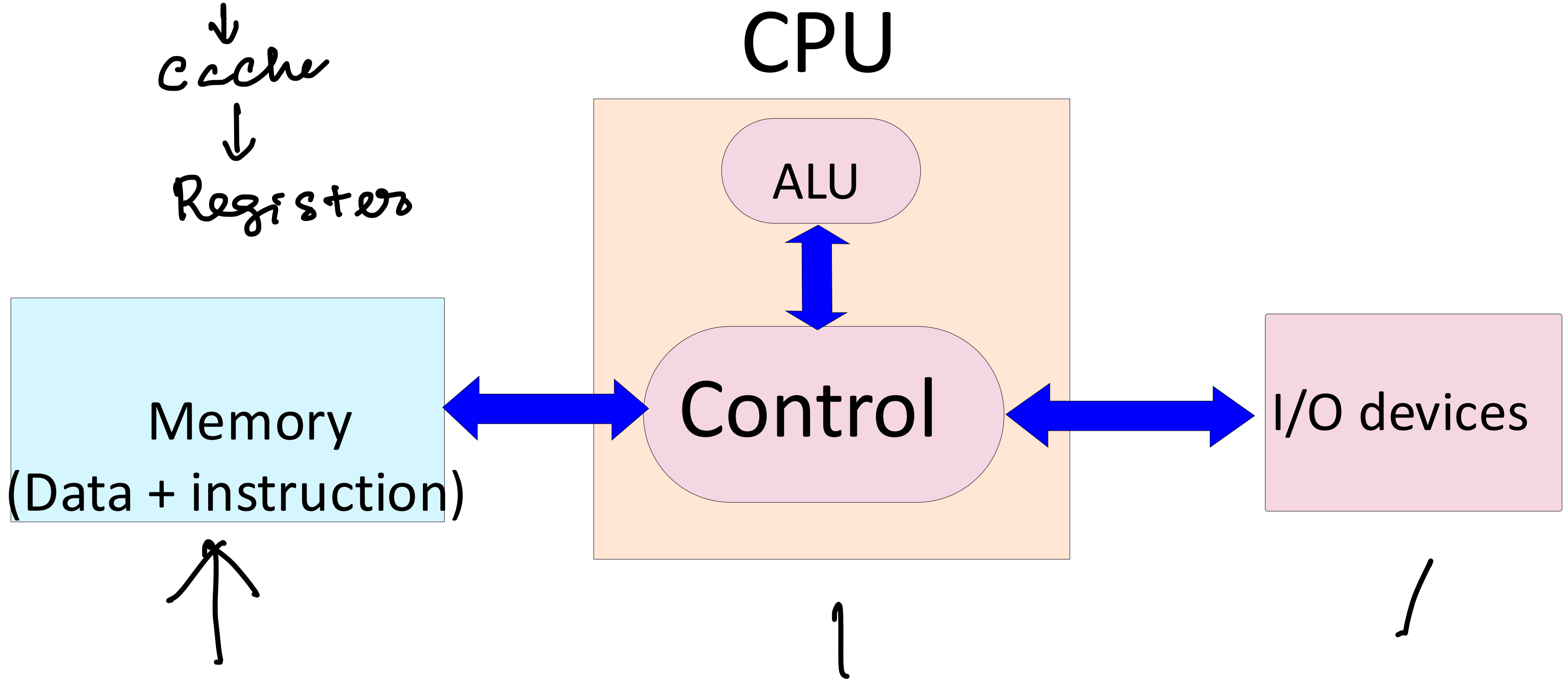
Designing Practical Machines

Harvard Architecture



Von-Neumann Architecture

Hard disk
↓
RAM
↓
cache
↓
Registers



Problems with Harvard/ Von-Neumann Architectures

* The memory is assumed to be one large array of bytes
  * It is very very slow

General Rule: Larger is a structure, slower it is

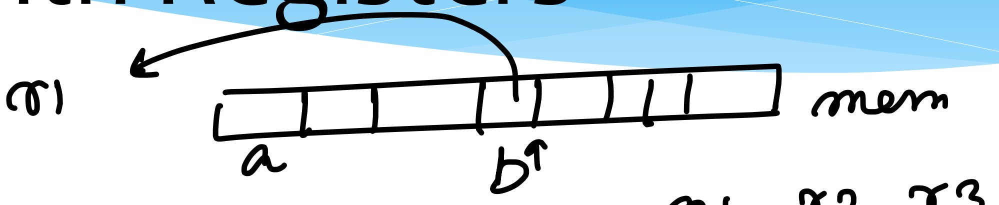
* Solution:
  * Have a small array of named locations **(registers)** that can be used by instructions
  * This small array is very fast

Insight: Accesses exhibit locality (tend to use the same variables frequently in the same window of time)

Uses of Registers

- A CPU (Processor) contains set of registers (16-64)
- These are named storage locations.
- Typically values are loaded from memory to registers.
- Arithmetic/logical instructions use registers as input operands
- Finally, data is stored back into their memory locations.

Example of a Program in Machine Language with Registers

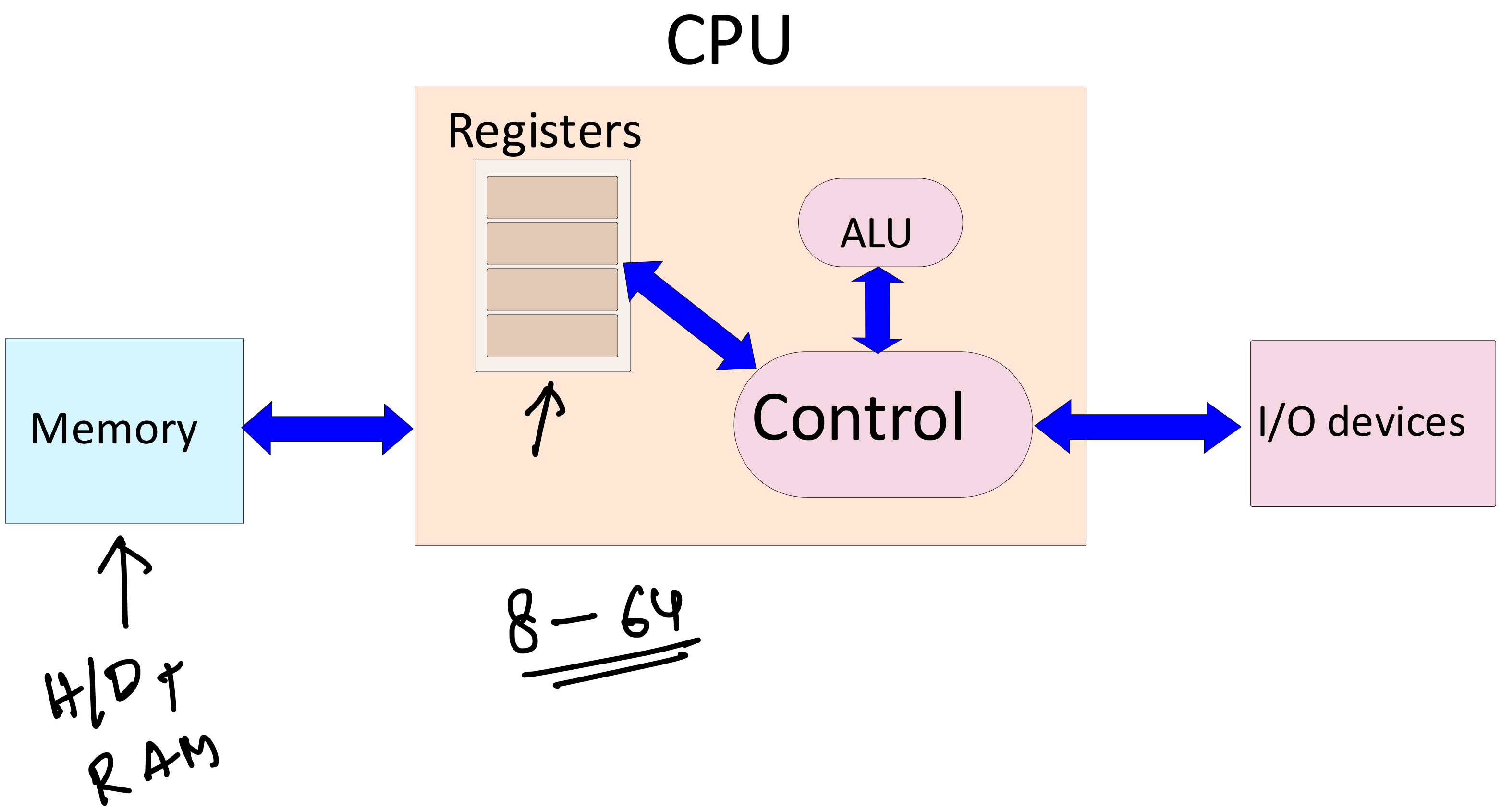


r1 r2 r3

```
1: r1 = mem[b] // load b
2: r2 = mem[c] // load c
3: r3 = r1 + r2 // add b and c
4: mem[a] = r3 // save the result
```

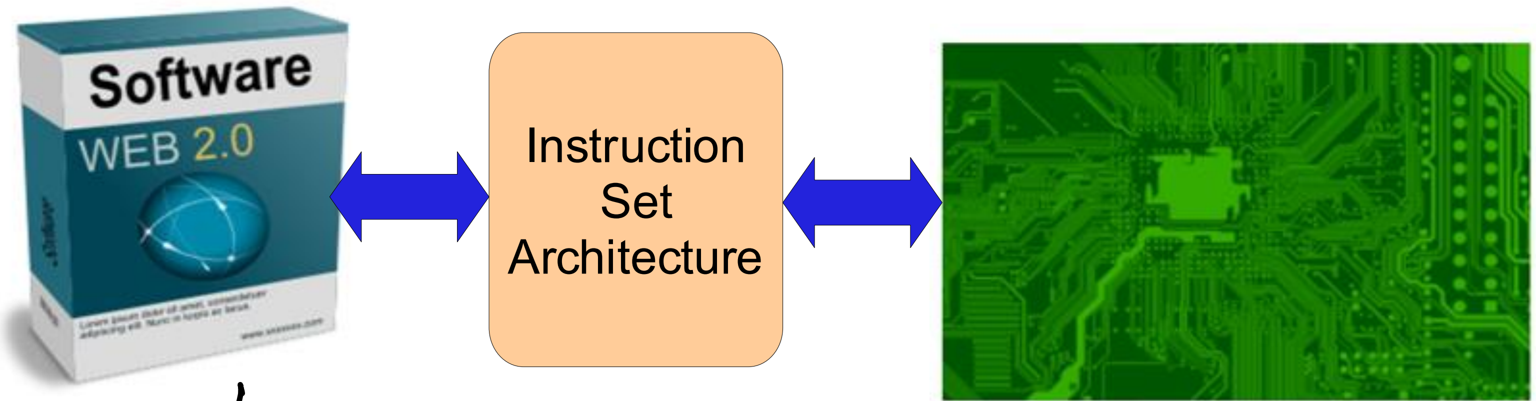
* r1, r2, and r3, are registers
* mem → array of bytes representing memory

Machine with Registers



Where are we …

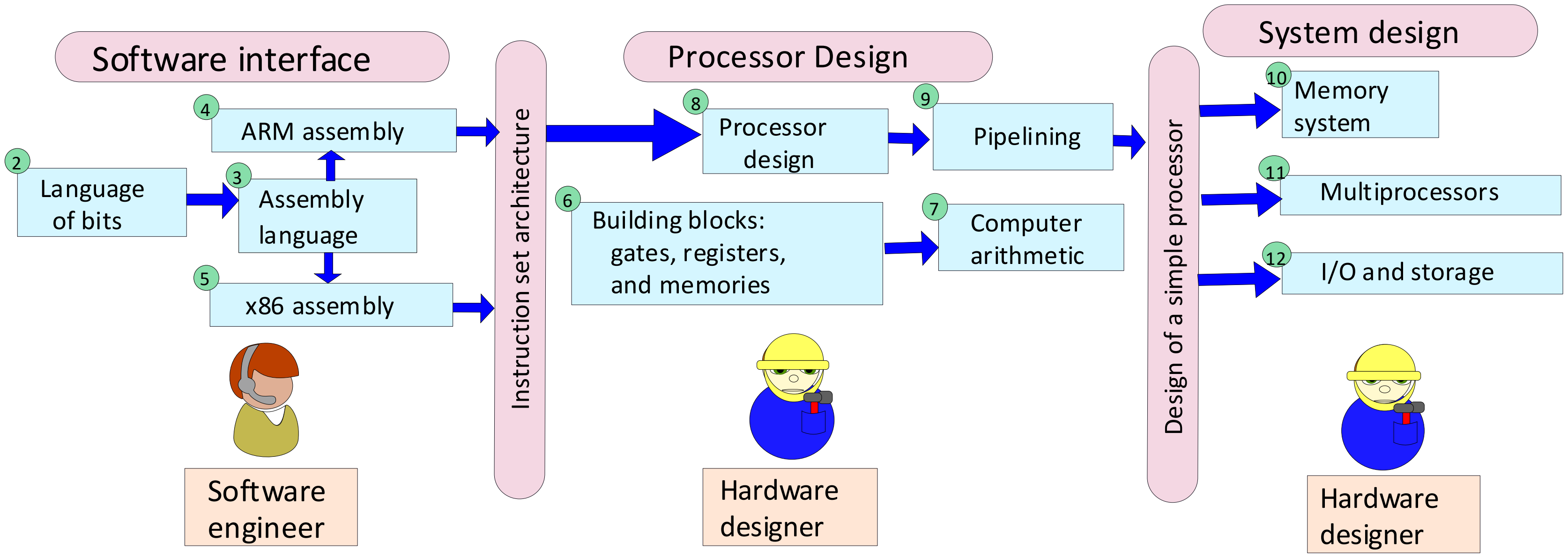
- We have derived the structure of a computer from <u>theoretical fundamentals</u>.
- It has a CPU with a program counter & registers, memory, and peripherals.
- The Instruction Set Architecture (ISA) is the link between hardware and software.



Instruction Set Architecture

* Interface between software and hardware
  * A compiler converts a program into machine instructions in the given ISA
  * The processor executes the instructions in the ISA
* We shall first look at the software aspect of the ISA (assembly programs)
* Then look at implementing the ISA by designing the processor
* Then, we shall make the computer more efficient by designing fast memory/ storage systems
* At the end, we will look at multiprocessors

Roadmap of the Course



THE END