1 Introduction

1.1 Course Introduction

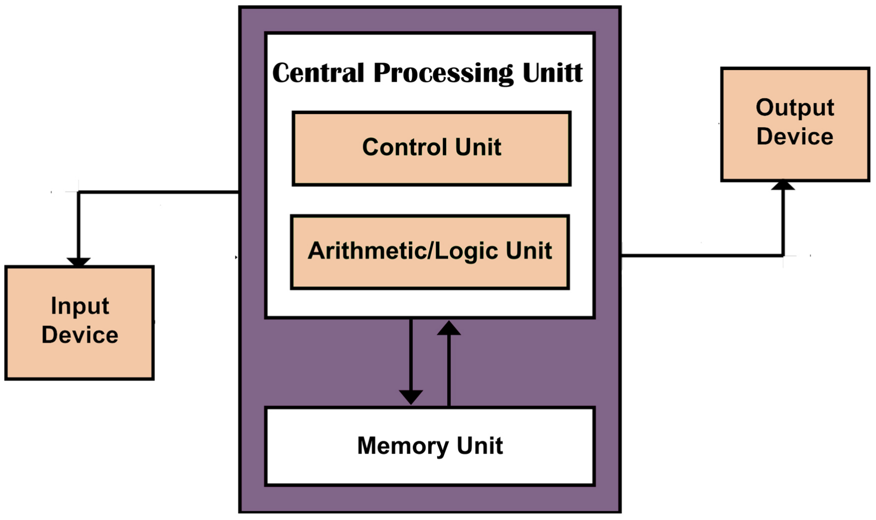
Definition (Abstraction) The process of removing physical, spatial, or temporal details or attributes in the study of objects or systems in order to focus attention on details of higher importance, it is also very similar in nature to the process of generalization.

This COURSE teaches: Programming Interface & Computer Organization.

1.2 Computer Organization

Von Neumann Architecture consists 3 main components:

- Input, Output (I/O devices)
- Memory (Memory)
- Arithmetic/Logic Unit, Control Unit (CPU, Central Processing Unit)



Von Neumann Architecture

> *Example*: How does a C program run in the Von Neumann Architecture?
>
> ```
> # include "stdio.h"
> int main() {
>  int x, y, z;
>  scanf("%d%d", &x, &y);
>  z = x + y;
>  print("%d\n", z);
>  return 0;
> }
> ```
>
> 1. Load the program in the *Memory Unit*.
> 2. *Control Unit* go through each statement sequentially:
>   1. Read the input numbers x and y from the *Input Devices* into the *Memory Unit*.
>   2. The *Arithmetic/Logic Unit* calculate the $x + y$, and store the result z in the *Memory Unit*.
>   3. Print the output number z to the *Output Device*.

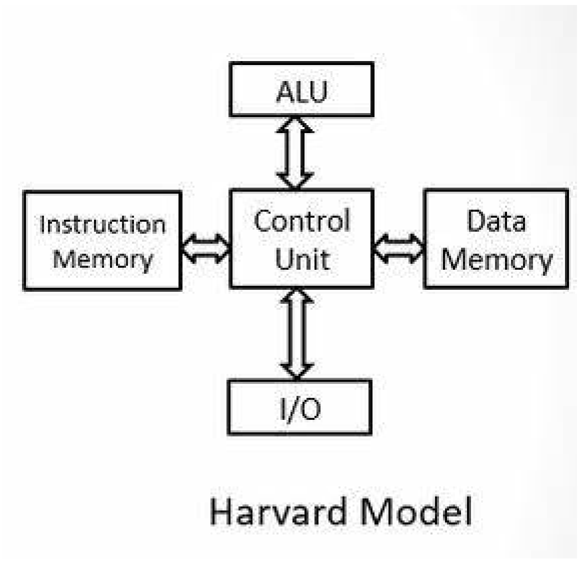
Key Abstractions of Von Neumann Architecture

- Data: Both instructions and data are stored in a *single* read-write memory.
- Instruction: The contents of memory are *addressable* by location, without regard to the type of data.
- Sequential execution model

> *Example*: The C programming language is just a higher level abstraction of Von Neumann architecture.
>
> - Data $\longleftrightarrow$ Variable
> - Instruction $\longleftrightarrow$ statement/code
> - Sequential execution model $\longleftrightarrow$ Sequential execution model (*in C programming language*)
> - More abstractions in C programming languages: function calls etc.

Interactions between the components and abstractions:

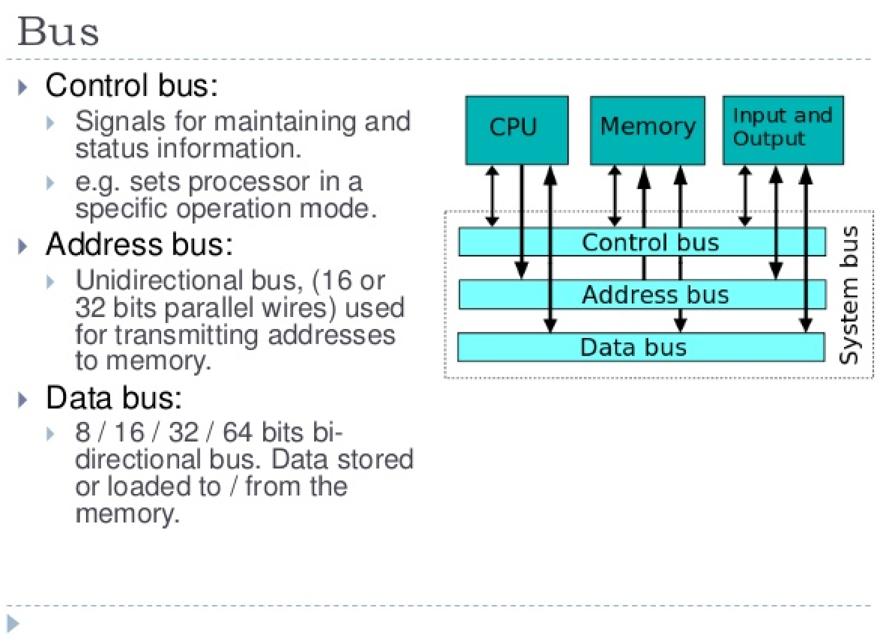
- Data and instructions are stored in the memory.
- CPU executes instruction in sequential order.
- Instruction can read and write memory.
- Instruction can perform arithmetic operations.

> *Example*: Another architecture: Harvard Architecture.
>
> 
>
>
> Harvard architecture separates Instruction Memory and Data Memory.
>
> **Advantages of Harvard Architecture**
>
> - Efficient Pipelining - Operand Fetch and Instruction Fetch can be overlapped.
> - Separate Buses for data and instructions.
> - Tailored towards an FPGA implementation.
>
> **Disadvantages of Harvard Architecture**
>
> - Not widely used.
> - More difficult to implement.
> - More pins.

Microcomputer

- **CPU**: processes information stored in the memory.
- **Memory**: stores both instructions and data.
- **Input/Output ports**: provide a means of communicating with the CPU.

- **BUS**: (*wires*) interconnecting all parts together. (Address bus, Data bus, Control bus)



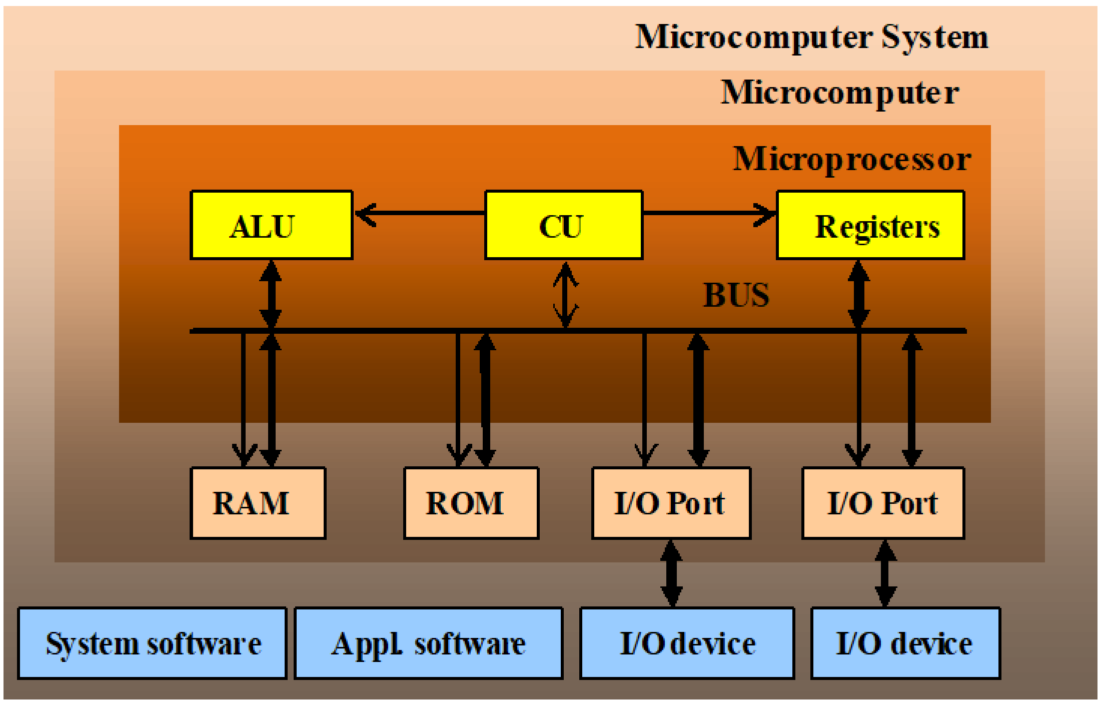
Bus

- Control bus:
  - Signals for maintaining and status information.
  - e.g. sets processor in a specific operation mode.
- Address bus:
  - Unidirectional bus, (16 or 32 bits parallel wires) used for transmitting addresses to memory.
- Data bus:
  - 8 / 16 / 32 / 64 bits bi-directional bus. Data stored or loaded to / from the memory.

CPU
Memory
Input and Output
Control bus
Address bus
Data bus
System bus

Microcomputer System

- Microcomputer
- Peripheral I/O devices
- **Software**
  - System Software: Interacting with hardware, such as OS, compilers, drivers.
  - Application Software: Interacting with users, such as Word, Excel, MATLAB.



1.3 CPU (Microprocessor)

> 4 layers of CPU structure:
>
> - Instruction Set Architecture (ISA)
> - Logical Gate

> - Transistor
> - Sand

Key concepts of CPU

1. **Core(s)** (each core includes ALU, CU, register) (*2005 - now: multicore*)
2. **Clock**
3. **ISA** (Intel x86/ARM/MIPS/RISCV)

ALU: multifunctional calculator.

- Arithmetic functions (add, substract, ...) and Logic functions (AND, OR, ...).
- Two inputs.
- Calculation result can be temporarily stored in one of registers.

Instruction: a pre-defined code which defines a specific operation, processing and exchanging information among CPU, memory and I/O devices.

CU: works under *instruction*, contains an *instruction decoder* and *program counter*.

- *instruction decoder*: decodes an instruction and generates all control signals.
- *program counter*: points to the address of the next instruction to be executed.

IS: Instruction Set

- *CISC: Complex Instruction Set Computer*: friendlier to programmer.
  - Variable instruction length (1 ~ n words)
  - Variable execution time of different format instructions.
  - More instruction formats.
  - Upwardly compatible (new instruction set contains earlier generation's instruction)
- *RISC (Reduced Instruction Set Computers)*: easier to implement.
  - Fixed size (1 word).
  - Fixed time for all instructions.
  - Easy to pipeline in RISC instructions (fast)
  - Fewer formats (simple hardware, shorter design cycle)