Freie Universitat Berlin
Erasmus Program

10 ECTS

Computer Architecture

Professor:
Larissa Groth

Autor:
Filippo Ghirardini

Winter Semester 2024-2025

Contents

Computer Architecture

Autor: Ghirardini Filippo

Winter Semester 2024-2025

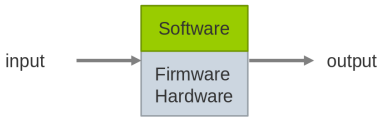
---

1 Introduction

1.1 Computer

A computer is made of three components:

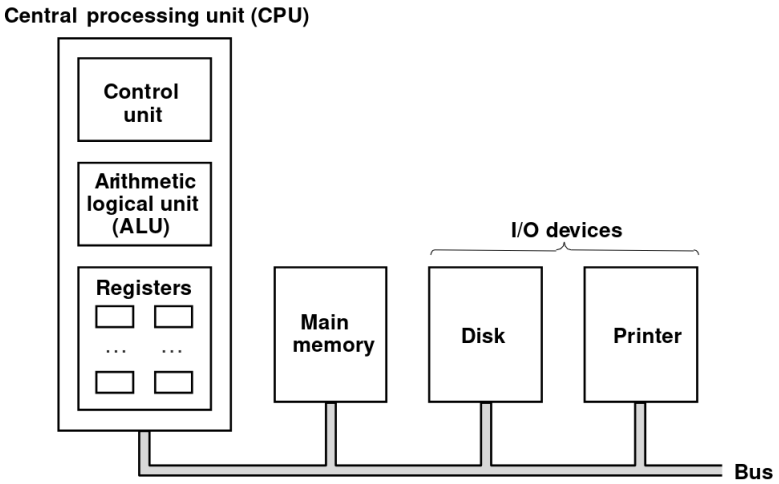
- **Hardware**: mechanical and electronic components
- **Software**: programs running on it
- **Firmware**: micro-programs stored in ROM



Note 1.1.0.1. Logical equivalence between HW and SW.

1.1.1 Micro-Computer

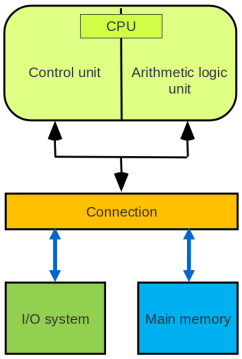
A micro-computer classical architecture is composed of different **modules**, each one dedicated to a feature (and to a corresponding physical part). They're all connected through a **bus**.



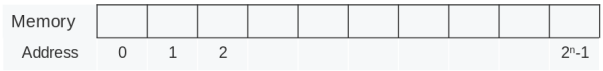
1.2 Architecture

1.2.1 Von Neumann

This architecture is one of the most wide spread. It has the following components:



- **Central Processing Unit**: controls the flow and the execution of all instructions. Consists of:
  - **Control Unit**: interprets the instructions and generates the control commands for other components
  - **Arithmetic Logical Unit**: executes the instructions, if needed with I/O and memory modules
- **Memory**: storage of **data** and **programs** as sequences of bits. It consists of memory cells with a fixed length, each one accessed individually with an address.

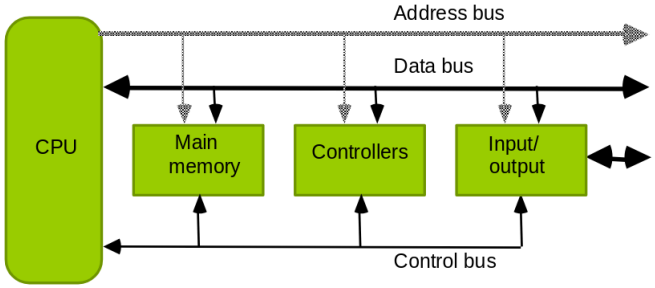


- **Input/Output** units: interface to the outside world that inputs and outputs the data

- **Interconnection**

  Observation 1.2.1 (Von Neumann Bottleneck)**.** Since there is no difference in memory between instructions and data and the CPU is way faster than the bus and the memory, the main interconnection is the central bottleneck.

IBM PC The IBM PC is a modified von Neumann architecture and was introduced by IBM fall 1981. The interconnection structure was realized by several busses.



Principle of operation This architecture follows the **Single Instruction Single Data** (SISD): at any time the CPU executes only a single instruction that can only manipulate a single operand.

There are **no memory protection** mechanisms, meaning that programs can destroy each other and access arbitrary data. Memory is **unstructured** and is addressed linearly. Interpretation of memory content depends on the context of the current program only.

Instruction processing follows a two phase principle:

- **Interpretation**: the content of a memory cell is fetched based on a program counter. This content is then interpreted as an instruction.
- **Execution**: the content of a memory cell is fetched based on the address contained in the instruction. This content is then interpreted as data.

Note 1.2.1.1. The instruction flow follows a strict **sequential** order.

Pros and cons The advantages of this architecture:

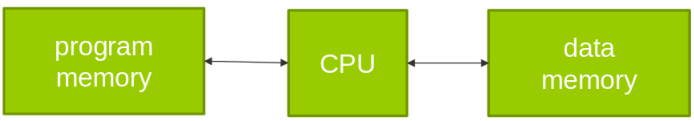
- Principle of minimal HW requirement
- Principle of minimal memory requirement

The disadvantages:

- Von Neumann bottleneck, also influencing programming approaches to deal with that
- Low structuring of data
- The instruction determines the operand type

1.2.2 Harvard

The main idea in this architecture is that there is a separation between data and program memory. Typically a processor uses this internally and then externally has a Von Neumann.



1.3 Performance

Definition 1.3.1 (Moore's Law)**.** *The number of transistors in an integrated circuit doubles about every two years.*

Observation 1.3.1 (Overclocking)**.** Given a certain processor with a clock speed, we can tune it up. The main consequence is that the energy used increases drastically, leading to higher and higher temperatures.

We need to assess the performance of a computer for different reasons: selecting a new one, tuning it o designing it. There are different methods to do so:

- Evaluation of **hardware parameters**, such as
  - Hypothetical maximum performance
  - **MIPS** o **MOPS**: millions of instructions/operations per second
  - **MFLOPS**: millions of floating point operations per second
  - **MLIPS**: millions of logical interferences per second
- **Run-time measurements** of existing programs through **benchmarks** that do different tasks (e.g. number crunching)
- **Monitoring** real cases of users through HW and SW monitoring
- Model **theoretic** approach with analytical methods or SW simulations

Benchmarks Here some example of benchmarks:

- **Sieve of Erathostenes**: uses the Ackermann function
- **Whetstone**: from 1970, uses FORTRAN with a lot of floating point arithmetic
- **Dhrystone**: from the begin of 1980s, focused on integers
- **Savage-Benchmark**: with mathematical standard functions
- **Basic Linear Algebra Subprograms** (BLAS), which is the core of the Linear Algebra Package
- **Lawrence Livermore Loops** for vectorizing compilers
- **SPEC**: Standard Performance Evaluation Corporation is a non profit organization created in 1989 that involves a lot of chip vendors. They focus on calculation speed and throughput.

1.4 Instruction Set Architecture

This type of architecture comprises the description of **attributes** and **functions** of a system from a machine language programmer's point of view. It enables the use of programs independent on the actual implementation. It defines:

- Instruction set
- Instruction formats
- Addressing modes
- Interrupt handling
- Logical address space
- Register and memory model accessible by the programmer

Note 1.4.0.1. It does not describe internal operations, components and implementation.

1.5 Micro Architecture

This type of architecture comprises the description of **hardware structure**, **data paths**, and **internal logic** of a specific realization of a processor. It defines:

- Number and stages of pipelines
- Usage of super scalar technology
- Number of ALUs
- Organization of cache memory

Definition 1.5.1 (Binary compatibility)**.** *All microprocessors following the same processor architecture specification are called* ***binary compatibles****.*

1.6 Classification

To classify different architectures we use the type of supported **parallelism**.
A parallel program can be seen as a partially ordered set of instructions, where the order is given by the dependencies among them. Independent instructions can be executed in parallel.
We have five levels of parallelism:

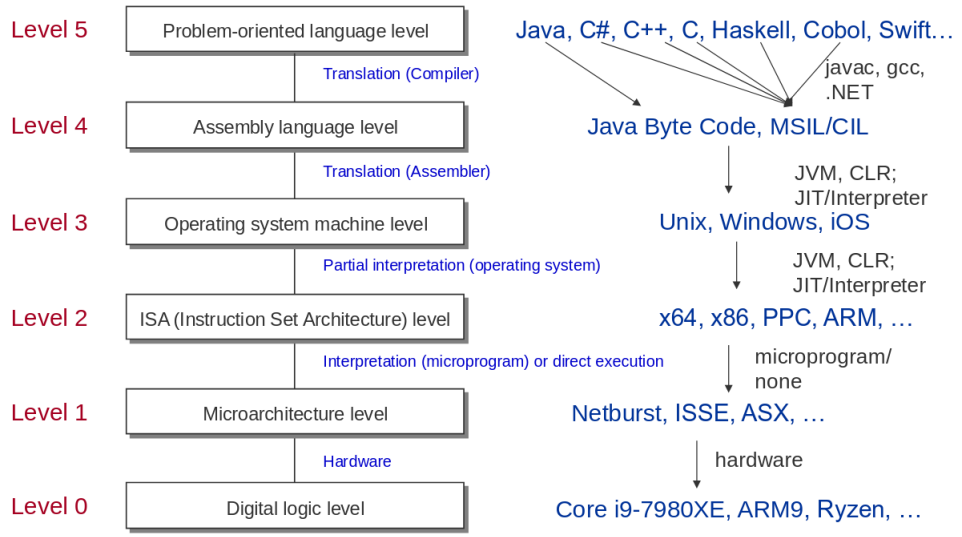
1. **Program**
2. **Process** or **task**: it involves heavy weighted processes or coarse-grained tasks
3. **Block**: it involves threads or light-weighted processes
4. **Instruction**: about the basic instructions of the chosen language or instruction set
5. **Sub-Operation**: basic instructions of a language or instruction set can be divided into sub-operations/micro operations by a compiler/the processor

Granularity The granularity depends on the relation of computation effort over communication or synchronization effort. Typically we have:

- **Large** grained: program, process, and thread level
- **Fine** grained: instruction and sub-operation

Note 1.6.0.1. Sometimes, the block or thread level is considered as **medium** grained.

1.7 6-Level model



2 Data arithmetic

2.1 Number systems

Definition 2.1.1 (Positional)**.** *Representation of numbers as a sequence of digits z_i, with the radix point between z_0 and z_{-1}:*

$$-z_n z_{n-1} \dots z_1 z_0 . z_{-1} z_{-2} \dots z_{-m} \tag{1}$$

Each position i of the sequence of digits is assigned a value, which is a power b_i of the base b of the numbering system:

$$X_b = z_n \cdot b^n + z_{n-1} \cdot b^{n-1} + \ldots + z_1 \cdot b^1 + z_0 \cdot b^0 + z_{-1} \cdot b^{-1} + \ldots + z_{-m} \cdot b^{-m} = \sum_{i=-m}^{n} z_i \cdot b^i \tag{2}$$

Common bases of numbering systems used in computer science are:

Base (b)	Number system	Alphabet
2	Binary	$0, 1$
8	Octal	$0, 1, 2, 3, 4, 5, 6, 7$
10	Decimal	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
16	Hexadecimal	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$

2.2 Conversion FROM decimal

2.2.1 Euclid

Considering the following representation of a number that we want to convert in a base b:

$$\begin{aligned} Z &= z_n \cdot 10^n + z_{n-1} \cdot 10^{n-1} + \ldots + z_1 \cdot 10 + z_0 + z_{-1} \cdot 10^{-1} + \ldots + z_{-m} \cdot 10^{-m} = \\ &= y_p \cdot b^p + y_{p-1} \cdot b^{p-1} + \ldots + y_1 \cdot b + z_0 + y_{-1} \cdot b^{-1} + \ldots + y_{-q} \cdot b^{-q} \end{aligned}$$

we generate the digits step by step starting with the most significant (leftmost):

1. Search p according to the inequation

   $$b^p \leq Z < b^{p+1}$$

   and assign $i = p$ and $Z_i = Z$

2. Derive y_i and the reminder R_i by division of z_i by b^i

3. Repeat step 2 for $i = p - 1, \ldots$ and replace each step Z_i with R_i until $R_i = 0$ or b^i is small enough that the precision is enough

2.2.2 Horner

This method is structured in two phases:

1. **Integer** part: factoring out the integer we get

   $$X_b = \sum_{i=0}^{n} z_i \cdot b^i = ((\ldots(((z_n \cdot b + z_{n-1}) \cdot b + z_{n-2}) \cdot b + z_{n-3}) \cdot b \ldots) \cdot b + z_1) \cdot b + z_0$$

2. **Decimal** part: we multiply the decimals of the number by base b to get the fractional digits y_{-i} from the most to the least significant position

   $$Y_b = \sum_{i=-m}^{-1} y_i \cdot b^i = ((\ldots(((y_{-m} \cdot b^{-1} + y_{-m+1}) \cdot b^{-1} + y_{-m+2}) \cdot b^{-1} + y_{-m+3}) \cdot b^{-1} \ldots) \cdot b^{-1} + y_{-1}) \cdot b^{-1}$$

2.3 Conversion TO decimal

We represent the values of the single positions of the number we want to convert in our common decimal system and sum all values.
The value X_b of the number is the sum of all single values of all positions $z_i \cdot b^i$, like in equation (2).

2.4 General conversion

If we want to convert from an arbitrary system to another, we first convert to decimal and then we use one of the other two methods.

Observation 2.4.1. If the base of one system is a power of the base of another system the conversion is done by Replacing a sequence of digits by a single digit or replace a digit by a sequence of digits, respectively.