Structure and functionality of computers – an overview

Topics:

- Levels of abstraction
- Compilers and interpreters
- Von-Neumann architecture
- How does a processor work?

Different Views on a Computer

External view (application):

„*Computer architecture*" deals with the functional behavior of a computer system as viewed by the programmer" [Murdocca, Heuring]

Internal view (implementation):

„*Computer organisation*" deals with structural relationships that are not visible to the programmer, such as interfaces to peripheral devices, the clock frequency, and the technology used for the memory" [Murdocca, Heuring]

Levels of abstraction

Both views can be applied on various levels of abstraction:

Top level:	high-level programming languages
Bottom level:	circuit (transistors, wires)

Model of a computer

Levels of abstraction and languages

High-level languages
C++, Java, ...

Operating system level
additional services (e.g. memory management, file IO)

Assembler
symbolic notation of machine instructions

Instruction set level (machine instructions)
lowest visible level; instructions consist of sequences of 0‘s and 1‘s

Micro program level
Instructions controlling control signals

Circuit level
hardware

Each level has it‘s own language!

Virtual machine

- Each level is represented by a ***virtual machine*** M_i, with an corresponding ***language*** L_i .

- A program written in language L_i will either
  - be ***compiled*** or
  - be ***interpreted***

  into language L_j ($j < i$)

- The virtual machine M_i makes the user believe that his program (written in L_i) is executed directly by the hardware

Levels of abstraction and languages

High-level language

a = b + 3;

translate

Assembler language

```
LOAD ACC 14     // load b into accumulator
ADDI 3          // add 3 to accumulator
STORE ACC 13    // store result in a
```

translate

Machine language

```
1001 0001 0000 0000 0000 0000 0000 1110
1101 0000 0000 0000 0000 0000 0000 0011
1000 0001 0000 0000 0000 0000 0000 1101
```

Circuit

„voltage on“, „voltage off“

Compiler versus Interpreter

Compilation

- A ***L_i-compiler*** a software program, which reads a program written in language L_i and returns an equivalent program in language L_j ($j < i$)
- If L_j is the machine language, we call the compiler a ***L_i-full compiler***.

Execution of a L_i-program

- Translate into equivalent L_j-program ($j < i$)
- Execute L_j-program (on level j)

Interpretation

cmd := first instruction of L_i-program;

repeat

translate *cmd* into sequence *cmd2* of commands of language L_j;

execute *cmd2* (on level j);

cmd :=next instruction of L_i-program;

until L_i-program executed;

Compiler versus Interpreter

- Interpreters are easier to build than compilers

  well suited for prototype implementations of new languages

- Compiled programs require more memory

  Memory for program in higher level language plus memory for interpreter < memory for program in machine code (for large programs)

- Interpreted programs can be independent of hardware platform

  e.g. internet-applications (Javascript)

- Compiled programs execute faster

  Code optimizations during compile time

Performance ratios

Higher level languages:	Runtime/response time of a program
Machine language level:	Millions of instructions per second (MIPS) Millions of floating-point operations per second (MFLOPS)
Circuit level:	Cycle time

Hardware, Software, Firmware

Some notations

- Hardware — electronic circuits, I/O-devices, memory, ...
- Software — all programs running on a computer
- Firmware — programs being used for starting the computer for testing and for control tasks. In general embedded during fabrication, stored in read-only memories (ROM)

Hardware vs. Software

In the early days implemented in software, today in hardware:

- Integer multiplication, division
- Floating point arithmetic
- ...

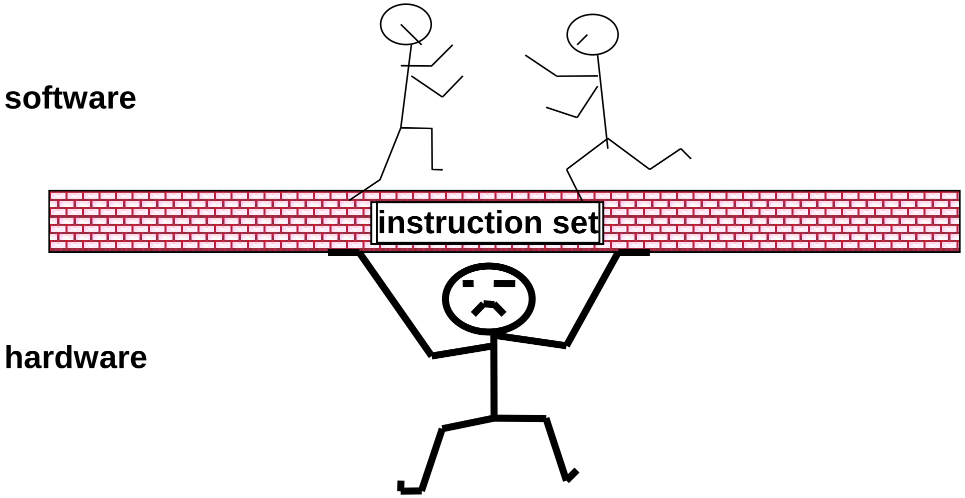
But also: RISC-concept (Reduced Instruction Set Computer)

- Simple but fast hardware, enables pipelining
- Complex operations in software

example: integer division in RISC-processors

Hardware vs. Software

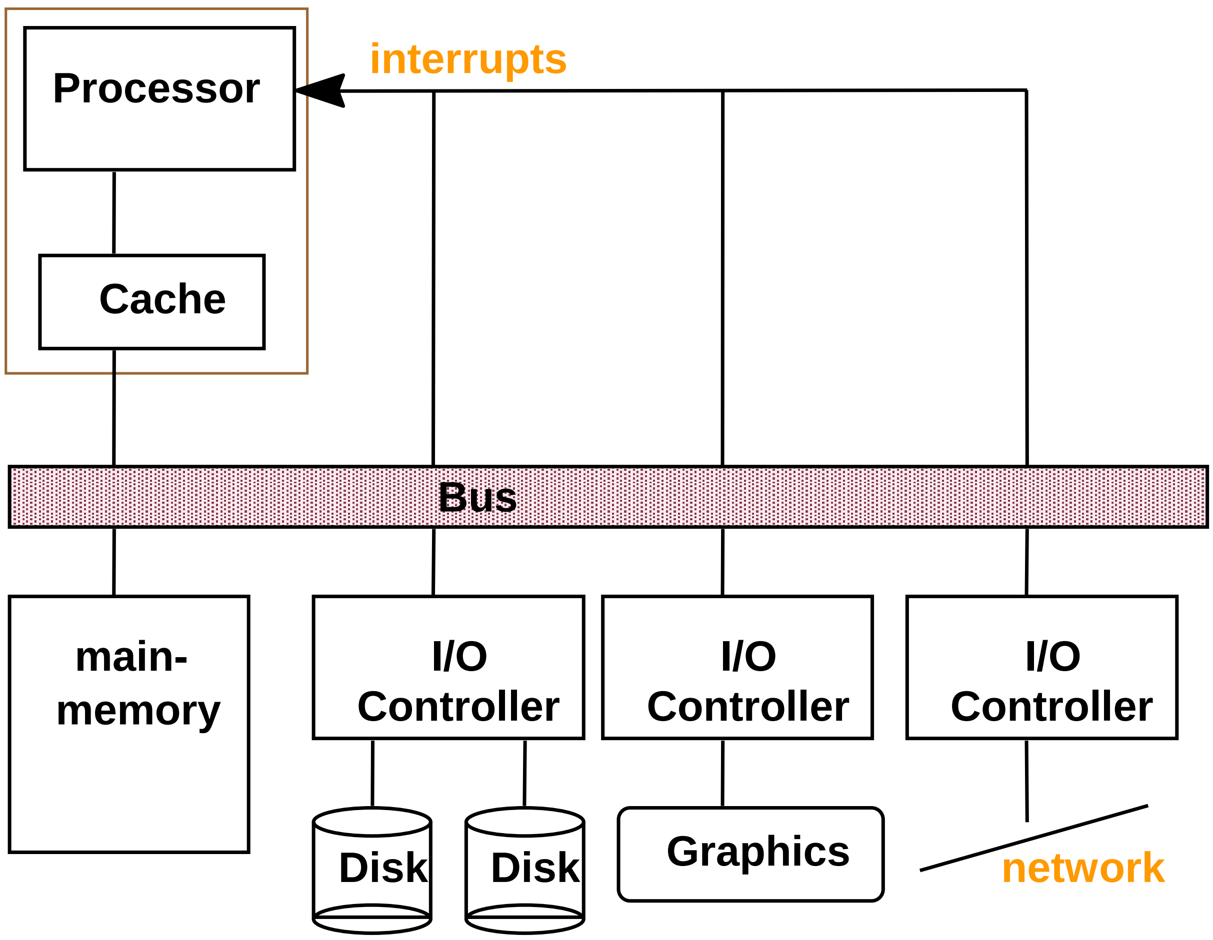
It depends on the instruction set whether a particular operation is implemented in hardware or has to be implemented in software by the programmer (or the compiler).



Example:
Realize division in hardware or in software?

Question of costs and performance!

Basic structure of a computer



Components:

- Processor (CPU)
- Main memory
- External memory
- Input devices (keyboard, mouse)
- Output devices (monitor,
- Busses

Model can be simplified further: von Neumann's architecture model

John von Neumann (1903 - 1957)

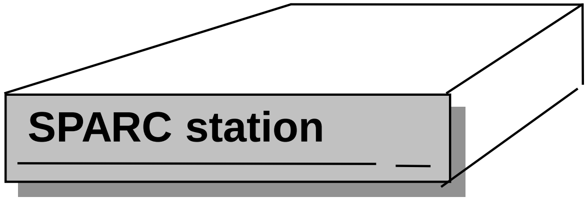
The architecture of modern computers has been influenced by the mathematician John von Neumann.

The von Neumann principle can be characterized as follows:

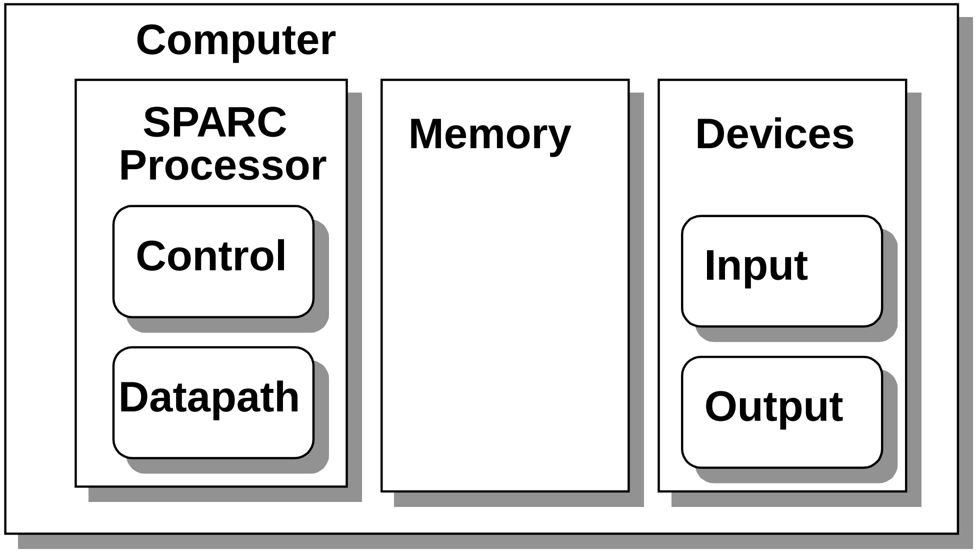
- Electronic computer
- Binary data representation
- Arithmetic logic unit
- Control unit
- Internal data and program memory

John von Neumann, 1932

Von Neumann-Computer

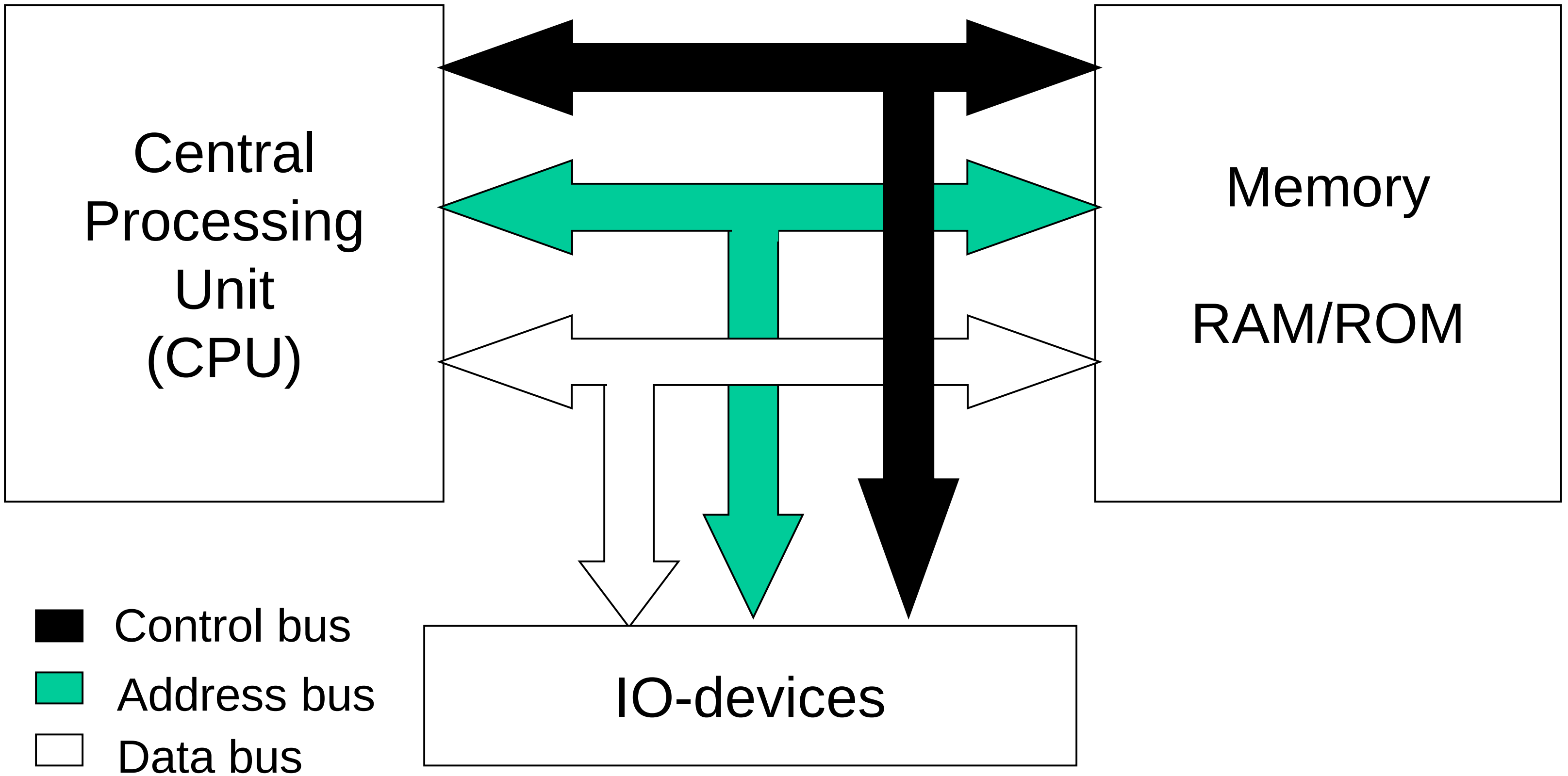


Major components of a von Neumann-computer:
- Processor (control unit and data path)
- Memory
- Input/Output



Bus Model

Another representation of the von Neumann-model



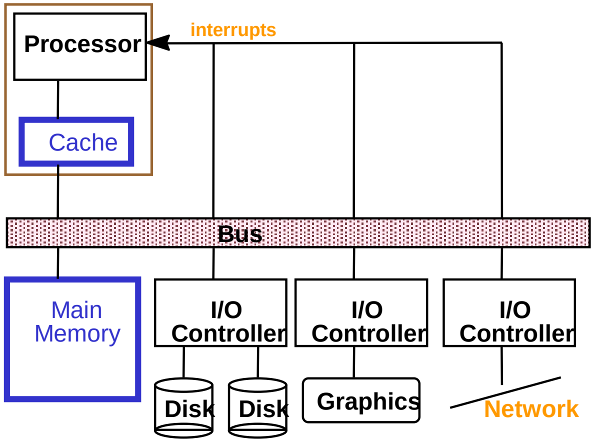
Options for a bus

Bus:
- Bundle of wires, each wire carries one binary signal (1 bit)
- Grouped after function (control bus, data bus, address bus)
- Standards for interpreting the signals on the wires (PCI, SCSI, ...)

- Using separate wires for address and data is faster
- Wide is faster
- Multiple Bus masters (arbiter needed)
- Synchronous transmission

- Multiplexing address and data signals is cheaper
- Narrow is cheaper
- Only one single bus master (no arbiter needed - cheaper)
- Asynchronous transmission

Memory



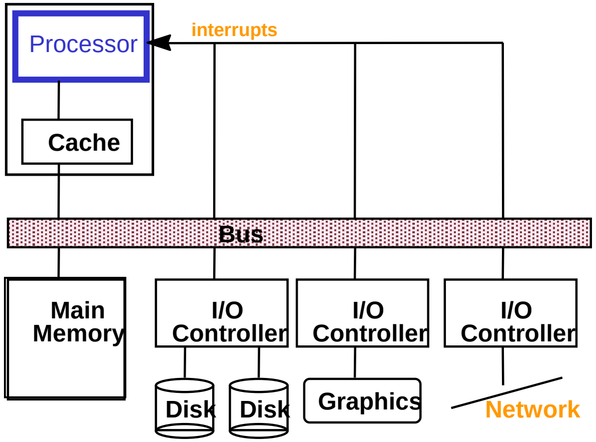
- Allows to store information (data and programs)
- Consists of *n* memory cells
- Each *memory cell*
  - Is the smallest addressable unit of the memory
  - Stores information consisting of *k* bits. *k* is called the *word size of the memory* (*k*=8,16,32,64)
  - Is assigned a unique *address* within the range [0... *n* -1]

Characterized by
- number of cells (words)
- word size
- access time

One **Bit** is the smallest amount of information that can be stored. Each bit can take one of two values, either 0 or 1.

One **Byte** consists of 8 Bit.

Processor



Control Unit

Sets control signals for:

- Fetching next instruction from main memory and storing it in instruction register
- Decoding instruction
- Executing instruction

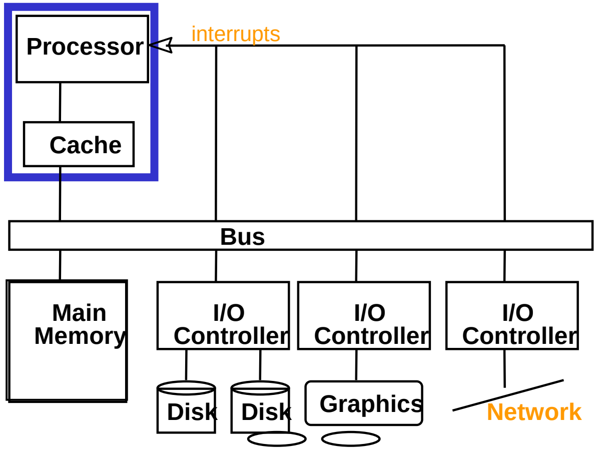
Data path (Arithmetic Logic Unit (ALU))

- Performs operations (e.g. comparison, addition, …) corresponding to the controlling signals set by the control unit

Register file

- Contains fast storage elements (registers)

Registers of a processor

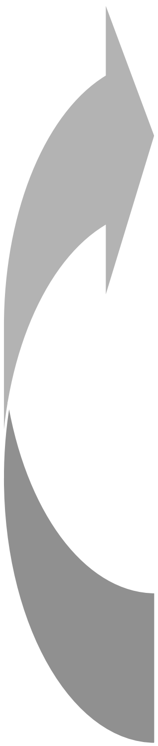


Data required by the processor:

- Instructions to be executed → **Instruction register (IR)**
- Address (in main memory) of the instruction to be executed next → **Program counter (PC)**
- Data to be processed by the ALU → **General purpose register**

Phases of an instruction

Fetch-Decode-Execute-Cycle



Fetch	fetches the next machine instruction from memory and stores it in the instruction register. The address of the instruction is found in the program counter.
Decode	decode instruction; load data required for execution
Execute	execute instruction and write back result

Phases of an instruction

Example: **Instruction: Add R_x and R_y , store the sum into R_z**

Step 1: **fetch instruction** ADD from memory and put it into the instruction register

Step 2: **decode instruction**

instruction register contains ADD instruction,
source operands are R_x, R_y,
target operand is R_z,

Step 3: **execute instruction**

send R_x, R_y to ALU,
instruct ALU to perform an addition,
add R_x to R_y
send result from ALU to R_z