

# Programming Computers

Wolfgang Schreiner  
Research Institute for Symbolic Computation (RISC)  
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at  
<http://www.risc.jku.at>

# Computer Systems

## Hardware

A computer has two core components.

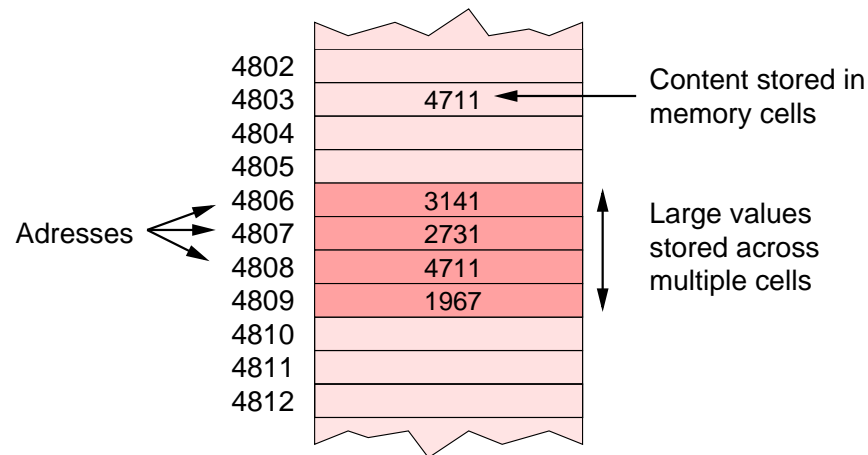
- Processor
  - Central processing unit (CPU).
- Main memory
  - Random access memory (RAM).

The processor executes a program that is stored in main memory and operates on data that are stored in main memory.

# Main Memory

- **Random Access Memory:**

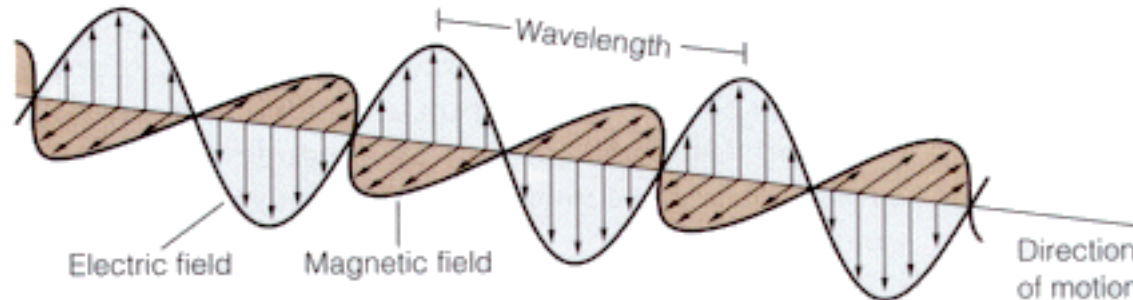
- Sequence of memory cells.
- Each cell holds a single natural number (the **content**).
- Each cell is addressed by a unique natural number (the **address**).
- Each cell can be independently read or written (random access).



## Digital Computers

Today's computers are digital.

- Digital: data are represented by discrete pieces.
  - Denoted by natural numbers: 0, 1, 2, 3 ...
- Analog: data are represented by continuous signals.
  - For instance: amplitude of electromagnetic waves.



## Character Encodings

- **ASCII**: American Standard Code for Information Interchange.
  - $128 = 2^7$  characters (letters and other symbols).
  - Represented by numbers  $0, \dots, 127$ .
  - Also non-printable characters: **LF** (line-feed).

ASCII code	Character
...	...
10	<b>LineFeed (LF)</b>
...	...
48–57	0–9
...	...
65–90	A–Z
...	...
97–122	a–z
...	...

## Character Encodings

There are various standards for character encodings.

- **ASCII** contains  $128 = 2^7$  characters.
  - First international standard.
  - Designed for representation of the English language.
- **ISO 8859-1** contains  $256 = 2^8$  characters.
  - Latin 1: characters of western-European languages.
  - First 128 characters coincide with ASCII standard.
- **Unicode** contains  $65534 = 2^{16} - 2$  characters.
  - Characters of most languages world-wide.
  - First 256 characters coincide with ISO 8859-1 standard.

The **ASCII** characters are the same in all encodings.

## Text

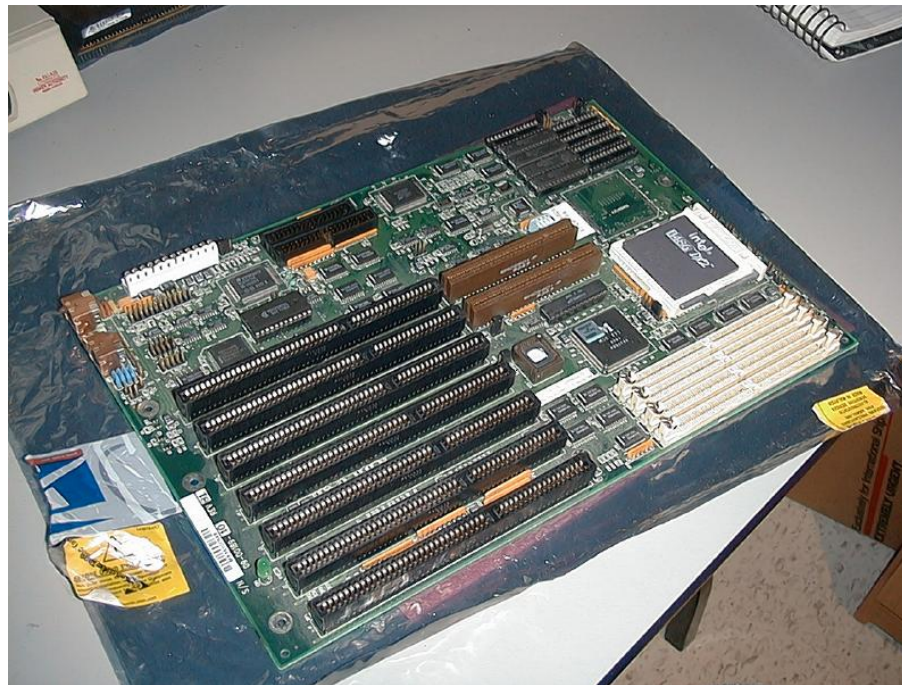
Text is a sequence of characters.

```
H i , H e a t h e r .  
72 105 44 32 72 101 97 116 104 101 114 46
```

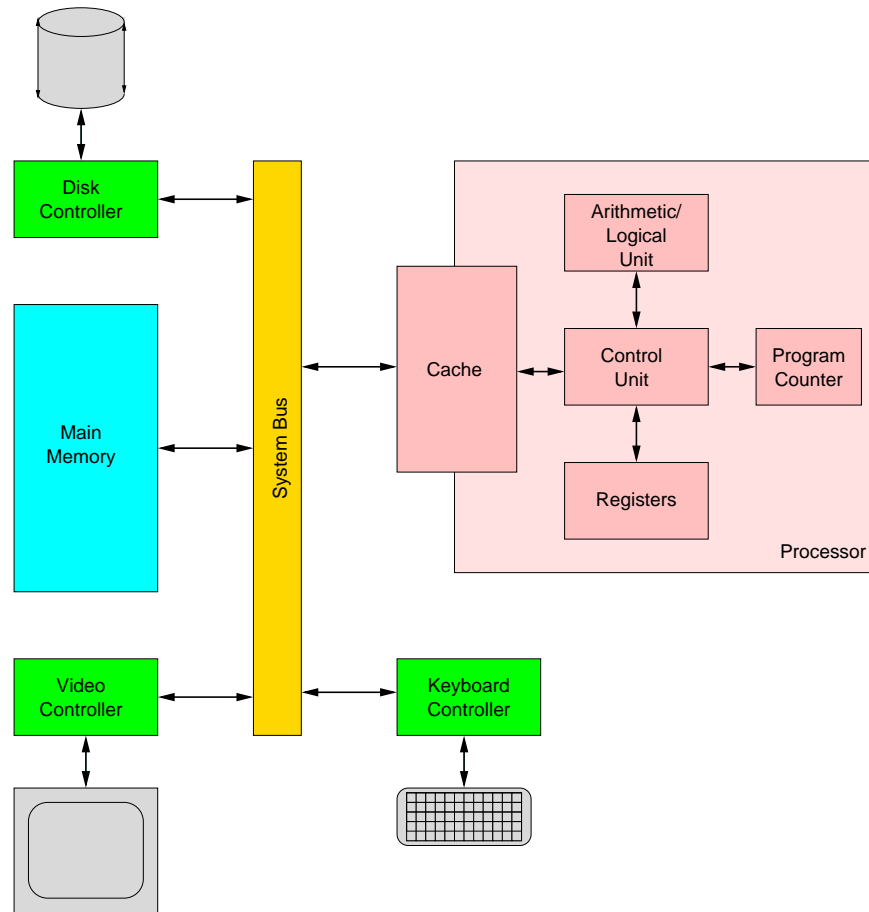
Text is encoded as a sequence of numbers.

## Processor

- Fetches **instructions** from memory and executes them.
- Processor and memory together sit on **main board**.



# Basic Computer Architecture



## Processor Architecture

- Control unit (CU):
  - Coordinates processing steps.
- Arithmetic/logical unit (ALU).
  - Performs calculations and makes decisions.
- Registers:
  - Small and very fast auxiliary memory within processor.
- Cache:
  - Fast buffer memory between processor and main memory.

## Program Execution

A program is a sequence of instructions stored in main memory.

- Each instruction is encoded by a computer word
  - Instruction: “load the word at address 100 into register 1”.
  - Symbolic name: `load r1,100`
  - Memory representation: 

17	1	0	100
----	---	---	-----
- Each instruction consists of multiple components.
  - 17: operation code for load.
  - 1: register 1.
  - (0, 100): 16 bit address 4 (100 binary).

In computer memory, programs are just data.

## Processor Instructions

`load  $r$ ,  $a$`

Load the word at memory address  $a$  into register  $r$ .

`store  $a$ ,  $r$`

Store the word from register  $r$  into the memory at address  $a$ .

`$op$   $r1$ ,  $r2$`

perform an arithmetic operation  $op$  on registers  $r1$  and  $r2$  and store the result in  $r1$ .

`jump  $p$`

Set the program counter to address  $p$ .

`cjmp  $r$ ,  $c$ ,  $p$`

Compare the content of register  $r$  with constant  $c$ . If the comparison is successful, set the program counter to address  $p$ , otherwise do nothing.

## Operation Principle

- **Program counter:**
  - Register which holds the address of the next instruction to be executed.
- **Processor operation:**
  - Continuous cycle of steps coordinated by the control unit.
- **Execution cycle:**
  1. fetch the instruction referenced by the program counter from the main memory,
  2. decode the instruction and increment the program counter.
    - PC references the next instruction in sequence,
  3. execute the instruction.

Von Neumann principle.

## Example

Exponentiation: compute  $a^b = a * a * \dots * a$  ( $b$  times).

- Inputs  $a$  and  $b$  are stored in words at addresses 100 and 104.
- Result  $a^b$  is to be stored at address 108.
- Program:

```
0: cnst r0, 1      load constant 1 into register r0
4: load r1, 100    load memory word a into register r1
8: load r2, 104    load memory word b into register r2
12: cjmp r2, 0, 28 if r2 is 0, jump to instruction 28
16: mult r0, r1    multiply r0 with r1 and store result in r0
20: diff r2, 1     subtract 1 from r2 and store result in r2
24: jump 12       jump to instruction 12
28: stor 108, r0   store register r0 as result
```

## Computation of $3^2$

step	pc	r0	r1	r2	100	104	108
0	0				3	2	
1	4	1			3	2	
2	8	1	3		3	2	
3	12	1	3	2	3	2	
4	16	1	3	2	3	2	
5	20	3	3	2	3	2	
6	24	3	3	1	3	2	
7	12	3	3	1	3	2	
8	16	3	3	1	3	2	
9	20	9	3	1	3	2	
10	24	9	3	0	3	2	
11	12	9	3	0	3	2	
12	28	9	3	0	3	2	9

# Programming Languages

## Programming

- **Machine program** is sequence of binary encoded instructions:

17	0	0	100
32	0	1	0
21	0	1	105

...

- Programs are actually not written in this style (any more):
  - Assembly language.
  - High-level languages, e.g., Java.

Since the 1950s, several generations of programming languages.

## Assembly Language

- Symbolic notation for machine programs:

```
0: cnst r0, 1
4: load r1, 100
8: load r2, 104
12: cjmp r2, 0, 28
16: mult r0, r1
20: diff r2, 1
24: jump 12
28: stor 108, r1
```

- One program line per machine instruction.
- **Assembler** constructs actual machine program.

Used for hardware access or efficiency, e.g. device drivers.

## High-Level Programming Languages

- Abstract symbolic notation:

```
int r = 1;
for (int i = 0; i < b; i++)
    r = r*a;
```

“Starting with  $r = 1$ , multiply  $b$  times  $r$  with  $a$ .”

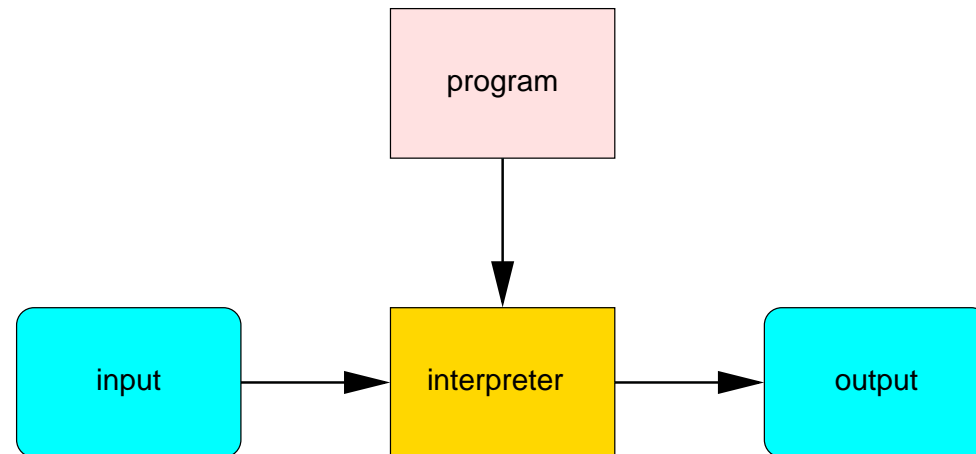
- One program line corresponds to many machine instructions.
- Various programming languages:
  - Fortran, Basic, Pascal, C, Modula-2, Simula, Ada, C++, Oberon, C#, AS3, Java, ...

How is such a high-level program executed?

## Interpretation

- **Interpreter:**

- Program that reads high-level program and executes it.
- A processor is an interpreter for machine language.



- About 100 times slower than machine program.

Used for execution of command scripts.

## Compilation

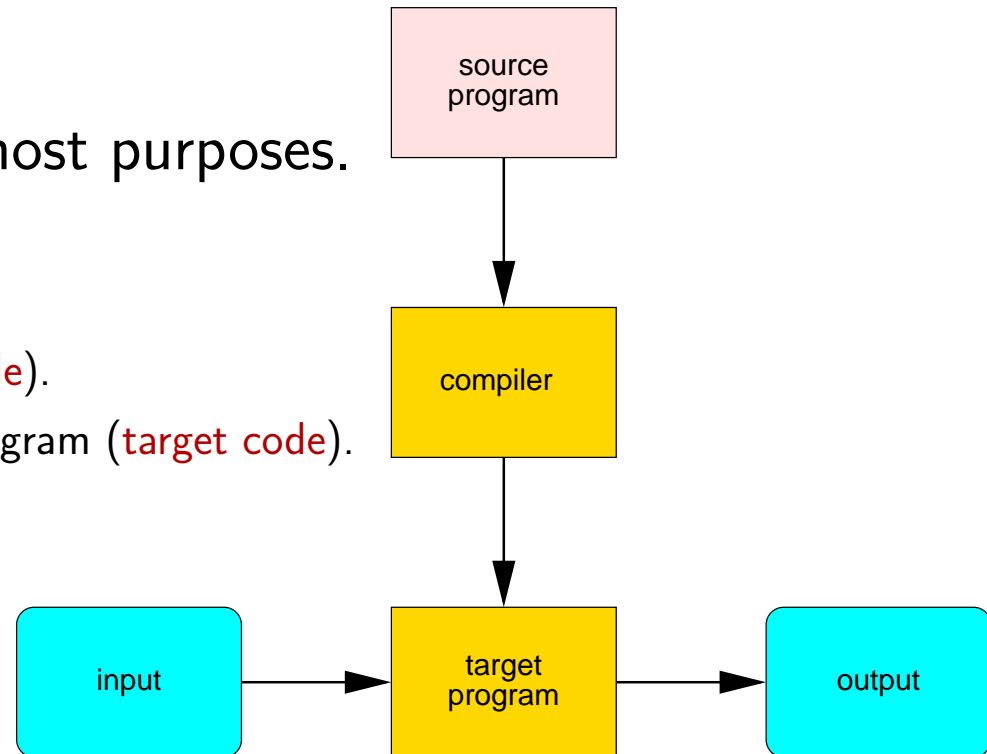
Interpretation is too slow for most purposes.

- **Compiler:**

- Reads high-level program (**source code**).
- Generates corresponding machine program (**target code**).

- **Optimization:**

- Sophisticated analysis techniques.
- Machine program rather efficient.
- Less than 30% performance loss.



Used for most real programming languages.

## Compilation: Linking

- High-level program decomposed into multiple modules.
  - Not just a single monolithic piece of code.
- **Module**: individual piece of code.
  - Modules can be independently developed and compiled.
  - Compilation of a module yields a piece of code in machine language called **object file**.
- **Linker**: program that combines object files.
  - Generation of a single executable program file.
- **Code libraries**: reuse of functionality.
  - Pre-compiled object files that can be used in many applications.
  - E.g.: input/output libraries, graphical libraries, mathematical libraries.

Program generation consists of compiling and linking.

## Compilation: Portability

- Machine program only executes on one brand of processor.
  - Intel Pentium, IBM PowerPC, Compaq Alpha, SUN Sparc, ...
- Source code has to be re-compiled/-linked for every platform.
  - Generated machine program should behave the same on every platform.
- A **portable** source program can be used for different platforms.
  - Fully portable program just need re-comilation and re-linking.
  - Not fully portable programs have to be ported (adapted) for each platform.
  - Incompatible libraries, differences in compilers, ...

Achieving portability with compiled programming languages is difficult.

## The Programming Language Java

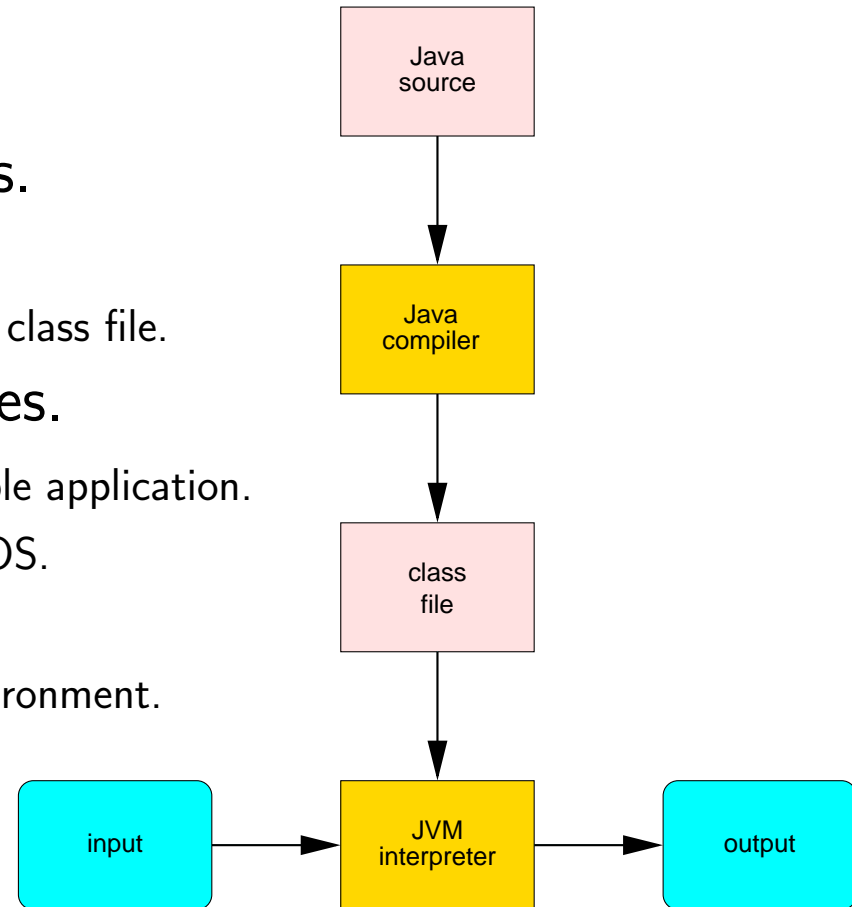
<http://www.oracle.com/technetwork/java>

- Developed in the beginning of the 1990's
  - Computer company Sun (now Oracle).
  - Gained popularity with the uprise of the World Wide Web.
  - “Java” is US slang for “coffee”.
- Idea: “write once, run everywhere”.
  - Target code should run in same way on all kinds of machines.
- Solution: Java Virtual Machine (JVM).
  - Abstraction layer between the Java programming language and the computer system.
  - Hypothetical machine that executes the JVM **byte code** language.

One of the most popular programming languages today.

## Execution of Java Programs

- Java compiler generates class files.
  - Object files in the JVM byte code language.
  - Each Java source module is compiled to one class file.
- JVM interpreter executes class files.
  - JVM dynamically links class files to executable application.
  - Defines abstract processor architecture and OS.
  - Completely hides real hardware and OS.
  - Java only uses the JVM as its execution environment.
- Interpretation overhead.
  - Byte code close to real machine code.
  - Only 2–3 times slower than machine program.



**JVM programs run in the same way on all architectures.**

## Application of Java

JVM interpretation is still too slow for many applications.

- **JIT** (Just in Time) compilation.
  - JVM interpreter dynamically compiles byte code to real machine code.
  - Execution of machine code rather than interpretation of byte code.
  - Execution overhead of Java programs has become rather small.
- Today: a versatile general-purpose programming language.
  - Application must not be extremely time-critical.
  - Application must not be too system oriented.
  - For such purposes, the programming language C++ is better.

Reasonable implementation language for most purposes.