Announcements

- Schedule for Unix/Linux seminars

  `http://www.cs.mcgill.ca/socsinfo/seminars/`

Binary to decimal conversion

- Problem: given a bit sequence $b = b_{n-1}b_{n-2}\cdots b_2b_1b_0$, compute its decimal representation $dec(b)$

- Algorithm:

$$dec(b) = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

- Examples:

$$\begin{aligned} dec(1101) &= 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 \\ &= 1\cdot 8 + 1\cdot 4 + 0\cdot 2 + 1\cdot 1 \\ &= 8 + 4 + 1 \\ &= 13 \end{aligned}$$

$$
\begin{aligned}
dec(101101) &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
&= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\
&= 32 + 8 + 4 + 1 \\
&= 45
\end{aligned}
$$

Decimal to binary conversion

- Problem: Given a natural number m, compute its bit sequence $b = b_{n-1}b_{n-2}\cdots b_2b_1b_0$ such that $m = dec(b)$

- Algorithm:

1. Let b be "" (the empty sequence)

2. Let q be m

3. While q is not 0 repeat the following:

   (a) Let *new_q* be q divided by 2, and
   (b) Let r be the remainder of q divided by 2
   (c) Append r in the front of the sequence b
   (d) Set q to be *new_q*
   (e) Repeat (from line 3)

4. Output: b

Algorithm execution

- Trace of execution
- Example: Consider the case $m = 26$

iteration	q	new_q	r	b
0	26			""
1	13	13	0	"0"
2	6	6	1	"10"
3	3	3	0	"010"
4	1	1	1	"1010"
5	0	0	1	"11010"

Elements of algorithms

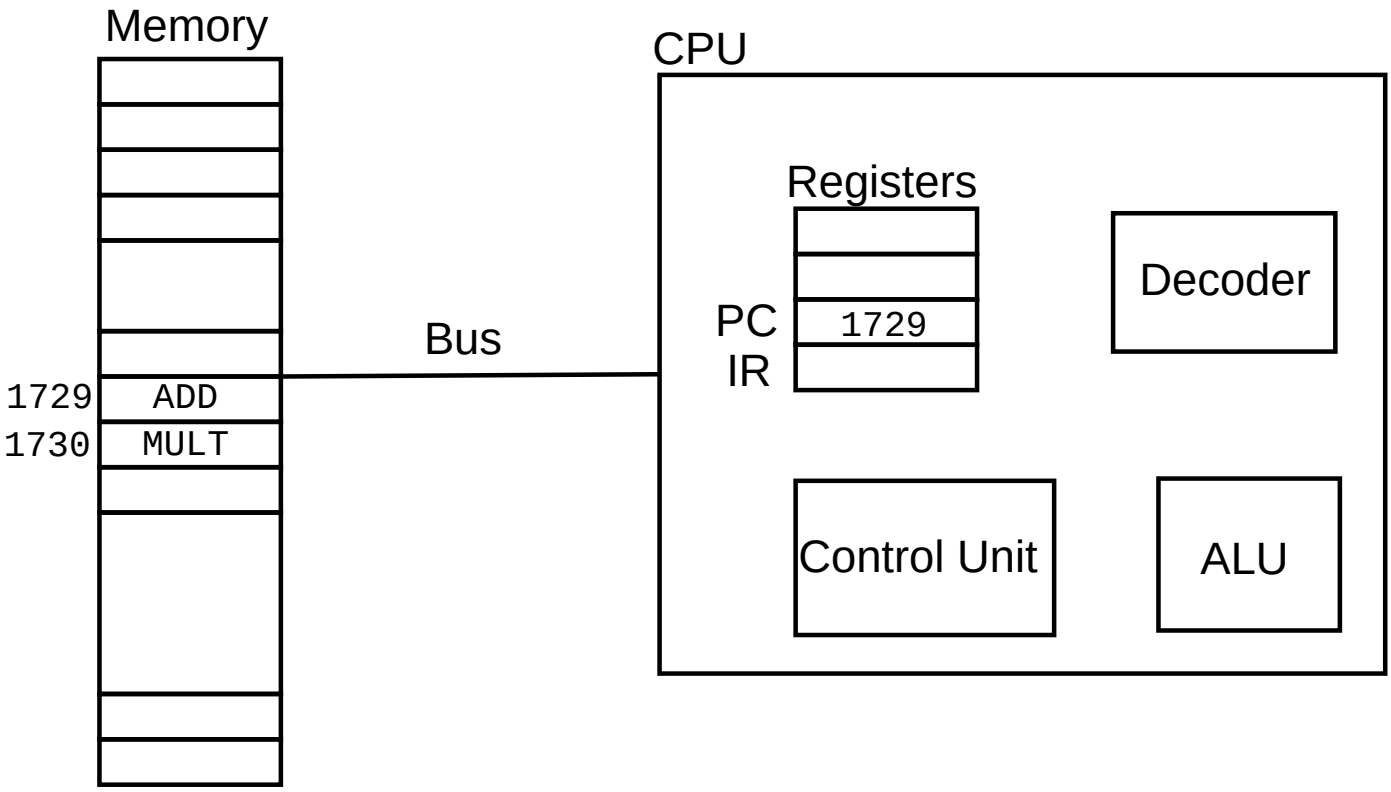
- Variables to store values (such as numbers, sequences, etc.)
- Instructions organized and executed in sequence: order of execution matters
- Instructions for:
  - computing values (e.g. divide by)
  - assigning values to variables
  - repeating a set of instructions
  - etc.
- Solving a problem: (General methodology)

1. Analysis: Understanding the problem
2. Design: Algorithm
3. Implementation: write a program which implements the algorithm using a programming language

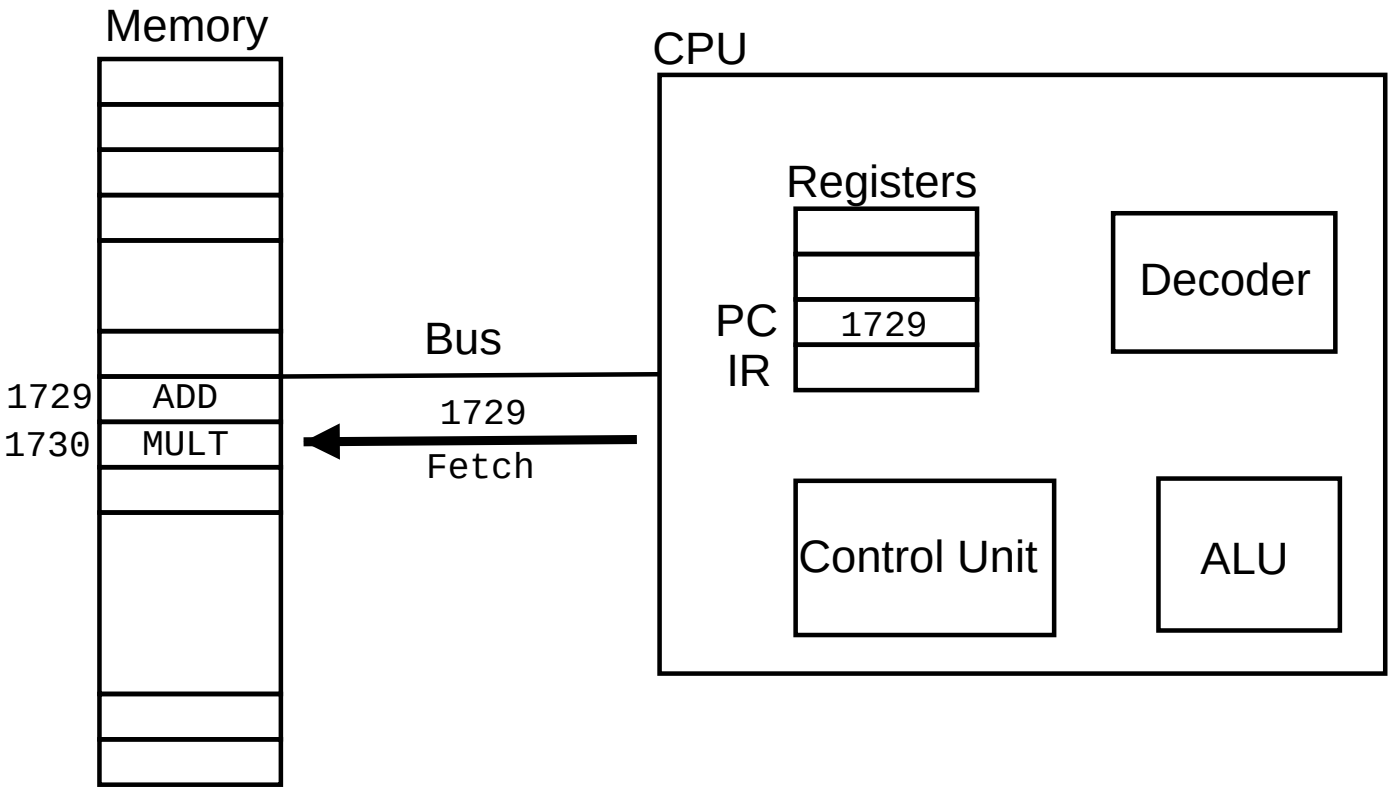
Computer Architecture

- CPU:
  - Registers (PC, IR, ...)
  - ALU (Arithmetic-Logic Unit)
  - Control Unit
  - Decoder
- Program execution:
  - Fetch: get an instruction
  - Decode: sent it to the appropriate component
  - Execute

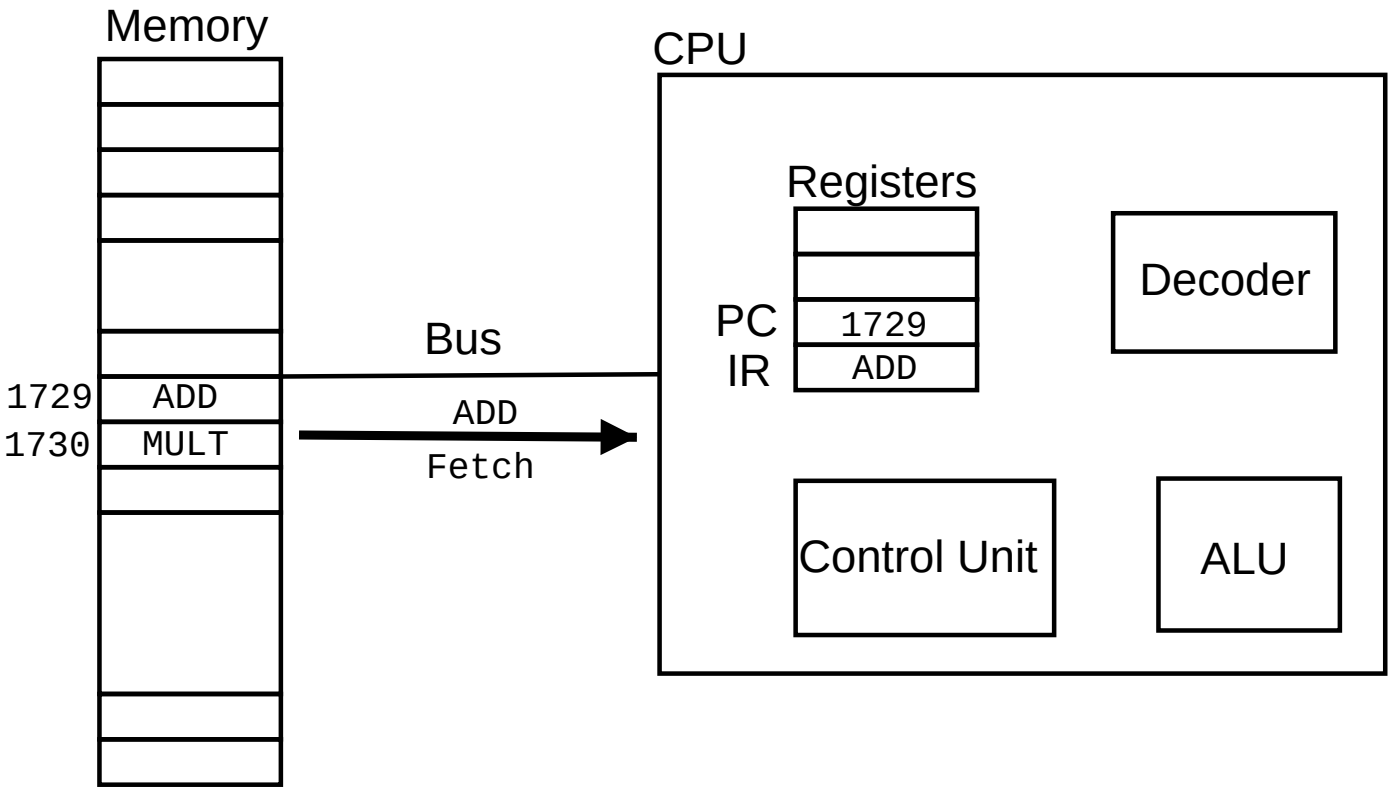
Computer Architecture



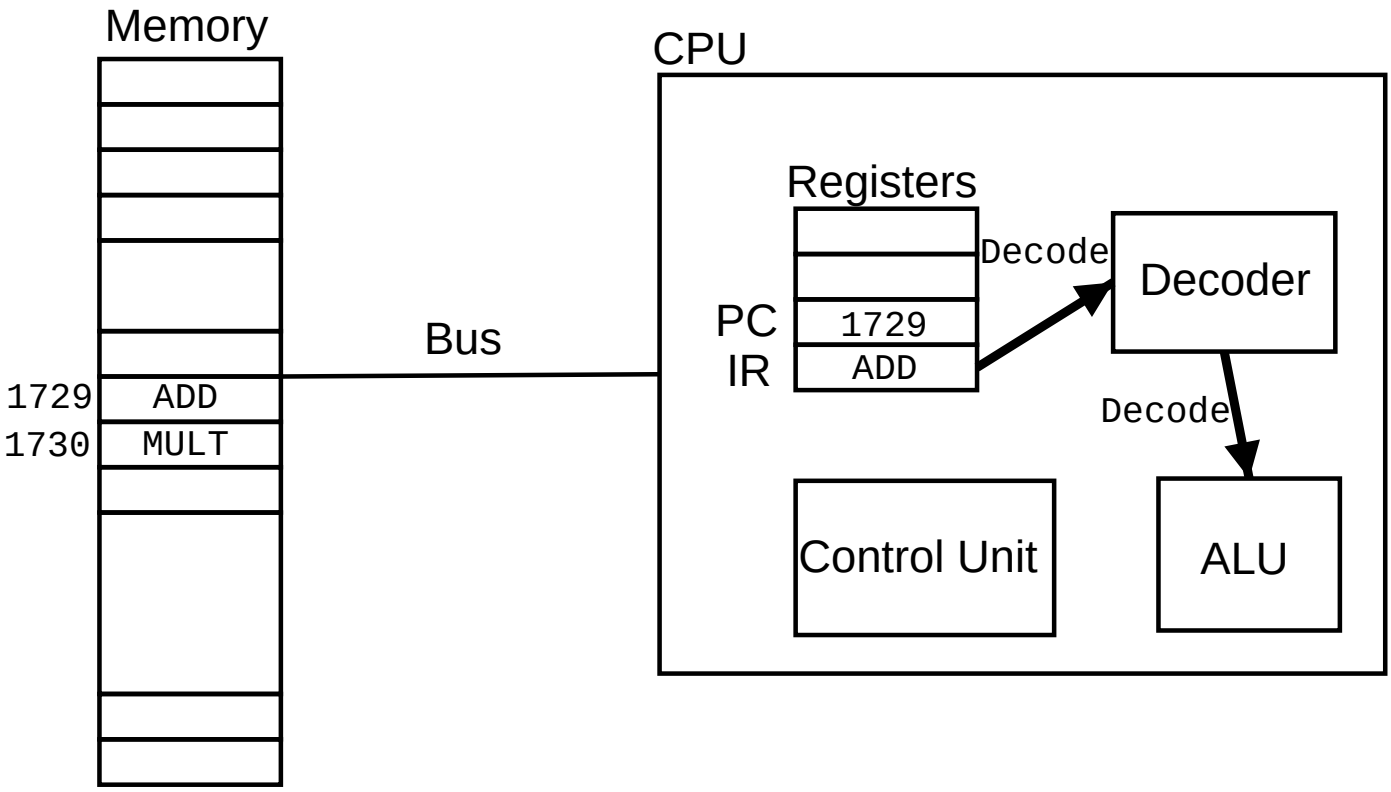
Computer Architecture



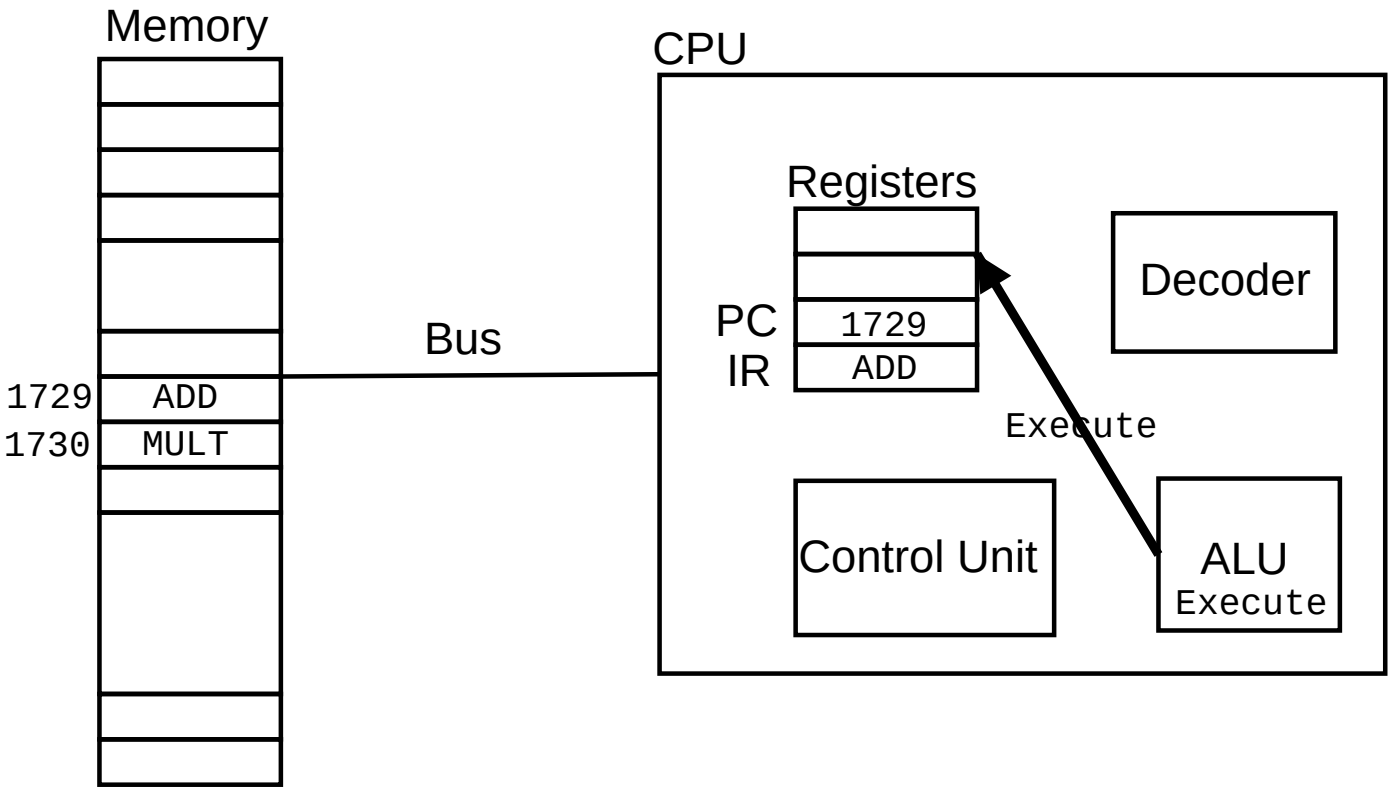
Computer Architecture



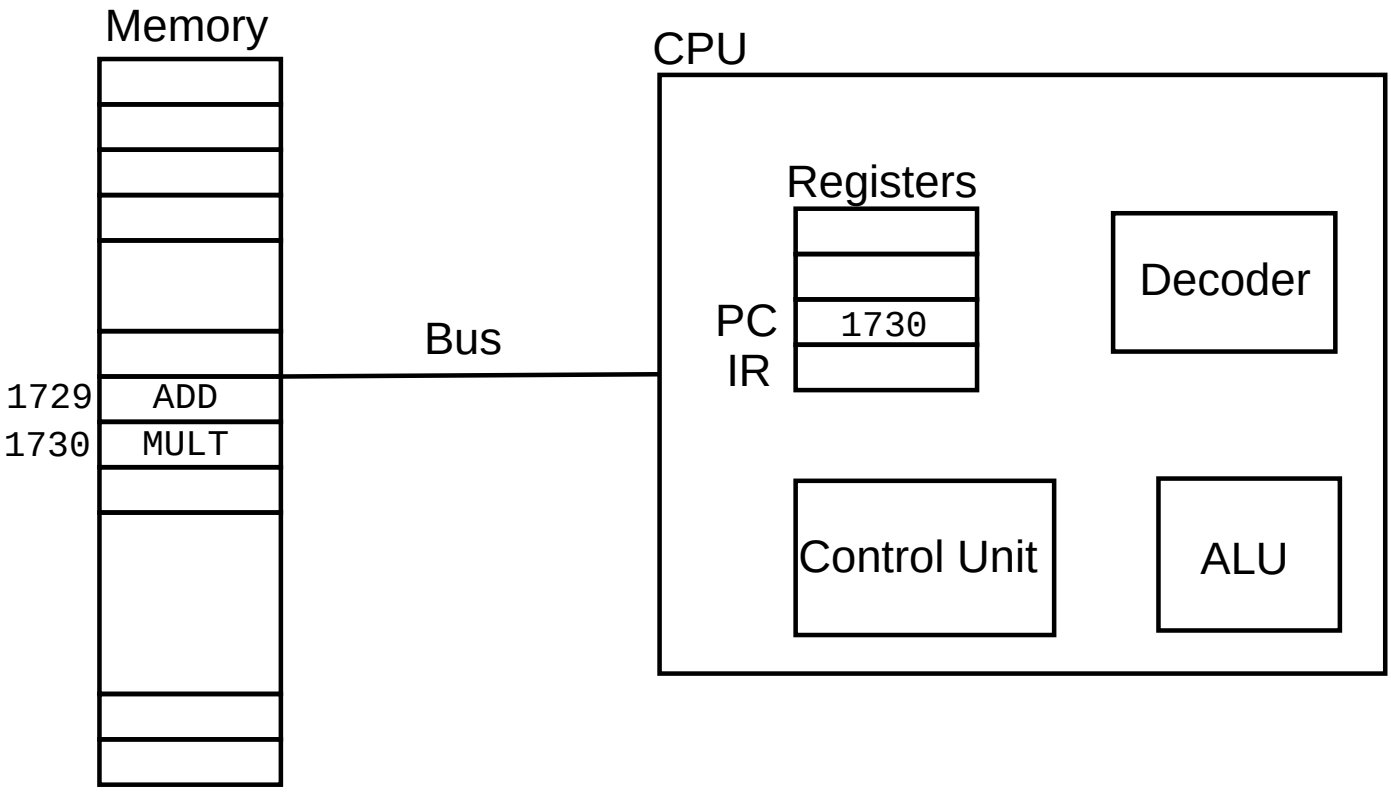
Computer Architecture



Computer Architecture



Computer Architecture



Computer Architecture

- Program instructions:
  - Instructions are numbers (ultimately in binary form)
    * 00110101 represents ADD (adding numbers)
    * 10101100 represents MULT (multiplication)
    * 01010111 represents LOAD (load data from memory to a register)
    * 10100111 represents STORE (stores data from a register to memory)

Computer Architecture

- Instructions, or operators may have parameters
  - Adding the contents of registers R1 and R2 and put the result in R3:

$$\underbrace{00110101}_{ADD}\ \underbrace{10001001}_{R1}\ \underbrace{10001010}_{R2}\ \underbrace{10001011}_{R3}$$

  - Loading data from memory cell 26 and put it in register 2

$$\underbrace{11111001}_{LOAD}\ \underbrace{00011010}_{26}\ \underbrace{10001010}_{R2}$$

Computer Architecture

- Different kinds of processors have different *instruction sets* (e.g. Pentium, PowerPC, Alpha, SPARC, Motorola)
  - Each instruction set has different instructions, and associates different numbers to each type of instruction
  - Hence, a program for one type of processor cannot be directly executed by a different processor.

- Portability: the ability to run (execute) a program in more than one type of processor.

Programming Languages

- A program as understood by the computer is a long sequence of words (bits):

```
1101100011101000100100010010100101001001001010
```

  – Machine Language

- But each instruction can be written in a fashion readable by humans:

```
LOAD [26], R1
LOAD 3, R2
ADD R1, R2, R3
STORE R3, [1700000029]
```

  – Assembly language

- Assembler: a program that translates an assembly language program into its machine language equivalent.

Programming Languages

- Assembly is a low-level language
- High-level languages abstract the components of the machine

  ```
  x = y + 3;
  ```

  – Java, C, C++, Python, Perl, ML, Scheme, Prolog, Ada, Pascal, Basic, Fortran, Cobol, ...
- Abstracting the components is good: when implementing an algorithm you don't have to think about the component of the computer. You focus on the problem.
- Compiler: a program that translates a high-level language program into its machine language equivalent.

A simple Java program

```
// This is a very, very, simple program

public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(“Hello, World!”);
    }
}
```

A simple Java program

```
// This is a very, very, simple program      // Comments

public class HelloWorld                      // Class declaration
{
    public static void main(String[] args)   // Method
    {
        System.out.println("Hello, World!"); // Statement
    }
}
```

A simple Java program

```
// This is a very, very, simple program

public class HelloWorld
{         // Definition of class HelloWorld begins here
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}         // Definition of class HelloWorld ends here
```

A simple Java program

```
// This is a very, very, simple program

public class HelloWorld
{
    public static void main(String[] args)
    {  // Definition of main method begins here
        System.out.println("Hello, World!");
    }  // Definition of main method ends here
}
```

A simple java program

- Java is case-sensitive:

  `HelloWorld`

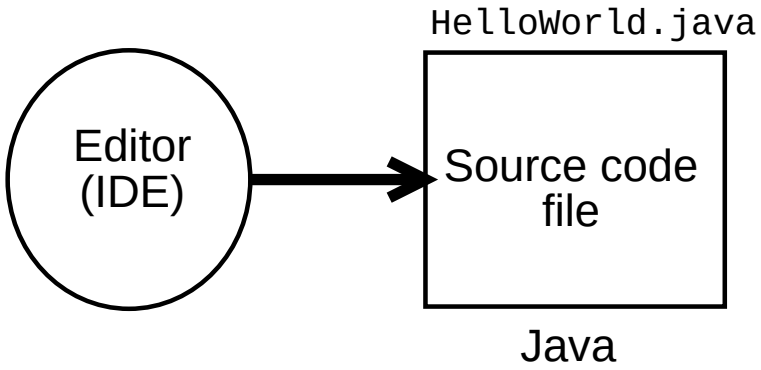
is not the same as

  `helloworld`

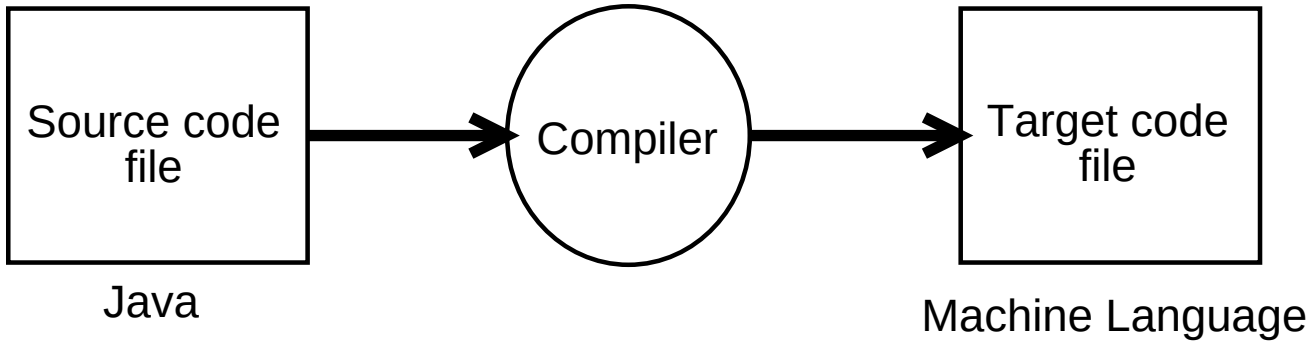
From code to a running program

- Editing
- Compilation/Interpretation
  - Compilation:
    - Translation
    - Execution
  - Interpretation:
    - Execution

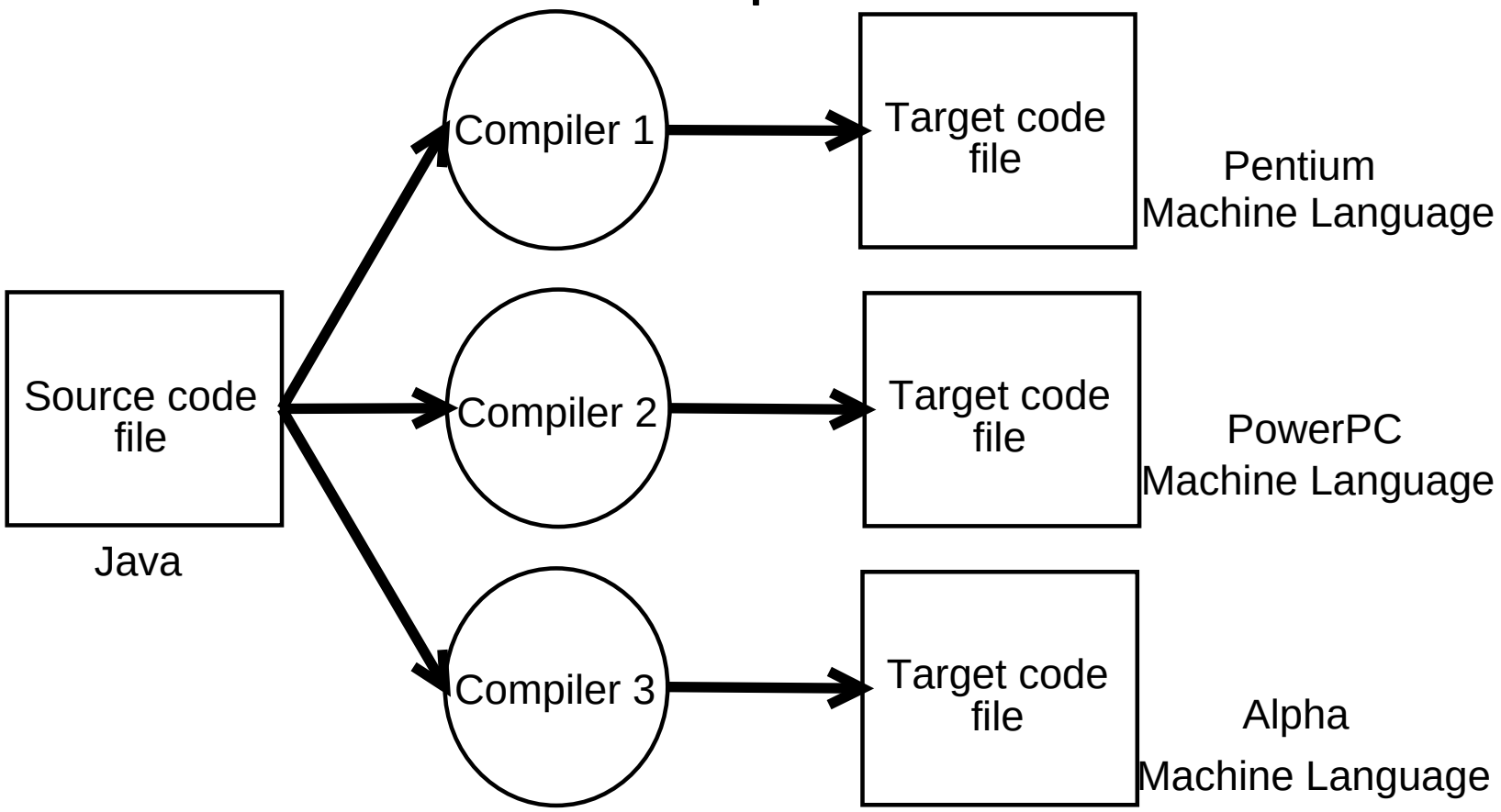
Editing



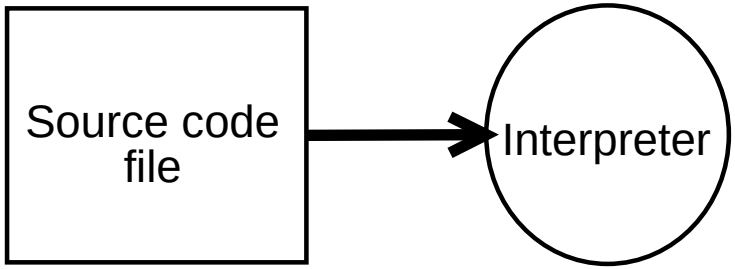
Compilers


Source code file
Compiler
Target code file
Java
Machine Language

Compilers

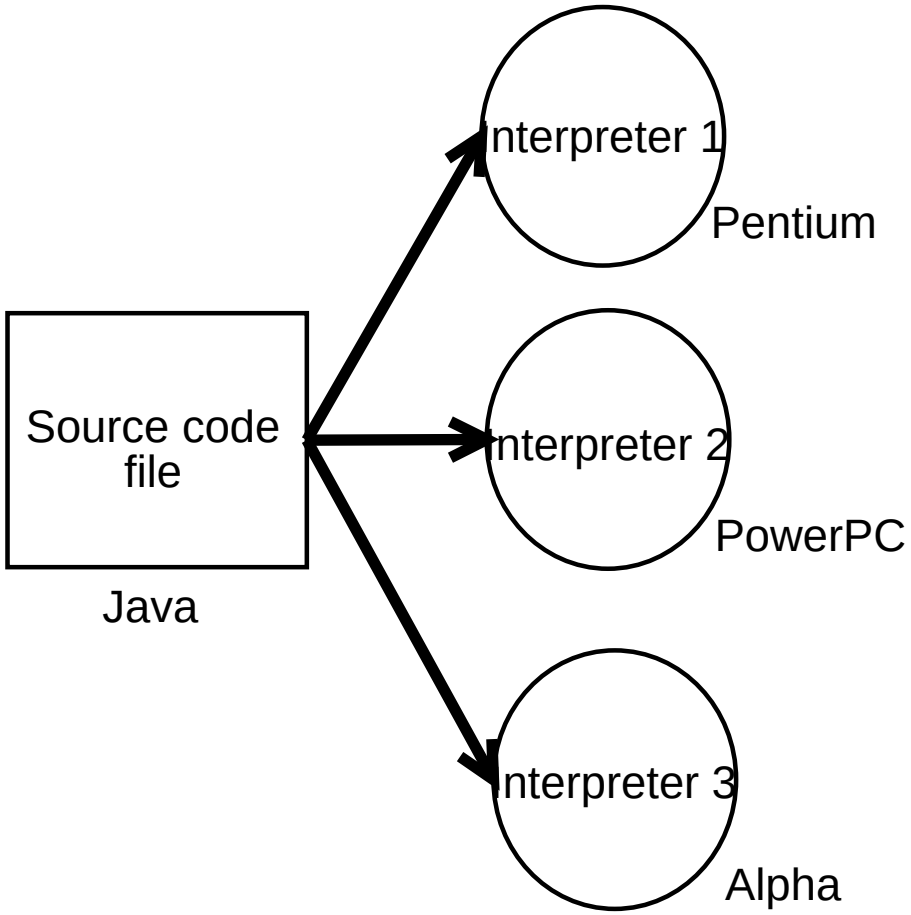

Compiler 1
Target code file
Pentium Machine Language
Source code file
Java
Compiler 2
Target code file
PowerPC Machine Language
Compiler 3
Target code file
Alpha Machine Language

Compilers

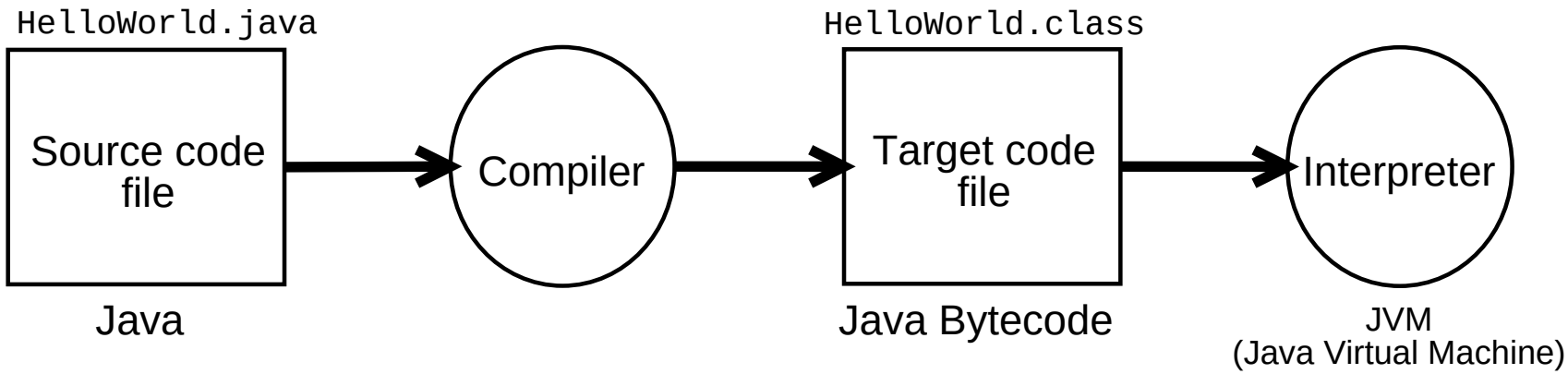


Java

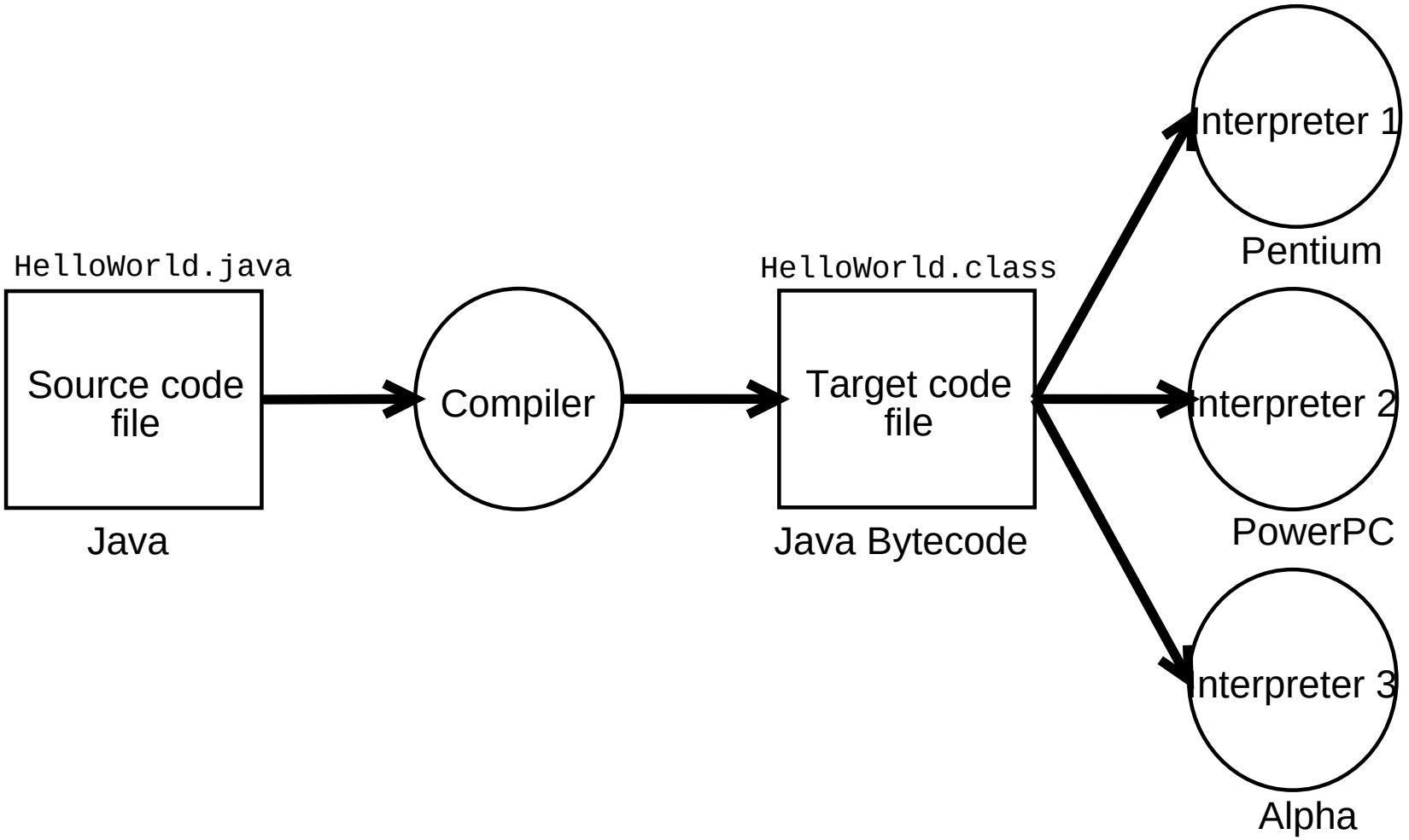
Compilers


Interpreter 1
Pentium
Source code file
Java
Interpreter 2
PowerPC
Interpreter 3
Alpha

Compilers


HelloWorld.java
Source code file
Compiler
HelloWorld.class
Target code file
Interpreter
Java
Java Bytecode
JVM
(Java Virtual Machine)

Compilers


HelloWorld.java
Source code file
Java
Compiler
HelloWorld.class
Target code file
Java Bytecode
Interpreter 1
Pentium
Interpreter 2
PowerPC
Interpreter 3
Alpha

Programming Languages

- A programming language is a formal language to describe algorithms
- A language is a means of communication
- A programming language is a means of communication between a human and a computer, but also between humans
- A programming language is formal: well-defined

Languages

- Elements of a language
  - Alphabet
  - Syntax (grammar)
  - Semantics (meaning)
- Elements of Java:
  - Alphabet of Java: ASCII
  - Syntax: 'constructs'
    - Class definitions
    - Method definitions
    - Statements
    - others
  - Semantics: computation

Errors

- Errors:
  - Compile-time errors
  - Run-time errors
    - Exceptions
    - Logical

Programming Languages

- Machine language (binary, processor dependent)
- Assembly language (textual, low-level, processor dependent)
- High-level languages (textual, abstract, processor independent)
  - There are many high-level languages: Java, C, C++, C#, ML, Haskell, Scheme, Prolog, Python, Perl, etc.
  - Different types of languages:
    - Imperative
      - Procedural
      - Object Oriented
      - Concurrent
    - Declarative:
      - Functional
      - Logic
    - Mixed

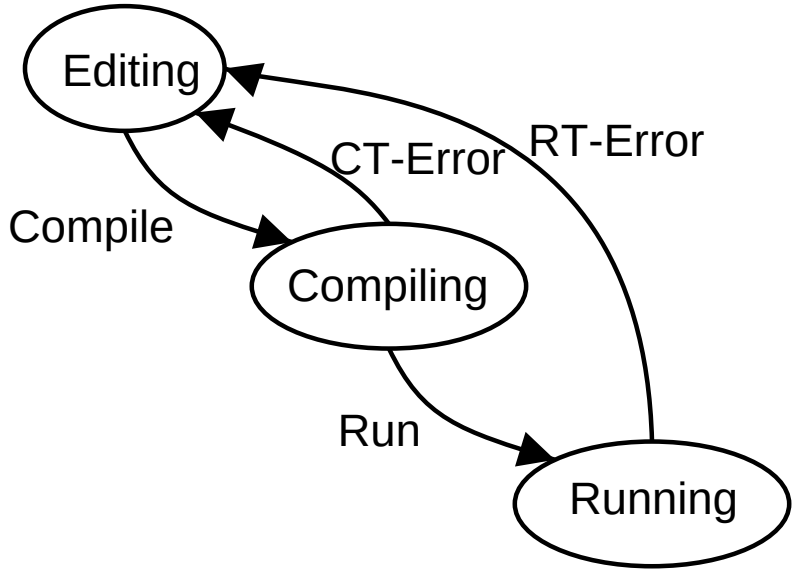
Executing programs

- Editing
- Compilation/Interpretation
  - (Native) Compilation: Translation to machine language + Execution
    - Advantages: Fast, processor specific code is generated
    - Disadvantage: Needs a compiler for each type of processor; generates a different target file for each type of processor
  - Interpretation: Direct execution
    - Advantages: Execution is processor independent. Does not generate a different target file for each possible processor (portability)
    - Disadvantage: Slow execution due to overhead of interpretation.
  - Combined: Translation to bytecode + interpretation of bytecode
    - Best of both worlds: Only one file is generated (portable) and it is faster to execute than direct interpretation (but slower than native compilation.)

Errors

- Errors:
  - Compile-time errors
  - Run-time errors
    - Exceptions
    - Logical

Errors



Basic Java Syntax

- A Java program is made up of one or more *class definitions*
- A class definition is made up of zero or more *method definitions*
- A method definition is made up of zero or more *statements* and *variable declarations*
- Roles:
  - Classes: Modules and Types of objects
  - Methods: procedures, functions, algorithms
  - Statements: instructions

Basic Java Syntax

```
public class ClassName
{
    // Body of ClassName
    // ...
    // List of method definitions
}
```

Basic Java Syntax

```
public class HelloWorld
{
   // Body of ClassName
   // ...
   // List of method definitions
}
```

Basic Java Syntax

```
public class Classname
{
    // method header
    {
        // method body: list of statements
    }
}
```

Basic Java Syntax

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello");
        System.out.println("Good bye");
    }
}
```