Programs

Machine code, Assembly, Compilers, Interpreters

Programs

Computers are **programmable.**

The programming can be **changed**.

That's what makes them useful.

So, what is a program?

High-level programs

Here's a bit of Python code:

```
import sys
name = sys.argv[1]
print 'Hello, ' + name + '!'
```

High-level programs

How about this one (Java):

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

High-level programs

Or this (C++):

```
#include <iostream>

int main(void) {
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

How are these executed by the CPU?

The CPU can’t execute these programs directly.

They need to be **compiled** or **interpreted** – sometimes both!

CPUs can only execute **machine code**.

Machine Code

- A very simple computer language
- Different for each computer architecture
  - e.g. different code for your phone, your laptop, your smart lights
- Machine code programs are
  - **Sequences of instructions**
  - **Stored in memory**
  - **Each instruction is one or more words**

Instruction Set Architecture

The set of instructions that a particular type of CPU understands is called its **Instruction Set Architecture (ISA)**.

What kinds of instructions?

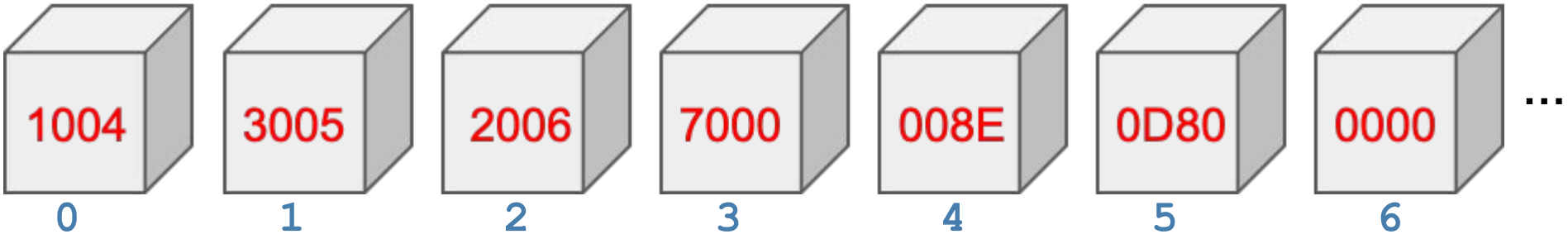
- Do some **maths** (add, subtract, multiply, compare etc.)
- **Move** data between memory, registers and I/O
- Execute **conditionals** and **loops**

Recap: registers

- Small number of very fast memory cells **inside** the CPU
- Each can (typically) hold one word of data
- General purpose register:
  - Use to store temporary values for calculations (e.g. ALU)
  - Often referenced by machine code instructions
- Special purpose register:
  - Used by the CPU internally
  - Usually can’t be used in programs directly

A quick introduction to memory

Memory is **external to the CPU**. Think of it as a sequence of boxes:



Each box contains a **value** (here: a 16-bit number).

This could be a machine code instruction, or data.

We give each box an **address**: the number of the box, starting from 0.

Assembly Language

Machine code is difficult to read and write.

Example: what does `0010000000000110` mean?

We use **assembly language**:

- Each machine code instruction has a mnemonic (easy to remember word)
- The assembler (a program) translates assembly into machine code
- This is easy: each assembly instruction is one machine code instruction

Assembly instructions

These are not real instructions, just examples.

- **`Load 0xA003, R0`**
  Load the number stored in memory at address `A003` into register `R0`
- **`Add R0, R1, R2`**
  Add the number stored in `R0` to the number stored in `R1`, store the result in `R2`
- **`Store R0, 0xA004`**
  Store the number in `R0` into memory address `A004`
- **`Jump 0x1000`**
  Continue program execution at address `1000`

Compilers and Interpreters

An interpreter is a program (usually machine code) that directly executes a high-level program.

A compiler is a program that translates programs from a higher level language into a lower level language.

Compilers can

- directly target machine code (e.g. C, C++)
- or produce simpler code for an interpreter (e.g. Java)

Summary

- Programs are what makes computers useful
- CPUs execute sequences of machine code instructions
- Machine code is stored in memory (together with data)
- Assembly language is a 1-to-1 mapping from machine code into a human-readable form
- Most programs are written in high-level languages
  - CPUs can’t execute them directly!
  - Need to be either translated into machine code (compiled) or executed by an interpreter

EOF