Programming for Engineers

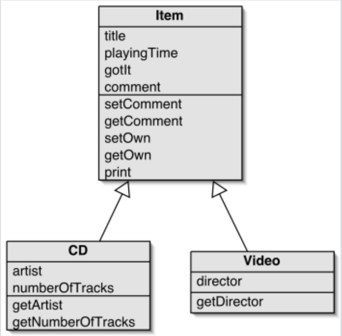
Lab class 9

Dr Ronald Grau School of Engineering and Informatics Autumn term 2019

Last week

Fundamental concepts in object-oriented modelling

- Inheritance
- Polymorphism



Number bases and bitwise operations

- Binary vs. decimal vs. hexadecimal numbers
- Bitwise operations / operators

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11

Today

Computer architecture

- Microarchitecture
- Instruction set architecture
- CPU operations

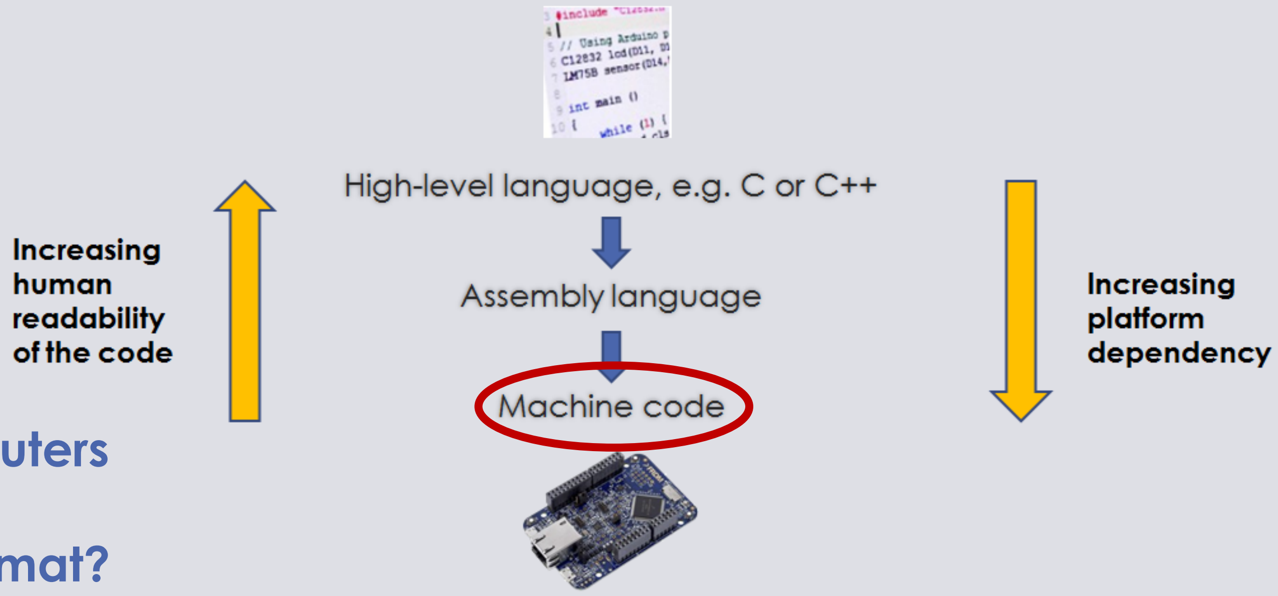
Logical & physical perspective on computers

How can we represent data in binary format?

How can we carry out instructions?

Machine code & Assembly language

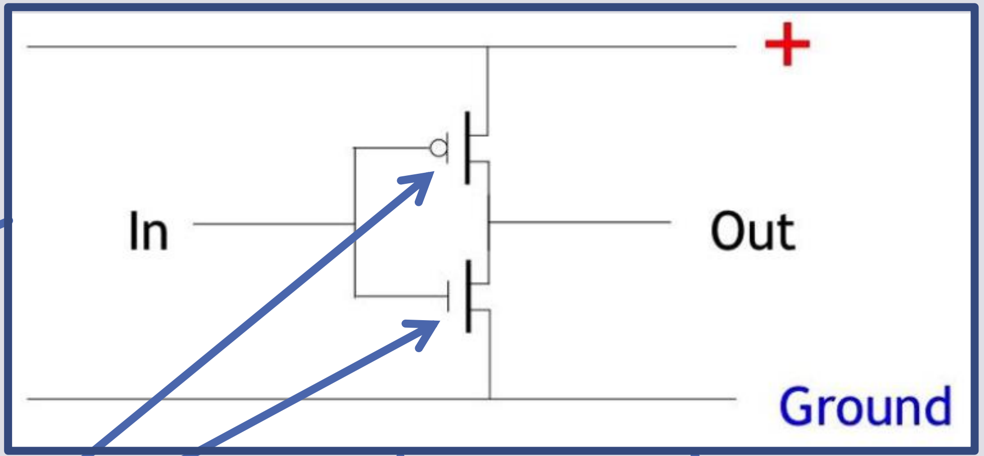
- The Brookshear machine emulator



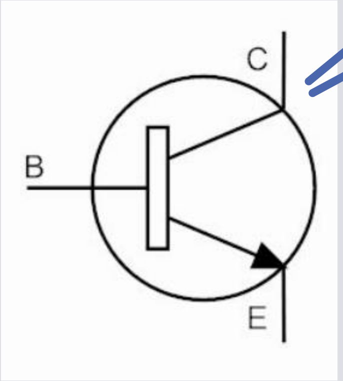
Back to Basics

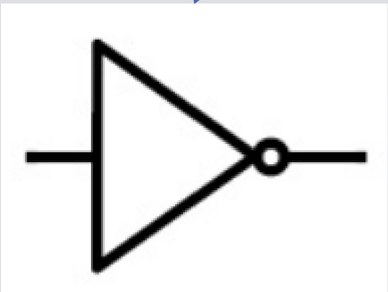
Ways of talking about a computing system

- Physical and logical perspectives
- Different levels of abstraction



Many!





In	Out
L	H
H	L

Basic Concepts

Representation of data and instructions as binary code

- Computers do not understand the human-readable code of "high-level" programming languages such as Java, Python, C, or PHP.
- Such code needs to be translated into instructions a computer can actually carry out

An integer

Data	00000000	00000001	00000010	00000011	00000100	00000101
Meaning	0	1	2	3	4	5

A character

Data	01000001	01000010	01000011	01000100	01000101	01000110
Meaning	A	B	C	D	E	F

Part of a machine instruction

Data	01011001	10000100	01101100	00001111	01010010	01000010
Meaning	LOAD	STORE	ADD	HALT	JUMP	TEST

Basic Concepts

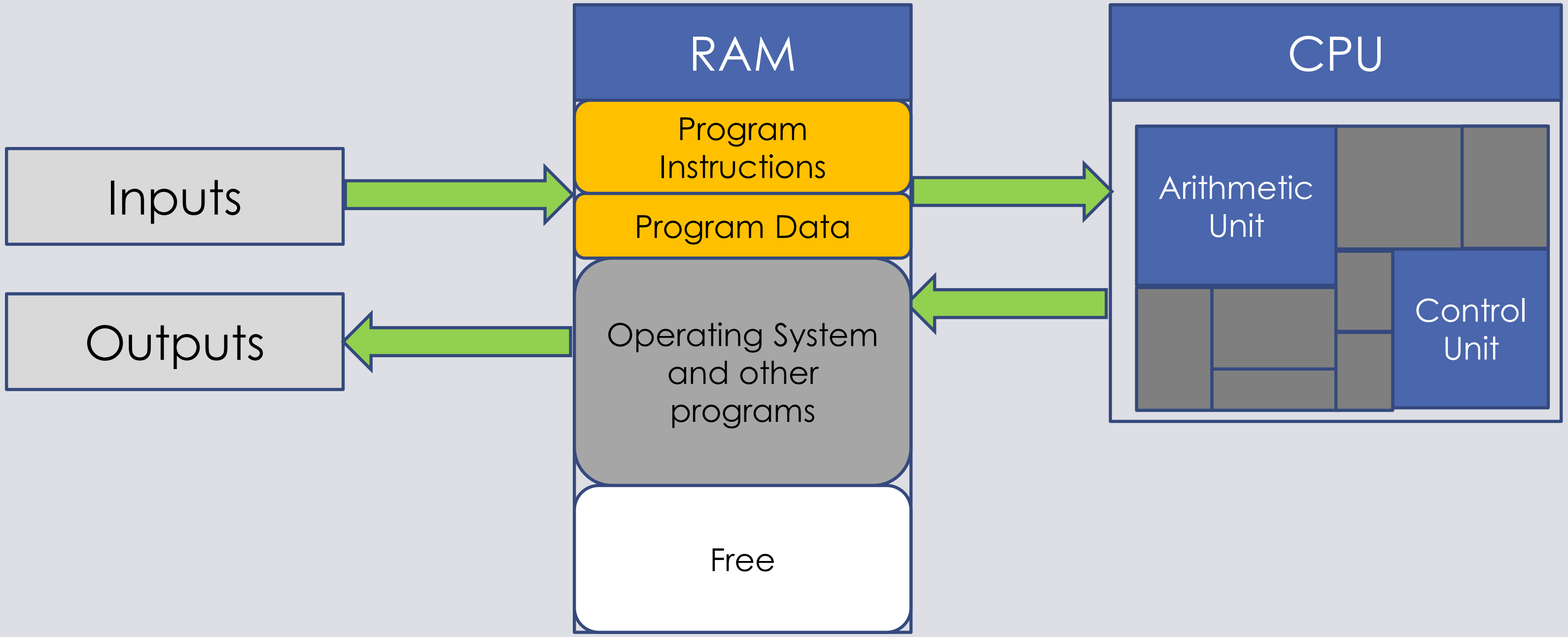
What are resources and what do we need them for?

- E.g.:
  - Memory → Storage of data
  - CPU → Processing of data
  - Networks & buses → Transmission of data / control

Basic Concepts

Why does running a program require memory?

- We need to store the data and the instructions of our program so that the CPU can access them as needed to run our program, and store any results of the computation, intermediate and final.

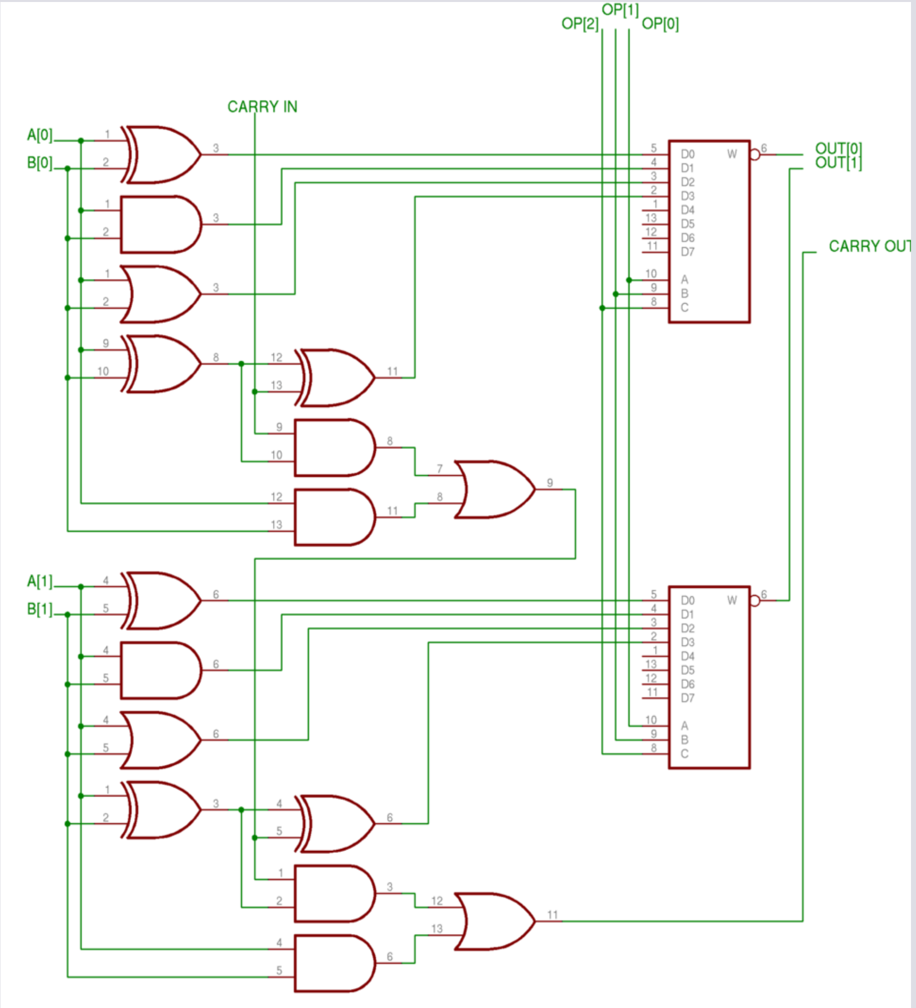


Basic Concepts

How does data processing work in principle?

- Machine code is binary (0 and 1) and translates into electrical signals (e.g., low and high voltage)
- The signals control the operation of transistors
- Transistors are arranged into logic circuits (CMOS)
- Logic circuits perform operations (manipulate multiple electric signals) in order to perform basic arithmetic or logic operations
- Electric signals can be interpreted as binary code

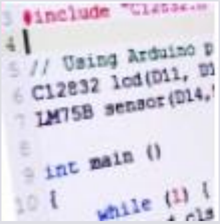
and so it continues…



A logic circuit for adding numbers

Hierarchy of control

What is the hierarchy of control?



Increasing human readability of the code

High-level language, e.g. C or C++

Assembly language

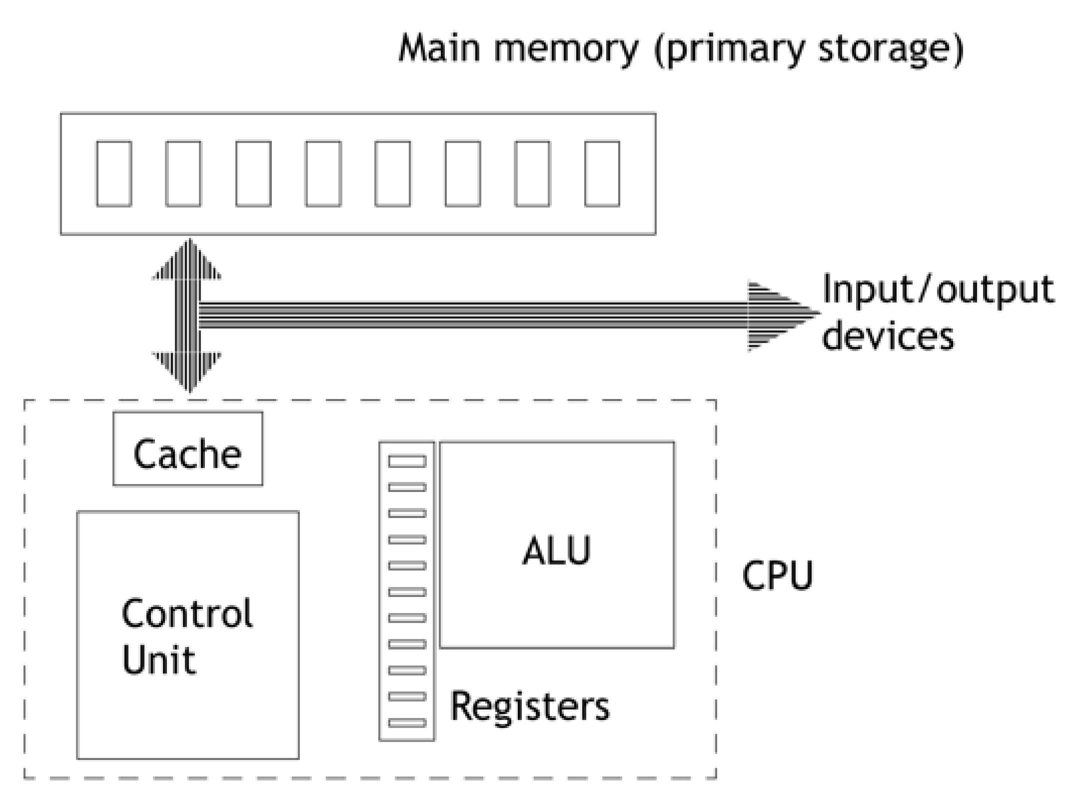
Machine code

Increasing platform dependency

Computer architecture

Microarchitecture

- The **Central Processing Unit** (CPU) contains (at least) a **Control unit** (CU) and an **Arithmetic Logic Unit** (ALU).
- In addition to that, there may be other units for special types of computation (e.g., FPU)
- Data is temporarily stored in the memory **cache** and in **registers**.



Computer architecture

ALU

- The ALU is built from logic circuits, typically made from CMOS transistors on a silicon wafer.
- It carries out fundamental operations on data. The set of possible operations is fixed ('hardwired'). The input to the ALU consists of data and control.

Computer architecture

ALU Operations

- An ALU carries out logical and arithmetic operations such as AND, OR, XOR, Addition, Subtraction, Comparison, applied to multi-bit inputs (typically 8 to 64 bits).
- These operations are sufficient to support different kinds of computing. Some ALUs may perform more complex operations, e.g. floating point arithmetic.
- The set of ALU operations is part of the **microarchitecture** of the computer.
- Conceptually, an ALU performs one operation at a time, but internally the situation is more complex.

Computer architecture

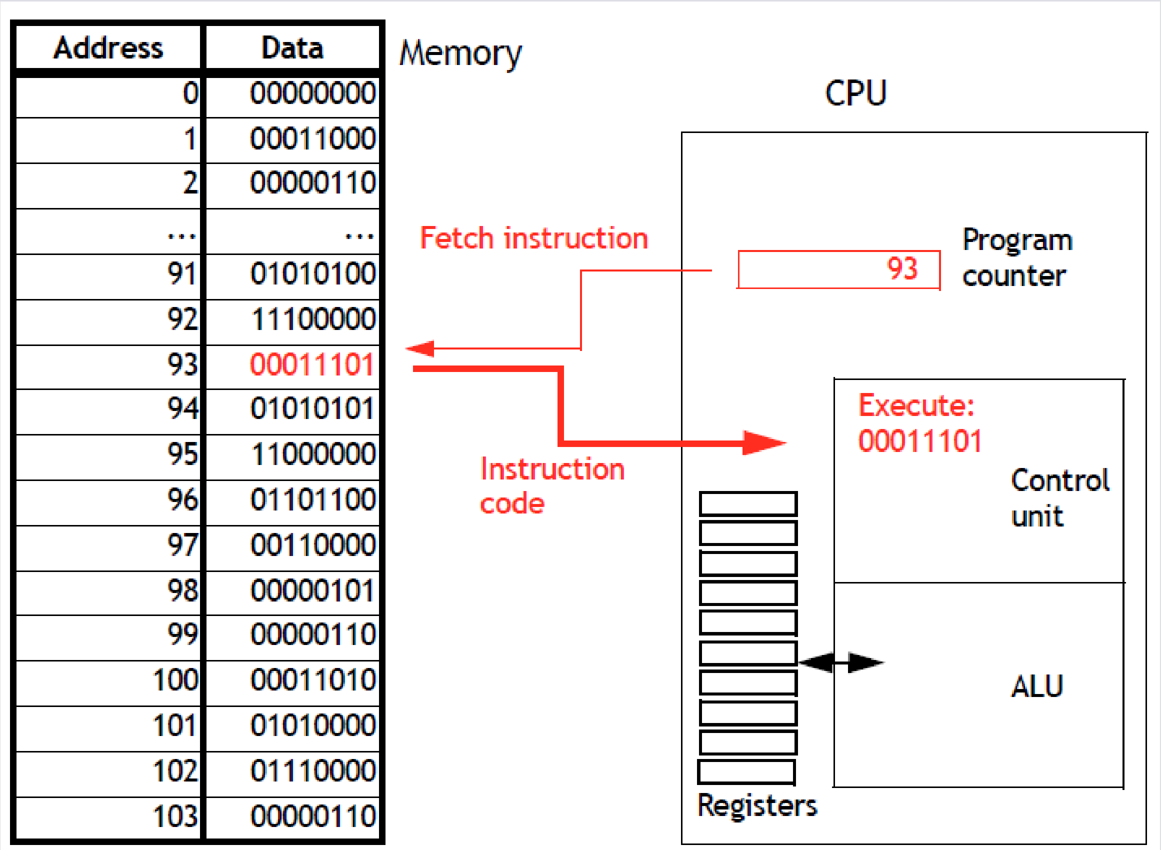
CPU

- Modern CPUs are also known as **microprocessors**, because all the circuitry comes on a single chip.
- The CPU carries out instructions in a cycle, driven by a **clock**:
- An instruction may:
  - Transfer data between main memory and a register
  - Cause the ALU to carry out a logical or arithmetic operation on data held in registers, storing the result in a register
  - Change the program counter
- For any CPU, the key information about it is the set of instructions that it obeys: its **instruction set architecture**.

Computer architecture

Basic CPU operation cycle

- Get an instruction from the memory address held in a special register, called program counter
- Execute the instruction
- Update the program counter to point to the next instruction



Computer architecture

Instruction sets: Complex Instruction Set Computer (CISC)

- A large variety of instructions, each possibly doing something quite complex (e.g. copying chunks of memory, extended-precision arithmetic)
- Needs a lot of transistors in the CPU to decode and execute instructions - adds expense
- Compilers (generators of CPU instructions) can take advantage of the fancy instructions to produce more compact code
- The CPU may have to get instructions from memory less often
- Each instruction may do a lot of computation
- An instruction may be slow to execute, so the CPU clock needs to be slower

Computer architecture

Instruction sets: Reduced Instruction Set Computer (RISC)

- Smaller variety of instructions, each doing something relatively simple
- Needs fewer transistors on the CPU — so CPU is cheaper
- Compilers have to be smarter to generate efficient code
- *Object code* generated by the compiler contains more instructions
- Each instruction is quick to execute, so the CPU clock can be faster
- Instruction format is simpler and more uniform

Computer architecture

CISC or RISC?

- Historically, CISC was the dominant approach for many years
  Mainframes (e.g. IBM 360 series) used CISC.
- Many Intel processors look like CISC from the outside (for compatibility), but are essentially RISC inside!
- Now, RISC is more successful — “lean and mean” philosophy of design — supported by go-faster techniques such as **pipelining**. Sun SPARC technology spearheaded RISC. IBM POWER processors and ARM processors are RISC.

Computer architecture

Kinds of CPU instructions

- **Data transfer**

  Often named “MOVE” instructions

  - Copy data from memory to register (LOAD)
  - Copy data from register to memory (STORE)
  - Copy data from one register to another
  - Set a register to a given constant value

  Also handles Input / Output

- **Control**
  - Take the contents of 2 registers. If they are equal, change the **program counter** to a specified value
  - Halt

Computer architecture

Kinds of CPU instructions

- **Arithmetic and logic**

  Take the contents of 2 registers, and carry out a logical or arithmetic operation on them:

  AND (bit by bit)
  OR (bit by bit)
  XOR (bit by bit)
  Addition
  …

  then store the result in a third register.

Computer architecture

Instruction formats

- Typically, machine instructions occupy several bytes of memory:
  - 1 or 2 bytes identifying the instruction: the **opcode**
  - Some bytes giving the address of a memory location, or identifying
  - a register, or containing a constant value

- **CISC machines:** instruction length may vary, depending on the operation and what extra information is needed. (E.g., HALT needs nothing extra, LOAD needs a memory address and a register address.)
  This is hard work for the CPU, as it has to decide what extra information is needed after decoding the opcode.
- **RISC machines:** Instructions lengths all the same (or at least more consistent), so a standard number of bytes can be fetched from memory on each cycle.

Machine code

Welcome to the lowest level of programming!

- Early computers allowed machine language programs to be entered in binary into memory directly from a row of switches on the front panel!
  In some cases, this was the only way to **boot** the computer.

DEC PDP 11/70 Minicomputer - https://en.wikipedia.org/wiki/PDP-11

Machine code

Hierarchy of control

Decreasing human readability of the code

High-level language, e.g. C or C++

Assembly language

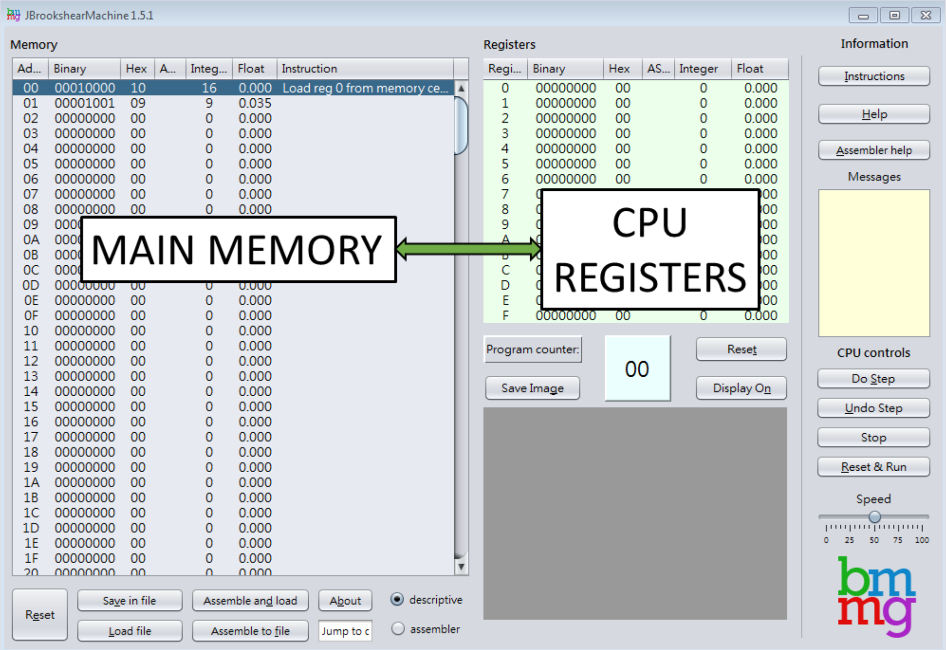
Machine code

Increasing platform dependency

Machine code

Meet the Brookshear machine

- Typing machine code in hexadecimal on a keyboard is quicker and less error-prone.
- The Brookshear Machine is an "idealized" RISC machine and has never existed in hardware. We are running it using an **emulator**, which also provides a programming environment with basic debugging facilities:
  - Visualisation of main memory and CPU registers
  - Built-in conversion between data representations
  - Assembler and disassembler
  - Step-wise execution of programs
  - Instruction undo



Data representation in memory

Signed Integers – Two's complement notation

- Binary addition for positive integers: no problem, apart from **overflow** if the result is too large for its destination.
- Negative numbers: use a bit to indicate sign. Most efficient (less logic needed) if negative numbers are represented in **two's complement** form.
  For 8-bit numbers, this looks like this →
- Using the standard circuitry (for bitwise addition with carry to the next bit position), this works for negative numbers just as well.

Decimal	Binary
00000000	0
00000001	1
00000010	2
…	…
01111110	126
01111111	127
10000000	-128
10000001	-127
10000010	-126
…	…
11111101	-3
11111110	-2
11111111	-1

Data representation in memory

Floating point numbers

- Floating point numbers are those that have fractional parts.
- In the example below, a floating point number is stored in just 8 bits for illustration. On real machines, 32, 64 or 128 bits are normal sizes for floating point numbers.

0 means number is positive;
1 means number is negative

Sign	Exponent	Mantissa
0	101	1101

interpret as an unsigned integer, call this ***m***

interpret as an unsigned integer ***and subtract 8***, call this ***e***

- The magnitude of the float being represented is $m * 2^e$

Data representation in memory

Floating point numbers

- The magnitude of the float being represented is $\boldsymbol{m * 2^e}$

- **`01011101`**$_2$ represents the floating point number $13 * 2^{-3} = 1.625_{10}$ (decimal).
- **`11101111`**$_2$ represents the float $-15 * 2^{-2} = -3.75_{10}$.

- There is an international standard for floating point number representations: *IEEE 754*.
- Floating point arithmetic is not exact — designing algorithms for some scientific and engineering computations requires careful study of how errors might propagate on the chosen computing platform.

Data representation in memory

Characters in hexadecimal notation

- A character can be represented in 8 bits, using the **ASCII** encoding.
  ASCII became an international standard in 1963, defining 128 characters. The ISO-8859-1 standard extended this to 256 characters.
- This is obviously not enough to cover all of the world's languages! The latest international standard for character sets is **Unicode**, which provides up to 1,114,112 **code points** (from 0 to 10FFFF), currently defining a little more than 110,000 of these.

Hexadecimal	Character
`0`	`Null char`
…	…
`20`	`SPACE`
…	…
`2E`	`.`
`2F`	`/`
`30-39`	`0-9`
…	…
`41-5A`	`A-Z`
…	…
`61-7A`	`a-z`
…	…

Demo

Negation of integers

Addition and subtraction of integers

Practice

Open the lab exercise sheet and complete SECTION 1

Practice

Open the lab exercise sheet and complete SECTION 2

Read

Basics (Optional):

- Brookshear and Brylow: Computer Science
  - Sections 2.1 – 2.3

Module Summary

What have we achieved?

- Basics of computing
  - Hardware & software, resources, hierarchy of control, binary systems
  - Computer microarchitecture, and computer instruction sets
- Basics of C programming
  - Foundational concepts, structures, and syntax of higher-level programming languages
- Basic elements of C++ programming (Object orientation, OO)
  - Classes and objects, and other basic concepts of OO
  - How to design OO programs
  - Inheritance and Polymorphism
- Practical programming
  - Problem-solving / basic search techniques in **C**
  - ARM mbed freescale platform in **C/C++**
  - Some hands-on experience with **Machine code**

Outlook

- Have a good break!
- There's no exam!

Note:

- CW2 is due in early January. See the instructions on Canvas.
- Submit your files on Canvas and hand in your mbed devices as well.