COMP110: Principles of Computing

2: Basic Principles for Computation

Learning outcomes

By the end of this week's sessions, you should be able to:

- **Use** binary, decimal and hexadecimal notation to represent and operate on numerical values
- **Explain** the basic architecture of a computer
- **Distinguish** the most common programming languages and paradigms in use today

Research journal

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them
- **Research** how this paper has influenced the field of computing
- **Write up** your findings
  - Maximum 1500 words
  - With reference to appropriate academic sources

Marking rubric

See assignment brief on LearningSpace/GitHub

Timeline

- **Peer review** in week 11 (4th December)
- **Deadline** shortly after (check MyFalmouth)
- Finding and reading academic papers takes time and effort — don't leave it until the last minute!

Binary notation

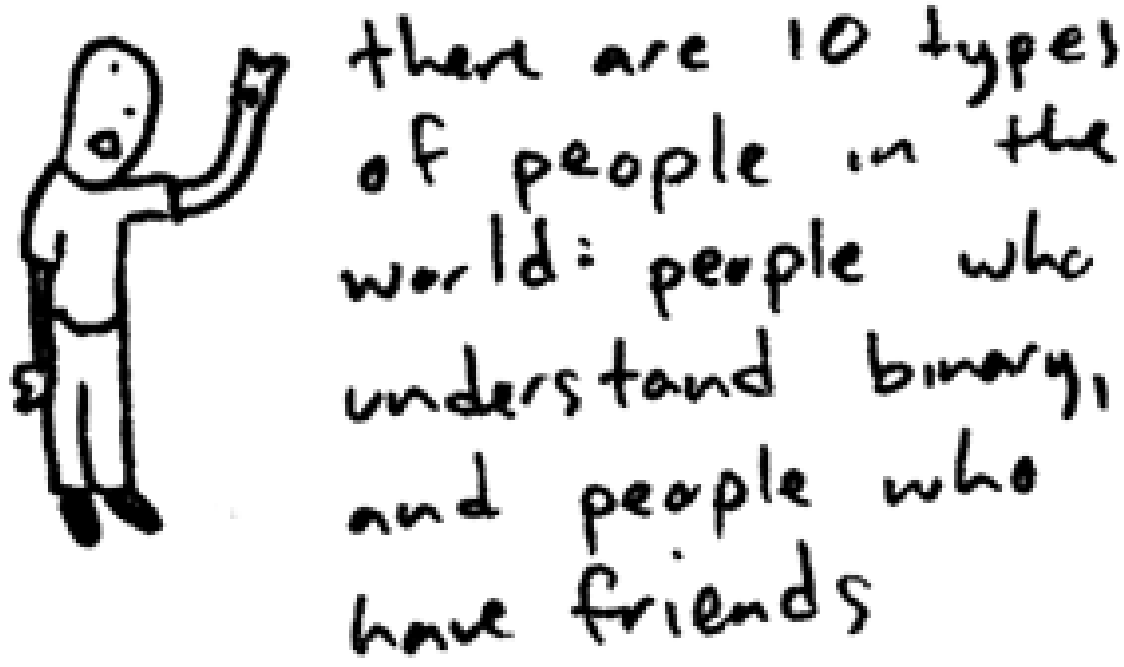


Image credit: `http://www.toothpastefordinner.com`

How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$
- When we write 6397, we mean:
  - Six thousand, three hundred and ninety seven
  - (Six thousands) and (three hundreds) and (nine tens) and (seven)
  - $(6 \times 1000) + (3 \times 100) + (9 \times 10) + (7)$
  - $(6 \times 10^3) + (3 \times 10^2) + (9 \times 10^1) + (7 \times 10^0)$
  -

Thousands	Hundreds	Tens	Units
6	3	9	7

Binary

- Binary notation works the same, but is **base 2** instead of **base 10**
- We have 2 **digits**: 0, 1
- When we write 10001011 in binary, we mean:

$$\begin{aligned}
&(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) \\
&+ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
&= 2^7 + 2^3 + 2^1 + 2^0 \\
&= 128 + 8 + 2 + 1 \text{ (base 10)} \\
&= 139 \text{ (base 10)}
\end{aligned}$$

Converting to binary

https://www.youtube.com/watch?v=OezK_zTyvAQ

Bits, bytes and words

- A **bit** is a binary digit
  - Can store a 0 or 1 (i.e. a boolean value)
- A **byte** is 8 **bits**
  - Can store a number between 0 and 255 in binary
- A **word** is the number of bits that the CPU works with at once
  - 32-bit CPU: 32 bits = 1 word
  - 64-bit CPU: 64 bits = 1 word
- An n-bit word can store a number between 0 and $2^n - 1$
  - $2^{16} - 1 = 65,535$
  - $2^{32} - 1 = 4,294,967,295$
  - $2^{64} - 1 = 18,446,744,073,709,551,615$

Addition with carry

In base 10:

$$\begin{array}{rcccc} & 1 & 2 & 3 & 4 \\ + & 5 & 6_1 & 7_1 & 8 \\ \hline & 6 & 9 & 1 & 2 \end{array}$$

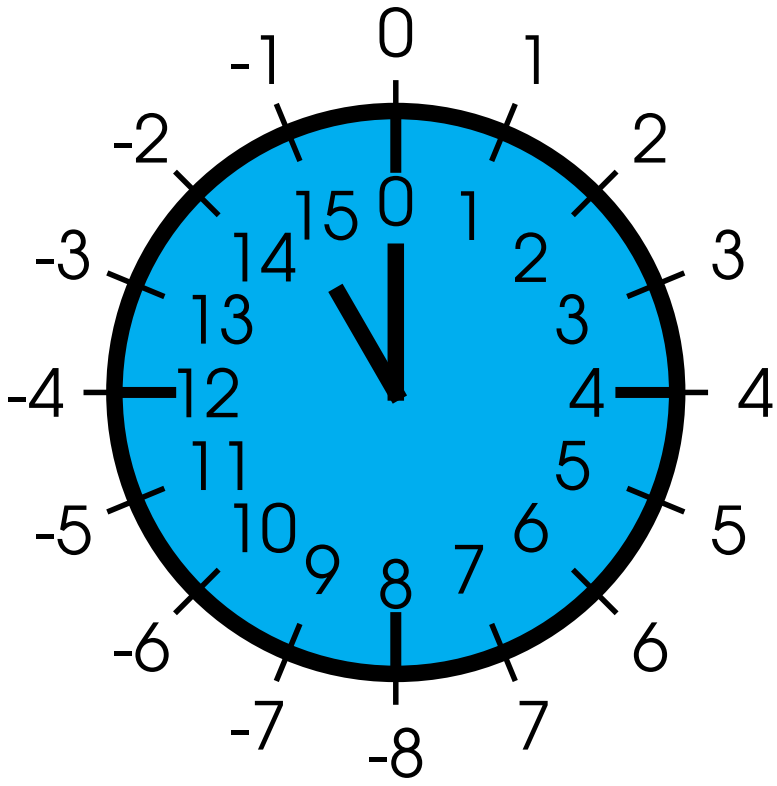
Addition with carry

In base 2:

$$\boxed{1+1=10 \qquad 1+1+1=11}$$

$$\begin{array}{cccccccccc} & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ + & 0_1 & 0_1 & 1 & 0_1 & 0_1 & 1_1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Modular arithmetic



- Arithmetic **modulo** N
- Numbers "wrap around" between 0 and $N-1$
- E.g. modulo 16:
  - $14+7=5$
  - $4-7=13$

2's complement

- How can we represent negative numbers in binary?
- Represent them modulo 2^n (for n bits)
- I.e. represent $-a$ as $2^n - a$
- Instead of an n-bit number ranging from 0 to $2^n - 1$, it ranges from -2^{n-1} to $+2^{n-1} - 1$
- E.g. 16-bit number ranges from -32768 to $+32767$
- Note that the left-most bit can be interpreted as a **sign** bit: 1 if negative, 0 if positive or zero

Converting to 2's complement

- Convert the absolute value to binary
- Invert all the bits (i.e. change $0 \leftrightarrow 1$)
- Add 1
- (This is equivalent to subtracting the number from 2^n... why?)
- This is also the process for converting back from 2's complement, i.e. doing it twice should give the original number

Why 2's complement?

- Allows all addition and subtraction to be carried out modulo 2^n without caring whether numbers are positive or negative
- In fact, subtraction can just be done as addition
- I.e. $a - b$ is the same as $a + (-b)$, where a and $-b$ are just n-bit numbers

Exercise Sheet i

Due next Tuesday!

Programming languages and paradigms

What is a programming language?

- A **program** is a sequence of instructions for a computer to perform a specific task
- A **programming language** is a formal language for communicating these sequences of instructions

Which is the best programming language?

- There is no "best" programming language
- There are hundreds of programming languages, each better suited to some tasks than others
- Sometimes your choice is dictated by your choice of platform, framework, game engine etc.
- To become a better programmer (and maximise your employability) you should learn several languages (but one at a time!)

Low vs high level

- **Low level languages** give the programmer direct control over the hardware
- **High level languages** give the programmer **abstraction**, hiding the details of the hardware
- High level languages trade efficiency for ease of programming
- Lower level languages were once the choice of game programmers, but advances in hardware mean that higher level languages are often a better choice

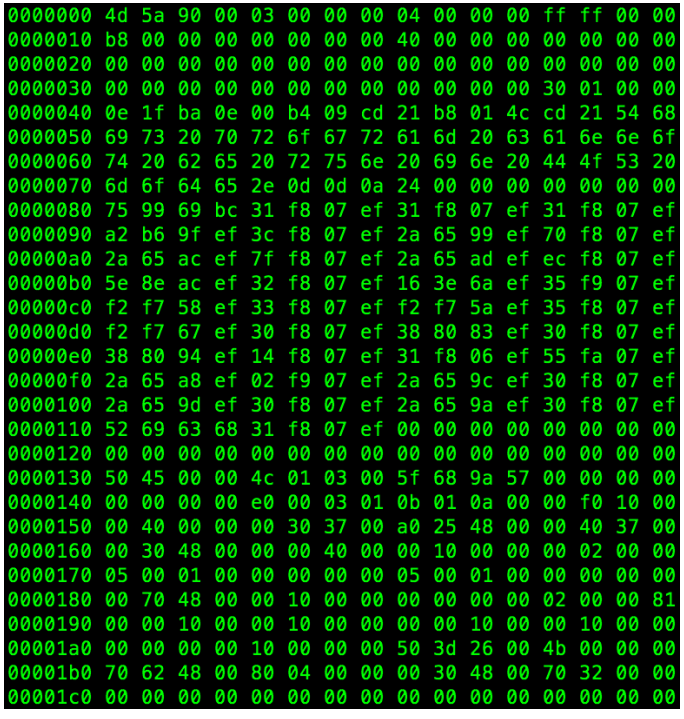
Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- **Procedural**: structured program is broken down into **procedures**
- **Object-oriented**: related procedures and data are grouped into **objects**
- **Functional**: procedures are treated as mathematical objects that can be passed around and manipulated
- **Declarative**: does not define the control flow of a program, but rather defines logical relations

Which paradigm?

- **Imperative** and **structured** languages are mainly of historical interest
- Most commonly used languages today are a mixture of **procedural** and **object-oriented** paradigms, with many also incorporating ideas from **functional** programming
- Purely **functional** languages are mainly used in academia, but favoured by some programmers
- Purely **declarative** languages have uses in academia and some special-purpose languages

Machine code



- Programs are represented as sequences of **numbers** specifying **machine instructions**
- More on this later in the module
- Nobody has actually written programs in machine code since the 1960s...

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code
- Commonly used for games in the 70s/80s/90s, but hardly ever used now
- Allows very fine control over the hardware...
- ... but difficult to use as there is no **abstraction**
- Also not portable between CPU architectures

C++

```cpp
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
	: game(game), sprite(sprite), isDead(false)
{
	x = rand() % CoinGame::WINDOW_WIDTH;
	y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
	sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
	double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
	return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

- Initially an object-oriented extension for the procedural language C
- Low level (though higher level than assembly)
- Used by developers of game engines, and games using many popular “AAA” engines (Unreal, Source, CryEngine, …)
- Also used by developers of operating systems and embedded systems, but falling out of favour with other software developers

High level languages

Often favoured by smaller indie teams for rapid development

- C# (XNA, Unity)
- Python (EVE Online, Pygame, Ren'py)
- JavaScript (HTML5 browser games)
- ActionScript (Flash games)
- Objective-C, Swift (iOS games)
- Java (Minecraft, Android games)

There are many others, but these are the most commonly used in game development

Scripting languages

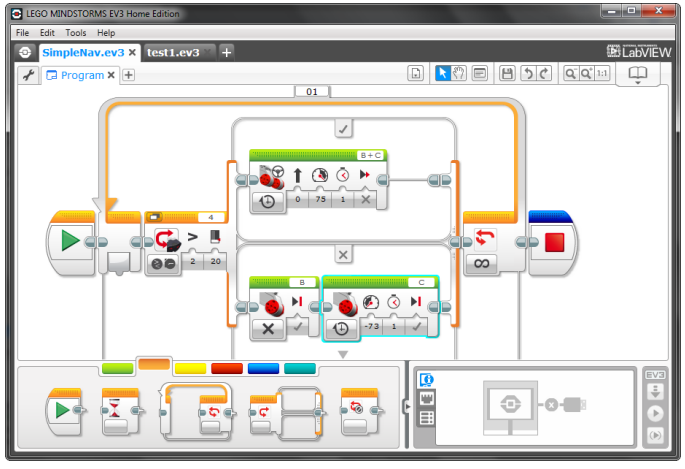
Many games use scripting languages in addition to their main development language

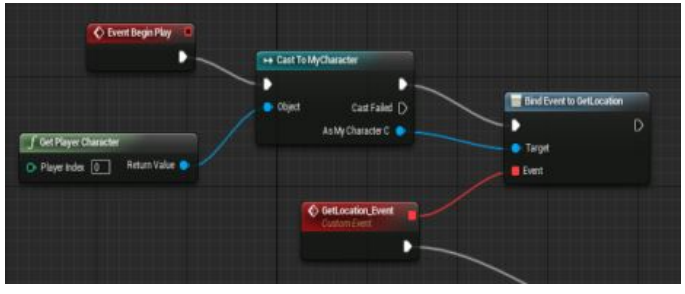
- Lua (many AAA games)
- Bespoke languages (many AAA games)

Some game engines have their own scripting language

- UnrealScript, Blueprint (Unreal Engine)
- GML (GameMaker)

Visual programming languages





Based on connecting graphical blocks rather than writing code as text

- Scratch (used for teaching in school)
- Lego Mindstorms
- Blueprint (Unreal)

Note: despite the name, Microsoft Visual Studio is **not** a visual programming environment!

Special purpose languages

- SQL (database queries)
- GLSL, HLSL (GPU shader programs)
- LEX, YACC (script interpreters)

Markup languages

Not to be confused with programming languages...

- HTML, CSS (web pages)
- LaTeX, Markdown (documentation)
- XML, JSON (data storage)

Which programming language is most popular?

`http://githut.info`

“Family tree” of programming languages

https://www.levenez.com/lang/lang.pdf

Turing machines

Turing machines

- Introduced in 1936 by Alan Turing
- Theoretical model of a "computer"
  - I.e. a machine that carries out computations (calculations)

Turing machine

- Has a finite number of **states**
- Has an infinite **tape**
- Each space on the tape holds a **symbol** from a finite **alphabet**
- Has a **tape head** pointing at one space on the tape
- Has a transition table which, given:
  - The current state
  - The symbol under the tape head

  specifies:
  - A new state
  - A new symbol to write to the tape, overwriting the current symbol
  - Where to move the tape head: one space to the left, or one space to the right

Activity

- In groups of 3-4
- Line up 5-10 chocolates of different colours — this is your **tape**
- Point your **Drumstick** lolly at the **leftmost** chocolate
  - The lolly is your **tape head**, and the type of lolly is your **state**
- Repeatedly apply the rules on the next slide
- What computation does this machine perform?
  - Hint: Milk $= 0$, White $= 1$, and remember yesterday's lecture...

Current lolly	Current chocolate	New lolly	New chocolate	Move direction
Drumstick	Blank	Fruit	Blank	$\leftarrow$
Drumstick	Milk	Drumstick	White	$\rightarrow$
Drumstick	White	Drumstick	Milk	$\rightarrow$
Fruit	Blank	Swizzels	White	$\rightarrow$
Fruit	Milk	Swizzels	White	$\leftarrow$
Fruit	White	Fruit	Milk	$\leftarrow$
Swizzels	Blank	Stop	Blank	$\rightarrow$
Swizzels	Milk	Swizzels	Milk	$\leftarrow$
Swizzels	White	Swizzels	White	$\leftarrow$

The Church-Turing Thesis

- If a calculation can be carried out by a mechanical process at all, then it can be carried out by a Turing machine
- I.e. a Turing machine is the most "powerful" computer possible, in terms of what is possible or impossible to compute
- A machine, language or system is **Turing complete** if it can simulate a Turing machine

Worksheet A review