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

FALMOUTH
UNIVERSITY

Research journal

Research journal

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them

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

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

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

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

Timeline

- **Peer review** in week 11 (4th December)

Timeline

- **Peer review** in week 11 (4th December)
- **Deadline** shortly after (check MyFalmouth)

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!

FALMOUTH UNIVERSITY

Binary notation

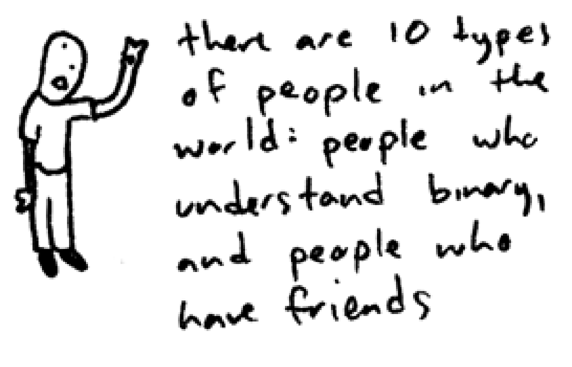

Image credit: `http://www.toothpastefordinner.com`

How we write numbers

How we write numbers

- We write numbers in **base 10**

How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$

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:

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

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)

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)
  - $(6 \times 1000) + (3 \times 100) + (9 \times 10) + (7)$

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)
  - $(6 \times 1000) + (3 \times 100) + (9 \times 10) + (7)$
  - $(6 \times 10^3) + (3 \times 10^2) + (9 \times 10^1) + (7 \times 10^0)$

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

- Binary notation works the same, but is **base 2** instead of **base 10**

Binary

- Binary notation works the same, but is **base 2** instead of **base 10**
- We have 2 **digits**: 0, 1

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:

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:
  $$(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

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:

$$\begin{aligned}&(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4)\\ &+ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)\\ &= 2^7 + 2^3 + 2^1 + 2^0\end{aligned}$$

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:
  $$(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4)$$
  $$+ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$
  $$= 2^7 + 2^3 + 2^1 + 2^0$$
  $$= 128 + 8 + 2 + 1 \text{ (base 10)}$$

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

Bits, bytes and words

- A **bit** is a binary digit

Bits, bytes and words

- A **bit** is a binary digit
  - Can store a 0 or 1 (i.e. a boolean value)

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**

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

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

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

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

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 $2^n - 1$

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 $2^n - 1$
  - $2^{16} - 1 = 65,535$

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 $2^n - 1$
  - $2^{16} - 1 = 65,535$
  - $2^{32} - 1 = 4,294,967,295$

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:

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

Addition with carry

In base 10:

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

Addition with carry

In base 10:

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

Addition with carry

In base 10:

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

Addition with carry

In base 10:

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

Addition with carry

In base 2:

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

$$\begin{array}{cccccccc c}
 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
+ & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
\end{array}$$

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Addition with carry

In base 2:

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

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

Modular arithmetic

Modular arithmetic

- Arithmetic **modulo** N

Modular arithmetic

- Arithmetic **modulo** N
- Numbers "wrap around" between 0 and $N - 1$

Modular arithmetic

- Arithmetic **modulo** N
- Numbers "wrap around" between 0 and $N - 1$
- E.g. modulo 16:

Modular arithmetic

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

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

2's complement

- How can we represent negative numbers in binary?

2's complement

- How can we represent negative numbers in binary?
- Represent them modulo 2^n (for n bits)

2's complement

- How can we represent negative numbers in binary?
- Represent them modulo 2^n (for n bits)
- I.e. represent $-a$ as $2^n - a$

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 $+2^{n-1} - 1$

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 $+32767$

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

Converting to 2's complement

- Convert the absolute value to binary

Converting to 2's complement

- Convert the absolute value to binary
- Invert all the bits (i.e. change $0 \leftrightarrow 1$)

Converting to 2's complement

- Convert the absolute value to binary
- Invert all the bits (i.e. change $0 \leftrightarrow 1$)
- Add 1

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?)

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?

Why 2's complement?

- Allows all addition and subtraction to be carried out modulo 2^n without caring whether numbers are positive or negative

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

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!

Turing machines

Turing machines

Turing machines

- Introduced in 1936 by Alan Turing

Turing machines

- Introduced in 1936 by Alan Turing
- Theoretical model of a “computer”

Turing machines

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

Turing machine

Turing machine

- Has a finite number of **states**

Turing machine

- Has a finite number of **states**
- Has an infinite **tape**

Turing machine

- Has a finite number of **states**
- Has an infinite **tape**
- Each space on the tape holds a **symbol** from a finite **alphabet**

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

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

Activity

- In groups of 2-3

Activity

- In groups of 2-3
- Line up 5-10 chocolates of different colours — this is your **tape**

Activity

- In groups of 2-3
- Line up 5-10 chocolates of different colours — this is your **tape**
- Put your **red** lolly under the **leftmost** chocolate

Activity

- In groups of 2-3
- Line up 5-10 chocolates of different colours — this is your **tape**
- Put your **red** lolly under the **leftmost** chocolate
- Repeatedly apply the rules on the next slide

Activity

- In groups of 2-3
- Line up 5-10 chocolates of different colours — this is your **tape**
- Put your **red** lolly under the **leftmost** chocolate
- Repeatedly apply the rules on the next slide
- What computation does this machine perform?

Current lolly	Current chocolate	New lolly	New chocolate	Move direction
Red	Blank	Orange	Blank	$\rightarrow$
Red	Milk	Red	Milk	$\leftarrow$
Red	Dark	Red	Dark	$\leftarrow$
Orange	Blank	Yellow	Dark	$\leftarrow$
Orange	Milk	Yellow	Dark	$\rightarrow$
Orange	Dark	Orange	Milk	$\rightarrow$
Yellow	Blank	Stop	Blank	$\rightarrow$
Yellow	Milk	Yellow	Milk	$\leftarrow$
Yellow	Dark	Yellow	Dark	$\leftarrow$

The Church-Turing Thesis

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

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

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