Experimental Methods I

Computing:

Data types and binary representation

M.P. Vaughan

UCC

Learning objectives

UCC

- Understanding
  - data types for digital computers
  - binary representation of different data types:
    - Integers (positive and negative)
    - Real numbers
    - Boolean variables
    - Text characters
  - How simple addition may be implemented in a digital computer

Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

ROINN NA FISICE
Department of Physics

Data types

- A **data type** defines a particular *kind* of data
- *Formally*
  - What kind of data (integer, real, character etc.)
  - The allowed values
- *Practically*
  - How the data is represented in terms of **bits**

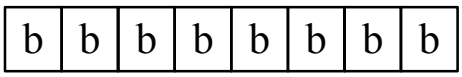
Bits

- A **bit** is a unit of data that (in binary digital computer) can either be ‘1’ or ‘0’
- The corresponds to ‘high’ or ‘low’ voltage in a digital circuit
  - E.G. the inputs and outputs to logic circuits (NAND, NOR etc.)

Bytes

- 1 **byte** is composed of 8 **bits**
- Since each bit can be either ‘1’ or ‘0’, the number of different values a byte can represent is

$$2^8 = 256$$

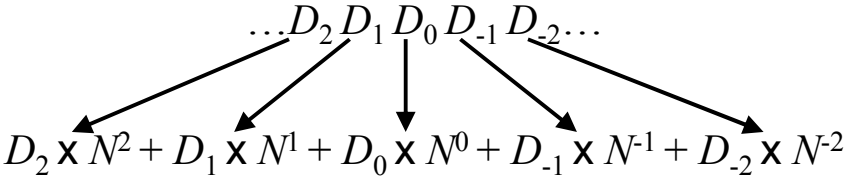


More bytes

- 1 **kilobyte** (kB) = 1024 bytes
- Why 1024 and not 1000?
  - $1024 = 2^{10}$, but 1000 is *not* a power of 2
- 1 **megabyte** (MB) = 2^{20} bytes
  - $2^{20} = 1024 \times 1024 = 1{,}048{,}576$
- 1 **gigabyte** (GB) = 2^{30} bytes
- 1 **terabyte** (TB) = 2^{40} bytes

Number bases

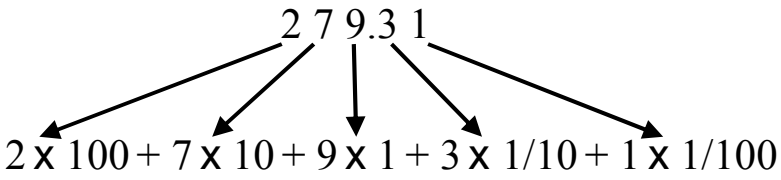
- In base *N* there are *N* digits. Each digit in a number represents a multiply of a power of *N*



(N.B. $N^0 = 1$)

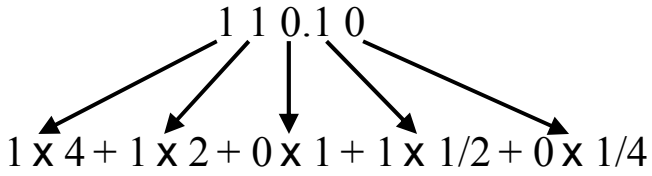
Decimal

- In **decimal**, there are 10 digits (0,1,2,3,4,5,6,7,8,9), so, for example



Binary

- In **binary**, there are 2 digits (0,1), so, for example



(6.5 in decimal)

Hexadecimal

- In **hexadecimal**, there are 16 digits (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
- Conversion between binary and decimal can be awkward
- Conversion between binary and hexadecimal easy:
  - since 4 bits can represent 16 numbers, it can always be represented by a *single* hexadecimal digit.

Four-bit numbers

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	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

Note: 2 digits to represent the numbers in red

Note: Only 1 digit for hexadecimal representation

Negative integers in binary

- Clearly, we can represent positive integers in bits
- What about negative integers?
- Consider the binary sequence of adding 1 to an integer represented by a byte…

Negative integers in binary

Sequence adding 1 to a byte

	1	1	1	1	1	1	1	0	(254)
	1	1	1	1	1	1	1	1	(255)
1	0	0	0	0	0	0	0	0	

Clocked around to zero again!

Negative integers in binary

Effectively, we have

$$255 + 1 = 0 \text{ (!)}$$

So (for a byte) the binary number

1	1	1	1	1	1	1	1

Is equivalent to -1.

Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland
ROINN NA FISICE
Department of Physics

The sign bit

- We may take the bit on the far left-hand-side to be the *sign* bit. If this is 1, the number is taken to be *negative*.

Sign bit →	1	0	1	1	1	1	1	1	(N_1)
	0	1	0	0	0	0	0	1	(65)

Adding these two numbers gives zero. So N_1 = -65

Ones and twos complements

- Making a binary integer negative
  - Step 1: Form 'ones complement' by inverting (NOT-ing) each digit
  - Step 2: Form 'twos complement' by adding 1.
- Example:

Start with 5	0	0	0	0	0	1	0	1	
Invert	1	1	1	1	1	0	1	0	
Add 1	1	1	1	1	1	0	1	1	(-5)

Range of integer data types

- Using 1 byte, an integer data type has the range
  - -128 to 127
  - (-2^7 to $2^7 - 1$)
- More often, an integer may be represented by 4 bytes (32 bits). This gives a range of
  - -2^7 to $2^{31} - 1$
  - (-2,147,483,648 to 2,147,483,647)

Real numbers

- How do we represent real numbers?
- We could choose a position in a group of bits to be the location of the *binary bit* separating the integral part from the fractional part of the number
- This is the *fixed point* representation
  - However, fixed point is inherently limited and is not used much

Scientific notation (base 10)

- Consider the scientific notation for representing a number, e.g.
  - $1.3806503 \times 10^{-23}$
- This consists of
  - The **mantissa** (here 1.3806503)
  - The **base** (here 10)
  - The **exponent** (here -23)

Changing to base 2

- In a digital computer, we use base 2. The number we had earlier may be written as
  - $1.043189872 \times 2^{-76}$
- In binary, the mantissa is
  - 1.1 1110 0010 1101 1101 0000 0111
  - In fact, we can drop the leading '1' since in this form it will always be there and may be implied. So the mantissa is
  - 1 1110 0010 1101 1101 0000 0111

Floating point numbers

- The mantissa and exponent may now be stored as integers with a sign bit (the mantissa is not usually converted to twos complement form)
- A number stored in this way is known as a **floating point number**
- Typically, such data types are given the name **float**

Precision of floating point numbers

- Since reals may be irrational (or require more digits than provided) floating point numbers are not necessarily exact (whereas integers are)
- We can increase the precision by increasing the number of bits used to store the number
- Typically, a data type known as a **double** has twice the number of bits as a **float**

Boolean variables

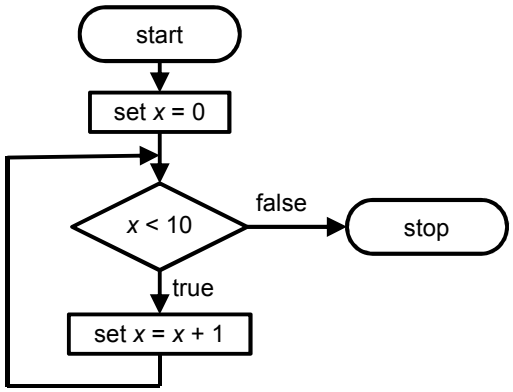
- A Boolean variable can take one of two values
- Often interpreted as *true* and *false*
- Examples seen in logic circuits (taking values ‘1’ and ‘0’
- However, using 1 byte, it is often the case that
  - ‘00000000’ = ‘false’
  - ‘11111111’ = ‘true’ (i.e. -1)

Program control

- Boolean variables are used to test cases for program control, e.g.

Algorithm for counting from 0 to 9

Note: (x < 10) evaluates to a Boolean variable, i.e. either ‘true’ or ‘false’.



Text characters

- Since we can represent numbers in binary, we can map a number to a printing character. For example
  - ASCII (American Standard Code for Information Interchange)
  - or, more recently, UNICODE

Data types in general

- In general, if a quantity can be represented by a number (or mapped by a number), then it can be represented in digital form and a data type defined for it.

Example of data processing

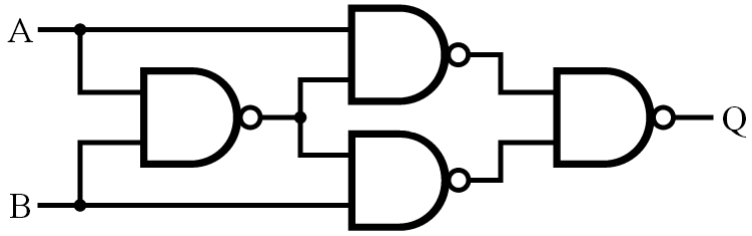
- Adding two numbers
- We shall use the XOR gate, shown in the next two slides

The XOR gate

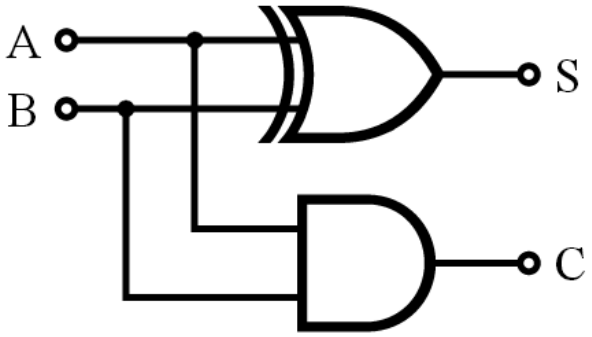
- The XOR or 'exclusive OR' gate has the truth table

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

The XOR gate in NAND gates

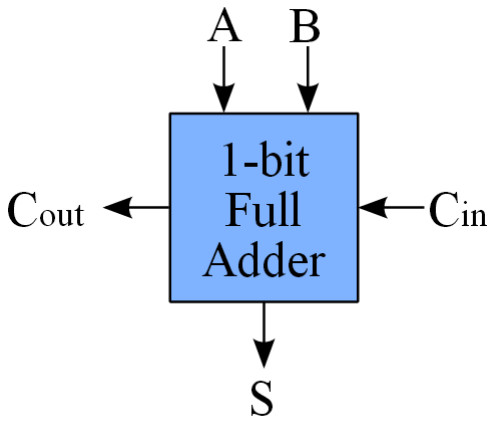


The half adder

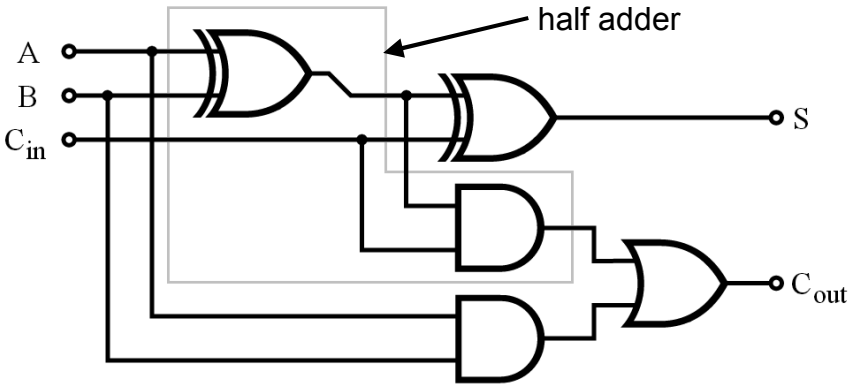


Adds two binary digits, giving the sum (S) and the carry (C) digits

The full adder – block diagram



The full adder – logic circuit



Full adder logic circuit. Note the half adder circuit within the grey lines