University of Nottingham
UK | CHINA | MALAYSIA

Computer Engineering and Mechatronics MMME3085

Dr Louise Brown

Appendix 2

All About Numbers (not in the book)

Numbers

On a computer, the basic building block for a number is the BYTE

- 8 bits
- Can range from 00000000 to 11111111
- Can be signed or unsigned

Numbers – byte sizes

Each variable is made up of a number of bytes, and this defines the range of numbers possible.

We can obtain the size using the **`sizeof`** function in C

Some machines will have the same size for variables that on other machines will be different (especially short int)

Numbers - range

The number of bytes used defines (in the case of integer types) the range of numbers that can be stored:

- 2 Byte (16 bits) has the range
  - 0 to (2^{16} – 1) or 0 to 65535 (unsigned)

  or
  - (0 – 2^{15}) to (2^{15} –1) or   -32768 to 32767 (signed)

Numbers - limits

The limits of a variable are machine specific (with the possible exception of char) as they depend on the number of bytes used for storage

With each machine/compiler is shipped a file ‘limits.h’ which has the permissible range

Decimal numbers

What do these numbers represent?
What are their values?

1056.2458

What is this called?

Fixed point numbers (1)

A method of representing fractions using binary numbers

Fixed point representation of fractions:

- In a fixed point representation, the **binary point** is understood to always be in the same position. The bits to the **left** represent the **integer** part and the bits to the **right** represent the **fraction** part :

- The integer parts go as 2^n (1,2,4,8 etc)

- The fraction parts go as 2^{-n} (0.5, 0.25, 0.125 etc) . The overall value is formed using a sum of these.

Fixed point numbers (2)

Example:
A fixed point system uses 8-bit numbers. 4 bits for the integer part and 4 bits for the fraction:

What number is represented by **00101100**?

Fixed point numbers (3)

Example:
A fixed point system uses 8-bit numbers. 4 bits for the integer part and 4 bits for the fraction:

What number is represented by **00101100**?

8	4	2	1	0.5	0.25	0.125	0.0625
0	0	1	0	1	1	0	0

Fixed point numbers (4)

Example:
A fixed point system uses 8-bit numbers. 4 bits for the integer part and 4 bits for the fraction:

What number is represented by **00101100**?

8	4	2	1	0.5	0.25	0.125	0.0625
0	0	1	0	1	1	0	0

NOTE : The headings for the fraction part are divided by 2 successively to the right...

The number **00101100** represents the number **2.75**.

Fixed point numbers (5)

Example:

Converting to Fixed Point – there is an easy way !

EG. 56.78125

Stage 1 : Integer part: easy

Sign	64	32	16	8	4	2	1
0	0	1	1	1	0	0	0

Fixed Point Numbers (56.78125)

Example: - fraction part

0	.	7	8	1	2	5
					x	2
(1)	.	5	6	2	5	0
					x	2
(1)	.	1	2	5	0	0
					x	2
(0)	.	2	5	0	0	0
					x	2
(0)	.	5	0	0	0	0
					x	2
(1)	.	0	0	0	0	0

You keep multiplying, ignoring the value in brackets until you get zero or run out of bits to use

So reading down we get

11001

So final answer is

00111000 11001000

Fixed Point Numbers - Example

Your Turn :

Convert the value 25.3 to a fixed point representation,
8 bit mantissa, 8 bit exponential

Fixed Point Numbers – Solution (1)

Example: - integer part

```
0 0011001
s      25
```

Fixed Point Numbers – Solution (2)

Example: - fraction part

0	.	3	0
(0)	.	6	0
(1)	.	2	0
(0)	.	4	0
(0)	.	8	0
(1)	.	6	0
(1)	.	2	0
(0)	.	4	0
(0)	.	8	0

Fixed Point Numbers – Solution (3)

Example: - fraction part

0	.	3	0	
(0)	.	6	0	0.5
(1)	.	2	0	0.25
(0)	.	4	0	0.125
(0)	.	8	0	0.0625
(1)	.	6	0	0.03125
(1)	.	2	0	0.015625
(0)	.	4	0	0.0078125
(0)	.	8	0	0.00390625

01001100
= 0.25 + 0.03125 + 0.015625
= 0.296575 ←

Not exactly 0.3!

Answer: 0 0011001 01001100
s 25 . 3

Limitations of fixed point representation

Fixed window limits representation of both very large and very small numbers

Prone to loss of precision when two large numbers are divided.

Scientific notation

Most common solution is to use scientific notation with a base and exponent:

- 123.456 -> 1.23456 x 10^2 in decimal
- 789.abc -> 7.89abc x 16^2 in hex
- 1010.110 -> 1.010110 x 2^3 in binary

- Giving a sliding scale of precision to maximise precision for both very large and very small numbers

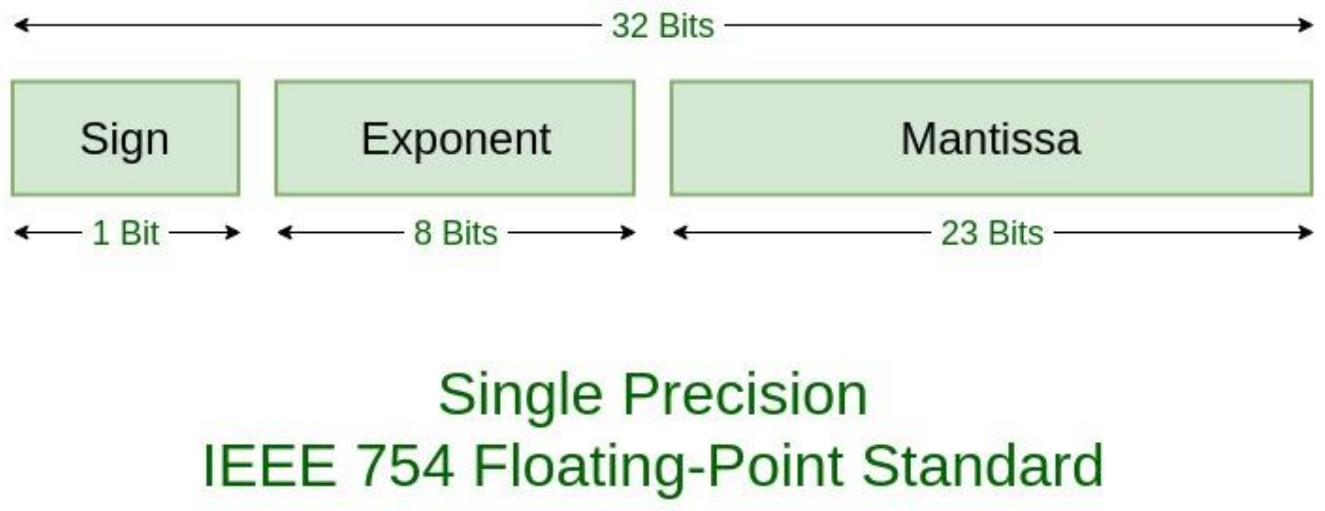
Floating Point Numbers

These are held using the IEEE 754 Floating Point Model which has 3 components:

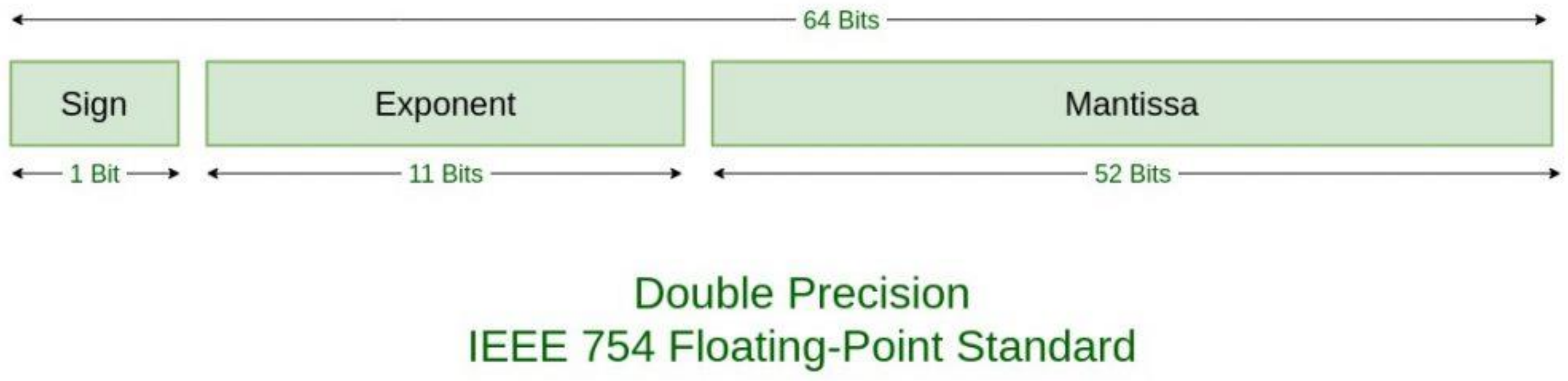
- Sign of mantissa – 0 for positive number, 1 for negative
- Biased exponent – addition of a bias enables both positive and negative exponents to be represented
- Normalised mantissa – part of number in scientific notation giving the significant digits. In binary this must be 0 or 1 so a normalised mantissa has 1 to the left of the decimal

- These may be single or double precision

Single and Double Precision Floating Point Numbers



Bias = 127



Bias = 1023

https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/

In summary

1. The sign bit is 0 for positive, 1 for negative.
2. The exponent base is two.
3. The exponent field contains 127 plus the true exponent for single-precision, or 1023 plus the true exponent for double precision.
4. The first bit of the mantissa is typically assumed to be 1, yielding a full mantissa of 1.*f*, where *f* is the field of fraction bits.

https://steve.hollasch.net/cgindex/coding/ieeefloat.html

Convert 25.3 to single precision floating point binary (1)

- 25 = 11001
- $0.3 = 0100\underbrace{1100}_{\text{This section will repeat}}110011\ldots$

0	.	3	0	
(0)	.	6	0	0.5
(1)	.	2	0	0.25
(0)	.	4	0	0.125
(0)	.	8	0	0.0625
(1)	.	6	0	0.03125
(1)	.	2	0	0.015625
(0)	.	4	0	0.0078125
(0)	.	8	0	0.00390625

Convert 25.3 to single precision floating point binary (2)

25 = 11001

0.3 = 0100110011001100011…

25.3 = 11001.0100110011001100011

$= 1.\underbrace{10010100110011001100110}_{} \times 2^4$

Normalised mantissa = 10010100110011001100110   (23 bits)

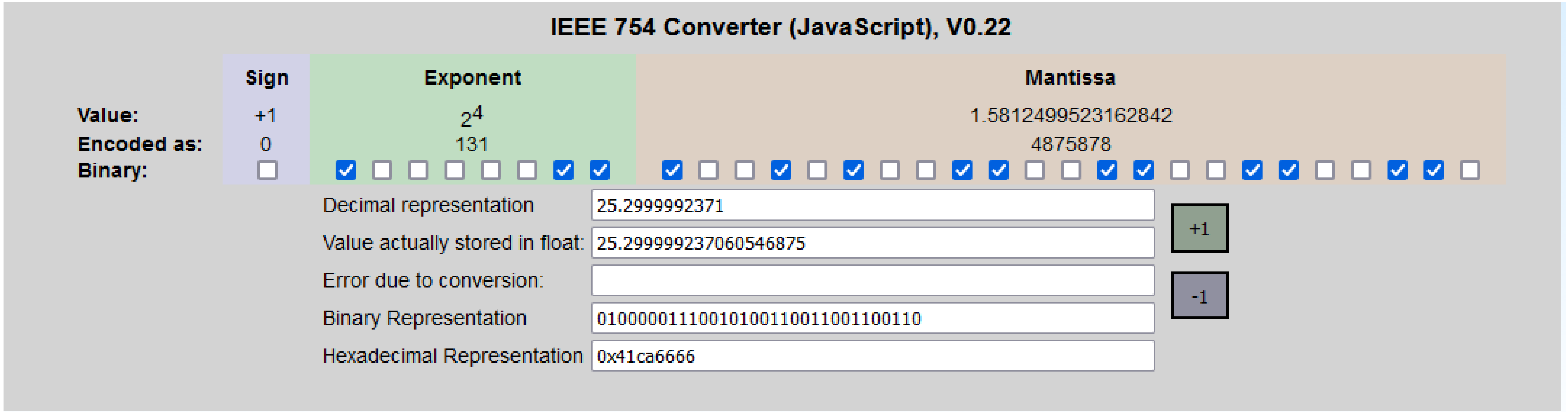
Sign = 0

Biased exponent = 127 + 4 = 131

= 10000011

Single precision is: **0 10000011 10010100110011001100110**

It’s not actually 25.3!



IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^4	1.5812499523162842
Encoded as:	0	131	4875878
Binary:	☐	✓ ☐ ☐ ☐ ☐ ☐ ✓ ✓	✓ ☐ ☐ ✓ ☐ ✓ ☐ ☐ ✓ ✓ ☐ ☐ ✓ ✓ ☐ ☐ ✓ ✓ ☐ ☐ ✓ ✓ ☐

Decimal representation	25.2999992371
Value actually stored in float:	25.299999237060546875
Error due to conversion:	
Binary Representation	01000001110010100110011001100110
Hexadecimal Representation	0x41ca6666

+1

-1

https://www.h-schmidt.net/FloatConverter/IEEE754.html

Floating point numbers

This representation however has limitations:

- Even with a 23 bit mantissa some numbers can appear the same
- We can extend to double precision (51 bit mantissa, 12 bit exponent) but we can still have inaccuracies
- Adding very large to very small numbers can cause considerable problems
  - https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
- Care should be taken with comparisons

Variable Selection

We aim to use the most suitable variable for the type of number we are storing

The choice of variable has consequences both for overall memory usage and speed of calculation

Integer mathematics is considerably faster than floating point calculations.

Byte ordering (1)

In an ideal world, all machines would use the same byte size for variables and would arrange the bytes in the same way

However… (as mentioned when we discussed binary files)

Byte ordering (2)

Different machines arrange the bytes used to make a numbers using one of two ordering types

- Big endian – high order byte comes first
- Little endian – low byte comes first

So for a two byte integer

- Big Endian - High byte, low byte
- Little Endian - Low byte, high byte

- For more information: https://developer.ibm.com/articles/au-endianc/

Project Planning Feedback

Project Planning Assignment Feedback

Generally very good planning ☺

- Don’t hard code the shape list – it is defined by the shapes in the file and these could change
- There was generally a lack of detail about how the shapes would be stored
- Many had missed that the pen up/down data needs to be stored as well as the x/y coordinates
- There was nothing in the brief to say that the grid size was an integer

Flowcharts

- Try to keep the main flow of the program moving vertically down the page
- Many flowcharts didn’t include checking for success opening file
  - Some checked in a function and returned an appropriate value but then didn’t do anything with this in the calling program
- Many flowcharts showed function calls but didn’t include the flowchart for the functions
- Don’t have crossing lines in the flowchart
- Always use a diamond box for decisions
- Make sure the outcome (yes/no, true/false) is shown on the exits from the decision boxes
- Only decision boxes should have multiple exits

Data Types

- The ‘Data Type’ should give the actual data type, eg int, float*, struct Shape
- Where structures are defined, give the actual definition and the data items that are stored within it
- The rationale should include the reason why you have chosen a given data type, e.g. int for pen up/down as it can only take integer values.

Test Data

This is probably the section which completed least well.

- Be precise with the test data
    - Give the actual data that would be passed into the function with the resulting output

- eg,

Function	Test Case	Input	Expected Output
ReadFile()	Invalid file	“nonexistent.txt”	Returns 0

- Ideally, there should be a test to cover each path through the program

January Exam

Exam Information

Exam weighting:

- Computer Engineering 40%
    - Two questions
    - These will be similar in length to previous years but there will only be two instead of three
- Mechatronics 60%

- Programs will be developed using VSCode on the Engineering Virtual Desktop.

- Upload program answers to Moodle

Accessing the Engineering Virtual Desktop

The computer engineering section of the January exam will require you to develop programs using VSCode. In order to access this from all exam rooms you will need to use the software via the Engineering Virtual Desktop.

Please make sure that you can access the Engineering Virtual Desktop (instructions are given here: https://www.nottingham.ac.uk/dts/communications/remote-working/virtual-desktop.aspx).

Also make sure that you can open VSCode and create and compile a C program.

There is a submission box on Moodle in the section **Test access to VSCode on Engineering Virtual Desktop**

Please upload a file from the Virtual Desktop to this submission box so that we know you are ready to sit the exam.

Hope you enjoyed the module!

Please take a few minutes to fill in the SEM survey

https://bluecastle-uk-surveys.nottingham.ac.uk