COMP20005 Workshop Week 9

1	`Representation of integers`
2	`Representation of floats`
3	`Numerical Computation & Root Finding`
4	`Small-group exercises`
5	`Implement:` • `Ex 9.11 if you think you remember physics` • `Output the bit-patterns of a double value`

Numeral Systems

$$2 \quad 1 \quad 1 \;.\; 3 \quad 9$$

$$2 \text{ x } 10^2 + 1 \text{ x } 10^1 + 1 \text{ x } 10^0 \quad + \quad 3 \text{ x } 10^{-1} + 9 \text{ x } 10^{-2}$$

→ *base* = 10

Other bases: binary (base= 2), octal (base= 8) ...

$21.3_{(8)} = 2 \text{ x } 8^1 + 1 \text{ x } 8^0 + 3 \text{ x } 8^{-1} = 17.375_{(10)}$

$1001_{(2)} = 1\text{x}2^3 + 0\text{x}2^2 + 0\text{x}2^1 + 1\text{x}2^0 = 9_{(10)}$

$1021_{(2)} = ?$

Changing Binary → Decimal

Just expand using the base=2:

$$\ldots b_3 \quad b_2 \quad b_1 \quad b_0 \,.\, b_{-1} \quad b_{-2} \ldots \;{}_{(2)}$$

is $\ldots\ b_3x2^3 + b_2x2^2 + b_1x\ 2^1 + b_0 \ + \ b_{-1}x2^{-1} + b_{-2}x2^{-2} \ldots$

so $1\ 0\ 1\ 0\ 0\ 1$ *and* $1\,.\,1\ 0\ 1$

are $2^5 + 2^3 + 1 = 41$ *and* $1 + 2^{-1} + 2^{-3} = 1.625$

Practical advise: remember

128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Changing Decimal → Binary

Remember and apply the power-of-two sequence:

128	64	32	16	8	4	2	1	.	0.5	0.25	0.125
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}

So:

$11_{(10)} = 8 + 0 + 2 + 1 = 1101_{(2)}$

$2.625_{(10)} = 2 + 0 + 0.5 + 0 + 0.125 = 10.101_{(2)}$

Algorithm: Decimal→Binary, Integer Part

Changing integer x to binary: Just divide x and the subsequent quotients by 2 until getting zero. The sequence of remainders, in reverse order of appearance, is the binary form of x.

Example: 23

operation	quotoion	remainder
23 :2	11	1
11:2	5	1
5:2	2	1
2:2	1	0
1:2	0	1

So: $23 = 10111_{(2)}$

Algorithm: Decimal → Binary, Fraction Part

Fraction: Multiply it, and subsequent fractions, by 2 until getting zero. Result= sequence of integer parts of results, in appearance order. Examples:

0.375				0.1		
operation	int	fraction		operation	int	fraction
.375 x 2	0	.75		.1 x 2	0	.2
.75 x 2	1	.5		.2 x 2	0	.4
.5 x 2	1	.0		.4 x 2	0	.8
				.8 x 2	1	.6
				.6 x 2	1	.2

So: $0.375 = 0.011_{(2)}$ $\quad 0.1 = 0.00011(0011)_{(2)}$

Now try convert: 6.875 to binary

Converting Decimal->Binary

- $17_{(10)}$ = $?_{(2)}$
- $34_{(10)}$ =
- $6.375_{(10)}$ =

Representation of integers

representation of integers in w bits

CASE	w data bits for:
x ≥ 0	binary form of x
x < 0	twos-compliment form of |x|

Examples using `w=4`*, representation:*

- *of* 2 *is* `0010`
- *of* 5 *is* `0101`
- *of* -5 *is* `1011`

Finding twos-complement representation in w bits for negative numbers in 3 step

Suppose that we need to find the twos-complement representation of $-$ `x`, *where* `x` *is positive, in* `w=16` *bits. It can be done easily in 3 steps:*

1) *Find binary representation of* |`x`| *in w bits*
2) *Take the result above, inverse* `1` *to* `0` *and vice versa*
3) *Add* `1` *to the above to get the final twos-complement representation*

***find the 2-comp repr of* –40**	**Bit sequence**			
1) bin repr of 40 in 16 bits	0000	0000	0010	1000
2) inverse	1111	1111	1101	0111
3) add 1	1111	1111	1101	1000

Note: Step 3 (adding `1`*) can be easily be done by:*

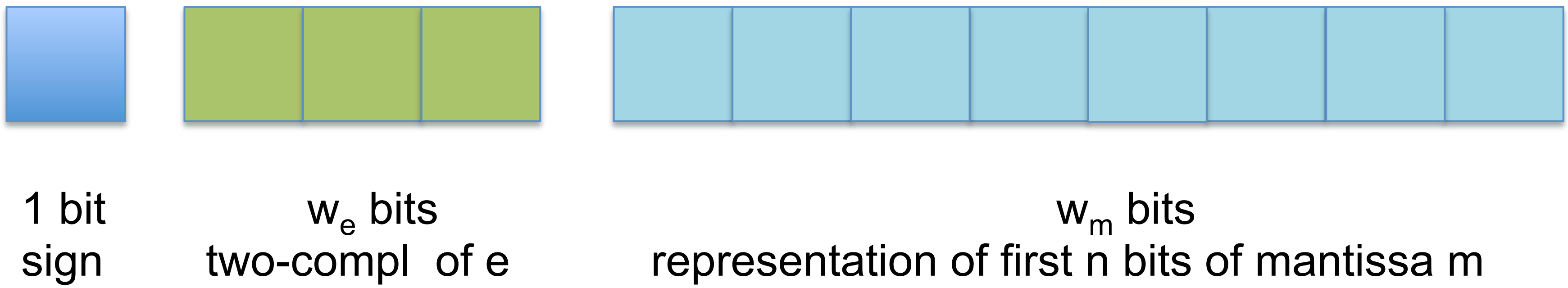
- *finding the right most* `zero-bit`*, then*
- *inversing all bits from this position to the right end.*

This note can be combined with steps 1-2 to make a shorter algorithm ☺

Ex: 2-complement representation in w=16 bits

Q: What are `17`, `–17`, `34`, and `–34` as 16-bit twos-complement binary numbers, when written as (a) binary digits, and (b) hexadecimal digits?

Representation of floats



- Convert $|x|$ to binary form, and transform so that:

  $|x| = 0.b_1b_2b_3\ldots \text{ x } 2^{\mathbf{e}}$ *with* $b_1 = 1$
- **e** is called exponent, **m**= $b_1b_2b_3\ldots$ is called mantissa
- Three components: sign, **e**, **m** are represented as in the diagram.

Ex: Representation of float

Q: Using the same representation as on page 27 (`w=16, ` w_s`=1, ` w_e`=3, ` w_m`=12`) of the numericA.pdf slides, what are the floating point representations of `-3.125`, `17.5`, and `0.09375`, when written as (a) binary digits, and (b) hexadecimal digits?

Quiz

What is the binary form of $13.625_{(10)}$?

```
A. 1101.101
B. 1011.101
C. 1101.11
D. 1011.11
```

Quiz

What is the binary form of $65535_{(10)}$?

```
A. 1000 0000 0000 0001
B. 1000 0000 0000 0000
C.   111 1111 1111 1111
D. 1111 1111 1111 1111
```

Numerical Computations

`int`, `float`, `double` all have some range:

- `int` 32 bits: about $-2\text{x}10^9$ – $2\text{x}10^9$
- `int` 64 bits:
- `float`
- `double`

computation on `float/double` is imprecise

- use `fabs(x) < EPS` instead of `x==0`
- adding a large value to a small value might have problems

Roof Finding for f(x)=0

Know about the methods:

- bracketing vs open
- bisection
- false position
- fixed-point iteration
- Newton-Raphson
- secant

Exercise

Re-examine the cube_root() function on page 77 of the textbook, croot.c. What method does it use? Explore what happens if: (a) very large numbers are provided as input; (b) very small (close to zero) numbers are provided; and (c) if ITERATIONS is made larger or smaller.

```
double cube_root(double v) {
    double x= 1.0;
    int i;
    …
    for (i=0; i<ITERATIONS; i++) {
        x= (2*x+v/(x*x))/3;
    }
    return x;
}
```

Lab

1. Ex 9.11 if you think you remember physics
2. Output the bit-patterns of a double value

Hints for 2): mimic Alistair's f`loatbits.c`, or DIY by

a) using a "`unsigned long long`" pointer to access the "`double`" variable

b) using bitwise operations to get value of a bit. For example, if `n` is an "`unsigned long long`" then:

```
( n >> (63-i) ) & 1
```

would give the value of the `i`-*th* bit of `n` (from the left).

Decimal→Binary: Integer Part

Changing integer x to binary: Just divide x and the subsequent quotients by 2 until getting zero. The sequence of remainders, in reverse order of appearance, is the binary form of x.

Example: 23

operation	quotoion	remainder
23 :2	11	1
11:2	5	1
5:2	2	1
2:2	1	0
1:2	0	1

So: $23 = 10111_{(2)}$

Decimal → Binary: Fraction Part

Fraction: Multiply it, and subsequent fractions, by 2 until getting zero. Result= sequence of integer parts of results, in appearance order. Examples:

0.375				0.1		
operation	int	fraction		operation	int	fraction
.375 x 2	0	.75		.1 x 2	0	.2
.75 x 2	1	.5		.2 x 2	0	.4
.5 x 2	1	.0		.4 x 2	0	.8
				.8 x 2	1	.6
				.6 x 2	1	.2

So: $0.375 = 0.011_{(2)}$ $0.1 = 0.00011(0011)_{(2)}$

Now try convert: 6.875 to binary