Encoding Information

Why do computers need to encode information?

Why do computers need to encode information?

- Because they can only deal with 0’s and 1’s

What do we mean by encoding information for computers?

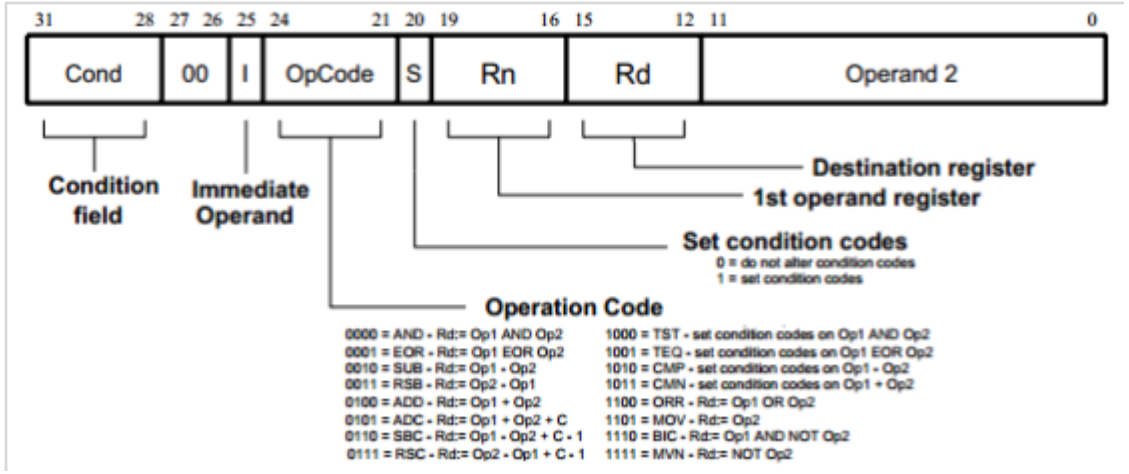
What do we mean by encoding information for computers?

- Numbers
  - (binary)

What do we mean by encoding information for computers?

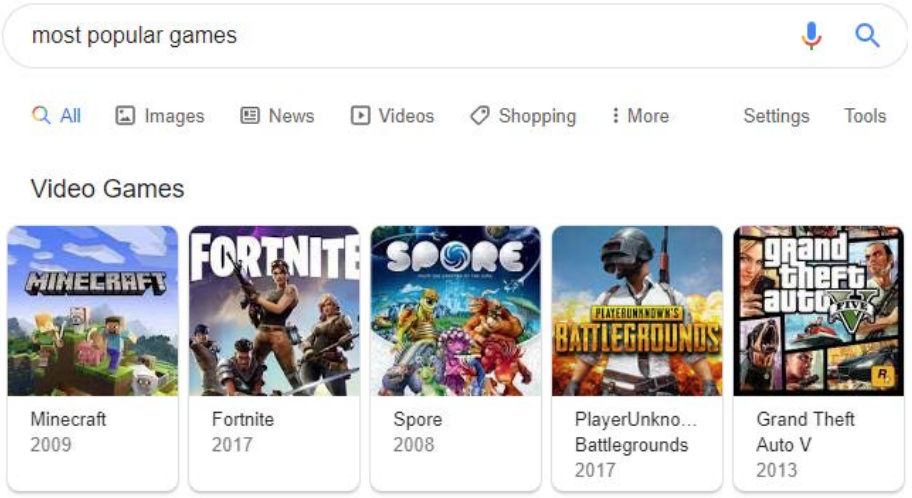
- Numbers
  - (binary)
- Colours
  - Red/green/blue
- Machine Code

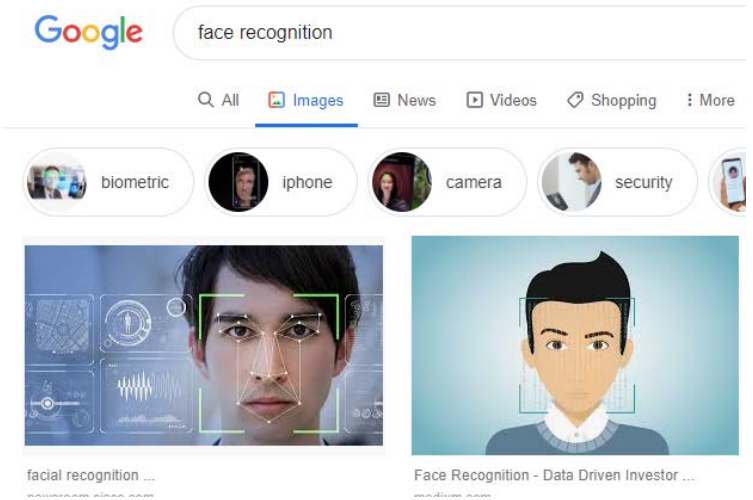
```
                Cond   I OpCd S  Rn Rd Op2
ADD    R0,  R1, R2    @ 1110|00|0|0100|0|0001|0000|000000000010
```



What do we mean by encoding information for computers?

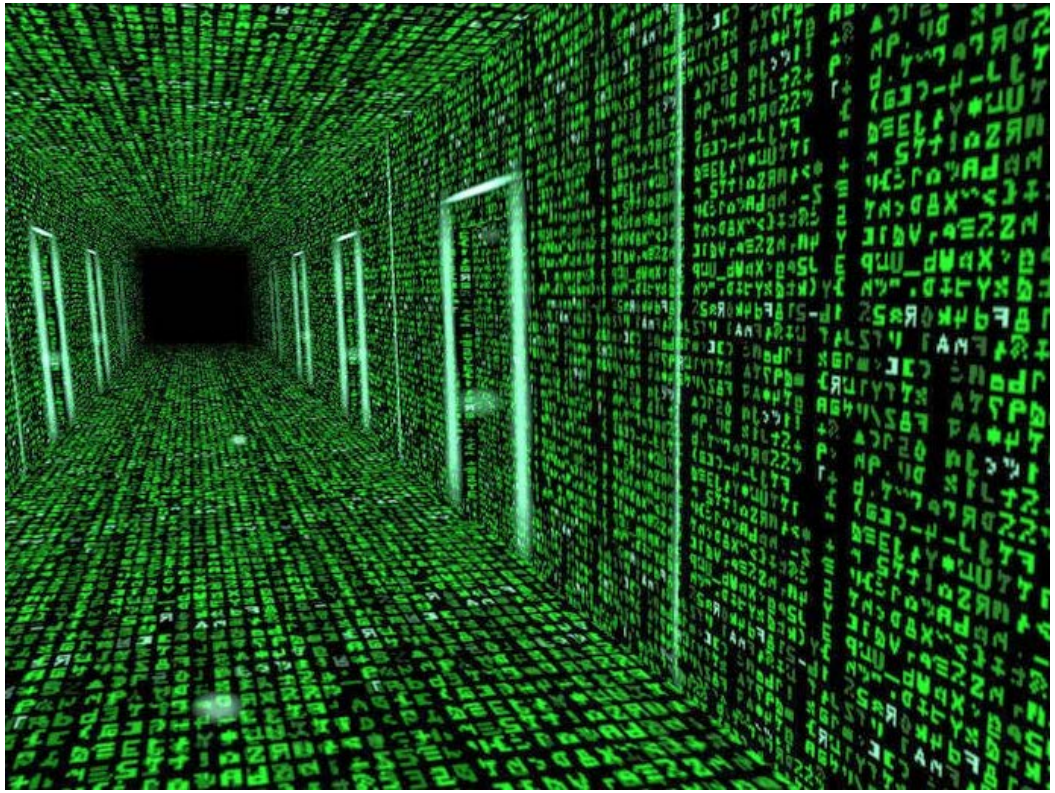
- Anything that you possibly want to do with a computer

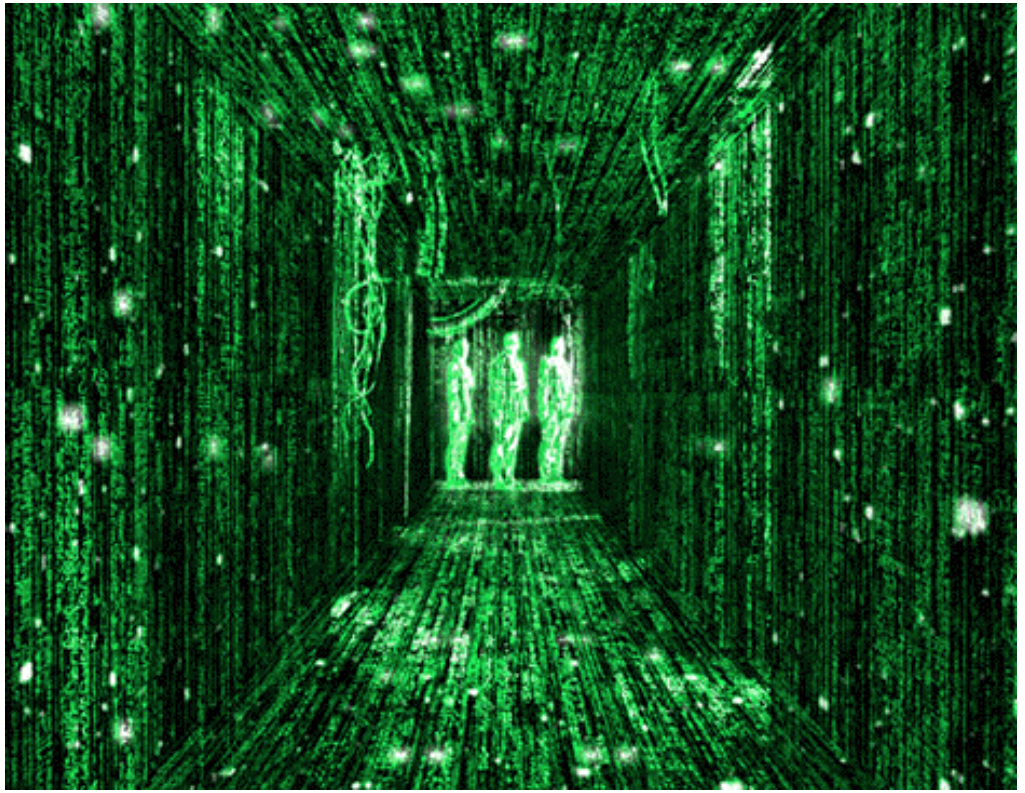




What do we mean by encoding information for computers?

- Literally everything…





Start with numbers…

- Why?
  - Computers very good at arithmetic
  - Computers do a lot of arithmetic

Start with natural numbers $\mathbb{N}$

- What are they?

How do we count numbers in base 10?

How do we count numbers in binary?

- Same way, but max of 2

What do numbers in base 10 mean

- Consider the number 1429
- How do we break it down?

What does a binary number mean

- Consider the number 111010 mean
- How do we break it down?

How do we convert numbers between bases

- Convert 3827_{10} to binary?

Why Octal/Hexadecimal

- How do you think this monitor is encoded?

Why Octal/Hexadecimal

- Make binary more human readable
- Conversion (from binary) is easy

First exercise from worksheet

Useful properties of base 10

- Is 1345 a multiple of 10? Is it a multiple of 100?
- Is 13450 a multiple of 10? Is it a multiple of 100?
- Is 10400 a multiple of 10? Is it a multiple of 100?

- Is this hard?

Useful properties of base 10

- How do you multiply 1274 by 10?
- How do you multiply 1830 by 100?

- Is this hard?

Useful properties of base 10

- How do you divide 10400 by 10?
- How do you divide 10400 by 100?

- Is this hard?

Useful properties of base 10

- Is 1345 a multiple of 7? Is it a multiple of 77?
- Is 13450 a multiple of 70? Is it a multiple of 777?
- Is 10400 a multiple of 70? Is it a multiple of 777?
- How do you multiply 1274 by 7?
- How do you multiply 1830 by 77?
- How do you divide 10400 by 77?
- How do you divide 10400 by 777?

- Is this hard?

Useful properties of base 2

- Is 101100 a multiple of 2? Is it a multiple of 4?

- How do you multiply 1010010 by 2?
- How do you multiply 1011110 by 4?

- How do you divide 10101010 by 2?
- How do you divide 10111100 by 4?

- Is this hard?

Useful properties of base 8, base 16

- Worksheet exercise 2

Couple of bonus questions

- What is 100_{16} -1 ?

How many different numbers can you represent?

- 10 digits?
- 10 octal values?
- 10 hex values?
- 10 bits?

Encoding integers $\mathbb{Z}$

- What is the difference between the set of integers and set of natural numbers?

Encoding integers $\mathbb{Z}$

- What is the difference between the set of integers and set of natural numbers?
- Sign-magnitude: the easiest way to represent negative numbers
  - Add a sign bit

Encoding integers $\mathbb{Z}$

- What is the difference between the set of integers and set of natural numbers?
- Sign-magnitude: the easiest way to represent negative numbers
  - Add a sign bit

- Is this a good representation?

How do we add sign-magnitude numbers

How do we add positive numbers

- How do we add 1503+1729?

How do we add positive binary numbers

- How do we add 1001+1011?

How do we add sign-magnitude numbers

- Suppose we have two binary numbers in sign-magnitude representation:

- A+B (A+ve, B+ve)
- A+B (A+ve, B-ve)
- A+B (A-ve, B-ve)

One’s complement

- Invert all bits
- Simpler addition

Two’s complement

- Easy to compute
  - Invert all bits and add 1
- Has a direct understanding
- Simpler addition still

Worksheet Q2

Why Floating-point?

Encoding Real Numbers $\mathbb{R}$

- Do we need floating-point to encode real numbers?

Why Floating-point?

- Can we represent natural numbers $\mathbb{N}$ with fixed point?
- Can we represent integer numbers $\mathbb{Z}$ with fixed point?

Why Floating-point?

- You have 8 bits. How do you represent numbers between 0 and 63?

Why Floating-point?

- You have 8 bits. How do you represent numbers between 0 and 63?
- You have 8 bits. How do you numbers between 0 and 126?

Why Floating-point?

- You have 8 bits. How do you represent numbers between 0 and 63?
- You have 8 bits. How do you numbers between 0 and 126?
- Is 8-bits sufficient to represent numbers between 0 and 63?

How many different values can you represent?

- 10 digits?
- 10 octal values?
- 10 hex values?
- 10 bits?

Why Floating-point?

- You want to represent a wide range with a fixed number of bits?

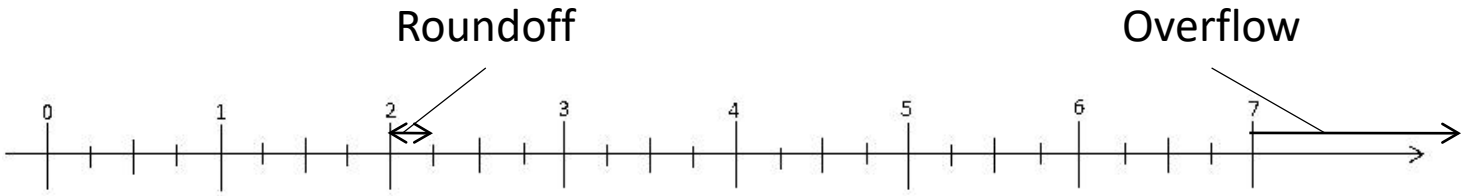
Why Floating-point?

- You want to represent a wide range with a fixed number of bits?

- Reason for floating-point: You want to represent a *variable* range with a fixed number of bits

Fixed vs Float

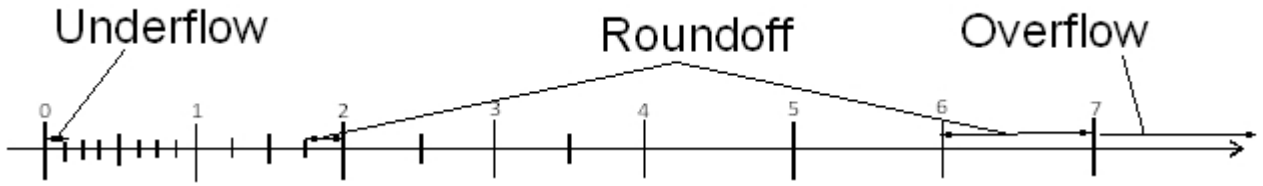
5 bit fixed point, 2 bits integer, 3 bits fractional



Number system must not overflow (error unbounded)
Accumulation of round-off errors must lie below tolerable region

$$x = \pm 2^{e} \times 0.b_1 b_2 \ldots b_m \qquad , b_i \in \{0,1\}$$

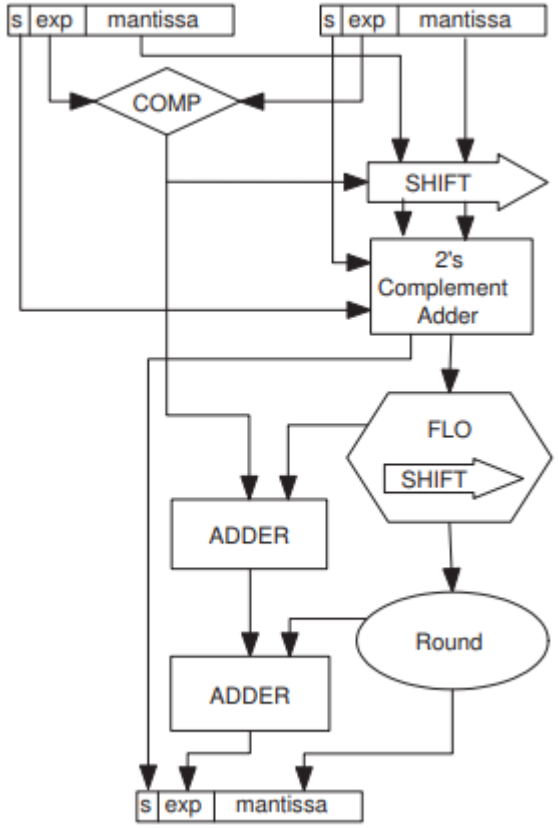
5 bit floating point, 2 bits exponent, 3 bits mantissa



Fixed vs floating point

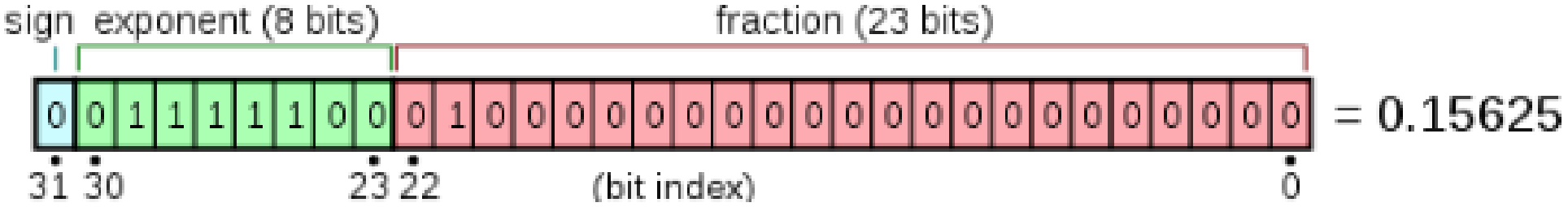
- Fixed point:
  - Number system must not overflow (error unbounded)
  - Accumulation of round-off errors must lie below tolerable region
- Floating-point similarities:
  - Number system must not overflow
  - Accumulation of round-off errors must lie below tolerable region
- Differences:
  - Bigger dynamic range
  - Underflow
  - Size of roundoff errors vary relative to the representable value
  - Floating point is not associative

Is floating-point good?

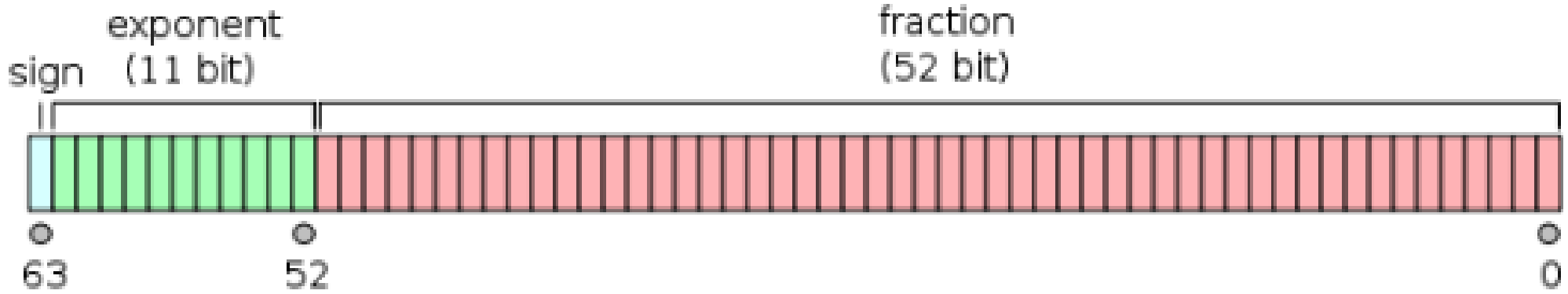


Notes vs IEEE 754 Standard Floating Point

- Single precision:

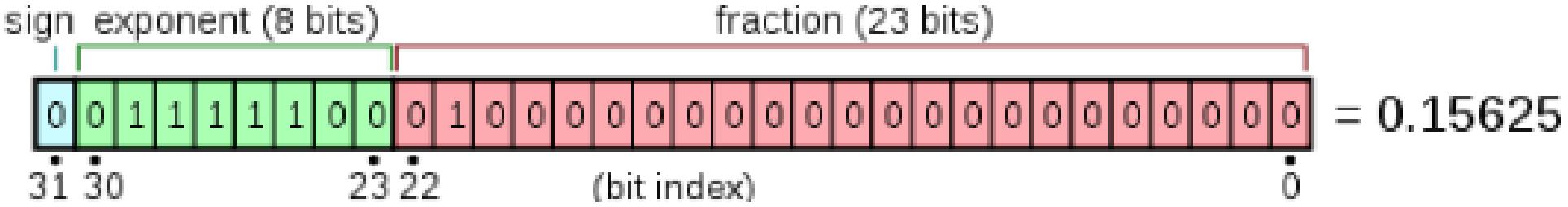


- Double precision

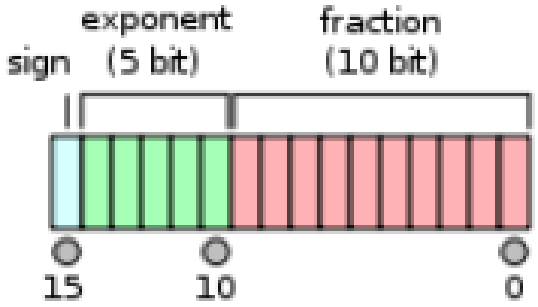


Notes vs IEEE 754 Standard Floating Point

- Single precision:



- Half precision



Notes vs IEEE 754 Standard Floating Point

- IEEE standard
  - Implicit leading 1
  - Special codes:
    - NaN (0/0, sqrt(-10))
    - Infinity (100/0, -100/0)
  - Exponent subtract a bias (127 for single gives range -126 to +127)

Important to remember

- Floating point needs its representation defined

Floating-point practice

- Worksheet Q3

Floating-point range/precision/underflow/overflow

- How do you increase range?
- How do you increase precision?
- What is overflow?
- What is underflow?

- Worksheet Q4

Floating-point games

Floating point is not associative

$$\sum_{i=1}^{1e7} i$$

Which answer is correct?

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        float j,k,l;
        int i;
        j=0;
        for (i = 1e7; i >= 0; i--)
        {
                j=j+i;
        }
        printf("j is %lf\n", j);
        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test3
j is 48714886414336.000000
```

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        float j,k,l;
        int i;
        j=0;
        for (i = 0; i <= 1e7; i++)
        {
                j=j+i;
        }
        printf("j is %lf\n", j);
        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test3b
j is 49423623127040.000000
```

Floating point is not associative

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        double j,k,l;
        int i;
        j=0;
        for (i = 1e7; i >= 0; i--)
        {
                j=j+i;
        }
        printf("j is %lf\n", j);
        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test3c
j is 50000005000000.000000
```

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        double j,k,l;
        int i;
        j=0;
        for (i = 0; i <= 1e7; i++)
        {
                j=j+i;
        }
        printf("j is %lf\n", j);
        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test3d
j is 50000005000000.000000
```

How do you add numbers in FP

Evaluate 3+0.625+0.875, with 3 bit mantissa

- Compute 3+0.625

$$x = \pm 2^{e} \times 0.b_1 b_2 \ldots b_m$$

  - 3 is 2^2x0.110
  - 0.625 is 2^0x0.101

- Align exponents
  - 3 is 2^2x0.110
  - 0.625 is 2^2x0.00101
- Perform addition
  - 2^2x0.11101
- Re-normalise to 5 bits
  - 2^2x0.111 = 3.5
  - Roundoff error of 0.125

- Compute 3.5+0.875.
  - 3.5is 2^2x0.111
  - 0.875 is 2^0x0.111
- Align exponents
  - 3.5 is 2^2x0.111
  - 0.875 is 2^2x0.00111
- Perform addition
  - 2^2x1.00011
- Re-normalise to 5 bits
  - 2^3x0.100 = 4
  - Roundoff error of 0.375

How do you add numbers in FP

Evaluate 0.625+0.875+3, with 3 bit mantissa

- Compute 0.625+0.875

$$x = \pm 2^{e} \times 0.b_1 b_2 \ldots b_m$$

  - 0.625 is 2^0x0.101
  - 0.875 is 2^0x0.111

- Align exponents
  - 0.625 is 2^0x0.101
  - 0.875 is 2^0x0.111
- Perform addition
  - 2^0x1.100
- Re-normalise to 5 bits
  - 2^1x0.110 = 1.5
  - No roundoff error

- Compute 1.5+3
  - Result is 2^1x0.110
  - 3 is 2^2x0.110
- Align exponents
  - 3 is 2^2x0.110
  - 0.875 is 2^2x0.0110
- Perform addition
  - 2^2x1.001
- Re-normalise to 5 bits
  - 2^3x0.101 = 5
  - Roundoff error of 0.5

Not just round-off errors can go wrong

$$\sum_{i=1}^{2^{25}} 2^{-25} = 1$$

Floating point games

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        float j,k,l;
        int i;
        k=1;
        l=k;
        for (i = 0; i < 25; i++)
        {
                k=k/2;
                l=l*2;
        }
        j=0;
        for (i = 0; i < l; i++)
        {
                j=j+k;
        }
        printf("j is %lf\n", j);
        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test1
j is 0.500000
```

$$\sum_{i=1}^{2^{25}} 2^{-25} = 1$$

```c
#include <stdio.h>
#include "stdlib.h"
int main ( int argc, char *argv[] )
{
        float j,k,l,m,n,o;
        int i;
        l=1;
        m=1;
        for (i = 0; i < 25; i++)
        {
                l=l/2;
                m=m*2;
        }
        j=0;
        k=0;
        for (i = 0; i < m/2; i++)
        {
                j=j+l;
                k=k+l;
        }

        printf("j is %lf\n", j);
        printf("k is %lf\n", k);
        j=j+k;
        printf("j is %lf\n", j);

        return 0;
}
```

```
davidb@goliath ~ $ ./simple_prec_test1b
j is 0.500000
k is 0.500000
j is 1.000000
```