Week 3

Outline

- A bit more on fixed vs floating-point
- Recap on why floating-point is a challenge

- More on encoding symbols

- How memory works

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 Point

- What numbers does 8-bit fixed-point represent?

Fixed Point

- Suppose I represent an 8-bit number that lies in the range [-2, 2] in fixed-point. What does that mean?

Why Floating-point?

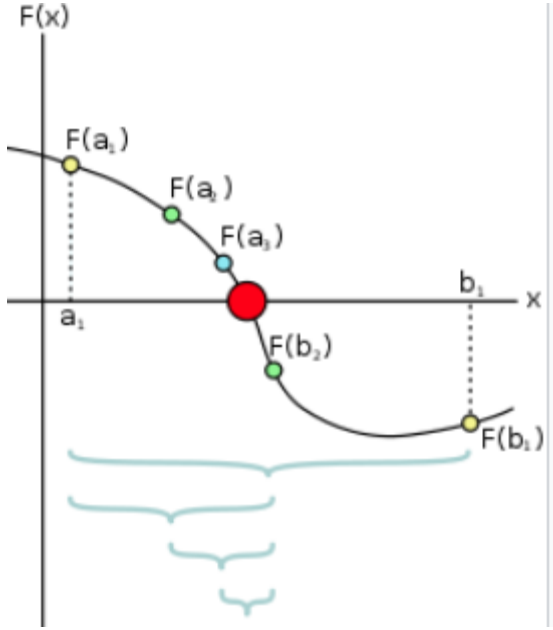
- You want to represent a wide range with a fixed number of bits?

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

- Your memory can store 16 bits.
- Your algorithm involves some calculations with many variables
- Variables include numbers as large as 30000.
- Your algorithm has an exit condition var1-var2>1e-6
- You need more fixed-point bits than you can handle

Bisection algorithm

```
N ← 1
While N ≤ NMAX # limit iterations to prevent infinite loop
  c ← (a + b)/2 # new midpoint
  If f(c) = 0 or (b – a)/2 < TOL then # solution found
    Output(c)
    Stop
  EndIf
  N ← N + 1 # increment step counter
  If sign(f(c)) = sign(f(a)) then a ← c else b ← c # new interval
EndWhile
Output("Method failed.") # max number of steps exceeded
```



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

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

Floating-point range/precision/underflow/overflow

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

- Worksheet Q4

Encoding symbols

- Why?
  - Computers cannot only work with numbers

- How do you encode any set of symbols?
  - Need to make a link between binary and a symbol

  - First define set
  - Work out minimum number of bits

Why ASCII

- 8-bits
  - Came up with 100 characters. Leave some room for redundancy.

Why ASCII

- 8-bits
  - Came up with 100 characters. Leave some room for redundancy.
    - Good thing: increase to approx. 200 (foreign characters)
- Problems?

Why ASCII

- 8-bits
  - Came up with 100 characters. Leave some room for redundancy.
    - Good thing: increase to approx. 200 (foreign characters)

- Problems?
  - What happens when add Chinese characters
  - Refine set of symbols, number of bits
  - Close to 36000
    - Different encodings
    - Unicode

ASCII vs Unicode

- Which is better?

Activity 1

- Benefits of RGB coding
- What is 1110 0010 1101 0100 0000 0101?
- Note: Order matter

Activity 2

Activity 2

- How many combinations can we encode?

Activity 2

- How many combinations can we encode?
  - 23 bits -> how many: 8million

Activity 2

- How many combinations can we encode?
  - 23 bits -> how many: 8million
- How many do we need:
  - 360*100*100 = 3.6 million

Activity 2

- How many combinations can we encode?
  - 23 bits -> how many: 8million
- How many do we need:
  - 360*100*100 = 3.6 million
- Could we just use 22 bits?

Activity 2

- How many combinations can we encode?
  - 23 bits -> how many: 8million
- How many do we need:
  - 360*100*100 = 3.6 million
- Could we just use 22 bits?
  - Hard look-up table
  - Fast encoding/decoding

Activity 2

- How many combinations can we encode?
- 23 bits -> how many: 8million
- How many do we need: 360*100*100 = 3.6 million
- Could we just use 22 bits?

Activity 2

- How many combinations can we encode?
- 23 bits -> how many: 8million
- How many do we need: 360*100*100 = 3.6 million
- Could we just use 22 bits?
- Why would we not?
  - Hard look-up table
- Simple rules vs efficient rules
  - Fast encoding/decoding

Encoding/decoding

- Advantage of RGB is that it can be easily used
- Digital designers make these decisions to maximise efficiency.

Memory

- Every computer has memory
- What operations does a memory perform

What does a memory conceptually look like

- Cells/number of cells

Memory notation

- What is a Byte?

General notation

- How many grams in a kilogram?

General notation

- How many grams in a kilogram?

- How many metres in a kilometre?

General notation

- How many grams in a kilogram?
- How many metres in a kilometre?
- How many bytes in a kilobyte?

Memory notation

- KB: 2^10 Bytes,
- MB: 2^20 Bytes,
- GB: 2^30 Bytes,

Memory visualisation

- How many cells in a 2GB memory, where each cell is 1 Byte.

Memory visualisation

- How many cells in a 2GB memory, where each cell is 1 Byte.
  - 2^30 *2 = 2^31

Memory visualisation

- How many cells in a 2GB memory, where each cell is 2 Bytes.

Memory visualisation

- How many cells in a 2GB memory, where each cell is 2 Bytes.
  - 2^30

Memory Read/Write

- If write cell it stays
- If read it, you get it back

Memory Read/Write

- If I write to a cell which already has a value, what happens to that value?

Memory Read/Write

- If I write to a cell which already has a value, what happens to that value
  - Write means overwrite

Memory Read/Write

- Say I write 4 bits into 8-bit cell. What happens?

Memory Read/Write

- Write 4 bits into 8-bit cell. What happens?
- Zero pad
- What if you don’t pad the 4-bits.
  - Overwrite 4, keep other
  - Remaining 4 unchanged
  - Doesn’t write at all

Address

- Every cell in memory must have an address
  - What type of number/encoding for address?
    - Base-2 natural numbers
  - How many bits for address?

Address Questions

- How many bytes does a memory with addresses 10 bits, cells 1 byte have?

Address Questions

- 4GB memory, 2 byte cells. How many cells?

Address Questions

- 4GB memory, 2 byte cells. How many cells?
  - 2^31

Address Questions

- 10 bit address, 16 bit cell. How many cells?

Activity 2

How does a memory work

- Address goes to all cells.
- Decoder chooses one cell

How do we store things in memory

- What if I want to store a big thing in memory

How do we store things in memory

- What if I want to store a big thing in memory
  - Use cells in consecutive locations (bytes)
    - Which order?

- Can a big endian architecture share memory with a little endian memory

Activity 3

How to store arrays

- What is an array?

How to store arrays

- Suppose it is an array of integers
- A[] = {10, 267, 39, 40}
- How many bits?
- Suppose you have a memory with 1 byte cells. What does it look like?

How to store arrays

- Memory has no notion of size
- It does not know how many bits are the integers
- It does not know they are storing integers

- How to know size of array if stored in memory?

How to find elements in memory

- Address base
- + bytes encoding size
- +3* size of element

How to store arrays

- Memory has no notion of size
- It does not know how many bits are the integers
- It does not know they are storing integers

- How to know size of array if stored in memory?

How to store arrays

- Memory has no notion of size
- It does not know how many bits are the integers
- It does not know they are storing integers

- How to know size of array if stored in memory?
  - Create a symbol to say end of array
    - Typically used for strings (null character)
    - Not great for integers (already have a 0)
  - Store the size
    - First put a number
  - Do nothing…
    - c

How to find elements in memory

- Address base
- + bytes encoding size
- +3* size of element

- E.g. look up A[5]
- What if look up A[7] in a array declared of size 6.
- First check it legal
  - Error message arrayindexoutofbounds
- Does this happen at run-time or compile time?

Activity 4

- Store in little endian
- Assume not storing size

Indirection

- Treasure hunt

Indirection

- Treasure hunt
  - Solve Clue 1, go to clue 2, solve that get to solution

Indirection

- Direct:
  - Read data in address 10.
  - Go to memory, read address 10. Done
- Indirect:
  - Read data in address 10.
  - Look up address described in address 10

Indirection

- Direct:
  - Read data in address 10.
  - Go to memory, read address 10. Done
- Indirect:
  - Read data in address 10.
  - Look up address described in address 10

- Can have multiple indirection

Why indirection?

- Multiple indirection.

```
1   public class Data {
2     int value;
3
4     public static void main(String[] args) {
5       Data obj1, obj2;
6
7       obj1 = new Data();
8       obj1.value = 1;
9
10      obj2 = obj1;
11
12      obj2.value = 2;
13
14      System.out.println(obj1.value);
15
16      return;
17    }
18  }
```