Introduction to Computer Science

Iain Styles
I.B.Styles@cs.bham.ac.uk

Recap

- Last time:
  - Representing numbers in the computer
    - Whole numbers in binary and hexadecimal notation
    - Positive real numbers in fixed-point binary
- This time:
  - Negative numbers
  - Arithmetic
  - Floating point

Arithmetic

- We have a representation, need manipulations
- Again, useful to understand decimal arithmetic:

```
        1 2 7
    +     8 4
(Carry 1 1    )
        2 1 1
```

- When a value is too large (>9) for a column, part of it is carried to the next column
- Same in binary, but carry at >1

Binary Addition

```
          1 0 0 1
   +      0 0 1 1
 (Carry 1 1     )
          1 1 0 0
```

- Try:
  - 1011 + 0010
  - 1101 + 0100

Overflow

- Numbers represented by fixed number of bits
- Large numbers “overflow” the maximum capacity of the number of allocated bits – computer will usually raise a hardware “error”
- Images typically use 8-bits to store whole number values representing image brightness
- 16, 32, 64 bit basic representations are now in common use (list in handout)

Subtraction

- Back to the decimals:

```
           2 1 1
       -     8 4
(Borrow 1 1    )
           1 2 7
```

- Binary

```
          1 1 0 0
     -    1 0 0 1
(Borrow      1    )
          0 0 1 1
```

Negative numbers

- We need to be able to represent negative numbers purely in binary
- In decimal representation, we have ten symbols plus the “minus” sign
- In binary, only two symbols
- So how do we denote the “sign” of a number?

Try something obvious

- Add a “sign” bit – assume on the left
- Then +5 = **0**101 and -5 = **1**101
- We know that +5 + -5 = 0

	0 1 0 1
+	1 1 0 1
(Carry	1 1)
	1 0 0 1 0

- We need to be a bit smarter about this...

An observation about overflow

- Let's investigate overflow a bit more closely

```
           0 1 1 1
   +       1 1 1 1
   (Carry  1 1 1   )
       (1) 0 1 1 0
```

- Interesting...try something else

```
           1 0 1 0
   +       1 1 1 1
   (Carry  1 1     )
       (1) 1 0 0 1
```

An observation about overflow

- And again

```
          0 1 1 1
     +    1 1 1 0
(Carry    1 1 1    )
      (1) 0 1 0 1
```

- And one more

```
          1 0 1 0
     +    1 1 0 1
(Carry    1 1      )
      (1) 0 1 1 1
```

Two's Complement

- Addition of a “large” number plus overflow looks like subtraction

  +111...111 $\rightarrow$ -1

  +111...110 $\rightarrow$ -2

  +111...101 $\rightarrow$ -3

- This leads us to the **two's complement (2C)** representation

Two's Complement Arithmetic

- In 2C the convention is:
  - All positive numbers start with 0
  - All negative numbers start with 1
  - Negation is achieved by:
    - Flipping all the bits
    - Adding 1 to the least significant (right-most)
- 011010 (positive) $\rightarrow$ 100110 (negative)

$$
\begin{array}{rcccccc}
 & 0 & 1 & 1 & 0 & 1 & 0 \\
+ & 1 & 0 & 0 & 1 & 1 & 0 \\
(\text{Carry} & 1 & 1 & 1 & 1 & 1 & \quad) \\
(1) & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Have a go

- 2C is used to implement subtraction
  - Easier to calculate 2C and add than to subtract
- 0001 - 0110?
- 1110 – 1001?

Range of 2C

- Given N bits, what is the largest value of a 2C number?
  - Recap: 111...111 = 2^N-1
  - Then 011...111 = 2^{N-1}-1
- What is the smallest (most negative) value?
  - 100..000 = (-)100...000 = -2^{N-1}
- Example: 8 bits
  - Maximum: 2^7-1 = 127
  - Minimum: -2^7 = -128

Fixed Point Arithmetic

- Everything is the same as for whole numbers
- Example: 01001.010 – 00010.100
- Take 2C and add:

```
          0 1 0 0 1 0 1 0
       +  1 1 1 0 1 1 0 0
(Carry  1       1          )
      (1) 0 0 1 1 0 1 1 0
```

- 00110.101 – 10110.010?

Floating Point

- Fixed point trades accuracy for range
- A possible alternative:

$$V = M \times 2^{E}$$

  - Why? Powers of 2 are easy to compute in binary!
  - M is the *mantissa:* 2C fixed point, one integer bit
  - E is the *exponent*: 2C integer

$$0101\ 0110 = 0.101 \times 2^{0110} = 0.625 \times 64 = 04$$

  - 1101 1110 = ?

Summary

- Binary arithmetic is like decimal arithmetic
  - But we are not practised so find it hard
- Negative numbers are tricky things
  - But we can use a few of our own tricks – 2C.
- Floating point is an alternative, but is very unnatural for us

Next time, we will begin to study how computer are organised, and how they execute programs