COMP2100 Workshop Week 6

Powers of two: How far can you go?

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, … Knowing the powers of two up to 1K (1024) is a minimum for a systems programmer, but you should aim higher…

2048, 4096, 8192, 16384, 32768, 65536 … that's 16 bits unsigned. Can you go further?

Practice your powers of two in the shower, on the train or bus, …

Bits, Bytes and Integers

The textbook contains practice exercises and solutions. Be sure that you understand how to solve each of the practice exercises, particularly the following. Ask about any you are unsure of!

- 2.1: Perform the following number conversions:
  A. 0x25B9D2 to binary

  B. binary 1010111001001001 to hexadecimal

  C. 0xA8B3D to binary

  D. binary 1100100010110110010110 to hexadecimal

- 2.2
  Without converting the numbers to decimal or binary, try to solve the following arithmetic problems, giving the answers in hexadecimal. Hint: Just modify the methods you use for performing decimal addition and subtraction to use base 16.

  A. 0x503c + 0x8 =

  B. 0x503c − 0x40 =

  C. 0x503c + 64 =

  D. 0x50ea − 0x503c =

- 2.3

  ```
  #include <stdio.h>

  typedef unsigned char *byte_pointer;

  void show_bytes(byte_pointer start, int len) {
        int i;
        for (i = 0; i < len; i++)
              printf(" %.2x", start[i]);
        printf("\n");
   }
  ```

  Consider the following three calls to `show_bytes`:

  ```
  int a = 0x12345678;
  byte_pointer ap = (byte_pointer) &a;
  show_bytes(ap, 1); /* A. */
  show_bytes(ap, 2); /* B. */
  show_bytes(ap, 3); /* C. */
  ```

  Indicate the values that will be printed by each call on a little-endian machine and on a big-endian machine:

A. Little endian:	Big endian:
B. Little endian:	Big endian:
C. Little endian:	Big endian:

Arithmetic and overflow

Unsigned addition

What is the sum of each of the following 8-bit unsigned addition operations (UAdd)? Show the sum as decimal, binary, octal and hex.

a. 100 + 1

b. 254 + 1

c. 255 + 1

d. 128 + 138

e. 255 + 255

Signed addition

1. What is the sum of each of the following 8-bit signed addition operations (TAdd)? Show the sum as decimal, binary, octal and hex.
   a. 100 + 1

   b. 10 + -11

   c. 126 + 1

   d. 127 + 1

   e. 127 + 127

   f. -128 + -1

Floating Point

The first priority in tutorials is **your questions**. Come with relevant questions to help you better understand the lectures and readings.

- Fill in the missing information in the following table:

Decimal	Binary	Decimal
$\frac{1}{8}$	0.001	0.125
$\frac{3}{4}$		
$\frac{25}{16}$		
	10.1011	
	1.001	
		5.875
		3.1875

- Consider a 5-bit floating-point representation based on the IEEE floating-point format, with one sign bit, two exponent bits (k = 2), and two fraction bits (n = 2). The exponent bias is 2^{2-1} − 1= 1.

  The table that follows enumerates the entire nonnegative range for this 5-bit floating-point representation. Fill in the blank table entries using the following directions:

e: The value represented by considering the exponent field to be an unsigned integer
E: The value of the exponent after biasing
2^E: The numeric weight of the exponent
f: The value of the fraction
M: The value of the significand
$2^E \times M$: The (unreduced) fractional value of the number
V: The reduced fractional value of the number
Decimal: The decimal representation of the number

Express the values of 2^E, *f*, *M*, $2^E \times M$, and *V* either as integers (when possible) or as fractions of the form $\frac{x}{y}$, where *y* is a power of 2. You need not fill in entries marked "—".

Bits	*e*	*E*	2^E	*f*	*M*	$2^E \times M$	*V*	Decimal
0 00 00								
0 00 01								
0 00 10								
0 00 11								
0 01 00								
0 01 01	1	0	1	$\frac{1}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	1.25
0 01 10								
0 01 11								
0 10 00								
0 10 01								
0 10 10								
0 10 11								
0 11 00	—	—	—	—	—	—		—
0 11 01	—	—	—	—	—	—		—
0 11 10	—	—	—	—	—	—		—
0 11 11	—	—	—	—	—	—		—

- The integer 3,510,593 has hexadecimal representation `0x00359141`, while the single-precision, floating-point number 3510593.0 has hexadecimal representation `0x4A564504`. Derive this floating-point representation and explain the correlation between the bits of the integer and floating-point representations.

- For a floating-point format with an *n*-bit fraction, give a formula for the smallest positive integer that cannot be represented exactly (because it would require an *n*+1-bit fraction to be exact). Assume the exponent field size *k* is large enough that the range of representable exponents does not provide a limitation for this problem.

  What is the numeric value of this integer for single-precision format (*n* = 23)?

- Play with `floatprint.c`