16.482 / 16.561: Computer Architecture and Design

Summer 2014

Homework #2

Due **Tuesday, 5/27/14**

Notes:

- While typed submissions are preferred, handwritten submissions are acceptable.
- Any electronic submission must be in a single file. Do not scan individual pages and attach each page; copy and paste the images into a single document. If multiple files are strictly necessary, combine all files into a .zip archive—please do not use .rar format.
- This assignment is worth a total of 100 points.

1. (25 points) For the examples below, show how binary multiplication would proceed using Booth's Algorithm. For each step, show what operations are performed in that step, and what the state of the product/multiplier register is at the end of each step. Assume each operand uses four bits.

a. 2 x 6
b. (-7) x 5
c. (-4) x (-3)

2. (25 points) Convert each of the following decimal values into single-precision IEEE floating-point format. Show all steps, including how you calculate the fraction and biased exponent stored in the number. *(Note: I encourage you to convert each result into hexadecimal, which will help ensure that your assignments are graded and returned relatively quickly!)*

a. 8.625
b. -23
c. 0.65625
d. -141.75
e. 2.014 (determine the closest approximation you can)

3. (25 points) Convert each of the following IEEE single-precision floating-point values into decimal values. Show all steps of your work.

a. 0x40d80000
b. 0xc0f80000
c. 0x3f380000
d. 0xbf550000
e. 0xabcd1234

4. (25 points) Compute the result of each floating-point arithmetic arithmetic operation below, in which each of the values is encoded in single-precision IEEE floating-point format. Recall that:

   - For floating-point addition, align the binary points, add the significands, then normalize the result.
   - For floating-point multiplication, add the exponents (taking care to only account for the bias once), multiply the significands, normalize the result, and then determine the sign.

All arithmetic should be done in binary, and results should be re-encoded in single-precision IEEE floating-point format.

a. `0x40980000 + 0x40100000`
b. `0x41700000 * 0x3f400000`
c. `0x3fd00000 + 0xc0a00000`
d. `0xc1680000 * 0xbe000000`