CMPT 295 Assignment 4 (2%)

Submit your solutions by Friday, February 15, 2019 10am.
Remember, when appropriate, to justify your answers.

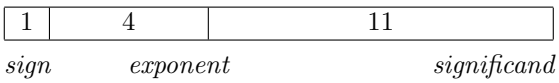
1. [5 marks] *Floating-Point Conversion*

   (a) [3 marks] Convert -255, -2.55 and 1/3 into the equivalent 32-bit floating-point. In all cases, show your steps and express your final answer both in binary and hex.

   (b) [2 marks] Convert the 32-bit floating-point `0x3e970a3d` and `0x3f7fffff` into their numerical equivalents. Express each as normalized base-2 scientific notation, as a base-10 rational fraction, and as a base-10 decimal value. *Round your answers to 10 decimals.* For example, if you were to convert `0x3f200000`, the expected answers would be: 1.01×2^{-1}, $\frac{5}{8}$, and 0.6250000000.

2. [6 marks] *Half-Precision Floating-Point*

   Suppose there was a floating-point code that fit in 16 bits, with 1 bit for the sign, 4 bits for the exponent and 11 bits for the significand.



   Answer the following questions. Be sure to justify your answers.

   (a) [1 mark] What is the range of normalized exponents?

   (b) [1 mark] Excluding $+\infty$, $-\infty$ and `NaN`, how many different values can be encoded?

   (c) [1 mark] What is the median value of the code?

   (d) [1 mark] What is the range of positive denormalized values?

   (e) [1 mark] What is the range of positive normalized values?

   (f) [1 mark] What is the median of the positive normalized values?

   (g) [1 BONUS mark] What is the median of the positive values?

3. [9 marks] *Convolution - Part 2*

This is a continuation of the coding question from Assignment 3. In that work, you implemented the function `conv()` to compute the reversed dot product of two arrays. In other words, given a pair of arrays `x[n]`, `h[n]`, it will return

$$\sum_{m=0}^{\mathtt{n}-1} \mathtt{x}[m] \cdot \mathtt{h}[\mathtt{n} - m - 1].$$

In this Assignment, you will write the assembly code that produces the full convolved signal. You will do this by calling the function `conv()` several times, as well as the function `min()`.

The Algorithm:

Open the care package. Within `conv_arr.s`, you will write the assembly code for the function `void conv_arr(char *x, int n, char *h, int m, char *result)`, which computes the convolution of the signal `x[n]` with the transfer function `h[m]`. The transformed signal will be placed in the array `result[n+m-1]`.

Your code should follow the pseudocode:

```
for i from 0 to n+m-2 do
   ladj <- min(i+1, m)
   radj <- m - min(m+n-(i+1), m)
   result[i] <- conv(x + (i+1-ladj), h + radj, ladj-radj)
```

Naturally, you will have to write a loop, but also you will have to call the supplied functions `min()` and `conv()`. Both obey the function call protocol: place the parameters in the correct registers, call the function, and collect the return value in `%rax`.

Hint: Remember to preserve any caller saved registers (scratch registers), lest they be overwritten by the functions!

The Specification:

- The function will obey the function call protocol, paying careful attention to scratch registers and callee saved registers.
- The function must have two calls to `min()` and one call to `conv()` per loop. They must be *meaningful* calls, in that the results of these calls will be used to computer your final result.
- The function must not overrun the buffer `result[m+n-1]` in either direction.

You will submit:

(a) [7 marks] an electronic copy of your `conv_arr.s` assembly source. This code will be tested for correctness with a selection of different inputs.

(b) [2 marks] a hard copy of your `conv_arr.s` assembly source. Your source should be well documented, so that any other programmer could read your code and understand it.