CMPT 295 Assignment 3 (2%)

Submit your solutions by Friday, February 1, 2019 10am.
Remember, when appropriate, to justify your answers.

1. [5 marks] *Carry Bits and Overflow*

   (a) [2 marks] Add the following 8-bit unsigned quantities, clearly indicating all carry bits. Indicate whether or not overflow occurs and, in the case(s) it does, explain why it occurs.

   i. $86_{10} + 115_{10}$
   ii. $251_{10} + 71_{10}$
   iii. $40_{10} + 206_{10}$

   (b) [2 marks] Add the following 8-bit two's complement quantities, again clearly indicating carry bits and explaining overflows when they occur.

   i. $-89_{10} + 35_{10}$
   ii. $69_{10} + 59_{10}$
   iii. $-97_{10} + -33_{10}$

   (c) [1 mark] Explain the functional difference between:

```
addq    (%rbx), %rax
```

   and

```
movl    (%rbx), %ecx
addq    %rcx, %rax
```

2. [5 marks] *Overflow Rules*

   (a) [2 marks] Let a and b be two numbers in the range $[0, 2^n - 1]$, i.e., they each have a valid representation in n-bit unsigned binary. To subtract b from a is equivalent to adding $-b$ to a, where $-b$ is represented by $2^n - b$. Overflow occurs only when $a < b$.
   Prove [mathematically] that $a < b$ if and only if the carry out of the MSB is 0.

   (b) [1 mark] Let x and y be two numbers encoded in n-bit 2's complement, such that $x < 0$ and $y \geq 0$. Clearly, the sum $x + y$ cannot generate an overflow because the magnitude of the result is moving closer to 0.
   Writing the binary equivalent of x with an MSB of 1 and y with an MSB of 0, there are two cases to consider:

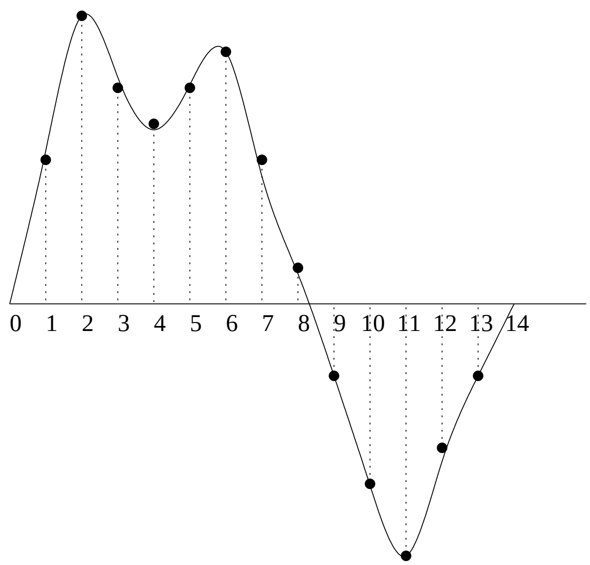
$$\begin{array}{rc|c|c|}\cline{3-4} x: & & 1^0 & \qquad\qquad \\ \cline{3-4} y: & + & 0 & \\ \cline{3-4} \multicolumn{4}{c}{} \\ \cline{2-4} \end{array} \qquad\qquad \begin{array}{rc|c|c|}\cline{3-4} x: & & 1^1 & \qquad\qquad \\ \cline{3-4} y: & + & 0 & \\ \cline{3-4} \multicolumn{4}{c}{} \\ \cline{2-4} \end{array}$$

   Either (Case 1) the carry in to the MSB equals 0, or (Case 2) the carry in to the MSB equals 1. Show that the overflow rule holds in either case, i.e., that the carry in equals the carry out.

   (c) [2 marks] Continuing with the same notation in part (b), there are 4 more cases to consider: where x and y are both negative or both positive, combined with whether or not the carry in to the MSB equals 0 or 1.
   Prove that the overflow rule holds in all 4 cases, i.e., that the carry in equals the carry out if and only if overflow did not occur.

3. [10 marks] *Convolution - Part 1*

The realm of *digital signal processing* (DSP) attempts to quantify the continuous world by using a sequence of discrete samples. Those samples are usually represented as an array of length N, say `char x[N]`.



As a function, the signal can have many turns — maxima and minima — and also inflection points, periodic behaviours, and perhaps other mathematically useful properties. To determine them, a *convolution* can be used, which is defined as follows:

Let `h[]` be a second array. The convolution of `x` with `h` is $(\mathtt{x} * \mathtt{h})[n] = \sum_{m=-\infty}^{\infty} \mathtt{x}[m] \cdot \mathtt{h}[n-m]$.

This is essentially a dot product where one array runs forwards and the other one backwards. Conventionally, array elements that are not defined have a value of 0.

E.g., Following the figure, let:

- $\mathtt{x[0..14]} = [0, 4, 8, 6, 5, 6, 7, 4, 1, -2, -5, -7, -4, -2, 0]$
- $\mathtt{h[0..2]} = [1, -2, 1]$

Then the convolution would be $(\mathtt{x} * \mathtt{h})\mathtt{[0..16]} = [0, 4, 0, -6, 1, 2, 0, -4, 0, 0, 0, 1, 5, -1, 0, -2, 0]$

$$\begin{aligned}\text{Specifically, } (\mathtt{x} * \mathtt{h})\mathtt{[3]} &= \sum_{m=-\infty}^{\infty} \mathtt{x}[m] \cdot \mathtt{h}[3-m] = \mathtt{x[1]} \cdot \mathtt{h[2]} + \mathtt{x[2]} \cdot \mathtt{h[1]} + \mathtt{x[3]} \cdot \mathtt{h[0]} \\ &= 4 \cdot (1) + 8 \cdot (-2) + 6 \cdot (1) = -6.\end{aligned}$$

Many arrays `h[]` are possible, each supplying some information about the characteristics of the signal. This particular `h[]` highlights changes in direction: large positive numbers indicate an upward swing (concave up); large negatives indicate a downward swing (concave down).

For this problem, you will implement a function `conv()` that computes the reversed dot product of two arrays. In other words, given a pair of arrays `x[n]`, `h[n]`, it will return

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

The Specification:

- The registers `%rdi` and `%rsi` will contain the base pointer of the two arrays, and `%edx` will contain the length of the arrays. Both arrays contain `char` values, i.e., one byte per value, each in the range $[-128, 127]$.
- The register `%al` will carry the return value, the summation as described above.
- Your code will need to use the `imul` instruction. Note, however, there is no `imulb` instruction.
- As per the function call protocol, you may only use the scratch registers `%rax`, `%rcx`, `%rdx`, `%rsi`, `%rdi`, `%r8`, `%r9`, `%r10` and `%r11`.

You will submit:

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

(b) [3 marks] a hard copy of your `conv.s` assembly source. Your source should be well documented, so that any other programmer could read your code and understand it. Your documentation shall include a synopsis of the algorithm you used to perform the computation.

(c) [3 BONUS marks] Because of the size of the return value, there is a chance of overflow, i.e., that the true result falls out of the range $[-128, 127]$. Though one solution might be to broaden the range, i.e., return a `short` or an `int`, another solution is to return whether or not the result generated an overflow by another means: by using a register.

For the BONUS marks, amend your `conv.s` so that when your subroutine returns, the register `%rdx` will contain the value 0 if and only if no overflow occurred. (The register `%al` should still hold the result of the computation.)