CMPT 295 Assignment 3 Solutions (2%)

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

   (a) [2 marks]

86_{10} =		0^1	1^1	0^1	1	0^1	1^1	1	0	
115_{10} =	+	0	1	1	1	0	0	1	1	
		1	1	0	0	1	0	0	1	no carry out of MSB ⇒ no overflow
251_{10} =		1^1	1^1	1^1	1^1	1^1	0^1	1^1	1	
71_{10} =	+	0	1	0	0	0	1	1	1	
	1	0	1	0	0	0	0	1	0	carry out of MSB ⇒ overflow
40_{10} =		0	0	1	0^1	1	0	0	0	
206_{10} =	+	1	1	0	0	1	1	1	0	
		1	1	1	1	0	1	1	0	no carry out of MSB ⇒ no overflow

   (b) [2 marks]

-89_{10} =		1	0^1	1	0	0^1	1^1	1^1	1	
35_{10} =	+	0	0	1	0	0	0	1	1	
		1	1	0	0	1	0	1	0	carry in = carry out ⇒ no overflow
69_{10} =		0^1	1^1	0^1	0^1	0^1	1^1	0^1	1	
59_{10} =	+	0	0	1	1	1	0	1	1	
		1	0	0	0	0	0	0	0	carry in ≠ carry out ⇒ overflow
-97_{10} =		1	0	0^1	1^1	1^1	1^1	1^1	1	
-33_{10} =	+	1	1	0	1	1	1	1	1	
	1	0	1	1	1	1	1	1	0	carry in ≠ carry out ⇒ overflow

   (c) [1 mark] The former adds to `rax` the 64-bit quantity referred to by the pointer `%rbx`; the latter adds to `rax` the 32-bit quantity (left padded with 0s).

2. [5 marks] *Overflow Rules*

   (a) [2 marks] The numerical result of the subtraction is $a - b = a + (2^n - b) = (a - b) + 2^n$. Thus when $a - b < 0$ (overflow), the numerical result is less than 2^n, i.e., the carry out of the MSB is 0. But when $a - b \geq 0$ (no overflow), the numerical result is greater than or equal to 2^n, i.e., the carry out of the MSB is 1.

   (b) [1 mark] In Case 1, $0+1+0 = 1$, thus the carry out of the MSB equals 0; in Case 2, $1+1+0 = 10_2$, thus the carry out of the MSB equals 1.

   (c) [2 marks] If $x, y < 0$, then the MSB equals 1 for both. The carry out is going to be 1 regardless of the carry in. If the carry in is 1 (Case 1), then the MSB of the result is 1, and no sign change means no overflow; if the carry in is 0 (Case 2), then the MSB of the result is 0, which is an overflow.

   A similar analysis works for $x, y \geq 0$, which would always have a carry out of 0: a carry in of 1 (Case 3) would change the sign (overflow); a carry in of 0 (Case 4) would not (no overflow).

   Thus the overflow rule holds in all four cases.

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

The algorithm moves two pointers: one forward through `x[]` and the other backward through `h[]`.

```
# Algorithm:
#
#   total <- 0
#   for i from n-1 downto 0 do
#     product <- x[n-i-1] * h[i]
#     total <- total + product
#

conv:
        xorl %eax, %eax            #   total <- 0
        leaq -1(%rsi, %rdx), %rsi  #   point %rsi to h[n-1]

loop:
        testl %edx, %edx           #   for i from n-1 downto 0 do
        je endloop

        movb (%rdi), %cl
        imulw (%rsi), %cx          #      product <- *h * *x
        addb %cl, %al              #      total <- total + product

        incq %rdi                  #      x++
        decq %rsi                  #      h--
        decl %edx

        jmp loop

endloop:
        ret
```