CMPT 295 Assignment 2 (2%)

Submit your solutions by Friday, January 25, 2019 10am.
Remember, when appropriate, to justify your answers.

1. [6 marks] *Binary Conversions*

   (a) [1 mark] Write the decimal numbers 106, 128 and 150 as 8-bit unsigned binary quantities and then as 2-digit hexadecimal quantities. Show all your work.

   (b) [2 marks] Write the decimal numbers -1, -106 and -128 as 8-bit two's complement binary quantities. Again, be sure to show your work.

   (c) [1 mark] Express the bit string 10101110_2 as a decimal quantity, first interpreted as an unsigned quantity, and second interpreted as a two's complement quantity.

   (d) [2 marks] Convert the bit strings 11001110_2 and 00110111_2 to hexadecimal, and then add them in hexadecimal. Repeat for 11111010_2 and 10101110_2.

2. [5 marks] `leal`

   Let the value of the register `%edi` be x, and let k be a positive integer constant. To multiply `%edi` by k can accomplished by `imul $k, %edi`, but there may be faster alternatives. Both add and shift are much faster operations, and the instruction `leal (%r, %r, s), %r` does both at once.

   The goal for both parts is to implement `%edi` $\leftarrow k \cdot x$, but by using one or two `leal` instructions. For each k, write the instruction or pair of instructions that yields `%edi` $\leftarrow k \cdot x$. You may use the scratch register `%eax` if you wish.

   (a) [2 marks] Using a single `leal` instruction, what values of k are possible?
   Note: The scaling factor s may only be 1, 2, 4, or 8.

   (b) [3 marks] Find a pair of `leal` instructions, to be executed one after the other, that has the effect of `%edi` $\leftarrow k \cdot x$, for
   - $k = 13$;
   - $k = 20$;
   - $k = 37$;

3. [9 marks] *The Root of the Problem*

   In this question, you will implement a subroutine, `sqrt`, that computes the integer square root of a number. You will write `sqrt` within the file `sqrt.s` (part of the care package).

   The Specification:

   - The register `%edi` will contain the argument `x`. It is an unsigned 32-bit quantity.
   - The subroutine `sqrt` will compute the integer square root of the parameter `x`. Mathematically, `sqrt(x)` $= \lfloor\sqrt{\texttt{x}}\rfloor$, where *the floor function* $\lfloor \texttt{y} \rfloor$ computes the largest integer that is less than or equal to `y`. For example, `sqrt(5)`$= 2$, `sqrt(16)`$= 4$ and `sqrt(24)`$= 4$.
   - The register `%eax` will carry the return value, a 32-bit unsigned quantity.
   - You may only use registers `%rax`, `%rcx`, `%rdx`, `%rsi`, `%rdi`, `%r8`, `%r9`, `%r10` and `%r11` as scratch registers. No other memory is allowed, including any of the other registers, or any external memory.

- Your code may not use any of the x86-64 division instructions.
- Your code may not use any of the x86-64 square root instructions, or any of the other floating point arithmetic.

The Algorithm:

Your program will build the result, one bit at a time, from the most significant bit to the least significant.

```
result <- 0
for k from 15 downto 0 do:
   change the kth bit of result to 1
   if result * result > x then:
      change the kth bit of result back to 0
return result
```

For example, say `x` was 2025. Then in the last 8 loops, the result becomes:

k	`result` before test	`result` after test
7	**1**000 0000	**0**000 0000
6	0**1**00 0000	0**0**00 0000
5	00**1**0 0000	00**1**0 0000
4	001**1** 0000	001**0** 0000
3	0010 **1**000	0010 **1**000
2	0010 1**1**00	0010 1**1**00
1	0010 11**1**0	0010 11**0**0
0	0010 110**1**	0010 110**1**

You will submit:

(a) [7 marks] an electronic copy of your `sqrt.s` assembly source. This code will be tested for correctness using the same mainline routine, but possibly with different inputs.

(b) [2 marks] a hard copy of your `sqrt.s` assembly source. Your source should be well documented with high-level comments, so that any other programmer could read and understand your code. This algorithm is well known. Include the common name of this algorithm among your high-level comments.

(c) [2 BONUS marks] an enhanced version of your work in part (a), except that `sqrtrd(x)` returns $\sqrt{\texttt{x}}$, rounded to the nearest integer. Place your source in the file `sqrtrd.s`.

Some Hints:

- You may need to use some of the bitwise operators presented on Monday, January 21.
- The debugger `gdb` has been enabled for `sqrt.s`. (Yes, it works for assembly source too.)
  - You can do `list`, `break`, `continue` and `print`, as before.
  - After a breakpoint, use `step` to run one instruction at a time.
  - To print the value of a register, name the register with a dollar sign prefix. E.g., to print the value of `rcx`, use `print $rcx`.
  - Because of the bitwise operations you will be performing, hex format may be more useful than decimal in some circumstances. E.g., `print /x $rcx`.
  - If you want the state of all registers in one shot, type `info registers`.