CS33 Fall 2014 Lab1

Due Oct 19, 2014 6PM

This assignment will help you to understand

1) how to convert from decimal to binary and back and
2) how a computer does addition and multiplication.

The C language does not accommodate a "bit" data type where you can actually input/store/change and output a single bit. For instance, in PL/I, a data type: bit(n) designates a bit string where n is the number of bits in the string. There are mechanisms for accessing any bits in the string (substr and array for example) and a constant type: '0'B and '1'B exists to set and compare bit values. In this assignment, we will work with integer/binary values so we will emulate the "bit" data type by using the C "int" data type and use only values 0 or 1 for values.

First: You must write two procedures:

```
void to_binary( int n, int w, int *x, int *o )
        and
void from_binary( int *x, int w, int *n )
```

to_binary shall convert the decimal integer "n" into an array of ints "x" of length "w" where "x" is the binary representation of "n" using zeroes and ones. Negative numbers shold be represented in two's complement. The error flag "o" shall denote that the integer "n" is larger than can fit into a binary representation of length "w".

from_binary shall convert a "binary" array of length "w" to an integer "n" whose value is the decimal value of the binary number.

Second: Write a procedure which takes two arrays of "binary" numbers produced by to_binary and adds them together using ***only*** Boolean functions: AND, OR, XOR and NOT and outputs a "binary" array:

```
void adder( int *x, int *y, int *z, int *o, int w )
```

"o" is the overflow flag, indicating that the result will not fit in an array of length "w", where "w" is the length of the binary arrays.

You must allow for either or both numbers to be negative according to the rules of 2's complement arithmetic.. Also, the program must set an error flag, but compute the added result if addition results in an overflow. Of course, you can use FOR loops and assignments.

Third: using your adder and conversion routines, write a procedure which takes two numbers

and multiplies them using only Boolean functions and assignments. Set an error flag when the multiplication overflows. (This is a bit more difficult than for addition.)

```
void mult( int *x, int *y, int *z, int *o, int w )
```

Overall Note: It does not matter, in to_binary, whether the most or least significant bit comes out in x[0] or x[w-1] just as long as you are consistent in all of your procedures.

Hint: for both the adder and multiplier, go back to your grammar school days and recall how you learned how to do it and emulate that. Also, multiplication tables are not required since multiplying by 0 is 0 and multiplying by 1 is 1 or 0. This may not be the most sophisticated adder and multiplier but it will show you how things work.

Use the lab1.c template. Do not change anything in the main() procedure!

Estimated solution time: 8 hours.