CS270 Programming Assignment 2
"Logic Simulation"

Program due September 30, 2011 (via Checkin by 2:59pm)
Revised 9/26/2011

Goals

In this assignment you will write a C program to simulate parts of the LC-3 ALU. The goals are:

- Learn how to simulate digital logic using C functions (and thus increase your understanding of digital logic).
- Start building components of the LC-3 simulator.
- Familiarize yourself with C `enum` data types and `typedef`s.
- Learn how to operate with pointers.

The Assignment

Make a subdirectory called PA2 for the programming assignment and copy the following files into the directory:
http://www.cs.colostate.edu/~cs270/Assignments/PA2/logic.c
http://www.cs.colostate.edu/~cs270/Assignments/PA2/logic.h
http://www.cs.colostate.edu/~cs270/Assignments/PA2/main.c
http://www.cs.colostate.edu/~cs270/Assignments/PA2/Makefile

The logic.h header file defines a `BIT` as an enumerated value; i.e., a variable of type `BIT` can take on values `ONE`, or `ZERO` (or alternately 1, 0). The header file also defines another variable type called `lc3_word_t` to represent an LC-3 word. The `lc3_word_t` type is an array of exactly 16 `BIT` variables, with the most significant BIT at index 0, and the least significant bit at index 15. You need to implement the following functions declared in logic.h:

```
void word_not(lc3_word_t *R, lc3_word_t *A);
```
- Takes two pointers to the LC-3 words A (i.e., A is of type `lc3_word_t*`) and R, and writes the negated `BIT`s of A into the LC-3 word pointed to by R.
- Note that you cannot use the C "~" operator to implement this function (because you are "simulating" the architecture).

```
void word_and(lc3_word_t *R, lc3_word_t *A, lc3_word_t *B);
```
- Takes three pointers to the LC-3 words A, B, and R, and writes the bitwise AND of A and B into the LC-3 word pointed to by R.
- Note that you cannot use the C "&" operator to implement this function.

```
void word_add(lc3_word_t *R, lc3_word_t *A, lc3_word_t *B);
```
- Takes three pointers to the LC-3 words A, B, and R, and writes the bitwise addition of these words into the LC-3 word pointed to by R.
- A, B, and R are in 2's complement form.
- You can disregard the carryout/overflow bit.

`void word_flad(lc3_word_t *R, lc3_word_t *A, lc3_word_t *B);`

- Takes three pointers to the LC-3 words A, B, and R, and writes the floating-point sum of these words into the LC-3 word pointed to by R.
- A, B, and R are in 16-bit "half-precision" floating point form.

`int four_bit_decoder(BIT b0, BIT b1, BIT b2, BIT b3);`

- Takes 4 `BIT`s, decodes them (just like a logic decoder) and returns the result as an `int`.
- `BIT b0` is the least significant bit, and `BIT b3` is the most significant bit

`int three_bit_decoder(BIT b0, BIT b1, BIT b2);`

- Takes 3 `BIT`s, decodes them and returns the result as an `int`.
- `BIT b0` is the least significant bit, and `BIT b2` is the most significant bit

We provided a Makefile and main.c file to compile your program. You do not need to modify main.c (but if you do modify main.c, we may overwrite your modifications).

To run the program, type the following command:

```
%> ./pa2
```

Submission Instructions

All programming assignments will be submitted directly to Checkin, same as the previous assignment. When you are done, your directory should have logic.h, logic.c, main.c and a Makefile. To package the files into a single file (PA2.tar.gz) for submission, type the following command inside PA2:

```
%> make pack
```

Submit the file PA2.tar.gz produced by the command shown above (the file will be created one directory above PA2).

Reminders:

1. Make sure that your tar file is named PA2.tar.gz.
2. Make sure that your tar file unpacks to a directory named PA2.
3. Test your tar file in another directory using `%> tar -zxvf PA2.tar.gz`
4. *Make sure your program compiles before you submit*.

Grading Criteria

Points will be awarded as follows:

- Implementing the functions described in logic.h: 90 points (15 for each function)
  - main.c has tests for each of the functions.
- Coding style and comments: 5 points
  - The grading factors we consider for coding style include having clear and concise comments, consistent indentation, and the minimal amount of code to solve the problem.
  - You need to ensure that every function (declared in logic.h) has a properly formatted description header.
- Following assignment directions: 5 points
  - You will lose these points if you submit the wrong tar file, or your program does not

compile with the Makefile you submitted.

Late Policy

Late assignments will be accepted up to 48 hours past the due date, but with 10% deduction for every 24 hours late. Assignments will not be accepted past this period. If you were unable to submit via Checkin for any reason, contact us via email and we may be able to help.