CS 277 Lab #3
Bit-Level Operations
Due 11:59pm PST Friday February 28th, 2014

Purpose

The purpose of this lab is to gain experience with bit manipulation at a fundamental level. Computer system engineers must design arithmetic processes that can perform operations using as few clock cycles as possible. In this lab, you will have to complete several different functions that perform bit-level operations, but you will do so using a very strict subset of C commands and operators. This will give you an idea of some of the challenges faced in the development of modern computer systems. As an aside, you will also be introduced to compiling your code using the *make* utility.

Assignment

The source file `bitlib.c` contains the definition of several C functions. Your job is to complete each function without using conditional statements and constructs (if, switch, etc.) and without using any loops (thats right… no while loops, no for loops, you get the idea…). Additionally, you may only use the following eight C operators:

1. ! logical negation
2. ~ bitwise NOT
3. & bitwise AND
4. | bitwise OR
5. ^ bitwise XOR
6. + addition
7. >> bitwise right shift
8. << bitwise left shift

And just because I am a little bit crazy some functions have additional restrictions, all functions have an upper limit on the # of operators you are allowed to use in order to make the function work properly, and you may not define any constants that are longer than 8 bits. Full descriptions appear in the comments before each function in the source file, but basic descriptions of each function and any additional limitations are as follows:

1. **int bitwise_nor(int x, int y):** implement ~(x | y). You may only use the & and ~ operators. Maximum # of operators is 8.
2. **int bitwise_xor(int x, int y):** implement a bitwise XOR (^) using only & and ~. Maximum # of operators is 14.
3. **int eval_not_equal(int x, int y):** implement x!=y. No additional restrictions. Maximum # of operators is 6.
4. **int get_byte(int x, int n):** Return byte n from word x. No additional restrictions. Maximum # of operators is 6.
5. **int copy_lsbit(int x):** Set all bits of result to the least significant bit of x. No additional restrictions. Maximum # of operators is 5.
6. **int bit_count(int x):** Count the # of bits in x that are set to 1. No additional restrictions. Maximum # of operators is 40.

To compile your code, you will use the provided *makefile* with the Unix *make* utility. To compile your solution, simply type `make` at the command prompt and press enter. If your code compiles without any errors or warnings, you should see something like this as output:

```
gcc -c *.c
gcc driver.o bitlib.o -o bitfun
```

The source is compiled into an executable named `bitfun`. To run the compiled executable, type `./bitfun` and press enter. Typing `make clean` and pressing enter will delete the executable and object files generated by the compilation process.

Submission

To submit, use make to delete the executable and object files as described above. Next create a .tar file named lab3-yourusername.tar containing the makefile and all source (.c) and header (.h) files and email it to mharmon@lclark.edu before the due date and time.

Hints and Tips

Start early. Some of the functions are fairly simple while others require a bit of thought. You want to leave yourself time to experiment, ask questions and seek help if needed.

When possible, Make good use of the identities of boolean algebra as covered in class.

I recommend doing your development on a lab computer for this assignment. Remember that the particulars of representation at the bit level can vary depending on the system. If you develop your solution on your own system, make sure you thoroughly test it on a lab system and make sure you get the same results. You've been warned, therefore statements like “but it worked on my computer” won't fly!

Evaluation

Submit on time and your lab will be evaluated as follows:

1. Code compiles (15 points) and runs (15 points) on a lab machine.
2. Each function evaluates correctly, follows restrictions and does not use more than the allowed # of operators. (60 points / 10 points per function).
3. Code style is clean, concise and uses commenting when appropriate (5 points).
4. You successfully used make clean before submitting your lab (5 points).