Assignment 1: Bit-level operations

Due: 11:59 PM Tuesday, October 8, 2024

Grader (TA): Nurjahan <nurja1@lsu.edu> (ask questions via emails)

Head TA: Misbah U Hoque <mhoqu12@lsu.edu> (ask questions during office hours)

Note: You must submit this assignment in **two** places: 1) the classes server (used for running your code) and 2) Moodle (used for automatically detecting potential cheating). There will be no credit if you submit in only one place.

In this assignment, you will become familiar with basic bit-level operations and get further practice with Unix-style programming. First, create the directory **`prog1`**, which will be used for submission. Change to this directory, and issue the following command to get the files you need for this assignment (Note: Do not ignore the period at the end of the command):

```
cp ~cs3501_lee/cs3501_F24/p1/* .
```

View and understand the provided `Makefile` and `tester.c` if you want to know more about compile and testing options.

You will write **two functions** prototyped below in the file `int2bitstr.c`:

```
char int2bitstr(int I, char *str);

int get_exp_value(float f);
```

Note: You are required to write a **comment** for **each meaningful line** to explain its purpose. A submission without comment will get a zero grade.

1) For the function `int2bitstr` in which `I` is the input and both `str` and `XOR_Result` are the outputs, you need to do two things.

a) You need to store the bit pattern of the 32-bit integer `I` in `str` as a 32-length string of 0s and 1s (remember that strings are null-terminated so you should put a null character like `str[32] = '\0'`). The first character in the string will be the most significant bit from `I`, and the last non-NULL character will be the least significant. You must use **bit-level** operations to determine whether a character should be a zero or one. The only allowed integer arithmetic is increment (++) or decrement (--) in the loop.

b) You need to return the exclusive OR (**XOR**) result between the **least significant** byte (LSB) and the **second least significant byte** (second LSB) of I. To do so, you can only use bit-wise operators such as right shift (>>), AND (&), and XOR (^).

Forbidden operations for `int2bitstr`:

- Switch statements, function calls, and macro invocations.

- Addition, subtraction, division, modulus, and multiplication.

2) For the function `get_exp_value` in which `f` is the input, you need to calculate and return the exponent value E of `f`. You may want to use the provided function `f2u` to get the bit pattern of `f`. The bias of `float` (single precision) is `127`, and you can return `-128` for special values such as infinity.

Forbidden operations for `get_exp_value`:

- Loops, switch statements, function calls (except `unsigned f2u(float f)`), and macro invocations.
- Division, modulus, and multiplication. You can use subtraction.
- Relative comparison operators (<, >, <=, and >=). You can use Equality (==) and inequality (!=) tests.

Here is an example run of my completed tester:

```
>make
gcc -Wall -g -m32 -c tester.c
gcc -Wall -g -m32 -c int2bitstr.c
gcc -Wall -g -m32 -o xtest tester.o int2bitstr.o

> ./xtest

Enter integer to convert to bits: 1
1 : 00000000000000000000000000000001
XOR result between LSB and Second LSB: 1
0x1 : 0000 0000 0000 0000 0000 0000 0000 0001
1.000000 : 00111111100000000000000000000000
exp : 0

Enter integer to convert to bits: -1
-1 : 11111111111111111111111111111111
XOR result between LSB and Second LSB: 0
0xffffffff : 1111 1111 1111 1111 1111 1111 1111 1111
-1.000000 : 10111111100000000000000000000000
exp : 0

Enter integer to convert to bits: 16
16 : 00000000000000000000000000010000
XOR result between LSB and Second LSB: 16
0x10 : 0000 0000 0000 0000 0000 0000 0001 0000
16.000000 : 01000001100000000000000000000000
exp : 4

Enter integer to convert to bits: 1024
```

```
1024 : 00000000000000000000010000000000
XOR result between LSB and Second LSB: 4
0x400 : 0000 0000 0000 0000 0000 0100 0000 0000
1024.000000 : 01000100100000000000000000000000
exp : 10

Enter integer to convert to bits: 0
0 : 00000000000000000000000000000000
XOR result between LSB and Second LSB: 0
0x0 : 0000 0000 0000 0000 0000 0000 0000 0000
0.000000 : 00000000000000000000000000000000
exp : -126
```

For each integer input, the tester will print five lines. The first line displays the entered integer value with its bit pattern. The second line prints the XOR result between LSB and the second LSB of I. The third line shows the hexadecimal value for the entered integer with its bit pattern, grouped by 4 bits. The fourth line prints the floating-point value (data type `float`), converted from the entered integer, with its bit pattern. The last line outputs the exponent value of the floating-point number. If you want to exit the tester, you need to enter zero. You should **not** modify the provided `tester.c` and `Makefile` files.

When you are ready to submit, you can do so with "**`make submit`**". You can submit multiple times, but don't do so after the due date! Note that you will be graded using the provided `Makefile`, not your copy, so do not count on any changes you make to any file except `int2bitstr.c`. **In addition, you must copy and paste** the content of your `int2bitstr.c` into Moodle. This Moodle submission will be used to automatically detect potential cheating (https://grok.lsu.edu/article.aspx?articleId=20589).

Important notes:

- You should **NOT** send any assignment-related emails to the instructor. The grader (TA) has full control over the assignment grading process. The instructor will NOT reply to assignment-related emails or answer assignment-related questions during office hours. You can still ask for high-level concepts.
- You should **NOT** send the TA (or the instructor) any emails enclosing your source code for review. In fairness to other students, the TA will only check your submitted file(s) after the assignment deadline. If you send any email enclosing your source code, there may be a penalty for your assignment grade. You can still ask for high-level concepts.
- If there is no comment in your code, a grade of "zero" will be recorded.
- Your program will be compiled and tested only on the *classes* server for grading. You need to ensure your code is running well on the server. In other words, even though your program runs well on another machine, that will not be considered during grading.

- It is your responsibility to remember and securely keep your password for the server. If you forget yours, you must check with the department computer manager to reset it. This process may take several days, and you cannot request an extension for this reason.
- Since this assignment was posted about two weeks before its deadline, last-minute requests for an extension will not be granted. Being busy can never be a valid reason to request an extension when you have about two weeks for the assignment. The instructor/TA will not reply to such requests.
- It is very likely that the TA will not answer last-minute questions sent on the due date. Note that you have about two weeks to work on this assignment.
- Lateness penalty: 20%, 40%, and 60% deduction submitted on Wednesday (10/9), Thursday (10/10), and Friday (10/11) respectively; a grade of zero if submitted after Friday

Only for Honors Option:

Write code to implement the following function and send your code **to the TA (Nurjahan) via email**:

```
/* Return 1 when x contains an odd number of 1s; 0 otherwise.
Assume w=32. */

int odd_ones(unsigned x);
```

Your code should contain a total of **at most 12** arithmetic, bitwise, and logical operations.

Forbidden operations:

- Conditionals (if or ?:), loops, switch statements, function calls, and macro invocations.
- Division, modulus, and multiplication.
- Relative comparison operators (<, >, <=, and >=).
- Casting, either explicit or implicit.