Assignment 1: Bit-level operations

Due: 11:59PM Thursday, September 29, 2022

Grader (TA): Ruxin Wang <rwang31@lsu.edu>

In this assignment you will become familiar with doing 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_F22/p1/* .
```

View and understand the provided `Makefile` and `tester.c` so that you can write your own in the future.

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

```
int 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.

1) For the function `int2bitstr` in which `I` is input and `str` is output, two things need to do: 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 either increment (++) or decrement (--) in the loop. b) you need to count the number of ones ("1") in the bit pattern of a given two's-complement integer and return the count. It is okay to use increment (++) for this.

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 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
#1s in the bit pattern: 1
0x1 : 0000 0000 0000 0000 0000 0000 0000 0001
1.000000 : 00111111100000000000000000000000
exp : 0

Enter integer to convert to bits: -1
-1 : 11111111111111111111111111111111
#1s in the bit pattern: 32
0xffffffff : 1111 1111 1111 1111 1111 1111 1111 1111
-1.000000 : 10111111100000000000000000000000
exp : 0

Enter integer to convert to bits: 16
16 : 00000000000000000000000000010000
#1s in the bit pattern: 1
0x10 : 0000 0000 0000 0000 0000 0000 0001 0000
16.000000 : 01000001100000000000000000000000
exp : 4

Enter integer to convert to bits: -1024
-1024 : 11111111111111111111110000000000
#1s in the bit pattern: 22
0xfffffc00 : 1111 1111 1111 1111 1111 1100 0000 0000
-1024.000000 : 11000100100000000000000000000000
exp : 10

Enter integer to convert to bits: 0
0 : 00000000000000000000000000000000
#1s in the bit pattern: 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 number of 1s in the bit pattern. 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`**". Note that 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 of it, so do not count on any changes you make to any file except `int2bitstr.c`.

Important notes:

- You should **NOT** send any assignment-related emails to the instructor. The TA has full control over the assignment grading process. The instructor will NOT reply to assignment-related emails and will NOT answer to assignment-related questions during his office hours. You can still ask high-level concepts.
- You should **NOT** send the TA (or the instructor) any emails enclosing your source code. 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 high-level concepts.
- If there is no comment in your code, a grade of "zero" will be recorded.

Only for Honors Option:

Write code to implement the following function and send your code **to the TA 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.