Characters, Strings, and Floats

Lab 2, due Fri Feb 8 (2400 hrs)

CS 350: Computer Organization & Assembler Language Programming

[2/2: Include month in due date]

A. Why?

- We need to represent textual characters in addition to numbers.
- We use floating-point numbers to represent non-whole numbers (numbers not evenly divisible by 1).

B. Outcomes

After this lab, you should be able to

- Translate floating-point numbers to/from binary, decimal and IEEE format.
- Understand some of the precision problems that come up with floating-point numbers.

C. Problems [75 points total]

1. [10 = 2*5 points] Do Question 2.3 in the textbook (page 43): (a) Assume that there are about 400 students in your class. If every student is to be assigned a unique bit pattern, what is the minimum number of bits required to do this? (b) How many more students can be admitted to the class without requiring additional bits for each student's unique bit pattern?
2. [12 = 2*6 points] Let hex BED0 0000 represent an IEEE 32-bit floating-point number[1]. (a) What binary scientific notation value does it represent? (b) What decimal value does it represent? (You can write the answer in fractional or decimal form, your choice.)
3. [7 points] What is the base 2 scientific notation representation of $5\,{}^{43}/_{64}$? (That's $5 + {}^{43}/_{64}$, not $5 \times ({}^{43}/_{64})$.)[2]

[1] In this problem and others, you can ignore any embedded blanks — they're just for readability.

[2] Have people stopped teaching mixed fractions?

4. [11 = 7+4 points] What is the IEEE 32-bit floating-point representation of $5\,{}^{43}/_{64}$, in (a) binary and (b) hex?

5. [15 = 3*5 points] Let X = 11.00000 00000 00000 00000 011_2; (a) Why is it impossible to represent X exactly in 32-bit IEEE floating-point? (b) and (c) What are the two binary numbers closest to X that we *can* represent?

6. [20 = 5*4 points] Let's say we're working with floating-point binary numbers with 5 significant bits. (E.g., we can represent 1.0000, 1.0001, 1.0010, and 1.0011, but not 1.00000 or 1.00001.) Let's assume we truncate bits if we can't represent them. E.g., we'd represent 1.00001 by 1.0000, and we say that truncation introduced an error. On the other hand, truncating 1.00000 to 1.0000 does not introduce an error; the result is "exact".

   (a) What is 1.1111 + .11111? Was there an (truncation) error?

   (b) What is .11111 + .11111? Was there an error?

   (c) What is 1.1111 + (.11111 + .11111)? Was there an error?

   (d) What is (1.1111 + .11111) + .11111)? Was there an error?

   (e) Is floating-point addition associative, in general?

D. Programming Problem [25 points]

The goal is to improve your ability to write functions that take and manipulate arrays and different radixes by completing a program that repeatedly asks for a decimal integer (≥ 1) and a base (≥ 2) and converts the integer into the given base (as an unsigned integer). Here's some sample output from a solution. (User input is underlined.)

```
Enter an integer and base (int < 1 or base < 2 to quit): 255 2
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
Enter an integer and base (int < 1 or base < 2 to quit): 255 8
0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 7 7
Enter an integer and base (int < 1 or base < 2 to quit): 255 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 15
Enter an integer and base (int < 1 or base < 2 to quit): 255 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0
```



```
Enter an integer and base (int < 1 or base < 2 to quit): 524288 2
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Enter an integer and base (int < 1 or base < 2 to quit): 1048575 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Enter an integer and base (int < 1 or base < 2 to quit): 6382179 256
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 97 98 99
Enter an integer and base (int < 1 or base < 2 to quit): 0 0
```

(Random note: Base 256 gives the ASCII representation of values; the three "digits" of `6382179` base `256` are `97`, `98`, `99`, which represent `'a'`, `'b'`, and `'c'`.)

To store the digits, the program uses an array of integers of length `ARRAYLEN`. As you convert, check to make sure that the value doesn't overflow the array. Since `ARRAYLEN` happens to be `20` and `1048575` is the largest value we can represent in 20 bits unsigned, an input of `1048576` causes an error because we'd want to place a `1` in the 21st bit:

```
Enter an integer and base (int < 1 or base < 2 to quit): 1048576 2
Array full but value 1 remains
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Enter an integer and base (int < 1 or base < 2 to quit):
```

To get the program to work, you must extend the skeleton `Lab02_skel.c` by completing two functions; `break_up` breaks up the value into a sequence and stores the sequence into an array; `print_digits` takes the array and prints it out. Note: In C, there's no runtime check for out-of-bounds array indexes, so be careful. If you get a runtime error `Segmentation Fault,` it likely means you have a bad array index.

Grading Scheme [Note: Your program gets tested on dijkstra.iit.edu]

- `break_up`: [14 = 2+4+2+2+2+2 points]: Call from main program; Function header (4 points); Loop over value; Calculate new digit; Calculate new value; Detect & report overflow.
- `print_digits`: [9 = 3+2+2+2 points]: Call from main program; Function header (3 points); Loop over array; Print each digit.
- Code clean and well-formatted, includes your name etc: 2 points.

