Other Bases and Data

CS 350: Computer Organization & Assembler Language Programming
Lab 2, due Fri Feb 7

A. Why?

- Octal and hexadecimal are convenient ways to representing long bitstrings.
- 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 between representations for integers in binary, octal, hex, and decimal (signed and unsigned) and to perform arithmetic using them.
- Translate floating-point numbers to/from binary, decimal and IEEE format.
- Show some of the precision problems of floating-point numbers.
- Declare, manipulate, and print signed and unsigned binary and hex integers of various lengths, in C.

C. Problems [80 points total]

1. [9 = 3*3 pts] 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. [9 = 3*3 pts] Convert 680_{10} to binary, octal, and hexadecimal.
3. [3 pts] What is the octal representation of $B25EA4D_{16}$?
4. [9 = 3*3 pts] As an 8-bit sign-magnitude number, FF_{16} (i.e., 11111111) represents -127_{10}. What decimal value does FF_{16} represent as an 8-bit number in (a) unsigned binary? (b) 2's complement? (c) 1's complement?

5. [9 = 3*3 pts] Show the results of each of the following steps: (a) Convert $4BD2_{16}$ to binary. (b) Then take the 2's complement negative. (c) Then convert back to hex. (Hint: Your answer to part (c) should equal the 16's complement of $4BD2_{16}$.)
6. [4 = 2*2 pts] (a) What hex string represents 110000100110011_2 ? (b) What sequence of two ASCII characters represents the same bitstring?
7. [8 = 2*4 pts] 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.)
8. [4 pts] What is the base 2 scientific notation representation of $5\,{}^{43}/_{64}$? (That's $5 + {}^{43}/_{64}$, not $5 \times ({}^{43}/_{64})$; have people stopped teaching mixed fractions?)
9. [8 = 2*4 pts] What is the IEEE 32-bit floating-point representation of $5\,{}^{43}/_{64}$, in (a) binary and (b) hex?
10. [9 = 3*3 pts] 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?

For problems 11 and 12, represent binary numbers using 5 significant bits. (One way to do this is to always represent numbers as $1.bbbb \times 2^n$ where the b's are bits and n is an exponent. So we can't represent 100001 because that would be 1.00001×2^5. It also means we can't carry out the addition 100000 + 1 without truncation (of a nonzero bit).

11. [4 = 2*2 pts] To calculate (.00001 + .00001) + 1.0000 requires two additions. (a) Does the first addition, .00001 + .00001, cause truncation (of nonzero bits) before or after the addition? If so, specify how and when (b) Does the second addition, [the value from part a] + 1.0000?
12. [4 = 2*2 pts] Repeat the previous question on .00001 + (.00001 + 1.0000). (Part a is the right-hand addition; part b is the left-hand addition.)

[1] In all these problems, you can ignore any embedded blanks: they're just for readability.

D. Programming Problem [20 pts]

The goal is to improve your ability to write functions that take and manipulate arrays and different radixes by completing a partially-written skeleton program **`Lab02_skel.c`**. Once complete, the program should repeatedly ask for a decimal integer (≥ 1) and a base (≥ 2) and convert the integer into the given base (as an unsigned integer). Here's some sample output from a solution. (User input is in **bold**.)

```
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 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 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 1 0 0 1 1 0
Enter an integer and base (int < 1 or base < 2 to quit): 65535 2
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): 43046720 3
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Enter an integer and base (int < 1 or base < 2 to quit): 65535 16
0 0 0 0 0 0 0 0 0 0 0 0 15 15 15 15
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 97 98 99
Enter an integer and base (int < 1 or base < 2 to quit): 0 0
```

(For this last input, base 256 gives the ASCII representation of a value; 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 `16` and $65535 = 1111111111111111_2$ (sixteen 1's) is the largest value we can represent in 16 bits unsigned, any larger input causes an error. For example, take $91934 = 10110011100011110$; we can represent the last 16 bits, but the leftmost bit, indicating 65536, doesn't fit:

```
Enter an integer and base (int < 1 or base < 2 to quit): 91934 2
91934 is too large for 16 digits in base 2; 65536 is leftover
0 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0
```

To write the program, you must extend the skeleton **`Lab02_skel.c`** by completing two functions; `convert_to_base` breaks up the value into a sequence and stores the sequence into an array; if the value is too big, it prints an error

message and indicates the leftover amount. The `print_digits` function takes the array and prints it out.

The skeleton may or may not compile; in any case, you need to fill in the missing parts, which are indicated by STUB; a stub is a comment or piece of code that indicates where we're missing something..

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: You can develop your program on other systems, but we'll test it by compiling it with `gcc` *yourfile*`.c -lm` on alpha, so make sure to test it there, too.
- A program that generates syntax errors or warnings earns ≤ 10/20 points.
- `convert_to_base`: [10 = 2+2+2+2+2 pts]: Call from main program; Loop over value; Calculate new digit; Calculate new value; Detect & report overflow.
- `print_digits`: [6 = 2+2+2 pts]: Call from main program; Loop over array; Print each digit.
- Code clean and well-formatted, includes your name etc: 4 points.
- Include your name & section in header comments (at the top of your) `*.c` file. Also include your name & section in your program's output. As with lab 1, include your name in the file you submit. –1 point if you forget any or all of these things.

