Other Bases and Data

CS 350: Computer Organization & Assembler Language Programming
Lab 3: Due Wed Feb 3

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.

C. Written Problems [30 of 50 points]

1. [6 = 3 * 2 points] Hex $AF_{16} = 1010\ 1111_2 = 175_{10}$ unsigned. What decimal value does AF_{16} represent in (a) sign-magnitude; (b) 1's complement; (c) 2's complement?
2. [6 = 3 * 2 points] Show the results of each of the following steps: (a) Convert $3B5D_{16}$ 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 our starting hex number.)
3. [4 = 2 * 2 points] (a) What hex string represents 110000100110011_2 ? (b) What sequence of two ASCII characters represents the same bitstring?
4. [4 = 2 * 2 points] Let hex BED0 0000 represent an IEEE 32-bit floating-point number*. (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

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

form, your choice. Also, feel free to abbreviate long strings of 0s or 1s: E.g. you can write $0^{(12)}$ to indicate 12 zero bits.)

5. [4 = 2 * 2 points] (a) Give the result of converting decimal $5\,^{43}/_{64}$ to normalized binary scientific notation[†]. (b) Convert the result from part (a) to 32-bit IEEE floating-point format and give the result.

6. [2 points] Let $X = 11.0^{(21)}11_2$. Note X has too many significant digits to be represented exactly as a 32-bit IEEE floating-point number. What are the two binary numbers closest to X that we *can* represent?

7. [4 = 2 * 2 points] (a) Calculate $1.0000 \times 2^5 + 1.0000 \times 2^5$ to 5 significant digits. Also, say if there was any truncation of 1 bits as you did this. (b) Repeat part (a) on $1.0000 \times 2^5 + 1.0000 \times 2^0$.

D. Programming Problem [20 of 50 points]

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 ***`Lab3_skel.c`***. There's a copy of the skeleton on ***`alpha`***: Once you log into ***`alpha`***, use

```
cp ~sasaki/CS350/Lab3_skel.c ~
```

to copy the file to your home directory.[‡] There's also a copy on the course website.

Once complete, the program should repeatedly ask for a decimal integer value (≥ 1) and a base (between 2 and 36) and convert the value into the given base (unsigned). Below is some sample output from a solution. (User input is formatted ***`in italics`***.)

```
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 255 2
11111111
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 255 8
377
Enter an integer and base:
```

[†] That's $5 + {}^{43}/_{64}$, not $5 \times {}^{43}/_{64}$

[‡] Plain tilde means "the user's home directory"; *~name* means "the home directory of *name*". So you're copying a file named ***`Lab3_skel.c`*** (from the ***`CS350`*** folder at the instructor's home directory) to your home directory.


```
(int >= 1 and 2 <= base <= 36 or we quit): 2147483647 2
11111111111111111111111111111111
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 2147483647 16
7FFFFFFF
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 366 2
101101110
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 46655 36
ZZZ
Enter an integer and base:
(int >= 1 and 2 <= base <= 36 or we quit): 0 0
```

Note that 2,147,483,647 = 2^{31}–1 is the largest `int` value.

To write the program, extend the skeleton `Lab3_skel.c` (You don't *have* to, but its code (and your solution code) could appear on a test. The skeleton may or may not compile; in any case, you need to fill each missing part indicated by "STUB".

Side discussion: A stub is a comment or piece of code that indicates where something needs to be filled in. Stub code replaces the actual operation by some limited calculation (like `x = 0;` instead of `x =` *actual formula;*) or output (like `STUB: Need to fill in calculation of x`).

Bounds Checking: In C, there's no runtime check for out-of-bounds array indexes, so be careful. If you get the runtime error `Segmentation Fault,` it likely means you have a bad array index.

Grading Scheme

- For all labs and the final project, programs that generates syntax errors on compilation earn 0 points.
- Include your name and section in header comments (at the top of your) `*.c` file. Also include your name and section in your program's output. Include your name in the file you submit. –1 point if you forget any or all of these things.
- [3 points] Loops correctly (while the value to convert is > 1 and 2 ≤ base ≤ 36).
- [6 points] Fills in digit array correctly to indicate converted value.

- Take the value to convert and divide by the base. The remainder is the next digit of the answer and the quotient is the new value to convert. Repeat until the value to convert is 0. Note: You fill in the digit array from right to left, at least conceptually.
- [6 points] Prints digit array out (note use “A” - “Z” for 10 – 36.
- [6 points] The code includes your name etc., is clean (understandable) and well-formatted.

