# *Other Bases and Data*

*CS 350: Computer Organization & Assembler Language Programming*

*Lab 2, due Wed Sep 17*[*]

[9/18: Footer]

## *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. 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

[*] No extension for attending lab Thu 9/11 or Fri 9/12

 

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†. (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 $1.00001 \times 2^5$. 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.)

† 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 (between 2 and 36) and convert the integer into the given base (as an unsigned integer). Here's some sample output from a solution. (User input is formatted **`like this`**.)

```
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:
(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 2147483647 = $2^{31}$–1 is the largest `int` value.

To write the program, you must extend the skeleton **`Lab02_skel.c`**

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.cs.iit.edu`, so make sure to test it there, too.
- A program that generates syntax errors or warnings earns ≤ 10/20 points.
- [5 pts] Loops correctly (while value to convert is > 1 and 2 ≤ base ≤ 36).
- [7 pts] Fills in digit array correctly to indicate converted value.
  - You need to take the value left to convert, divide by the base and look at the quotient and remainder; the remainder is the next digit of the answer, and the quotient becomes the value left to convert.
- [5 pts] Prints digit array out (note use "A" - "Z" for 10 – 36.
- [4 pts] Code clean and well-formatted, includes your name etc.
- 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.

 

### *Solution to Written Problems*

1. (a) Need 9 bits to get $512 \geq 400$ bitstrings. (c) Can handle $512 - 400 = 112$ more students
2. $680_{10} = 101\ 010\ 1000_2 = 1250_8 = 2A8_{16}$
3. $B25EA4D_{16} = 1011\ 0010\ 0101\ 1110\ 1010\ 0100\ 1101 = 1\ 011\ 001\ 001\ 011\ 110\ 101\ 001\ 001\ 101 = 1311365115_8$
   $A63CB59_{16} = 1010\ 0110\ 0011\ 1100\ 1011\ 0101\ 1001 = 1\ 010\ 011\ 000\ 111\ 100\ 101\ 101\ 011\ 001 = 1230745531_8$
4. $FF_{16} = 11111111_2$. (a) Unsigned, $11111111 = 255_{10}$. (b) In 2's comp, $11111111 = -1$. (c) in 1's comp, $11111111 = -0$.
5. (a) $4BD2_{16} = 0100\ 1011\ 1101\ 0010_2$; (b) $-0100\ 1011\ 1101\ 0010_2 = 1011\ 0100\ 0010\ 1110_2$; (c) $B42E_{16}$
6. Hex 6133 = decimal 24,883.
7. $BED0\ 0000_{16} = 1011\ 1110\ 1101\ 0^{(20)}$ [where by $0^{(20)}$ I mean 20 zeros] has a sign bit of $S = 1$ followed by an exponent field $E = 0111\ 1101 = 125_{10} = 127 - 2$ (so the fraction will be multiplied by $2^{-2}$), and a fraction field $F = .1010^{(20)}$, so the number is $(-1)^S \times (1 + F) \times 2^{-2} = -1.1010 \times 2^{-2} = -.011010_2 = -{}^{13}/_{32}$.
8. $5\,{}^{43}/_{64} = 101_2 + 101011/2^6 = 101 + .101011 = 101.101011_2$.
9. $5\,{}^{43}/_{64} = 101.101011_2 = 1.01101011 \times 2^2 = 1.01101011 \times 2^{129-127}$ is represented by $0\ 10000001\ 01101011\ 0^{(14)} = 0100\ 0000\ 1011\ 0101\ 1000\ 0000\ 0000\ 0000 = 40B5\ 8000_{16}$.
10. It takes up 25 bits, and the IEEE representation can only handle 24 bits (23 bits in the fraction plus the implicit leading "1."). The two closest numbers we can represent are the results of rounding the value up or down to 24 bits: $11.00000\ 00000\ 00000\ 00000\ 01_2$ (rounding down) and $11.00000\ 00000\ 00000\ 00000\ 10_2$ (rounding up).
11. (a) $1.0000 \times 2^{-5} + 1.0000 \times 2^{-5} = 10.0000 \times 2^{-5} = 1.0000 \times 2^{-4}$ causes no truncation. (b) $1.0000 \times 2^{-4} + 1.0000 \times 2^0 = 0.0001 \times 2^0 + 1.0000 \times 2^0 = 1.0001 \times 2^0 = 1.0001$ also causes no truncation.
12. (a) $.00001 + 1.0000 = 1.0000$ with truncation of .00001; (b) is the same as (a), so we get $.00001 + 1.0000 = 1.0000$ with truncation of .00001.x

 