Binary Integers, C Compiling

CS 350: Computer Organization & Assembler Language Programming

Lab 1, due Fri Jan 24 (2400 hrs)[1]

A. Why?

- A basic way to store information in a computer is to use binary integers.
- There are multiple ways to represent negative integers.
- You're going to be writing C programs; learning to compile them will help.

B. Outcomes

After this lab, you should be able to

- Represent negative binary integers in sign-magnitude, 1's complement, or 2's complement; to list the pros and cons of each of these three systems; to explain why hardware commonly uses 2's complement to represent negative integers.
- List the representations of the most positive and negative values in each system; know when overflow occurs and how to recognize it.
- Be able to edit, compile, and run a simple C program on the `alpha` machine.

C. Written Problems [90 points total]

For Problems 1 – 4, answer each question for the three representation schemes (a) Sign-magnitude, (b) 1's Complement, and (c) 2's Complement.

1. [12 = 4*3 pts] In *<representation scheme>*, what decimal value does 110111 represent, and what bitstring is the negative of 110111?
2. [9 = 3*3 pts] In *<representation scheme>*, what 6-bit string(s) represent zero?
3. [6 = 3*2 pts] In *<representation scheme>*, what is the most negative 6-bit string, and what is its decimal equivalent?

[1] Have you read the syllabus to find out how to get an automatic extension to Monday?

4. [3 pts] In *<representation scheme>*, does taking the negative of the most negative number cause overflow?

More questions

5. [3 pts] Using unsigned 6 bit addition, what bitstring results from 111011 + 001110?
6. [3 pts] Take your bitstring from Question 5; that bitstring is the 2's complement representation of what decimal value?
7. [3 pts] 111011 is the 2's complement representation of what decimal value?
8. [3 pts] 001110 is the 2's complement representation of what decimal value? (Hint: Your answers for Questions 7 and 8 should add up to your answer for Question 6.

For Problems 9 – 12, rewrite the following additions and subtractions in 6-bit 2's complement. For example, 3 – 5 = –2 is 000011 – 000101 = 000011 + (–000101) = 000011 + 111011 = 111110 = –(000010) = –2

9. [12 pts] 12 + 18 = 30
10. [12 pts] 13 – 30 = –17
11. [12 pts] –25 – 7 = –32
12. [12 pts] 24 + 10 = ??? (Be sure to show the decimal result; you should get overflow.)

D. Programming Problem [10 pts]

First, copy the program `Lab01_skel.c`, to the `alpha` machine (`alpha.cs.iit.edu`). Run the C compiler on it to find the syntax errors. Edit the program to fix the problems and compile and run your program. (Your program should get no compiler error messages or warnings, and it should behave as described in the program's comments.) Also fix any semantic errors. You'll want to read up on the `printf` function. (Hint: how do `%d`, `%x`, `%#x`, and `%08x` differ as formats?)

Rename your program to `Lab01_YOUR_NAME.c`, (using your name) and submit it. (Don't submit the executable or output files.)

E. What to submit

For the problems in Part C, either (1) Write your solutions as comments in your `*.c` file for Part D or (2) Write your solutions in a separate pdf file and zip the *.c file and your solution. Either way, don't bother including an object file or executable file or program output run. (See http://cs.iit.edu/~cs350 → syllabus.)

Submit your file using Blackboard: Find Lab 1 under Assignments and press the link for uploading your solution.

Solutions to Written Problems

1. (a) Sign-mag: 111001 = –11001 = -25_{10}; $+25_{10}$ = 011001

   (b) 1's comp: 111001 = –000110 = -6_{10}; $+6_{10}$ = 000110

   (c) 2's comp: 111001 = –000111 = -7_{10}; +7 = 000111

2. 000000 is zero in all 3 systems; 100000 is "negative" zero in s-mag; 111111 is "negative zero" in 1's comp.

3. Sign-mag: 111111 = -31_{10}. 1's comp: 100000 = -31_{10}; 2's comp: 100000 = -32_{10}.

4. In sign-mag and 1's comp, –(–31) = +31 = 011111_2 (no overflow). In 2's comp, –(–32) = –100000 = 100000 = –32 (overflow).

5. (a) Unsigned:

```
  101101
+ 000110
--------
  110011
```

6 2's comp: 101101 = –010011 = –19

7 2's comp: 000110 = 6

8 2's comp: 110011 = –001101 = –13 (and note: –19 + 6 = –13)

9. 12 + 18 = 001100 + 010010 = 011110 = 30

10. 13 – 30 = 001101 – 011110 = 001101 + 100010 = 101111 = –17

11. –25 – 7 = 100111 – 000111 = 100111 + 111001 = 100000 = –32

12. 24 + 10 = 011000 + 001010 = 100010 = –(011110) = –30

