CS 240 SI Worksheet Solutions
Valerie Zhao
Session #3
2/2/17

Integer Representation

1. Using <u>8-bits</u> (which is 1 **byte** *[fill in the blank]*), what's **-25$_{10}$** in:
   a. Unsigned integer representation?
      Not possible

   b. Signed integer representation?
      10011001

   c. Two's complement representation?
      11100111

      i. What's **25** in two's complement?
         00011001

2. *Without looking at your notes or any other materials*, fill in the following table for an 8-bit binary integer:

Integer Representation	Minimum value (in base 10)	Maximum value (in base 10)
Unsigned	0	255
Signed	-127	127
Two's Complement	-128	127

3. Why is signed integer representation flawed? (2 reasons)
   - Normal addition involving negative integers doesn't produce the right results
   - Two representations of 0 (+/- 0)

   a. How does two's complement remedy this?
      - Addition involving negative integers is (usually) correct, i.e. if no overflow
      - Only one representation of 0

4. Interpret the numbers given under “Integer in binary” according to the 3 different representations, then record the base-10 value it encodes:
   (for example, 0100 is 4 in all 3 encodings.)

Integer in binary	**Unsigned**	**Signed**	**Two’s Complement**
1010	10	-2	-6
0111	7	7	7
1111	15	7	-1
0000	0	0	0
1000	8	0	-8

5. Calculate **0010 - 0111**:

   0010 - 0111 = 1011

   Steps of the borrow algorithm: (like subtraction in base 10, but in binary)

   0 0 1 0 (“A”)
   - 0 1 1 1 (“B”)
   1 0 1 1

   1. In the rightmost column (least significant bit), 0 in “A” is smaller than 1 in “B”, so 0 in “A” borrow from the bit to the left of it to become 10. **10 - 1 = 1**.
   2. Moving leftward, in the 2nd-to-rightmost column, 1 in “A” became 0 because of the borrowing from step 1, which is smaller than 1 in “B”, so it borrows from the bit to the left of it to become 10. **10 - 1 = 1**.
   3. In the 3rd-to-rightmost column, 0 in “A” became 1 because of the borrowing from step 2 WHICH required this 0 to also borrow from the bit to the left of it (most significant bit). In other words, this 0 in “A” borrowed from the most significant bit to become 10 before step 2 borrowed from it and it became 1 (It’s like subtraction in base 10, ex. 123 - 49.). **1 - 1 = 0**.
   4. In the leftmost column, just like in step 3, 0 in “A” had to become 10 by borrowing from the bit to its left (imagine there was a 1 to its left--it’ll maybe make sense in 4a) to become 10, then after step 3’s borrowing it became 1. **1 - 0 = 1**.
      a. Essentially we’re treating the equation as **10010 - 00111**. Try doing the reverse, i.e. calculating **1011 + 0111** → *what do you get if you keep all bits of the result?*

   Steps of the two’s complement algorithm: (even if the terms are in signed representation -- *why?*)

   1. -0111 = +(-0111) → -x = ~x+1 → 1001
   2. 0010 + 1001 = **1011**

a. What's the answer (in base 10) if this expression was in <u>signed</u> integer representation?

$1011_2 = -3$

b. In <u>two's complement</u>?

$1011_2 = -5$

c. How did <u>overflow</u> apply to what you did in parts a and b?

If you calculate 0111 + 1011 (this is the reverse of the given subtraction 0010 - 0111 = 1011 with the answer known), the complete result has 5 bits instead of 4. In signed integer representation, this indicates overflow. In two's complement, overflow was not an issue because despite the extra bit, the carry-in was equal to the carry-out, and it is not the case that the two terms being added have the same sign bit while the sum has the opposite sign bit.