ECE 204 - BINARY NUMBERS AND CODES - INVESTIGATION 10
BINARY ADDITION AND SUBTRACTION - PART II

FALL 2003 **A.P. FELZER**

To do "well" on this investigation you must not only get the right answers but must also do neat, complete and concise writeups that make obvious what each problem is, how you're solving the problem and what your answer is. You also need to include drawings of all circuits as well as appropriate graphs and tables.

From the last Investigation we know how to build logic circuits to add positive binary numbers. And we know how to convert 2's complement numbers to base 10. The main objective of this Investigation is to show how to put positive and negative numbers in 2's complement form and how to add and subtract with them. The main result of this Investigation is that 2's complements give us a straightforward way to add and subtract positive and negative binary numbers.

1. We begin with a review problem. Convert each of the following 2's complement numbers to base 10
   a. 11011^2
   b. 011^2
   c. 1110^2
   d. 1111^2

2. Find the 8 bit 2's complement numbers that have the same values as
   a. 001010^2
   b. 101010^2

3. Make use of your results in Problem (2) to find the affect of
   a. Adding 0's on the left side of a positive 2's complement number
   b. Adding 1's on the left side of a negative 2's complement number

4. The objective of this and the next couple of problems is to show how to put numbers in 2's complement form. We begin with positive numbers because positive numbers in 2's complement form are basically the same as "regular" binary numbers except that the MSB is zero. Express each of the following positve numbers as 6-bit 2's complement numbers
   a. 11101_2
   b. 1101_2
   c. 5_{10}

5. As we saw in Problem (4) it's easy to express positive numbers in 2's complement form. But negative numbers are a little more work. Suppose, for example, that we want to express A = –3 as a 5-bit 2's complement number. We know that the MSB is a one and so

$$A = -3 = 1x_3x_2x_1x_0{}^2 = -2^4 + x_3(2^3) + x_2(2^2) + x_1(2^1) + x_0(2^0)$$

   The problem is to find the values of the x's. The trick is to realize that

$$A = -3 = -2^4 + (2^4 - 3)$$

   So the problem reduces to finding the 4-bit positive binary number

$$2^4 - 3 = 10000_2 - 0011_2 = (1111_2 + 0001_2) - 0011_2 = (1111_2 - 0011_2) + 0001_2$$

a. First verify that we can obtain

$$1111_2 - 0011_2$$

by simply complementing every bit of 0011_2 - by replacing every zero by a one and every one by a zero. We call this the **1's complement** of the binary number A.

b. Now complete the calculations to obtain A=–3 as a 2's complement number
c. Express B=–6 as a 2's complement number

6. Generalizing on the result of Problem (5) we can show that if A is a 2's complement number - either positive or negative - then we can obtain the 2's complement form of –A by

(1) First complementing every bit of A (taking the 1's complement of A)
(2) And then adding +1

Use this algorithm to put the following numbers in 6-bit 2's complement form

a. -011010^2
b. -111010^2
c. –5 (Hint - first find 5 in 2's complement form)
d. –13

7. When we say "*take the 2's complement of A*" we mean find –A in 2's complement form. Take the 2's complements of

a. $A = 011010^2$
b. $A = 111010^2$

8. An alternate way to find the 2's complement of a 2's complement number like 001010 is to

(1) Start from the LSB and copy every 0 until you reach the first 1
(2) Copy the 1
(3) Complement all the rest of the bits

Using this algorithm we see that the 2's complement of

001010 is 110110

Now use this algorithm to

a. Find the 2's complement of 01011^2
b. Find the 2's complement of 11010^2

9. Make use of the fact that it can be shown that the sum A+B of two 2's complement numbers can be obtained as follows

(1) Add A and B as if they were "regular" binary numbers
(2) Discard the carry from the sum of the two most significant bits of A and B

to obtain the following sums

a. Calculate 00101 + 01001. Then verify your result in base 10
b. Calculate 11101 + 01001. Then verify your result in base 10
c. Calculate 10101 + 00101. Then verify your result in base 10
d. Calculate 10101 + 11101. Then verify your result in base 10

10. The objective of this problem is to identify overflow when we add 2's complement numbers

a. How can we tell if A+B has overflowed when they're both positive 2's complement numbers. Hint - look at the sign of the MSB of the sum. Do an example to illustrate.

b. How can we tell if A+B has overflowed when they're both negative 2's complement numbers. Do an example to illustrate.
c. Can the sum of a positive and a negative number ever overflow. How can you tell

11. From the algorithm in Problem (9) for adding 2's complement numbers it follows that the difference A–B of 2's complement numbers A and B can be calculated as follows

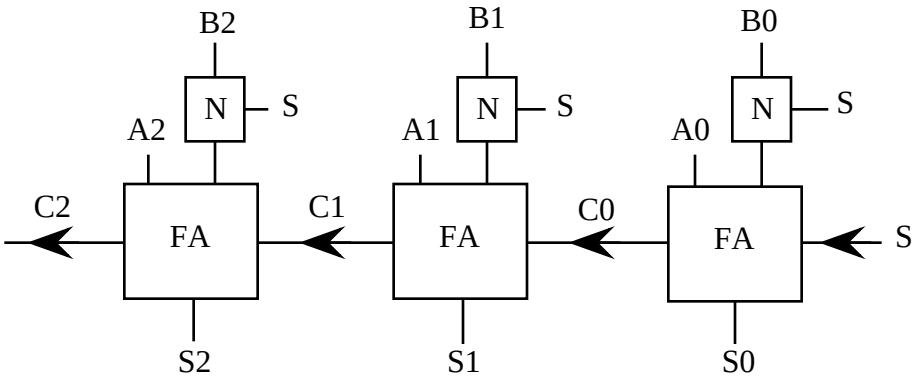
(1) Add A to the 2's complement of B as if they were "regular" binary numbers
(2) Discard the carry from the sum of the two MSB bits

Make use of this algorithm to calculate each of the following differences. Then verify your results in base 10

a. $00101_2 - 01001_2$
b. $11101_2 - 01001_2$
c. $10101_2 - 00101_2$
d. $10101_2 - 11101_2$

12. The objective of this problem is to identify overflow when we subtract 2's complement numbers
a. How can we tell if A-B has overflowed if A is a positive 2's complement number and B a negative 2's complement number. Do an example to illustrate.
b. How can we tell if A-B has overflowed if A is a negative 2's complement number and B a positive 2's complement number. Do an example to illustrate.

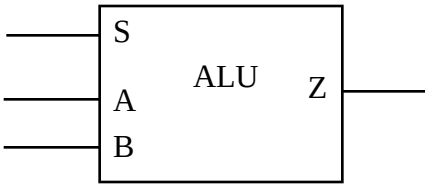
13. Design N in the following circuit for adding and subtracting 3 bit 2's complement numbers A and B as follows



Assume

(1) The inputs A and B are already in 2's complement form
(2) The output $S_2S_1S_0$ is to be in 2's complement form
(3) S=0 when A and B are to be added: A+B
(4) S=1 when A and B are to be subtracted: A-B

14. An **ALU** (**A**rithmetic **L**ogic **U**nit) is a circuit found in microprocessors and computers that performs both arithmetic and logic functions. **Memorize** this term. Then design a simple ALU as follows



with AOI gates that produces

(i) Z = A and B when S=0

(ii) Z = A plus B (without the carry) when S=1