ECE 204 - BINARY NUMBERS AND CODES - INVESTIGATION 9
BINARY ADDITION AND SUBTRACTION - PART I

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.

In the last Investigation we showed to convert decimal numbers to binary and vice versa. The objective of this and the next Investigation is to develop methods for adding and subtracting binary numbers using 2's complement arithmetic.

1. We begin with a review of number conversion. But first note that we refer to the base 2 number system as well as all other base number systems as **weighted codes** since the value of any given number is the sum of the digits times the *weights* of their places. **Memorize** this definition. Then
   a. Convert 79 to binary.
   b. Convert the binary number 101.10 to decimal.

2. Given a binary number like 10110 we call the bit on the far left the **Most Significant Bit (MSB)** and the bit on the far right the **Least Significant Bit (LSB)** as follows

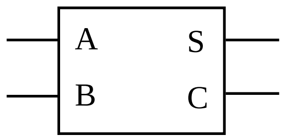
   10110

   MSB = 1 LSB = 0

   Memorize the definitions of MSB and LSB
   a. Why do we call the bit on the far left the MSB
   b. Why do we call the bit on the far right the LSB

3. The objective of this problem is to illustrate the adding of positive binary numbers.
   a. Convert $a = 3$ and $b = 5$ to binary
   b. Add your binary numbers in part (a). Be sure to show the carries.
   c. Verify that your binary sum in part (b) is equal to 8

4. The objective of this problem is to show how to design a logic circuit as follows



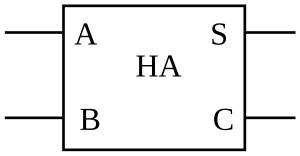
   that adds single bit binary numbers A and B to produce the sum S and carry C
   a. Complete the following truth table for S and C as functions of A and B

A	**B**	**S**	**C**
0	0		
0	1		
1	0		
1	1		

   b. Write logic equations for S and C

c. Design a circuit for S and C using AND and OR gates and INVERTERS.

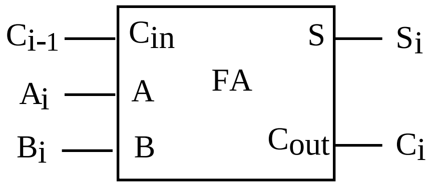
5. The binary adder from Problem (4) - which we refer to as a **half-adder (HA)** as follows



is great but when we want to add two binary numbers like the following

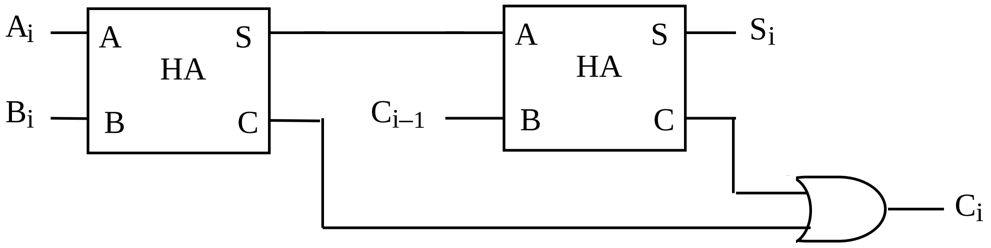
$$\begin{array}{r} B_2\ B_1\ B_0 \\ +\ \underline{A_2\ A_1\ A_0} \end{array}$$

we of course need to add in the carries. For example, when we add B_1+A_1 we must also add the carry C_0 from the previous sum. So we need **full-adders (FA)** as follows

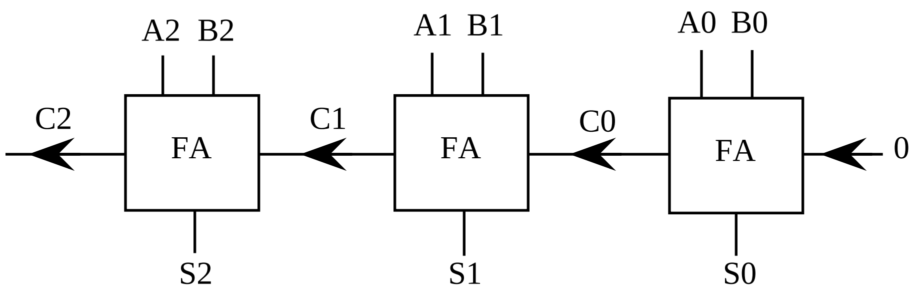


where $C_{in} = C_{i-1}$ = carry out from the previous full-adder. And $C_{out} = C_i$ = carry out from this full-adder.

a. Write the Truth Table for a full-adder
b. Verify that a full-adder can be made from two half-adders HA and an OR gate as follows



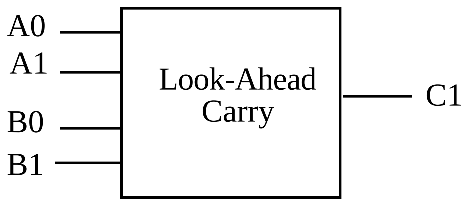
6. When we put *identical* full-adders FA together to build a binary adder as follows



$$\begin{array}{r} A_2\ \ A_1\ \ A_0 \\ +\ \ B_2\ \ B_1\ \ B_0 \\ \hline C_2\ \ S_2\ \ S_1\ \ S_0 \end{array}$$

we call it a **parallel** adder since all the bits are inputted at once. Now this is great but the full-adder in the middle can't generate S_1 and C_1 until it receives C_0. And similarly the full-adder on the far left can't generate S_2 and C_2 until it receives C_1. We say that the circuit can't finish adding the numbers until the carries **ripple through** the circuit. We refer to such circuits as **iterative** circuits. **Memorize** these terms. Now suppose each FA takes 10 µsec to produce its sum and carry

a. How long will it take the circuit to calculate the complete sum $C_2S_2S_1S_0$
b. One way to speed up our adder is with a Look-Ahead Carry - a circuit like the following



which is specifically designed to speed up the calculating of carries - in this case C_1. How fast will our adder above calculate the sum if a Look-Ahead Carry circuit is added that can produce C_1 in 15 µsec

7. When we build a circuit from identical modules like we built the binary adder in Problem (6) from Full Adders then we say the circuit is an **iterative** circuit. as being **iterative**. **Memorize** this term. What are the tradeoffs of iterative circuits.

8. As everyone knows the memories in calculators and computers for storing numbers are finite. And when the result of a calculation exceeds the limit we get an *overflow* error message.
   a. How would you make use of the carry output of the last FA in an iterative binary adder like in Problem (6) to tell when the sum of two positive binary integers has overflowed
   b. How big a positive integer (in base 10) could be stored in a computer with a 32-bit memory before it overflowed

9. Up to now we've just been working with positive binary numbers. An easy way to specify positive and negative binary numbers is using **Signed Magnitude**. We put a 0 on the far left if the number is positive and a 1 if it's negative. So

$$0100_{SM} = +4 \qquad \text{and} \qquad 1100_{SM} = -4$$

Memorize the definition of Signed Magnitude. Then
   a. Express +7 as a signed magnitude binary number with a total of 5 bits
   b. Express –7 as a signed magnitude binary number with a total of 5 bits

10. As we've seen in the last several problems it's pretty straightforward to build circuits that add positive binary numbers. And it's pretty easy to specify positive and negative binary numbers with the Signed Magnitude code. The challenge is coming up with a way to "easily" add and subtract positive and negative binary numbers. Now one way to do a calculation like $x = 3 - 5$ is to "simply"

   (1) Determine which number is larger
   (2) Subtract the smaller number from the larger (which in general requires borrowing)
   (3) Attach the correct sign to the result

Now this is all great but it's also pretty cumbersome. The objective of this problem is to introduce how to write binary numbers in what's called 2's complement form so that we'll be able to more efficiently add and subtract positive and negative binary numbers.

Two's complement numbers are basically the same as "regular" positive binary numbers except that the MSB has a negative weight. Whereas the "regular" binary number 1101_2 has the value

$$1101_2 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 13_{10}$$

the two's complement number 1101^2 (which we denote with a superscript 2) has the value

$$1101^2 = 1x(-2^3) + 1x2^2 + 0x2^1 + 1x2^0 = -3_{10}$$

Find the values of each of the following 2's complement numbers in base 10

a. 1010^2
b. 11010^2
c. 10101^2
d. 01101^2