1. More on MUXes
   a. Truth table for simple 2-bit 2 to 1 MUX

| Data 0	Data 1	Select	Output
10	11	0	10
01	00	1	00
11	01	0	11
10	01	1	01
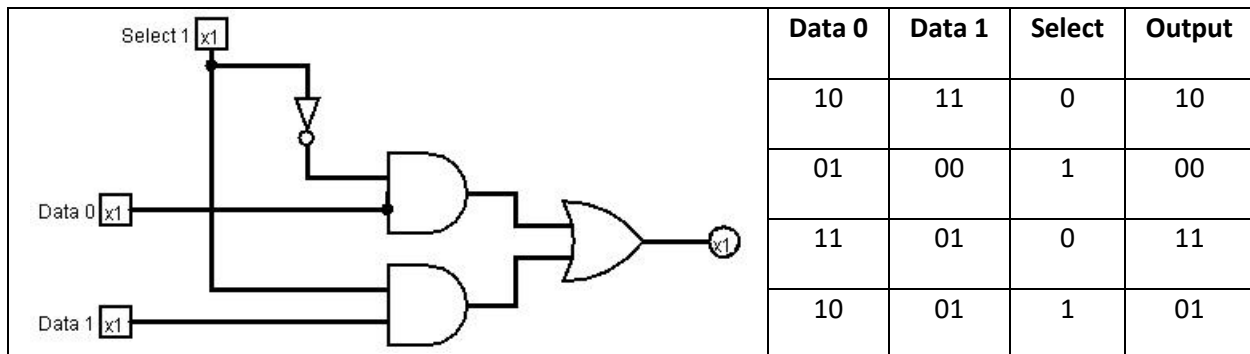
   b. Can use MUXes to implement functions
      i. Hook up constant 0s or 1s to each input
      ii. MUX takes in input bits and outputs corresponding constant for that input
   iii. Example below: implement an AND gate using a 2 to 1 MUX
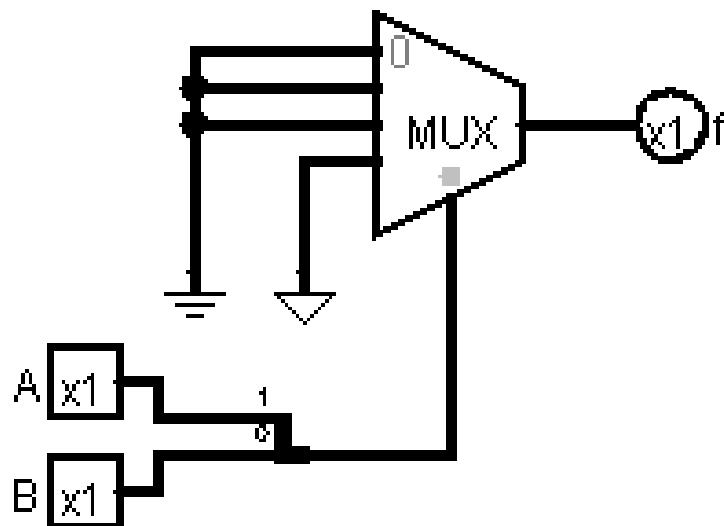
2. Decoders
   a. Decoders convert binary information from *n* input lines to a maximum of 2^n unique output lines
   b. One-hot encoded – only one output is asserted at a time
      i. Used in memory circuits
      ii. Give an encoded address, need to select a memory location
      iii. Use a n to 2^n decoder to convert the selected address on the bus to the correct row select line
   c. Additional input attached to all AND gates can be used in two ways
      i. If you use data, we turn the decoder into a *demultiplexer*
         1. Guides the input data into a specific output
   ii. If we treat the line as an enable, we can turn the decoder on and off
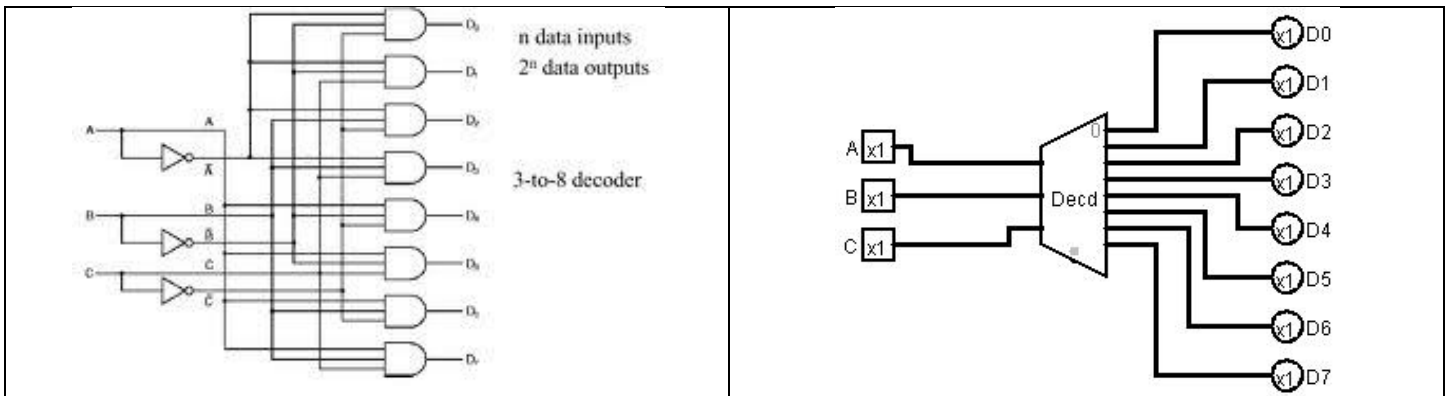
3. Adder
   a. Digital circuit that adds two numbers
   b. *Half adder* – adds two single binary digits, *A* and *B*
      i. Two outputs, the sum *S* and carry *C*
   c. *Full adder* – add binary numbers, account for carry in and carry out
      i. Longest (worst-case) path (A to C_{out}) goes through three gates
      ii. Involved in determining clock speed
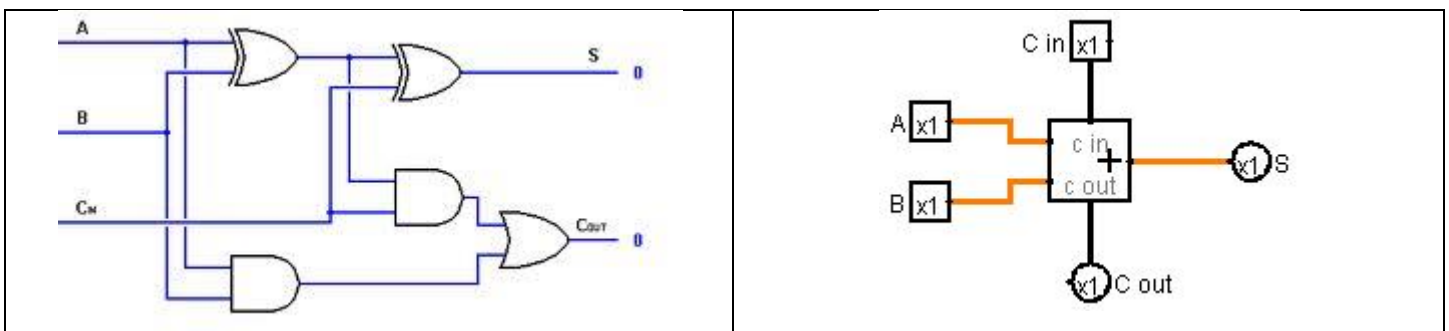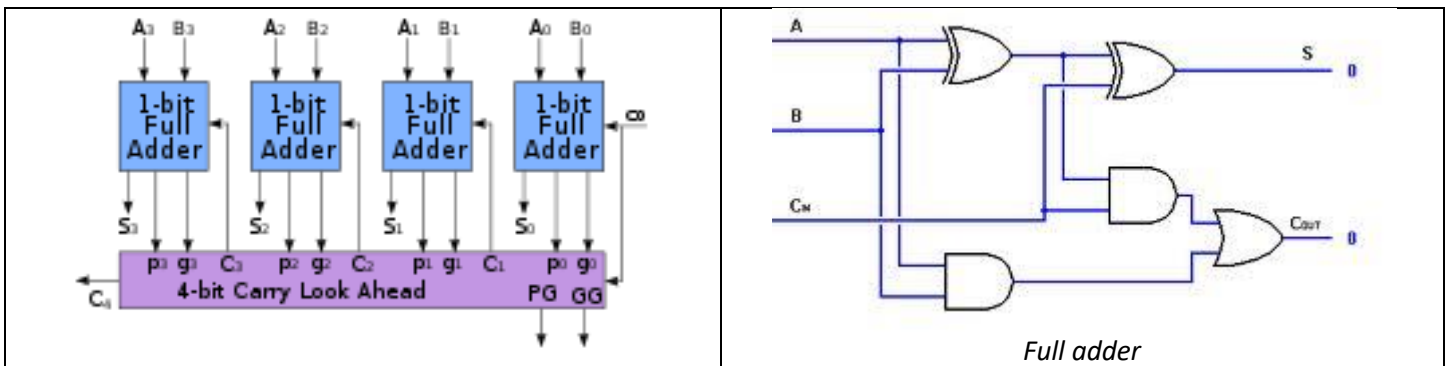
    d. Types of multiple-bit adders
        i. *Ripple-carry* – adder made up of a bunch of full adders hooked up in sequence
            1. Use the C_{out} of the previous bit being added as its C_{in}
            2. Causes a delay as you need to wait for the carry signal to propagate through
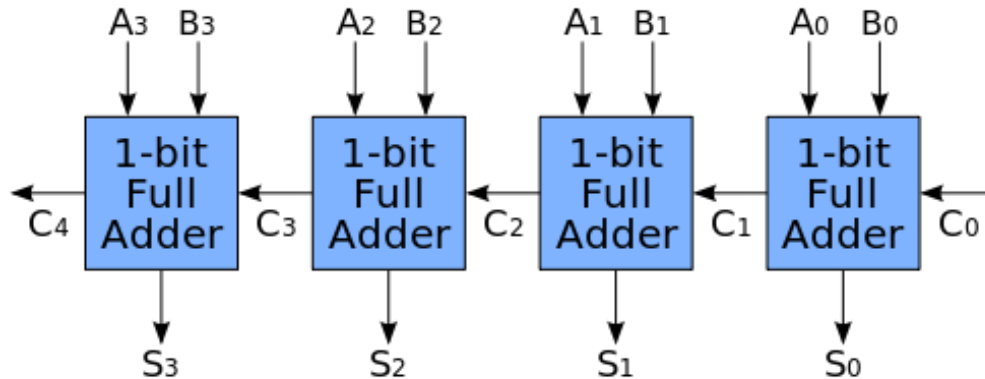
    4. Carry-lookahead adders (CLA)

Full adder

    a. Want to provide all carry bits for an adder at the same time
        i. Don't want to have to wait for them to ripple through
    b. Generate two signals for each bit position
        i. *Generate*, or *g*
            1. Addition will always carry, doesn't matter if there's an input carry or not
            2. $G(A, B) = A * B$
        ii. *Propagate*, or *p*
            1. Addition will carry whenever there is an input carry
            2. $P(A, B) = A \oplus B$
    c. $C_{i+1} = G_i + P_i C_i$, where C is the carry
   i. $G_i = A_i B_i$, $P_i = A_i \oplus B_i$
        ii. Can expand this out
            1. $C_1 = G_0 + P_0 * C_0$
            2. $C_2 = G_1 + P_1 * C_1 = G_1 + P_1 * (G_0 + P_0 * C_0) = G_1 + P_1 * G_0 + P_1 * P_0 * C_0$
            3. $C_3 = G_2 + P_2 * C_2 = G_2 + P_2 * (G_1 + P_1 * C_1) = G_2 + P_2 * (G_1 + P_1 * (G_0 + P_0 * C_0))$
              $= G_2 + (P_2 * G_1) + (P_2 * P_1 * G_0) + (P_2 * P_1 * P_0 * C_0)$
            4. $C_4 = G_3 + P_3 * C_3 = G_3 + P_3 * G_2 + (P_3 * P_2 * G_1) + (P_3 * P_2 * P_1 * G_0) + (P_3 * P_2 * P_1 * P_0 * C_0)$
        iii. C_0 is the only carry that must be known for all of these calculations

        iv. All of these expressions can be implemented with two levels of gates
1. Longest path could always be two gates
2. However, more inputs in an AND gate means it takes longer for the gate to resolve, as there are more transistors in series
3. For more than 7 inputs, binary tree of AND gates could be faster, even though path is longer

d. Process
    i. Each bit adder i calculates its P_i and G_i
       1. Takes one gate delay
    ii. CLA unit simultaneously calculates all carry for its adders
       1. Takes two gate delays
       2. Also calculates the carry for the next group of adders (C_4 in the picture above)
    iii. With the calculated carries, each bit-adder in the group simultaneously calculates its sums
       1. Only takes one gate delay, as carry and the XOR was already calculated for $P = A_i \oplus B_i$

e. Can expand this 4-bit adder to further levels, like a 64-bit unit
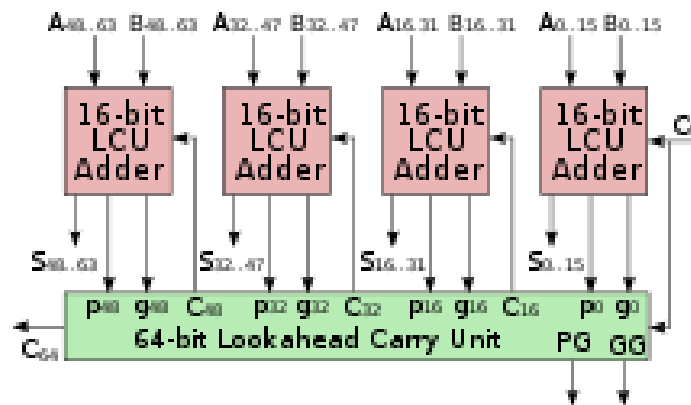
5. Subtractors
a. Create the 2's complement of the number to be subtracted
b. Then add to other number
c. Circuit below allows for that
    i. When wanting to subtract, D is set to 1
    ii. Multiplexers invert the entire number
    iii. Add one by setting carry in C_0 to 1