ECE 361 Homework 2
Fall 2004
Due: 10/26/04

1. Given the bit pattern: `1000 1111 1110 1111 1100 0000 0000 0000` what does it represent, assuming that it is

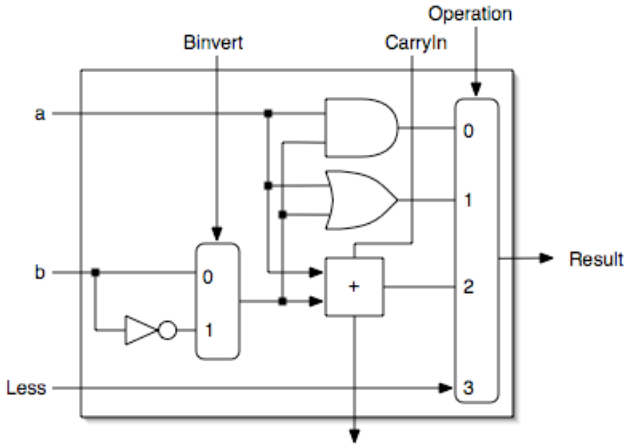
   a. a two's complement integer?
   b. an unsigned integer?
   c. a MIPS instruction?

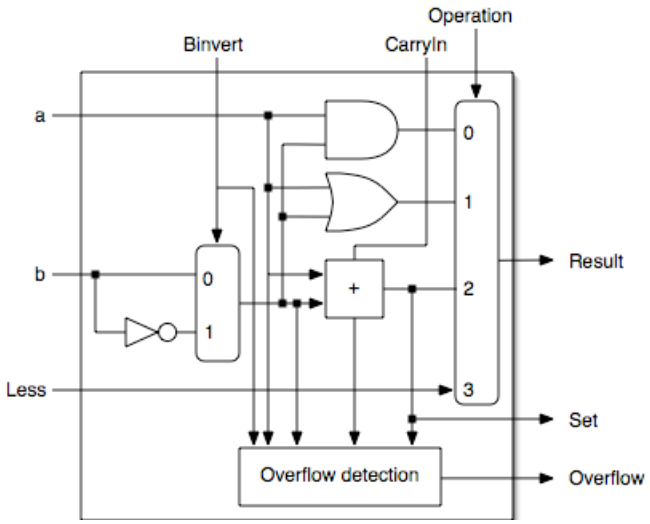
2. The ALU supported set on less than (`slt`) using just the sign bit of the adder. Let's try a set on less than operation using the values -7_{ten} and 6_{ten}. To make it simpler to follow the example, let's limit the binary representations to 4 bits: 1001_{two} and 0110_{two}.

   $1001_{two} - 0110_{two} = 1001_{two} + 1010_{two} = 0011_{two}$

   This result would suggest that -7 > 6, which is clearly wrong. Hence we must factor in overflow in the decision. Modify the 1-bit ALU in Figure 1 to handle `slt` correctly. Make your changes on the supplied pdf of Figure 1 to save time.

   (Fig 1: Top – A 1-bit ALU that performs AND, OR, and addition on a and b and not b. Bottom – a 1-bit ALU for the most significant bit.)





3. The full MIPS instruction set has two more logical operations not mentioned thus far: `xor` and `nor`. The operation `xor` stands for exclusive OR, and `nor` stands for not OR. The table that follows defines these operations on bit-by-bit basis.

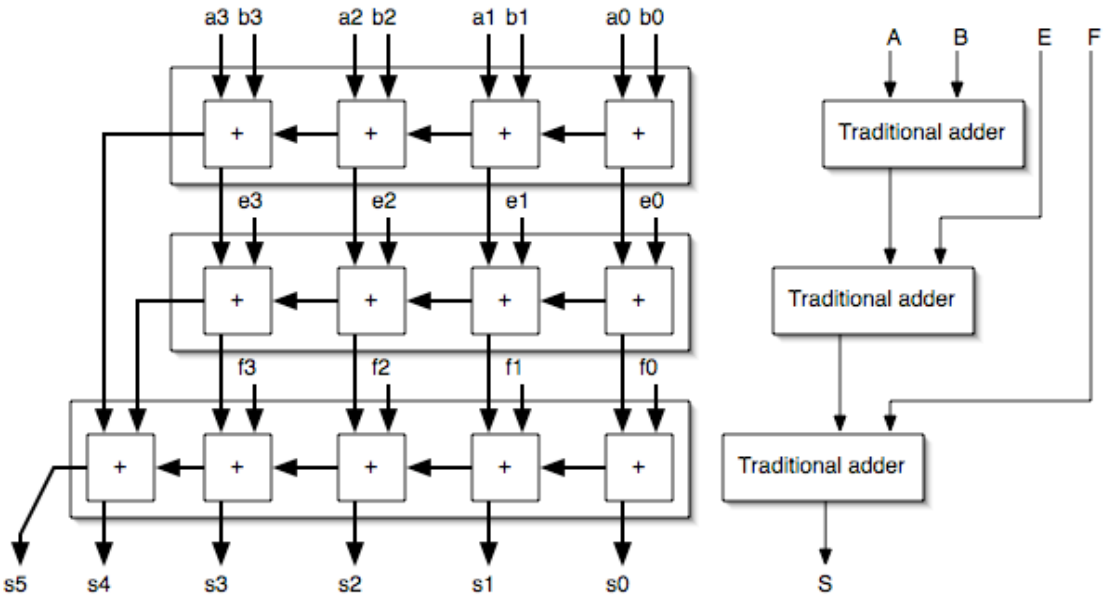
A	B	A xor B	A nor B
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	0

Show the minimal MIPS instruction sequence for a new instruction called `swap` that exchanges two registers. After the sequence completes, the Destination register has the original value of the Source register, and the Source register has the original value of the Destination register.
Convert this instruction:

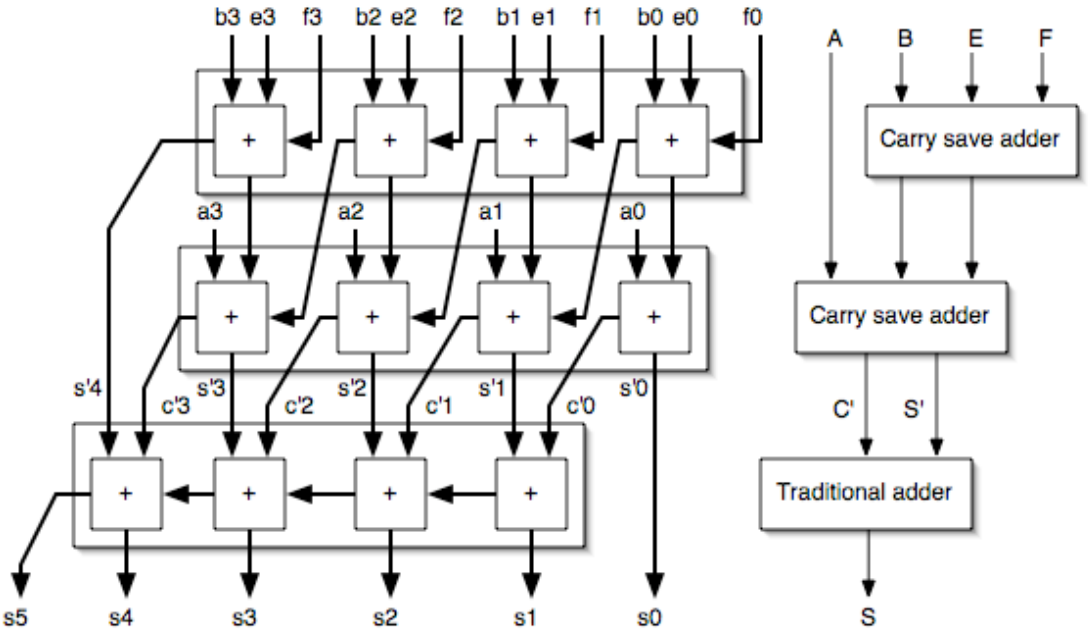
```
swap $s0, $s1
```

The hard part is that this sequence *must use only these two registers!* (Hint: It can be done in three instructions if you use the new logical instructions. What is the value of (`A xor B xor A`)?)

4. Assume that the time delay through each 1-bit adder is 2T. Calculate the time of adding four 4-bit numbers to the organization at the Figure 2a the organization in the in Figure 2b.
(Fig 2a)



(Fig 2b)



5. The original reason for Booth's algorithm was to reduce the number of operations by avoiding operations when there were strings of 0s and 1s. Revise the algorithm on page 260 (2nd edition) or on the supplemental CD under "In More Depth" 3:5-9 (3rd edition) to look at 3 bits at a time and compute the product 2 bits at a time. Fill in the following table to determine the 2-bit Booth encoding:

Current bits		Previous bit	Operation	Reason
ai+1	ai	ai-1		
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Assume that you have both the multiplicand and 2 x multiplicand already in registers. Explain the reason for the operation on each line, and show a 6-bit example that runs faster using this algorithm. (Hint: Try dividing to conquer; see what the operations would be in each of the eight cases in the table using 2-bit Booth algorithm, and then optimize the pair of operations.)