ECE 2300 Digital Logic and Computer Organization

Topic 5: Number Systems

http://www.csl.cornell.edu/courses/ece2300
School of Electrical and Computer Engineering
Cornell University

revision: 2025-10-05-14-17

List of Problems

Problem 1. Recall

In this problem, we'll talk about the features of different number systems.

Part 1.A Limits

How many different numbers *can* be represented with 4 bits? 8 bits? n bits?

What is the largest 16 bit *unsigned* binary number? How about 32 bits? How about n bits?

What is the *range* (in the format $[a, b]$) of the following n bit binary number systems?

System	Range
Unsigned	
Sign Magnitude	
Two's Complement	

Part 1.B Conversion

Convert the following decimal numbers into each number system. **Represent each number with 6 bits**. If it is not possible to represent the decimal number in that system, please denote so with an $\times$.

Number	Unsigned	Sign-magnitude	2's Complement
0			
1			
-1			
9			
26			
47			
-7			
-19			

Convert the following *binary* numbers *from* each number system. **Assume each number has 6 bits**.

Binary	Unsigned	Sign-magnitude	2's Complement
000000			
000101			
011010			
001011			
100000			
111111			
110101			
101001			

Problem 2. Subtraction!

In this problem, we consider *subtraction* and how to efficiently implement it for *unsigned* numbers.

Part 2.A Sign Magnitude

In the spirit of *regularity*, we may consider re-using our adder (from lecture) to do subtraction.

A perhaps naive approach to compute $a - b$ would be to rewrite it as $a + (-b)$. Of course, with *unsigned* numbers, representing $-b$ might be difficult. Instead, we turn to *Sign Magnitude* numbers.

This yields a simple "algorithm" of sorts. **Use the same, smallest possible bitwidth for a and b**. Prepend a bit to both a and b (that is, place it in the most significant position). Prepend a 0 to a and a 1 to b to negate it. Then, we use our fantastic adder implementations to add a and $-b$, yielding $a - b$.

But, does this work? **Let's find out by trying a few examples.**

Try to compute $7 - 4$ and $11 - 7$ with this method.

Does this method work? Why or why not?

Part 2.B "Half" of the problem

The dissapointing results from *Signed Magnitude* motivates specialized hardware for subtraction.

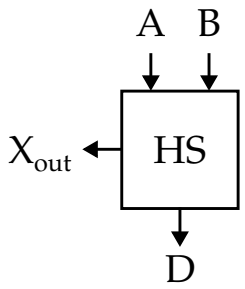
In fact, we can implement a *subtractor* using a structure analogous to an adder. However, we abandon the notion of a ***carry*** for that of a ***borrow***. Otherwise, the interface is almost *identical* (*modularity*).

A borrow means the current bit of a is not large enough, so we must "borrow" from the next bit.

A *Half Subtractor* subtracts two one-bit numbers a and b.

It produces the *difference* $d = a - b$ and a *borrow* bit.

Complete the truth table for this building block.



A	B	X_{out}	D
0	0		
0	1		
1	0		
1	1		

Hint: when you borrow, you are "using" the next bit (2× larger), so, in this case, what should D be?

Write the boolean expressions for D and X_{out}.

Complete a gate-level implementation of a Half Subtractor.

Part 2.C A Transistor Schematic

Let's consider the transistor schematic for X_{out} from our *Half Subtractor*.

Complete the transistor schematic for $\mathbf{X_{out}}$

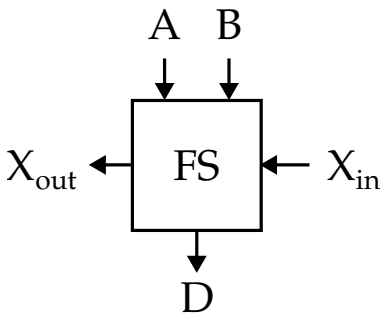
It's *likely* that your schematic uses 8 transistors. However, we can optimize this.

Your implementation *likely* uses a AND/OR and an inverter. But, it would be much more efficient if we used an NAND/NOR and an inverter. **Simplify the boolean expression to use a NAND/NOR and a *single inverter* and draw the new transistor schematic.**

Hint: try negating your current expression twice, *then use De Morgan's to simplify the expression*.

Part 2.D The "Full" Solution

Let's extend our half subtractor into a *Full Subtractor* so that we can chain them together. We extend our previous interface to also include a borrow-in. **Complete the truth table for this building block**.



A	B	X_{in}	X_{out}	D
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Write the boolean expressions for D and X_{out}.

Complete a gate-level implementation of a Full Subtractor.

Part 2.E Timing Analysis

Use the provided timing model. Each path should be specified by the input port, each gate along the path, and the output port. **Find the propagation delay for each input/output pair. Also identify the propagation delay and contamination delay for every path in the network. And identify the critical and short paths.** You may not need every row in the table.

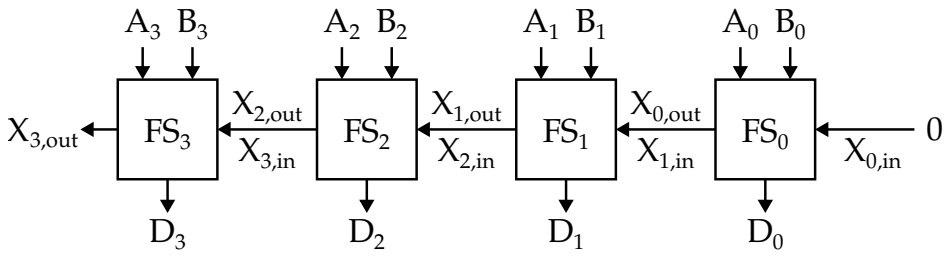
Path	t_{pd}
$A \rightarrow D$	
$B \rightarrow D$	
$X_{\text{in}} \rightarrow D$	
$A \rightarrow X_{\text{out}}$	
$B \rightarrow X_{\text{out}}$	
$X_{\text{in}} \rightarrow X_{\text{out}}$	

Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ
AND2	3τ	1τ
OR2	4τ	1τ
XOR2	7τ	6τ

Path	**Propagation Delay**	**Contamination Delay**	**Critical Path?**	**Short Path?**

Part 2.F Adding Subtractors?

We can put several *Full Subtractors* together to create a *Ripple Carry Subtractor*.



Using the timing analysis from before, what is the propagation delay through this network?

Generally, if we have an n-bit Ripple-Carry, what is the propagation delay?

That's not so good... **How can we make our subtractor faster?**

Part 2.G An alternative?

Recall from lecture, that subtraction can be implemented with an adder in one of the number systems. **Which number system was this?**

How do we use this number system to implement subtraction?

Part 2.H Comparison

Compare both subtractor implementations (dedicated vs 2's complement). Is one definitely better than the other? Which would you use in an add/sub unit? Which better fits our course principles of modularity, hierarchy, and regularity?

Hints

Compare qualitative and/or quantitative area and timing (see T02 practice problems for a simple area model). For a n bit unsigned number, we must include a sign bit for 2's complement addition. Do our subtractor need this?

Good luck for Prelim 1. You got this!