C335 Homework #5 Solution

Part I (8 points)

Assume that X consists of 3 bits, x2, x1, and x0. Write four logic functions that are true if and only if

(A) X contains only one 0
(B) X contains an even number of 0s
(C) X when interpreted as an unsigned binary number is less than 4
(D) X when interpreted as a signed (two's complement) number is negative

X2	X1	X0	F(A)	F(B)	F(C)	F(D)
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	0	1	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	0	1

F(A) = x2'x1x0 + x2x1'x0 + x2x1x0'
F(B) = x2'x1'x0 + x2'x1x0' + x2x1'x0' + x2x1x0
F(C) = x2'x1'x0' + x2'x1'x0 + x2'x1x0' + x2'x1x0 (= x2')
F(D) = x2x1'x0' + x2x1'x0 + x2x1x0' + x2x1x0 (= x2)

Part II (8 points)

With x = 01011011 (bin) and y = 00001101(bin) representing two's complement signed integers, perform, showing all work:

(A) x + y
(B) x – y
(C) x * y
(D) x/y

(The steps are skipped here)

(A) 0110,1000
(B) 0100,1110
(C) 100,1001,1111
(D) Quotient: 111, remainder: 0

Part III (6 points)

Do the unsigned multiply for 0011 * 0111, using the *Multiply Algorithm Version 3* (check the lecture notes). Show the contents of registers for multiplicand (4 bits) and product (8 bits) step by step.

Multiplicand		Product
0011		00000111
	+	0011
		00110111
	→	00011011
	+	0011
		01001011
	→	00100101
	+	0011
		01010101
	→	00101010
	→	00010101

Part IV (6 points)

Do the unsigned multiply for 0011 * 0111, using the *Booth's Algorithm* (check the lecture notes). Show the contents of registers for multiplicand (4 bits) and product (8 bits) step by step.

Multiplicand		Product	
0011		00000111	0
	-	0011	
		11010111	
	→	11101011	1
	→	11110101	1
	→	11111010	1
	+	0011	
		00101010	1
	→	00010101	0

Part V (6 points)

In Lecture Notes c335-ALU-1.pdf, we discussed how we could modify the 4-bit ALU to support detecting overflow but did not give the proof. Now you are expected to prove you can detect overflow by simply check if Carry into MSB ! = Carry out of MSB.
(**Hint:** consider the two cases that will cause an overflow)

We prove it by using truth table:

Sign bit of operand A	Sign bit of operand B	Carry into MSB	Sign bit of result	Carry out of MSB	Overflow?	Comments
0	0	0	0	0	No	
0	0	1	1	0	Yes	Positive (A) + positive (B), but get negative (sum)
0	1	0	1	0	No	
0	1	1	0	1	No	
1	0	0	1	0	No	
1	0	1	0	1	No	
1	1	0	0	1	Yes	Negative (A) + negative (B), but get positive (sum)
1	1	1	1	1	No	

The above table gives us a shortcut solution to judge if an overflow happens or not: just compare Carry into MSB and carry out of MSB.

Part V (6 points)

Do necessary modifications on the following 1 bit ALU, then use it to construct a 4-bit ALU that supports the SLT instruction.

Hint: read text appendix **B** section **B.5** to find clue.

Problem description: to implement instruction slt

Pseudo Instruction: slt C, A, B

What it does: if (A < B) then C = 0001

else C = 0000

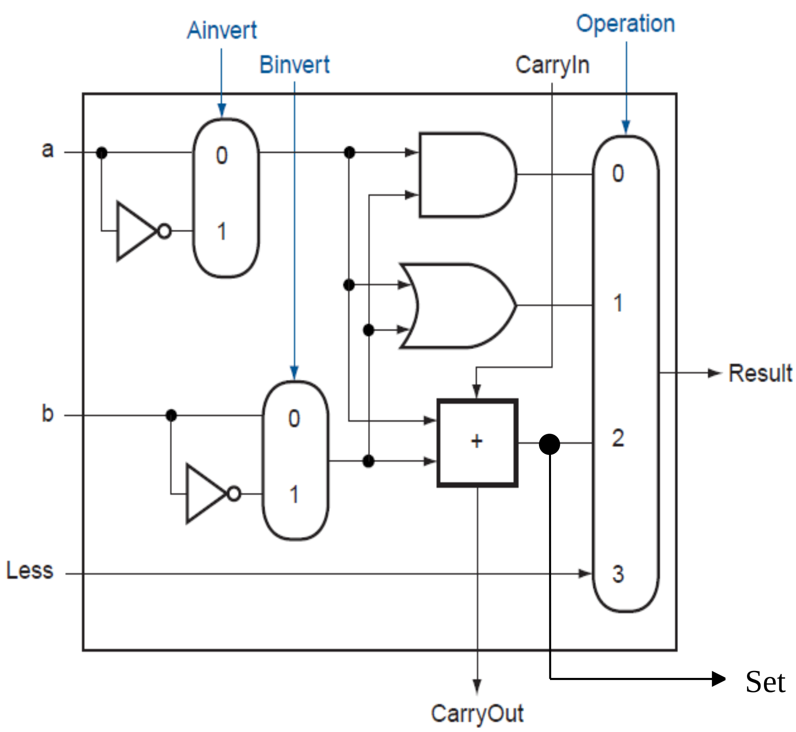
Basic Idea:

Step (1) do A – B

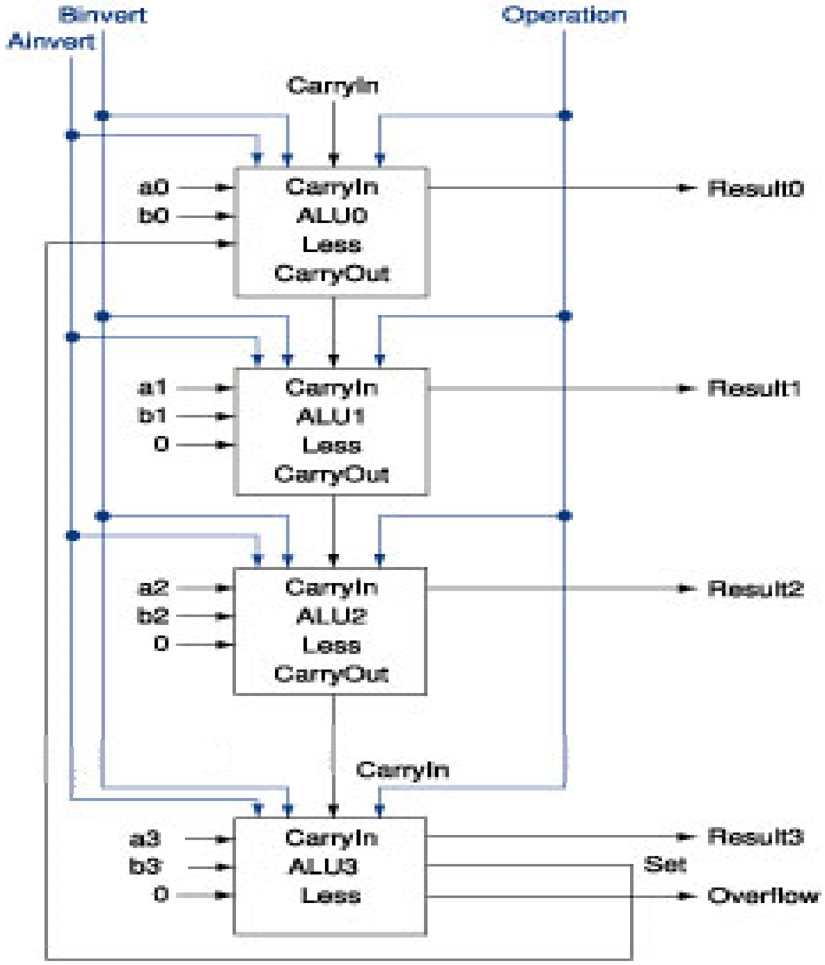
Step (2) check the result of step 1, if result is negative (sign bit is 1), then output 0001, else output 0000. (use the sign bit of the result as the feedback input for the last bit)

Implement:

Modifications on 1-bit ALU



Put 4 1-bit ALUs together:



Binvert
Ainvert
Operation
CarryIn
a0
b0
CarryIn
ALU0
Less
CarryOut
Result0
a1
b1
0
CarryIn
ALU1
Less
CarryOut
Result1
a2
b2
0
CarryIn
ALU2
Less
CarryOut
Result2
CarryIn
a3
b3
0
CarryIn
ALU3
Less
Result3
Set
Overflow