BEE 271 Midterm Answer Key, Spring 2017

Name (please print): _____

This exam is closed book, closed notes, no cellphones, computers or calculators. It is unproctored. Please try sit next to students you don't normally. You have 2 hours to answer as many questions as you can. Some questions are easy and some are hard but each is worth the same 4 points, so pick and choose. Most correct answers can be short. If you need more space, additional sheets may be stapled to your exam. I will be outside the room if you need to discuss something.

Please copy the following statement *in your own handwriting* and s*ign your name* below it:

> On my honor, I will neither accept nor give unauthorized aid on this exam.
>
> _____
>
> _____
>
> _____
> Signed

1. Convert 193 base 10 to binary and hex following the procedure discussed in class. What is the minimum number of bits needed to represent this as an unsigned number?

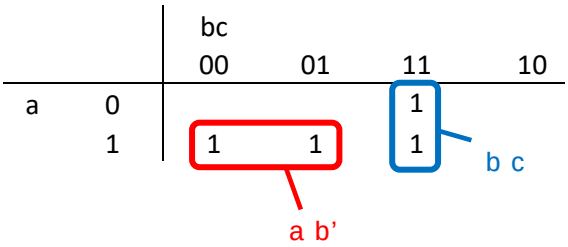
	Remainder
193 ÷ 2 = 96	1 LSB
96 ÷ 2 = 48	0
48 ÷ 2 = 24	0
24 ÷ 2 = 12	0
12 ÷ 2 = 6	0
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1 MSB

Resulting bitstring = 1100 0001 = hex C1

It requires 8 bits.

2. Derive the simplest SOP equation that implements f(a, b, c) = Σm(3, 4, 5, 7) by using a Karnaugh map.

Collecting the ones, f = a b' + b c

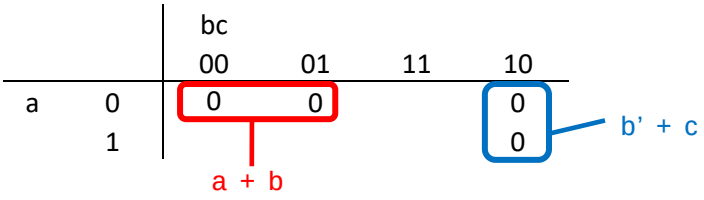


3. Derive the simplest SOP equation that implements f(a, b, c) = Σm(3, 4, 5, 7) by algebraic manipulation.

Σm(3, 4, 5, 7) = a' b c + a b' c' + a b' c + a b c
= (a b' c' + a b' c) + (a' b c + a b c)
= a b' (c' + c) + (a' + a) b c
= a b' + b c

4. Derive the simplest POS equation for f(a, b, c) = ΠM(0, 1, 2, 6) using a Karnaugh map.

Collecting the zeroes, f = (a + b)(b' + c)



5. Derive the simplest POS equation that implements f(a, b, c) = ΠM(0, 1, 2, 6) by algebraic manipulation.

ΠM(0, 1, 2, 6) = (a + b + c) (a + b + c') (a + b' + c) (a' + b' + c)

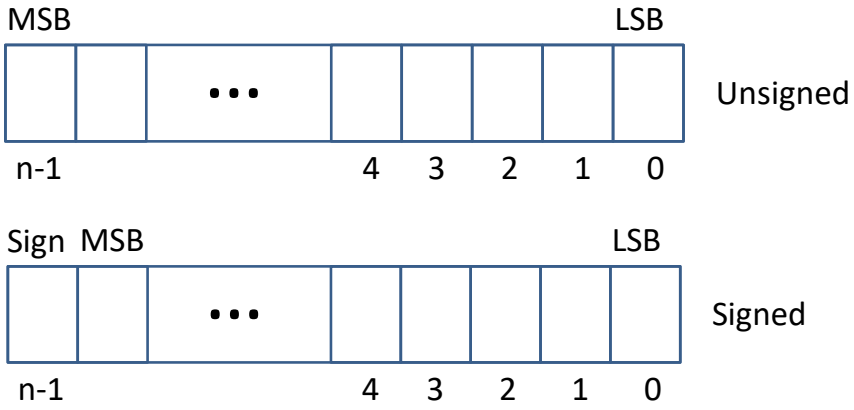
By the Combining Theorem,

(x + y) (x + y') = x x + x y' + x y + y y'
= x + x (y' + y) + 0
= x

Therefore,

ΠM(0, 1, 6, 7) = (a + b + c) (a + b + c') (a + b' + c) (a' + b' + c)
= (a + b) (b' + c)

6. Draw diagrams of signed and unsigned n-bit numbers, numbering the bits. What do LSB and MSB stand for? Mark them and any sign bits on your diagrams. In an unsigned number, a 1 in position ***k*** would have what value?



LSB = Least significant bit
MSB = Most significant bit

A one in position k in an unsigned number = 2^k

7. What is the difference between overflow and carry?

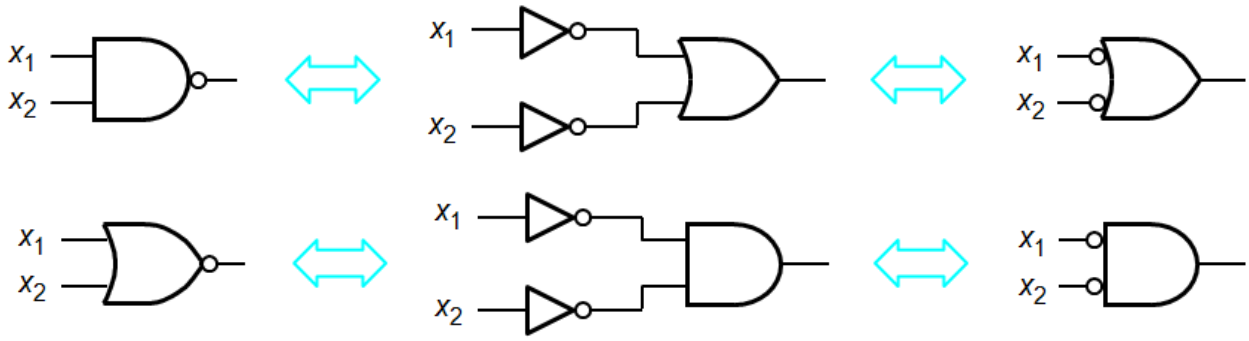
In unsigned binary addition, a carry is from the MSB and means the number was too large for the number of bits.

In signed binary addition, an overflow is a carry from the MSB into the sign bit and results in an incorrect result. A carry is out of the sign bit and can be ignored.

8. Draw an example of bubble-pushing. What theorem is this based on?

It's based on DeMorgan's Theorem:

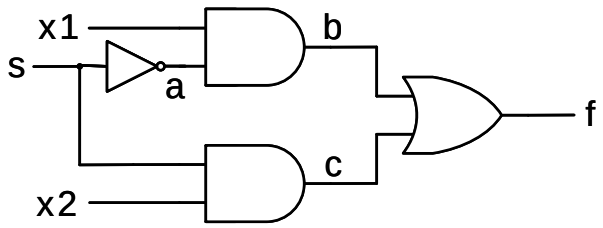
(a b)' = a' + b'
(a + b)' = a' b'



9. What is the difference between an implicant, a prime implicant and an essential prime implicant?

In SOP implementation, an implicant is a product term for which the output is a 1. In a POS implementation, an implicant is a sum term for which the output is a 0. A prime implicant is one that cannot be combined with another that has fewer literals. An essential prime implicant is one that must be included in any cover.

10. What does this circuit do? Is there a name for it? Write it as a Verilog module using only built-in gates and give it an appropriate name.



It's a 2:1 multiplexer. It selects either x1 or x2 based on the value of s.

```
module Mux2To1A( input s, x1, x2, output f );

   wire a, b, c;
   not ( a, s );
   and ( b, x1, a );
   and ( c, s, x2 );
   or ( f, b, c );

endmodule
```

11. Write the same function as a Verilog module using continuous assignment.

```
module Mux2To1B( input s, x1, x2,
      output f );

   assign f = s ? x2 : x1;

endmodule
```

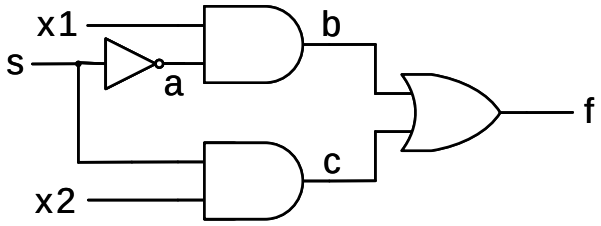
```
module Mux2To1C( input s, x1, x2,
      output f );

   assign f = ~s & x1 | s & x2;

endmodule
```

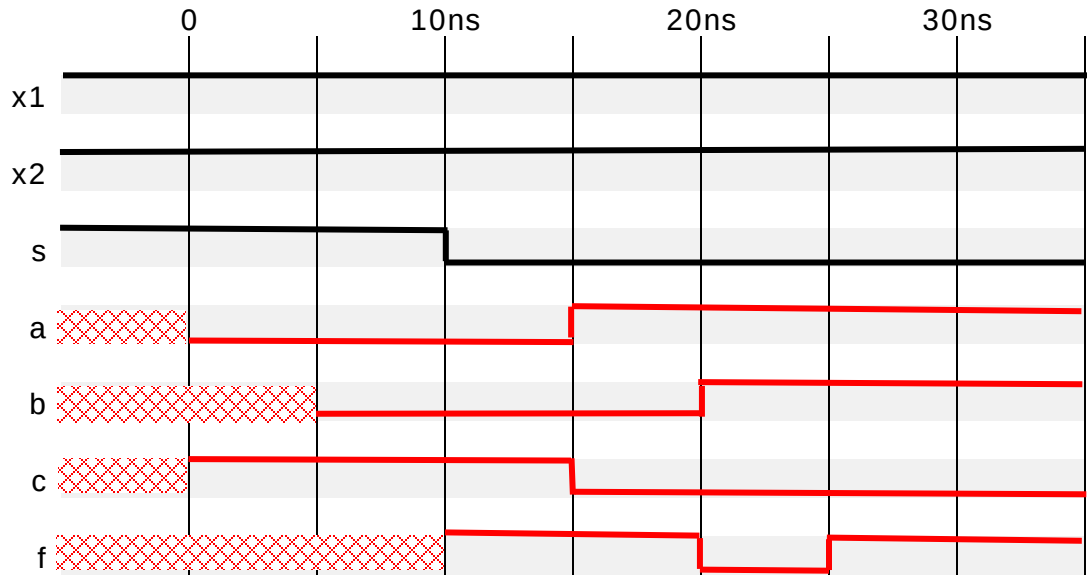
12. Write the same function as a Verilog module using a behavioral specification.

```
module Mux2To1D( input x1, x2, s,
      output reg f );

   always @( * )
      if ( s )
         f = x2;
      else
         f = x1;

endmodule
```

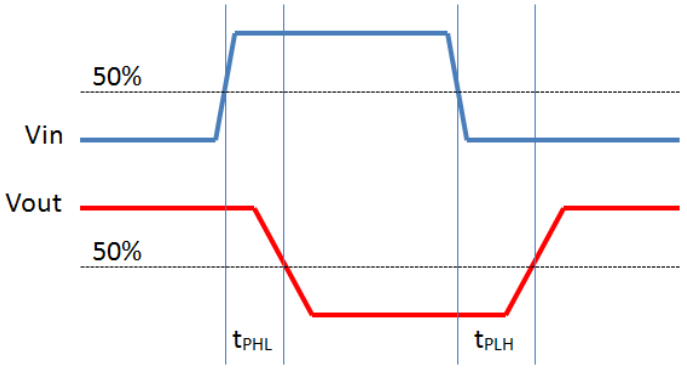
```
module Mux2To1E( input s, x1, x2,
      output f );

   always @( * )
      case ( s )
         0: f = x1;
         1: f = x2;
      endcase

endmodule
```

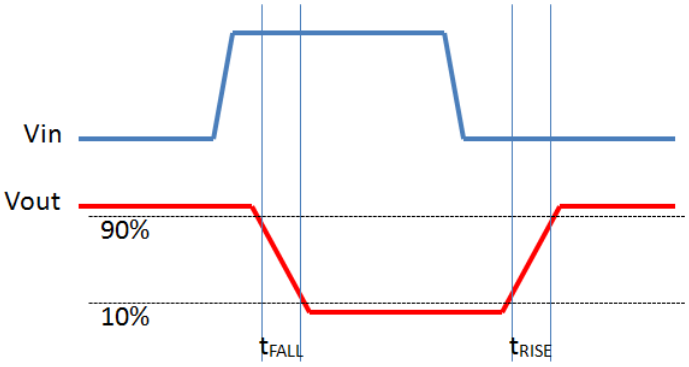
13. Again referring to this circuit, fill in the timing diagram below, showing what happens to signals a, b, c and f. Assume all gate delays are 5 ns. You may not assume anything about the input signals prior to what's shown, so please indicate by crosshatching any signals that are unknown.





14. Draw simple sketches of Vin and Vout versus time for an inverter, showing how t_{RISE}, t_{FALL}, t_{PLH} and t_{PHL} are measured.





15. For the function g defined by this Karnaugh map, write the minterm equation, g = Σm(…) + D(…), identify the prime implicants and any essential prime implicants and then write the simplified SOP equation.

g		b1 b0			
		00	01	11	10
b3 b2	00			d	
	01			1	d
	11			d	1
	10	1	1	d	

g = Σm(7, 8, 9, 14) + D(3, 6, 11, 15)

The red and blue prime implicants are essential. Green and purple are not.

g = b3 b2' b1' + b2 b1

16. For the function h defined by this Karnaugh map, write the Maxterm equation, h = ΠM(…) + D(…), identify the prime implicants and any essential prime implicants and then write the simplified POS equation.

h		b1 b0			
		00	01	11	10
b3 b2	00		0		
	01		d		
	11		d	0	
	10		0	d	

h = ΠM(1, 9, 15) + D(5, 11, 13)

Both implicants are essential prime.

h = (b3' + b0') (b1 + b0')

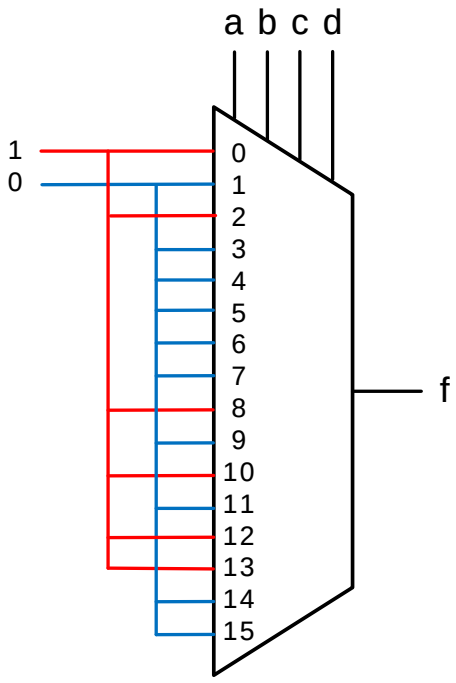
17. Implement f = a b c' + b' d' using a 16:1 multiplexer. If groups of inputs are all the same, you may simplify your drawing by indicating the list of inputs tied together.

Plotting this on a Karnaugh map, we can see that f = Σ m(0, 2, 8, 10, 12, 13), meaning this function can be implemented by tying those inputs to 1 and the others to 0.

f		cd			
		00	01	11	10
ab	00	1			1
	01				
	11	1	1		
	10	1			1

b' d'

a b c'



18. Implement your 16:1 decoder solution to f = a b c' + b' d' as directly as you can in Verilog using a case statement and the concatenation operator.

From the previous step, f = Σ m(0, 2, 8, 10, 12, 13).

```
module funcF ( input a, b, c, d, output reg f );

   always @( * )
      case ( { a, b, c, d } )
         0, 2, 8, 10, 12, 13: f = 1;
         default: f = 0;
      endcase

endmodule
```

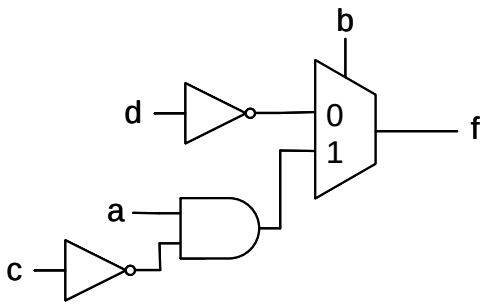
19. What is Shannon's expansion? What is a co-factor?

If f = f(x1, x2, x3, …, xn) then f = x1' $f_{x1'}$ + x1 f_{x1}
where $f_{x1'}$ and f_{x1} are cofactors of f with respect to x1' and x1 such that
$f_{x1'}$ = f(0, x2, x3, …, xn) and f_{x1} = f(1, x2, x3, …, xn)

20. Use Shannon's expansion to implement f = a b c' + b' d' using a 2:1 multiplexer, an AND gate and two inverters.

f = a b c' + b' d'

$f_{a'}$ = b' d'
f_{a} = b c' + b' d'
$f_{b'}$ = d'
f_{b} = a c'
$f_{c'}$ = a b + b' d'
f_{c} = b' d'
$f_{d'}$ = a b c' + b'
f_{d} = a' b c'



21. What are 3 different ways to represent a signed binary number? How would -17 (minus 17 decimal) be encoded as an 8-bit number under each method?

Sign + magnitude, 1s complement and 2s complement.

As 8-bit numbers, -17 (decimal) would be encoded as:

Sign + magnitude	1001 0001
1s complement	1110 1110
2s complement	1110 1111

22. What are the steps to convert a 2's complement number to sign + magnitude so you make sense of it? Given the 2's complement value 16'hFF75, what is the signed decimal value?

If the sign bit = 0, it's positive and needs no conversion. If the sign bit = 1, take the 1s complement (invert all the bits) and add 1.

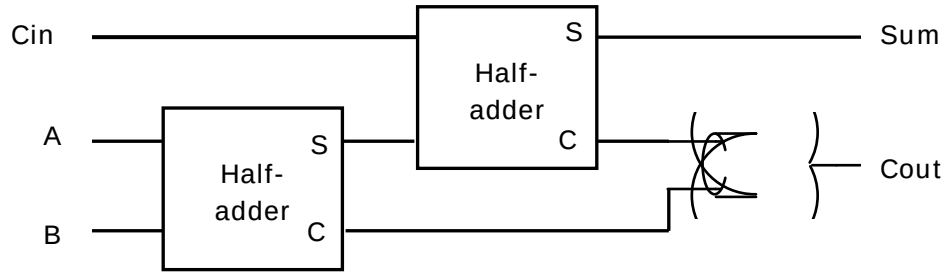
FF75 = 1111 1111 0111 0101 (Sign bit = 1)

inverting and adding one

= 0000 0000 1000 1010
\+ 1
= 0000 0000 1000 1011
= hex 008B
= 8 *16 + 11 = 139 decimal

FF75 = -139 decimal

23. Draw a circuit for a full-adder using two half-adders and any additional gates you need.



24. Write a Verilog module using a casex statement that takes a 4-bit value as input and outputs the number of ***trailing 1's***, e.g., 0111 → 3, 1011 → 2, 1100 → 0, etc.

```
module CountTrailingOnes( input [ 3:0 ] v, output reg [ 2:0 ] p );

   always @( * )
      casex ( v )
         4'bxxx0: p = 0;
         4'bxx01: p = 1;
         4'bx011: p = 2;
         4'b0111: p = 3;
         4'b1111: p = 4;
      endcase

endmodule
```

25. Create a module in Verilog that adds a 5-bit ***signed*** binary number to a 9-bit ***unsigned*** binary number, producing a 10-bit ***signed*** result using the + operator and ***explicit sign extension*** using replication and concatenation (not the signed keyword). Can it ever overflow? Give an example to explain your answer.

```
module AddS( input [ 4:0 ] A, input [ 8:0 ] B, output [ 9:0 ] sum );

   // Sign-extend A, replicating the sign bit A[ 4 ] through 5
   // positions.

   assign sum = { { 5{ A[ 4 ] } }, A } + B;
endmodule
```

Yes, it can overflow. For example:

A = 01 = +1
B = 1FF = largest 9-bit unsigned number
Sum = 3FE, which requires 10 bits + 1 bit for the sign = 11 bits.