CSE370 HW1 Solutions (Winter 2010)

General Note: I noticed a lot of people did the wrong problem and didn't do the extra bits as described on the assignment webpage, please take care when reading the assignment and problems.

1. **CLD2e, A.1**
   a. **Part c**
      0101011_2
      $= 1(2^0) + 1(2^1) + 0(2^2) + 1(2^3) + 0(2^4) + 1(2^5) + 0(2^6)$
      $= 1(2^0) + 1(2^1) + 1(2^3) + 1(2^5)$
      $= 1(1) + 1(2) + 1(8) + 1(32)$
      $= 1 + 2 + 8 + 32$
      $\boxed{= \mathbf{43_{10}}}$

   b. **Part d**
      757_8
      $= 7(8^0) + 5(8^1) + 7(8^2)$
      $=7(1) + 5(8) + 7(64)$
      $= 7 + 40 + 448$
      $\boxed{= \mathbf{495_{10}}}$

   c. **Part g**
      FFA_{16}
      $= 10(16^0) + 15(16^1) + 15(16^2)$
      $= 10(1) + 15(16) + 15(256)$
      $= 10 + 240 + 3840$
      $\boxed{= \mathbf{4090_{10}}}$

   Tip: To figure out something like 8^3 or 16^2 in your head just remember your powers of 2! E.g. $8 = 2^3$ and so $8^3 = (2^3)^3 = 2^9 = 512$…similarly $16 = 2^4$ so $16^2 = (2^4)^2 = 2^8 = 256$!

2. **CLD2e, A.2**
   a. **Part c**
      129 to base 2
      129 / 2 = 64 r 1
      64 / 2 = 32 r 0
      32 / 2 = 16 r 0
      16 /2 = 8 r 0
      8 / 2 = 4 r 0
      4 / 2 = 2 r 0
      2 / 2 = 1 r 0
      1 / 2 = 0 r 1
      $129_{10} \boxed{= \mathbf{10000001_2}}$
         We can check our work by converting because to base 10:
         $= 1(2^0) + 1(2^7)$
         $= 1 + 128$
         $= 129_{10}$

b. **Part g**

1023 to base 16

1023 / 16 = 63 r 15

63 / 16 = 3 r 15

3 / 16 = 0 r 3

$1023_{10} \boxed{= \mathbf{3FF_{16}}}$

Again, always a good idea to check your work:

$= 15(16^0) + 15(16^1) + 3(16^2)$

$= 15(1) + 15(16) + 3(256)$

$= 15 + 240 + 768$

$= 1023_{10}$

3. **CLD2e, A.4**

When converting to binary from a base that is a power of two you should first realize that each digit in the original base will convert to an integral number of digits in binary … as a result, each digit can be converted to binary independently and then concatenated together. The number of digits depends on the original base. Base 8 is base 2^3 which means you need 3 binary digits to represent it. Similarly, for base 16 = base 2^4 you need 4 binary digits to represent each digit in that base. Note that this trick only works because our original base is a power of two.

Some people first converted to base 10 and then to binary, this is a lot more work and slower.

a. **Part a**

252_8

|2|5|2|

$2_8 = 010_2$

$5_8 = 101_2$

$2_8 = 010_2$

|010|101|010|

$\boxed{\mathbf{010101010_2}}$

Check:

= 2 +8 + 32 + 128

$= 170_{10}$

170 / 8 = 21 r 2

21 / 8 = 2 r 5

2 / 8 = 0 r 2

$=252_8$

b. **Part d**

$AFE0_{16}$

|A|F|E|0|

$A_{16} = 1010_2$

$F_{16} = 1111_2$

$E16 = 1110_2$

$016 = 0000_2$

|1010|1111|1110|0000|

$\boxed{\mathbf{1010111111100000_2}}$

Check:

= 32 + 64 + 128 + 256 + 512 + 1024 + 2048 + 8192 + 32768
= 45024

45024 / 16 = 2814 r 0
2814 / 16 = 175 r 14
175 / 16 = 10 r 15
10 / 16 = 0 r 10

= $AFE0_{16}$

4. CLD2e, A.7

a. Part b

$110111_2 + 101_2$

First I am going to pad the second number out so that it has the same number of digits:
= $110111_2 + 000101_2$

We will perform the addition starting from the least significant digits and moving to the more significant digits, just like addition in base 10. And just like in addition in base ten we will keep track of whether we need to carry a 1 or not.

1 + 1 = 0 + carry(c)
c + 1 + 0 = 0 + c
c + 1 + 1 = 1 + c
c + 0 + 0 = 1
1 + 0 = 1
1 + 0 = 1

Reading off the column after = we get the answer:

$\boxed{\mathbf{= 111100_2}}$

We can check the result by performing the addition in base 10:
$55 + 5 = 60_{10}$
60 / 2 = 30 r 0
30 / 2 = 15 r 0
15 / 2 = 7 r 1
7 / 2 = 3 r 1
3 / 2 = 1 r 1
1 / 2 = 0 r 1
= 111100_2

b. Part e

$11011100110_2 + 10011001_2$

First I will pad the second number so it has the same number of digits:
$11011100110_2 + 00010011001_2$

I will follow the same procedure as before adding the number digit by digit starting with the least significant digit:

0 + 1 = 1
1 + 0 = 1
1 + 0 = 1
0 + 1 = 1

0 + 1 = 1
1 + 0 = 1
1 + 0 = 1
1 + 1 = 0 + carry(c)
c + 0 + 0 = 1
1 + 0 = 1
1 + 0 = 1

Reading off the column after = we get the answer:

$\boxed{\mathbf{= 11101111111_2}}$

This time let's check by adding in base 16:

$6E6_{16} + 099_{16}$

6 + 9 = F
E + 9 = 7 + carry(c)
c + 6 + 0 = 7

$= 77F_{16}$

=|0111|0111|1111|

$= 11101111111_2$

5. **CLD2e, 1.3 and 1.4**
   a. **1.3**

There are a lot of potential encodings for the cards. I will describe three common ones, but there may be others. Some people just reordered their bits for their second encoding. This is technically another encoding, but the point of this problem (along with 1.4) is to show how easy/hard it can be to represent different concepts using different encodings.

For all the encodings, the suit order we will use is hearts, diamonds, clubs, spades. The rank order is Ace to King.

Number-Per-Card Encoding

Since there are 52 cards, we can encode this with 6 bits which can encode up to 64 values. Each card could just be uniquely defined. This would be a good choice if we never needed to distinguish between cards of the same suit…on the other hand if we want combinatorial logic to do something special when a card has a certain suit then this type of encoding would make that logic more complex (see part 1.4)

E.g. 0_{10} = 000000 = A of Hearts
1_{10} = 000001 = 2 of Hearts
…
50_{10} = 110010 = Q of Spades
51_{10} = 110011 = K of Spades

Rank-Suit Encoding

The design rational behind this encoding is that we can use 2 bits for the suit and 4 bits (16 values, of which we only need 13) to represent the rank. We then append the 6 bits together to get our value. This is just as compact as the previous representation, but as you will see from 1.4 it makes it easier to represent all cards of a single suit.

E.g. 0000 00 = A of Hearts
0001 00 = 2 of Hearts
…
1011 11 = Q of Spades
1100 11 = K of Spades

One-Hot Encoding
Each suit can be represented by a one-hot encoding of 4 bits. Also, each rank can be encoded using a one-hot encoding of 13 bits. Appending these together gives us 17 bits total. As alternatives, some people used a “one-hot encoding” for just the suit or just the rank and a binary encoding for the other. This encoding has the benefit that it is easy to write an expression for a single rank or single suit (see 1.4), but uses a lot of wires.

E.g. 0000000000001 0001 = A of Hearts
0000000000010 0001 = 2 of Hearts
…
0100000000000 1000 = Q of Spades
1000000000000 1000 = K of Spades

b. 1.4
Note: A lot of people had issues with this. The problem states that you should come up with a “logic expression” for different cases for *each* encoding above. A lot of people only did one of their encodings. Also, a lot of people just gave the value for each of the examples, instead of the logic expression. A logic expression is a formula i.e. F=ABCD

For encoding 1, let us name the bits A B C D E F.
For encoding 2, let us name the bits A B C D S2 S1.
For encoding 3, let us name the bits R_{12}-R_0 S C D H

Also, my formulae assume no invalid input.

i. A jack of diamonds
Number-Per-Card: J_D = A’B’CDE’F ((13*1+10)-1 = 25_{10} = 001101)
Rank-Suit: J_D = AB’CD’S2’S1
One-Hot: J_D = R_{10}D

ii. A seven of any suit
Number-Per-Card: Seven = A’B’C’DEF’ + A’BC’D’EF + AB’C’D’E’F’ + AB’CDE’F
Rank-Suit: Seven = A’BCD’
One-Hot: Seven = R_6

iii. Any card of hearts suit
Number-Per-Card: Heart = A’B’C’D’E’F’ + A’B’C’D’E’F + A’B’C’D’EF’ + A’B’C’D’EF + A’B’C’DE’F’ + A’B’C’DE’F + A’B’C’DEF’ + A’B’C’DEF + A’B’CD’E’F’ + A’B’CD’E’F + A’B’CD’EF’ + A’B’CD’EF + A’B’CDE’F’
Rank-Suit: Heart = S2’S1’

One-Hot: Heart = H

6. CLD2e, 1.9 then realize your equation using AND, OR, and NOT gates (width as many inputs as needed)

Note: A lot of people wrote out the formulae using words for the month e.g. "February". Also some people used "m2" for February etc. You need to make sure you specify things in terms of the bits themselves. E.g. February = $m_8'm_4'm_2m_1'$

This changes the truth table to be:

m_8	m_4	m_2	m_1	L	d_{28}	d_{29}	d_{30}	d_{31}
0	0	0	0	X	X	X	X	X
0	0	0	1	X	1	1	1	1
0	0	1	0	0	1	0	0	0
0	0	1	0	1	1	1	0	0
0	0	1	1	X	1	1	1	1
0	1	0	0	X	1	1	1	0
0	1	0	1	X	1	1	1	1
0	1	1	0	X	1	1	1	0
0	1	1	1	X	1	1	1	1
1	0	0	0	X	1	1	1	1
1	0	0	1	X	1	1	1	0
1	0	1	0	X	1	1	1	1
1	0	1	1	X	1	1	1	0
1	1	0	0	X	1	1	1	1
1	1	0	1	X	X	X	X	X
1	1	1	0	X	X	X	X	X
1	1	1	1	X	X	X	X	X

From this truth table we can see that:

$d_{28} = 1$ *"every month"*

$d_{29} = (m_8'm_4'm_2m_1'L')' = m_8 + m_4 + m_2' + m_1 + L$ *"every month except Feb. in a non-leap year"*

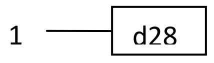
$d_{30} = (m_8'm_4'm_2m_1')' = m_8 + m_4 + m_2' + m_1$ *"every month except Feb."*

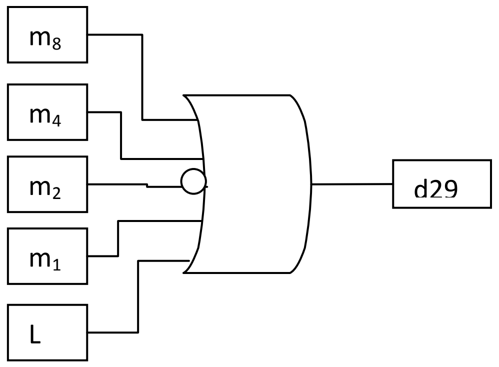
$d_{31} = m_8'm_4'm_2'm_1 + m_8'm_4'm_2m_1 + m_8'm_4m_2'm_1 + m_8'm_4m_2m_1 + m_8m_4'm_2'm_1' + m_8m_4'm_2m_1' + m_8m_4m_2'm_1' = m_8m_1' + m_8'm_1$

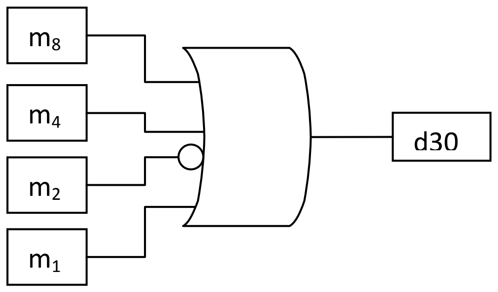
Note that in my solution I am setting the "don't cares" to 1 when beneficial. Some people just used all the minterms and that is fine, but this is a more compact representation.

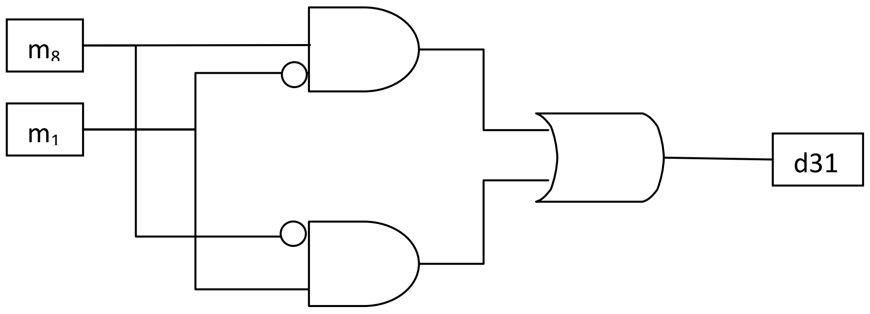
A couple notes:
1) Assume you can always just set an output to true or false as I did with d_{28}
2) One could have written the outputs for d_{29} and d_{30} with minterms, but doing it with maxterms results in a smaller circuit.

The circuit diagrams for these is as follows:



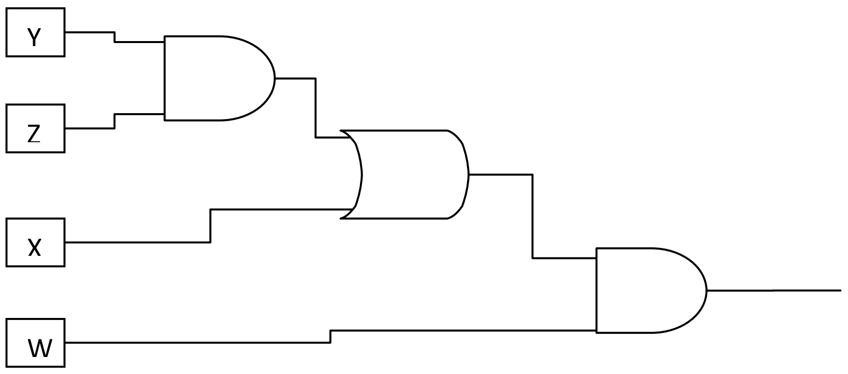






7. CLD2e, 2.2, part e

Draw a schematic for W(X + YZ)



When asked to draw these schematics I like to start with the smallest terms, in this case "Y AND Z" and then work outward to the larger terms. So next we will add the "OR X", and then finally, we can perform the final AND.

8. CLD2e, 2.9

Prove:

BC + A’B’ + A’C’ = ABC + A’

BC + A’B’ + A’C’	
(B’+C’)’ + (A+B)’ + (A+C)’	DeMorgan’s Law [12] (x3)
((B’+C’)(A+B)(A+C))’	DeMorgan’s Law [12D]
((B’+C’)(A + BC))’	Distributive Law [8D]
(A(B’+C’) + BC(B’+C’))’	Distributive Law [8]
(AB’ + AC’ + B’BC + BCC’)’	Distributive Law [8] (x2)
(AB’ + AC’ + 0C + 0B)’	Theorem of Complementarity [5D] (x2)
(AB’ + AC’)’	Operations with 0 [2D] (x2)
(A(B’+C’))’	Distributive Law [8]
A’ + (B’+C’)’	DeMorgan’s Law [12D]
A’ + BC	DeMorgan’s Law [12]
A’ + ABC	Simplification Theorem [11D]
ABC + A’	Commutative Law [6]

9. CLD2e, 2.12 then derive an XOR implementation using only NOR gates

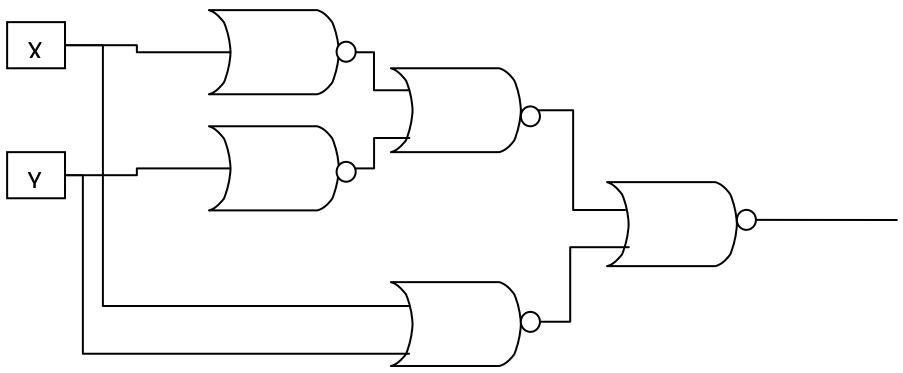
From the diagram we can construct a term from each of the gates:

((X (XY)’)’ (Y (XY)’)’)’	
(X (XY)’) + (Y (XY)’)	DeMorgan’s Law [12D]
X(X’+Y’) + Y(X’+Y’)	DeMorgan’s Law [12D] (x2)
XX’ + XY’ + X’Y + YY’	Distributive Law [8] (x2)
0 + XY’ + X’Y + 0	Theorem of Complementarity [5D] (x2)
XY’ + X’Y	Operations with 0 [1] (x2)
X XOR Y	Definition of XOR

To construct the circuit using only NOR gates we can work backwards:

XY’ + X’Y	
0 + XY’ + X’Y + 0	Operations with 0 [1] (x2)
XX’ + XY’ + X’Y + YY’	Theorem of Complementarity [5D] (x2)
X(X’+Y’) + Y(X’ + Y’)	Distribitive Law [8] (x2)
(X+Y)(X’+Y’)	Distributive Law [8]
(((X+Y)(X’+Y’))’)’	Involution Theorem [4]
((X+Y)’ + (X’+Y’)’)’	DeMorgan’s Law [12D]
((X+Y)’ + ((X+X)’ + (Y+Y)’)’)’	Idempotent Theorem [3] (x2)

Now we have a formula that only uses NOR operations so we can draw the circuit:



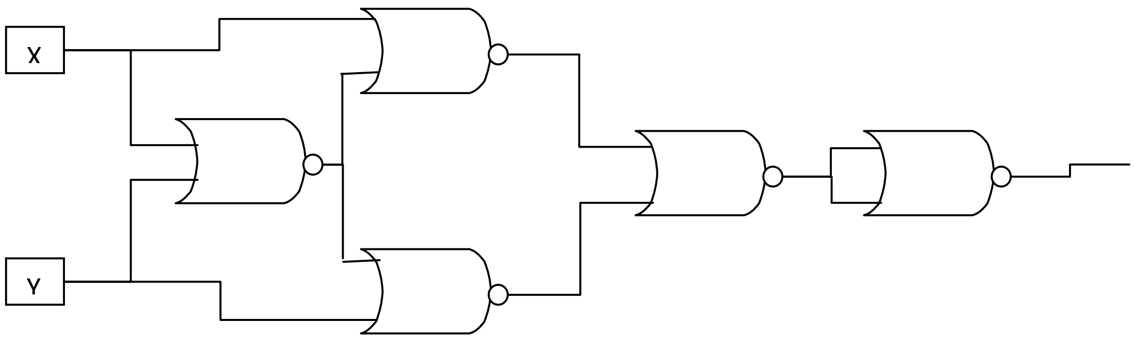
There is an **alternative circuit** that also computes XOR using NOR gates and can be derived as follows:

XY’ + X’Y	
0 + XY’ + X’Y + 0	Operations with 0 [1] (x2)
YY’ + XY’ + X’Y + XX’	Theorem of Comp. [5D] (x2)
Y’(X+Y) + X’(X+Y)	Distributive Law [8] (x2)
(Y + (X+Y)’)’ + (X +(X+Y)’)’	Involution Theorem [4] (x2)

We are close at this point but we only have an OR not a NOR at the top level…

((Y + (X+Y)’)’ + (X +(X+Y)’)’)((Y + (X+Y)’)’ + (X +(X+Y)’)’)	Idempotent Theorem [3D]
(((Y + (X+Y)’)’ + (X +(X+Y)’)’)’ + ((Y + (X+Y)’)’ + (X +(X+Y)’)’)’)’	DeMorgan’s Law [12D]

What we are doing in the last step is as follows:
Let A = (Y + (X+Y)’)’ + (X +(X+Y)’)’
A = AA = ((AA)’)’ = (A’ + A’)’
…so our A is negated, turning the OR into a NOR, and the whole thing is itself another NOR.
The circuit diagram for this is as follows:



Note that although the formula is quite large we can reuse values in the circuit giving us a fairly compact final circuit.

Some people arrived at this diagram by adding “bubbles” to the NAND diagram. This usually resulted in inverting the X and Y inputs. However, this is actually unnecessary since XOR with inverted inputs is still XOR, i.e. X XOR Y = X’ XOR Y’