2. Practice Problem Solutions

The following are solutions to the set of practice problems presented separately. Use these only after attempting the problems on your own.

HW2 - Number Systems and Conversions

1.

a. 1110110.010111_2
$= 1*2^6 + 1*2^5 + 1*2^4 + 1*2^2 + 1*2^1 + 1*2^{-2} + 1*2^{-4} + 1*2^{-5} + 1*2^{-6}$
$= 118.359375_{10}$
$= 001 \mid 110 \mid 110.010 \mid 111_2 = 166.27_8$
$= 0111 \mid 0110.0101 \mid 1100_2 = 76.5C_{16}$

b. $15B.35_{16}$
$= 1*16^2 + 5*16^1 + 11*16^0 + 3*16^{-1} + 5*16^{-2} = 347.20703125_{10}$
$= 0001 \mid 0101 \mid 1011.0011 \mid 0101_2$
$= 101 \mid 011 \mid 011.001 \mid 101 \mid 010_2 = 533.152_8$

2. Use the "Making Change" method
   a. Start by listing the powers of 2 and then find the coefficients of the number by starting with the largest powers and working toward lower powers determining which ones sum to the desired value.

0	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**.**	**1**	**0**	**1**
1024	512	256	128	64	32	16	8	4	2	1		.5	.25	.125

$(923.625)_{10} = (1110011011.101)_2$

3.

a. 4530.152_8
$= 100\ 101\ 011\ 000.001\ 101\ 01_2$
$= 1001 \mid 0101 \mid 1000.0011 \mid 0101_2 = 958.35_{16}$

4.

a. $DABB.AD00_{16}$
$= 1101\ 1010\ 1011\ 1011.1010\ 1101_2$
$= 001 \mid 101 \mid 101 \mid 010 \mid 111 \mid 011.101 \mid 011 \mid 010_2 = 155273.532_8$

b. $BAD.A_{16} = 11*16^2 + 10*16^1 + 13*16^0 + 10*16^{-1} = 2989.625_{10}$

5. Use the "Making Change" method: Start by listing the powers of 5 and then find the coefficients of the number by starting with the largest powers and working toward lower powers determining which ones sum to the desired value

a.

0	**3**	**2**	**0**	**3**	**3**	**.**	**3**	**1**
3125	625	125	25	5	1		.2	.04

$(2143.64)_{10} = (32033.31)_5$

HW3 - Boolean Algebra, Logic Functions, and Canonical Representation, 2-Level Implementations

1. **Probably the easiest method is perfect induction (i.e. a truth table)**
   F = X + X' = 1

X	X'	F
0	1	1
1	0	1

2.

a.

F = WXYZ • (WXYZ' + WX'YZ + W'XYZ +WXY'Z)
= WWXXYYZZ' • WWXX'YYZZ + W'WXXYYZZ + WWXXYY'ZZ — T8
= WXYZZ' + WXX'YZ + WW'XYZ + WXYY'Z — T3'
= 0 + 0 + 0 + 0 + 0 — T5'
= 0 — A4'

b.

F = AB + ABC'D + ABDE' + ABC'E + C'D
= AB • (1 + C'D + DE' + C'E) + C'D — T8
= AB • (1 + DE' + C'E) + C'D — T6
= AB • (1) + C'D — T2
= AB + C'D — T1'

3.

a. F = X'Y + X'Y'Z'

XYZ	X'	X' • Y	Y'	Z'	X'• Y' • Z'	F
000	1	0	1	1	1	1
001	1	0	1	0	0	0
010	1	1	0	1	0	1
011	1	1	0	0	0	1
100	0	0	1	1	0	0
101	0	0	1	0	0	0
110	0	0	0	1	0	0
111	0	0	0	0	0	0

b. F = W’ + X' • (Y' + Z)

WXYZ	W’	Y’	Y’+Z	X’	X’ • (Y’+Z)	F
0000	1	1	1	1	1	1
0001	1	1	1	1	1	1
0010	1	0	0	1	0	1
0011	1	0	1	1	1	1
0100	1	1	1	0	0	1
0101	1	1	1	0	0	1
0110	1	0	0	0	0	1
0111	1	0	1	0	0	1
1000	0	1	1	1	1	1
1001	0	1	1	1	1	1
1010	0	0	0	1	0	0
1011	0	0	1	1	1	1
1100	0	1	1	0	0	0
1101	0	1	1	0	0	0
1110	0	0	0	0	0	0
1111	0	0	1	0	0	0

c. F = (A' + B' + CD) • (B + C' + D')

ABCD	A’	B’	CD	A' + B' + CD	C’	D’	B + C' + D’	F
0000	1	1	0	1	1	1	1	1
0001	1	1	0	1	1	0	1	1
0010	1	1	0	1	0	1	1	1
0011	1	1	1	1	0	0	0	0
0100	1	0	0	1	1	1	1	1
0101	1	0	0	1	1	0	1	1
0110	1	0	0	1	0	1	1	1
0111	1	0	1	1	0	0	1	1
1000	0	1	0	1	1	1	1	1
1001	0	1	0	1	1	0	1	1
1010	0	1	0	1	0	1	1	1
1011	0	1	1	1	0	0	0	0
1100	0	0	0	0	1	1	1	0
1101	0	0	0	0	1	0	1	0
1110	0	0	0	0	0	1	1	0
1111	0	0	1	1	0	0	1	1

d.

F = (((A’ + B)' + C')' + D)'
= ((A’ + B)’ + C') • D’ DEMORGANS
= (A • B’ + C’) • D’ DEMORGANS
= AB’D’ + C’D’ T8

ABCD	B’	D’	A•B’•D’	C’	C’•D’	F
0000	1	1	0	1	1	1
0001	1	0	0	1	0	0
0010	1	1	0	0	0	0
0011	1	0	0	0	0	0
0100	0	1	0	1	1	1
0101	0	0	0	1	0	0
0110	0	1	0	0	0	0
0111	0	0	0	0	0	0
1000	1	1	1	1	1	1
1001	1	0	0	1	0	0
1010	1	1	1	0	0	1
1011	1	0	0	0	0	0
1100	0	1	0	1	1	1
1101	0	0	0	1	0	0
1110	0	1	0	0	0	0
1111	0	0	0	0	0	0

4.

a.

F = Σ_{XYZ} (0,2,3)
= m0 + m2 + m3
= X’Y’Z’ + X’YZ’ + X’YZ

= $\prod_{XYZ}$ (1,4,5,6,7)
= M1 • M4 • M5 • M6 • M7
= (X+Y+Z’)•(X’+Y+Z)•(X’+Y+Z’)•(X’+Y’+Z)•(X’+Y’+Z’)

b.

F = $\prod_{ABC}$ (1,2,4,6)
= M1 • M2 • M4 • M6
= (A + B + C’) • (A + B’ + C) • (A’ + B + C) • (A’ + B’ + C)

= $\Sigma_{A,B,C}$ (0,3,5,7)
= m0 + m3 + m5 + m7
= A’B’C’ + A’BC + AB’C + ABC

c.

F $= \Sigma_{ABCD}(1,2,5,7)$
$= m1 + m2 + m5 + m7$
$= A'B'C'D + A'B'CD' + A'BC'D + A'BCD$

$= \prod_{ABCD}(0,3,4,6,8,9,10,11,12,13,14,15)$
$= M0 \bullet M3 \bullet M4 \bullet M6 \bullet M8 \bullet M8 \bullet M9 \bullet M10 \bullet M11 \bullet M12 \bullet M13 \bullet M14 \bullet M15$
$= (A + B + C + D) \bullet (A + B + C' + D') \bullet (A + B' + C + D)$
$\bullet (A + B' + C' + D) \bullet (A' + B + C + D) \bullet (A' + B + C + D')$
$\bullet (A' + B + C' + D) \bullet (A' + B + C' + D') \bullet (A' + B' + C + D)$
$\bullet (A' + B' + C + D') \bullet (A' + B' + C' + D) \bullet (A' + B' + C' + D')$

d.

F $= X' + YZ' + YZ' = X' + YZ'$ T3
$= X'(Y + Y')(Z + Z') + (X + X')(YZ')$ T1'/T5
$= X'Y'Z' + X'Y'Z + X'YZ' + X'YZ + XYZ' + X'YZ'$ T8
$= m0 + m1 + m2 + m3 + m6$
$= \Sigma_{XYZ}(0,1,2,3,6)$

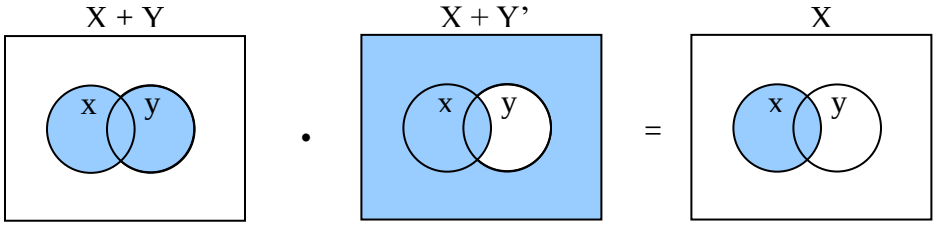
$= \prod_{X,Y,Z}(4,5,7)$
$= M4 \bullet M5 \bullet M7$
$= (X' + Y + Z) \bullet (X' + Y + Z') \bullet (X' + Y' + Z')$

e.

F $= A'B + B'C + AC'$
$= (A'BC + A'BC') + (AB'C + A'B'C) + (ABC' + AB'C')$ T10
$= m3 + m2 + m5 + m1 + m6 + m4$
$= \Sigma_{ABC}(1,2,3,4,5,6)$

$= \prod_{ABC}(0,7)$
$= M0 \bullet M7$
$= (A + B + C) \bullet (A' + B' + C')$

5.



6. **One possible solution:**

XY + X'Z + YZ	
XY + X'Z + YZ • 1	T1'
XY + X'Z + YZ • (X + X')	T5
XY + X'Z + XYZ + XYZ	T8
(XY + XYZ) + (X'Z + X'ZY)	T6
XY + X'Z	T9

7.

a.

F	= X'Z' + (Y(X'+Z))'	
	= X'Z' + Y' + (X'+Z)'	DEMORGANS
	= X'Z' + Y' + XZ'	DEMORGANS
	= (X' + X) • Z' + Y'	T8
	= 1 • Z' + Y'	T5
POS	**= Z' + Y'**	T1'

b.

G	= XY + Y'Z'	
	= (XY + Y')(XY + Z')	T8'
	= (X + Y')(Y + Y')(X + Z')(Y + Z')	T8'
	= (X + Y')(X + Z')(Y + Z')	T3'
	= (X + Y')(Y + Z')	T11'
POS	**= (X + Y') • (Y + Z')**	

c.

H	= AB • (CD)' + A + D	
	= (AB • (C' + D')) + A + D	DEMORGANS
	= (AB + A + D) • (C' + **D'** + A + **D**)	T8'
	= (A(B+1) + D) • (1)	
	= (A + D) • 1	T1'
POS	**= A + D**	

8.

Z	= AB + (C' + A'B')' + A'(AB + AC'D')	
	= AB + (C' + A'B')' + A'AB + A'AC'D'	T8
	= AB + C• (A+B) + 0 + 0	DEMORGANS, T5'
SOP	**= AB + AC + BC**	T8

9.

a. F = x'y' + xy'z + z'

Let us simplify first

= y'(x' + xz) + z'	
= y'(x'(z + z') + xz) + z'	T1', T5
= y'(x'z + x'z' + xz) + z'	T8

= y'(x'z + x'z + x'z' + xz) + z'	T3 (replicate terms)
= y'(x'(z+z') + z(x'+x)) + z'	T8
= y'(x' +z) + z'	T5,T1'
Now let us convert to POS using T8'	
= (z'+y')(z' + x' + z)	T8'
= (z'+y')	T5,T1'

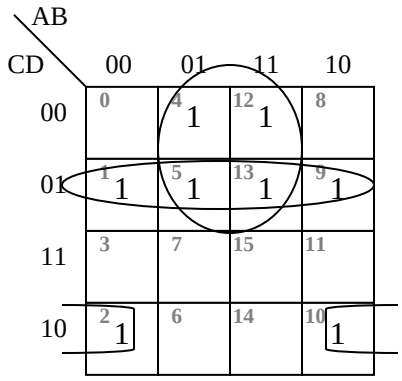
b. G = (x'+y')(y)(w'+y+z)
Convert to SOP using T8 (simplifying as we go)

= (x'y + y'y)(w' + y + z)	T8
= x'y(w' + y + z)	T5'/T1
= w'x'y + x'y + x'yz	T8
= x'y(w' + 1 + z)	T8
= x'y	T2 / T1'

HW4 - Circuit Design w/ Karnaugh Maps

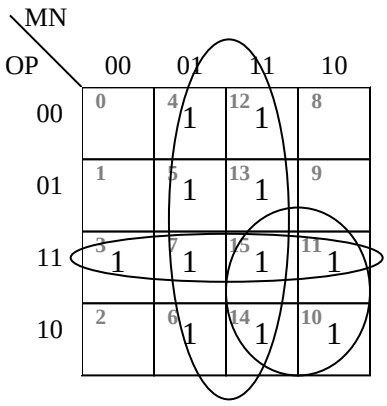
1.

a. **F = Σ_{ABCD}(1,2,4,5,9,10,12,13)**



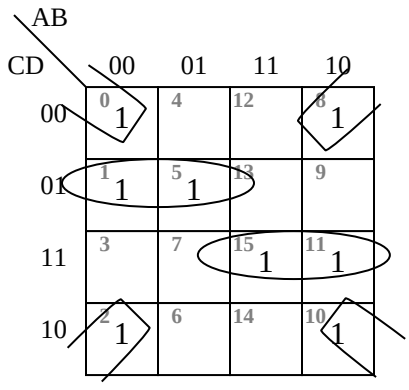
$F_{SOP} = \overline{C}D + B\overline{C} + \overline{B}C\overline{D}$

b. **F = $\prod_{MNOP}$(0,1,2,8,9)**



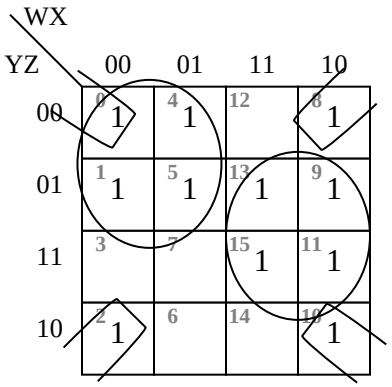
$F_{SOP} = N + OP + MO$

c. **F = Σ_{ABCD} (0,1,2,5,8,10,11,15)**



$$F_{SOP} = \overline{A}\overline{C}D + ACD + \overline{B}\overline{D}$$

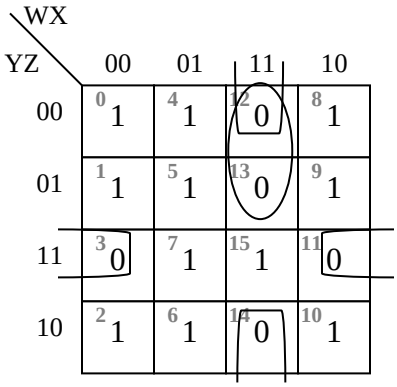
d. **F = ∏WXYZ (3,6,7,12,14)**



$$F_{SOP} = \overline{W}\overline{Y} + WZ + \overline{X}\overline{Z}$$

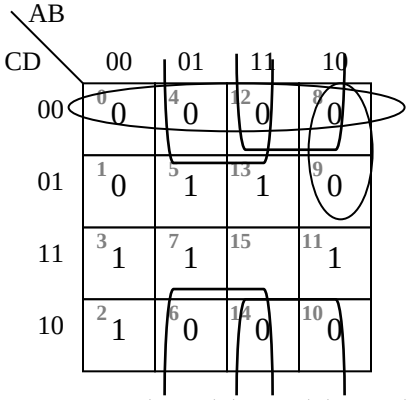
2.

a. F = X'Y' + X'Z' + W'X + XYZ



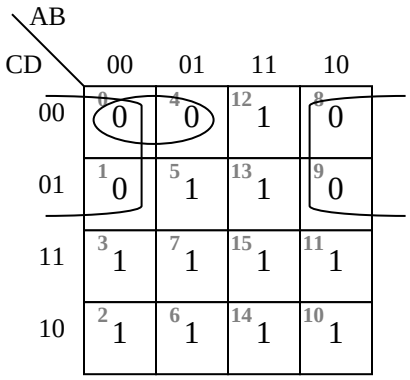
$$F_{POS} = (W' + X' + Y)(X + Y' + Z')(W' + X' + Z)$$

b. The 4-bit prime numbers are: 2, 3, 5, 7, 11, 13



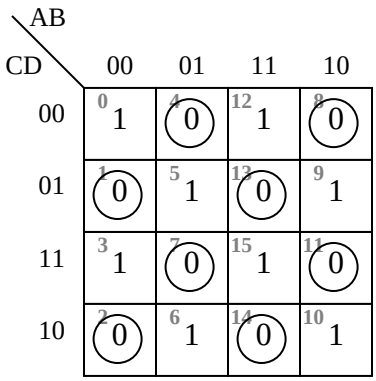
$$F_{POS} = (C+D)(B'+D)(A'+D)(A'+B+C)$$

c. The 4-bit numbers that are not perfect squares or cubes are: 2, 3, 5, 6, 7, 10, 11, 12, 13, 14, 15.



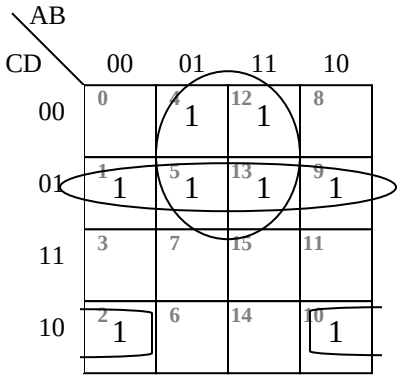
$F_{POS} = (B + C)(A + C + D)$

d. The 4-bit numbers divisible by 3 or 5 are: 0, 3, 5, 6, 9, 10, 12, 15

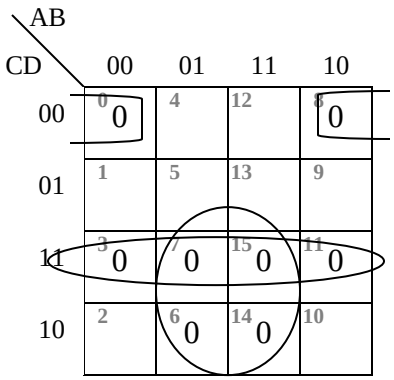


F_{POS} = (A+B+C+D')(A+B+C'+D)(A+B'+C+D)(A+B'+C'+D')(A'+B+C+D)•
(A'+B+C'+D')(A'+B'+C+D')(A'+B'+C'+D)

3. **The K-Maps are below:**

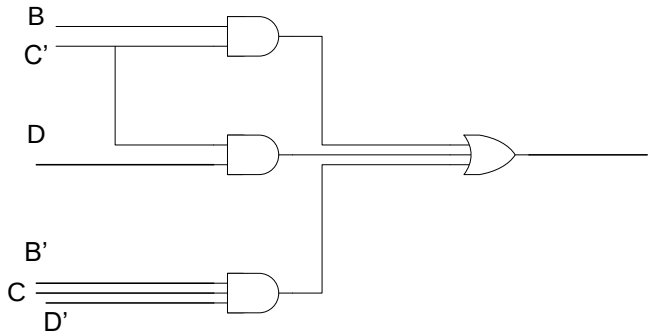


$F_{SOP} = \overline{C}D + B\overline{C} + \overline{B}C\overline{D}$

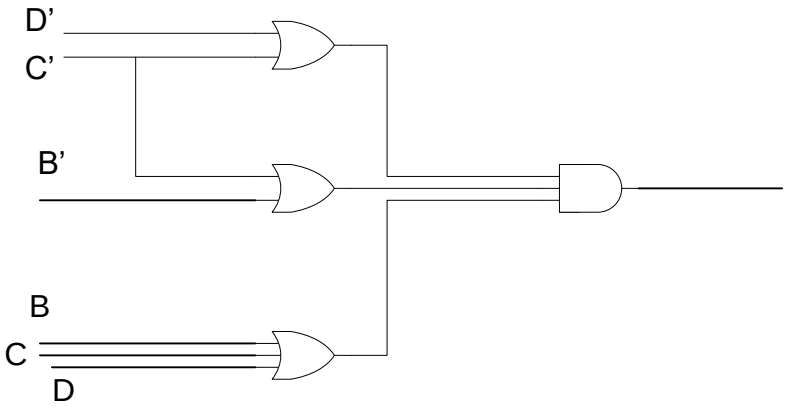


$$F_{POS} = (\overline{C} + \overline{D})(\overline{B} + \overline{C})(B + C + D)$$

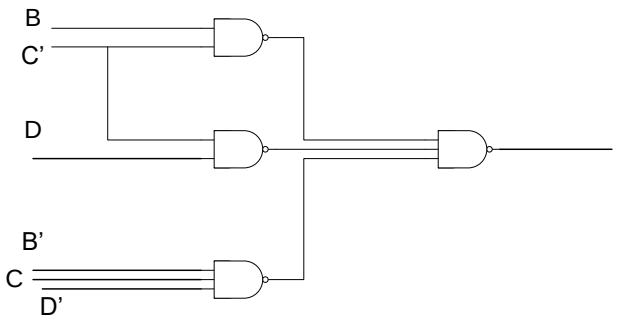
a. AND – OR Implementation



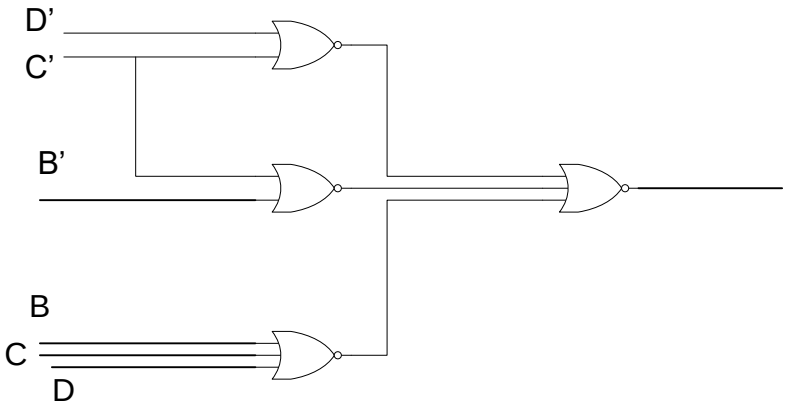
b. OR - AND Implementation



c. NAND - NAND Implementation

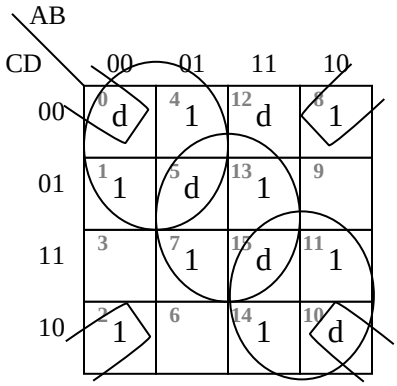


d. NOR - NOR Implementation



4.

a. The values with an odd number of 1s are: 1, 2, 4, 7, 8, 11, 13, 14. The Don't Care numbers are: 0, 4, 5, 8, 10, 12, and 15.

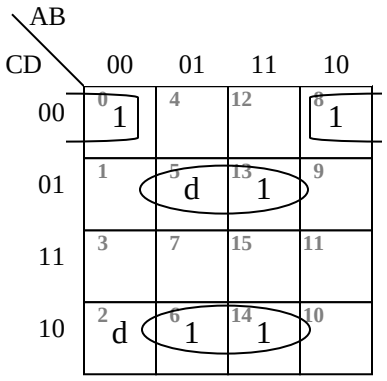


SOP = A'C' + BD + AC + B'D'

b. F = B'C'D' + BCD' + ABC'D, d = A'BC'D + A'B'CD'

F = (A+A')B'C'D' + (A+A')BCD' + ABC'D
= AB'C'D' + A'B'C'D' + ABCD' + A'BCD' + ABC'D

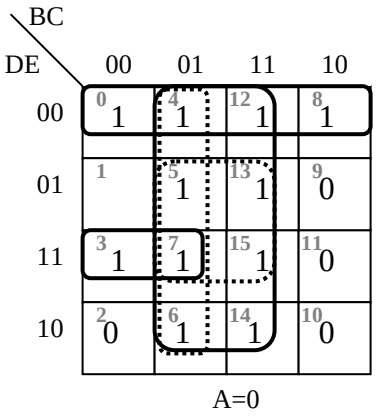
d = A'BC'D + A'B'CD'

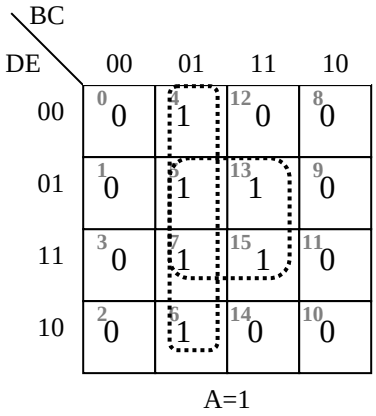


F_{SOP} = B'C'D' + BC'D + BCD'

5.

a. F = A’C + ACE + CAB’ + B’A’DE + A’ D’ E’

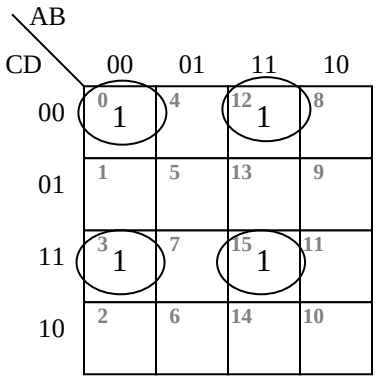




F_{SOP} = A’C + A’D’E’ + A’B’DE + B’C + CE

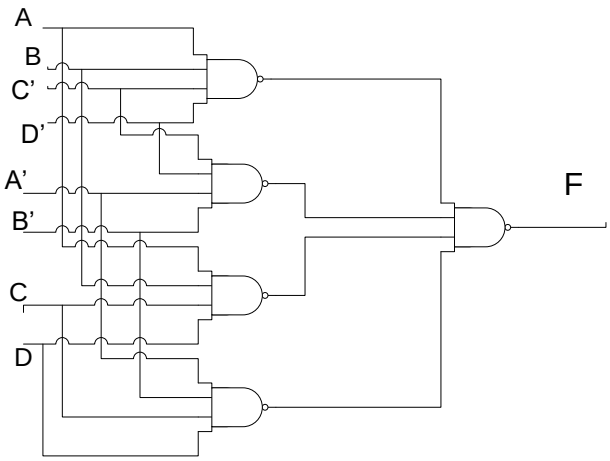
b. F = (AB + A’B’) (C’D’ + CD)
= ABC’D’ + A’B’C’D’ + ABCD + A’B’CD

- K-Map:



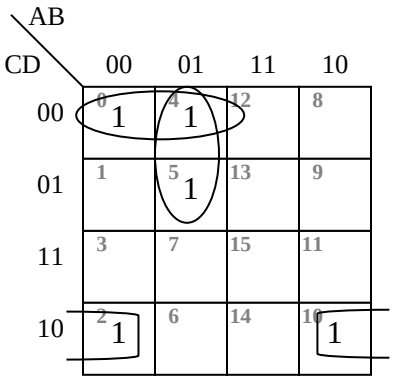
F_{SOP} = ABC’D’ + A’B’C’D’ + ABCD + A’B’CD

- NAND – NAND Implementation



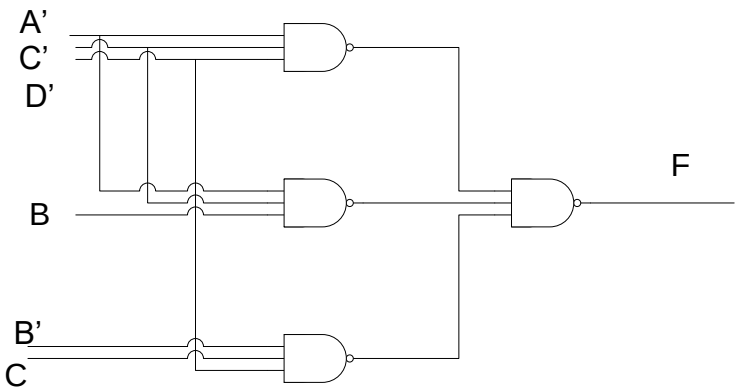
c. F = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + AB'CD'

- K-Map:



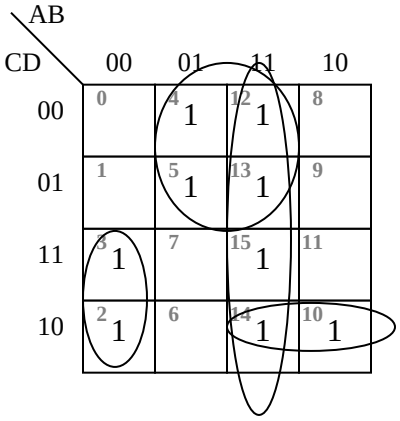
F_{SOP} = A'C'D' + A'BC' + B'CD'

- NAND – NAND Implementation



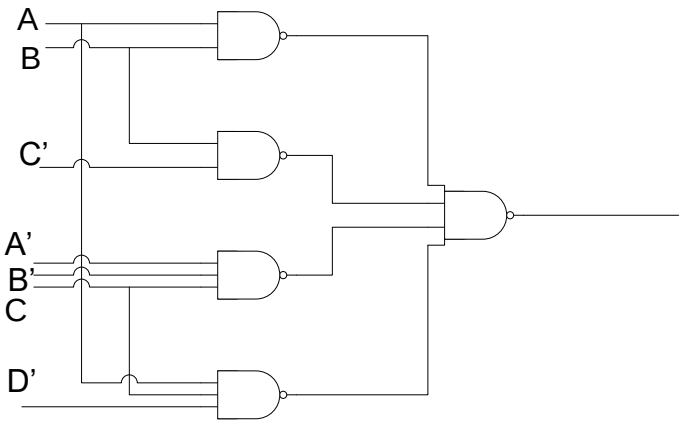
d. The numbers are: 2, 3, 4, 5, 10, 12, 13, 14, 15.

- K-Map:



SOP = AB + BC’ + A’B’C + ACD’

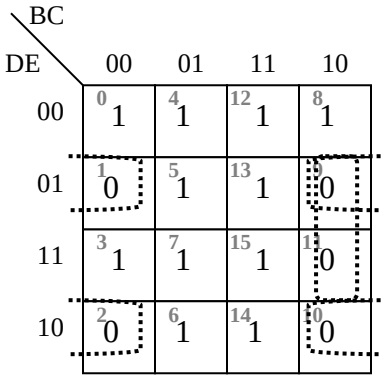
- NAND – NAND Implementation



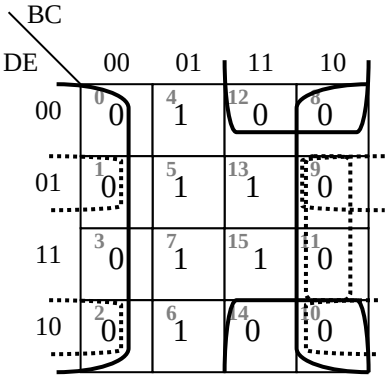
6.

a. F = A’C + ACE + CAB’ + B’A’DE + A’ D’ E’

- K-Map:



A=0

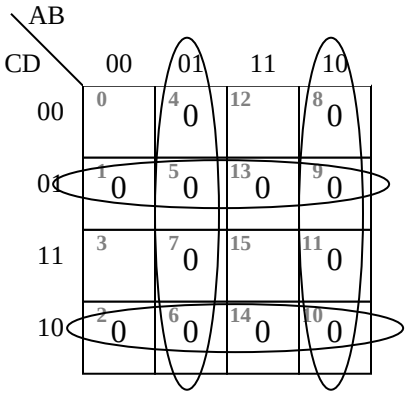


A=1

$F_{POS} = (A' + C')(A' + B' + E)(B'+C+E')(C + D + E')(C + D' + E)$

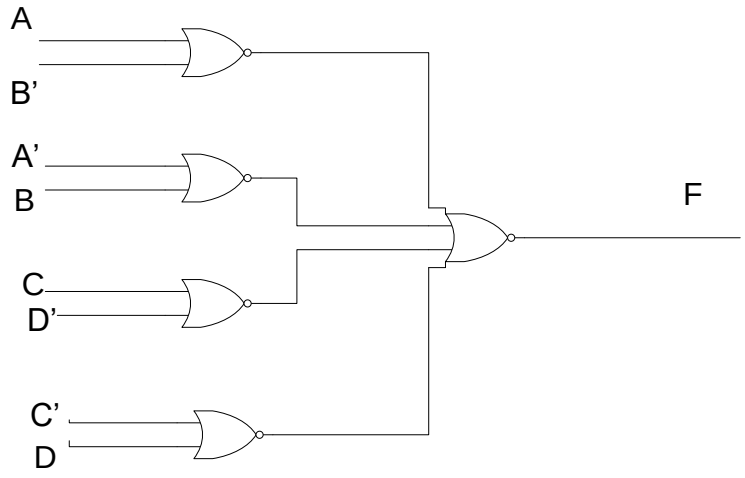
b. F = (AB + A’B’) (C’D’ + CD)

- K-Map:



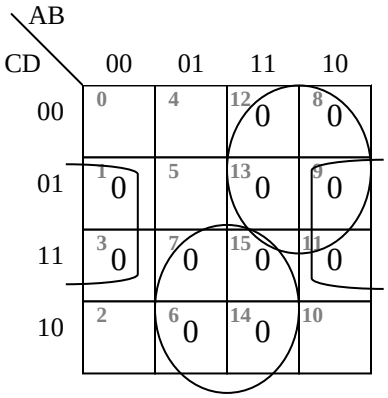
$F_{POS} = (A + B')(A' + B)(C + D')(C' + D)$

NOR – NOR Implementation



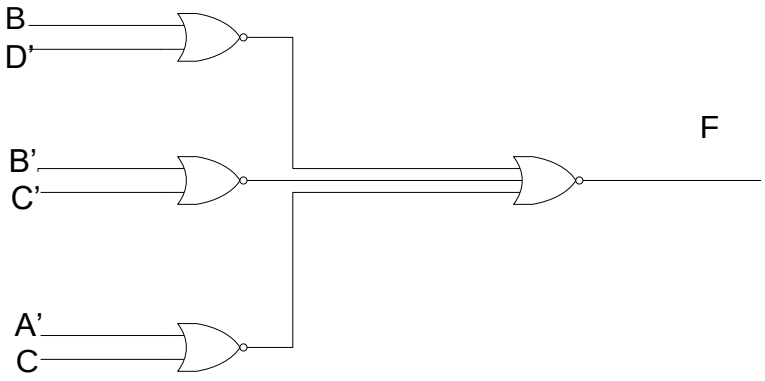
c. F = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + AB'CD'

- K-Map:



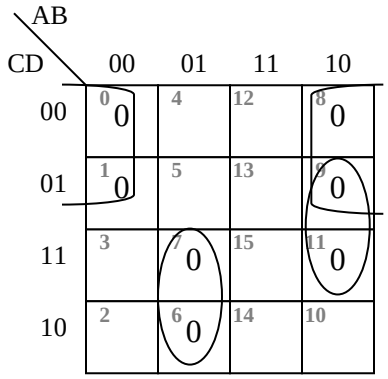
$F_{POS} = (B + D')(B' + C')(A' + C)$

- NOR – NOR Implementation:



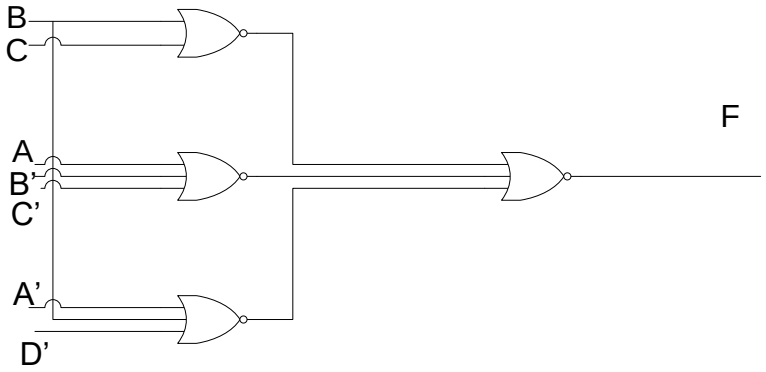
d. The numbers are: 2, 3, 4, 5, 10, 12, 13, 14, 15.

- K-Map:



POS = (B + C)(A + B’ + C’)(A’ + B + D’)

- NOR – NOR Implementation:



7. A and B are both 2-bit signed numbers. They range from -2 to +1. Thus F = A + B is ranging from -4 to +2. We need 3 bits to represent the signed number F.

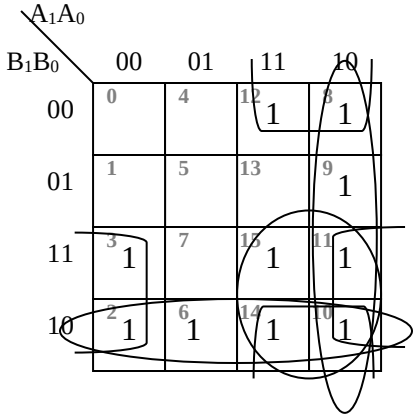
a. The block diagram is below:

b. The truth table is below:

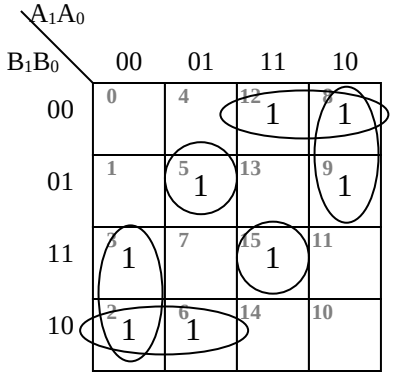
A1	A0	B1	B0	F2	F1	F0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	1	1	0
0	0	1	1	1	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	0
1	0	0	1	1	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

c.

K-Map of output "F_2"



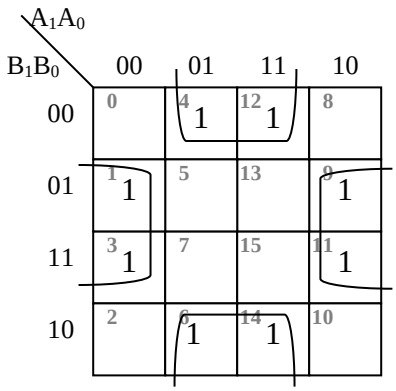
K-Map of output "F_1"



$F2_{SOP} = A_1A_0' + B_1B_0' + A_1B_1 + A_1B_0' + A_0'B_1$

$F1_{SOP} = A_1B_1'B_0' + A_1A_0'B_1' + A_1'A_0'B_1 + A_1'B_1B_0' + A1'A_0B_1'B_0 + A_1A_0B_1B_0$

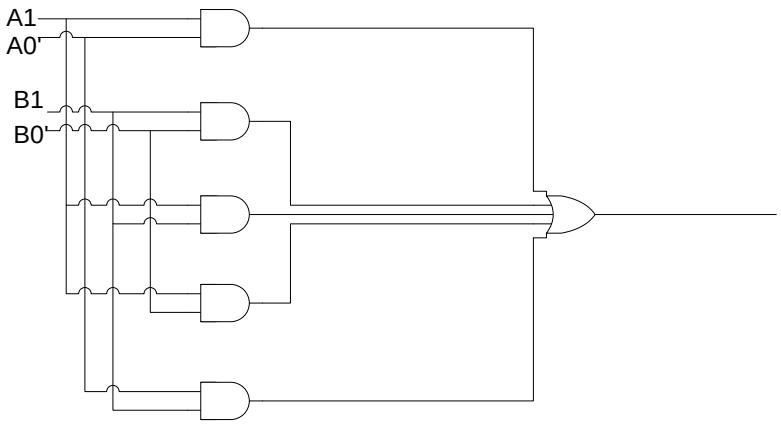
K-Map of output "F_0"



$F0_{SOP} = A_0B_0' + A_0'B_0$

d. AND-OR implementation

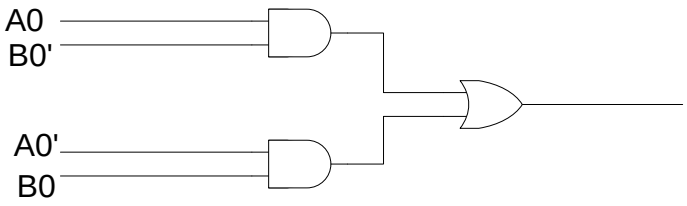
Implementation of F_2



Implementation of F_1

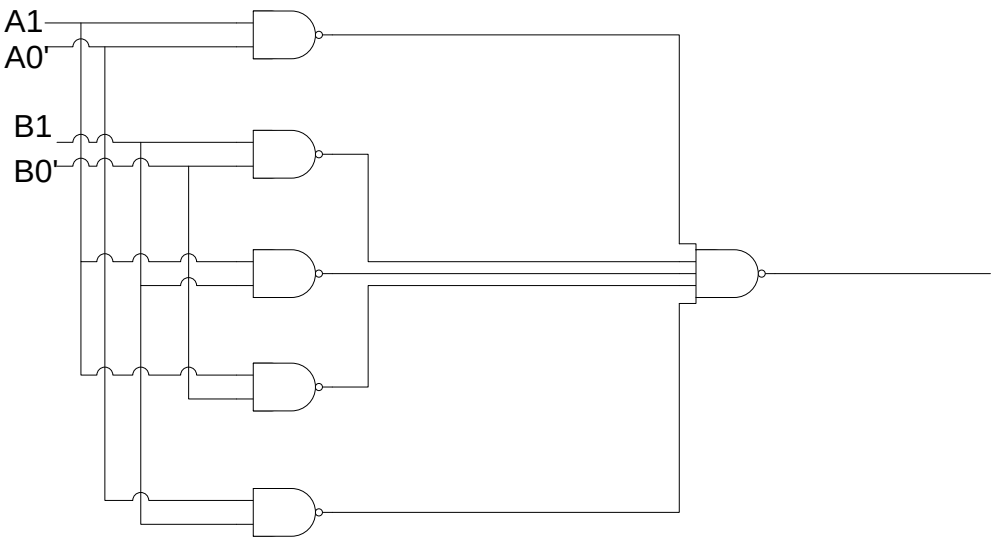
The implementation consists of four 3-input AND gates, two 4-input AND gates and one 6-input OR gate.

Implementation of F_0



NAND – NAND Implementation

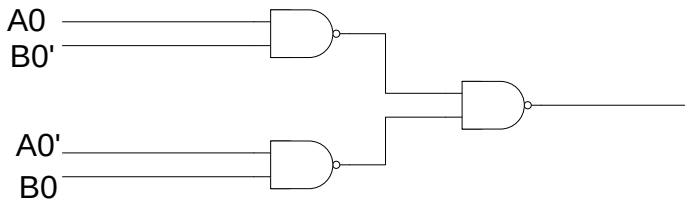
(1) Implementation of F_2



(2) Implementation of F_1

The implementation consists of four 3-input NAND gates, two 4-input NAND gates and one 6-input NAND gate.

(3) Implementation of F_0



8. X is a 3-bit unsigned number. It ranges from 0 to 7. Thus Y = 3X + 1 is ranging from 1 to 22. We need 5 bits to represent the unsigned number Y.

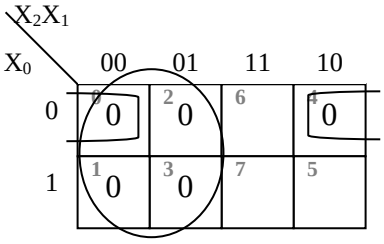
a. The block diagram is below:

b. The truth table is below:

X	X_2	X_1	X_0	Y_4	Y_3	Y_2	Y_1	Y_0	Y
0	0	0	0	0	0	0	0	1	1
1	0	0	1	0	0	1	0	0	4
2	0	1	0	0	0	1	1	1	7
3	0	1	1	0	1	0	1	0	10
4	1	0	0	0	1	1	0	1	13
5	1	0	1	1	0	0	0	0	16
6	1	1	0	1	0	0	1	1	19
7	1	1	1	1	0	1	1	0	22

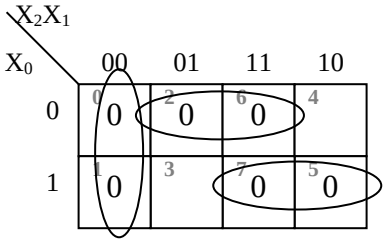
c.

K-Map of output "Y_4"



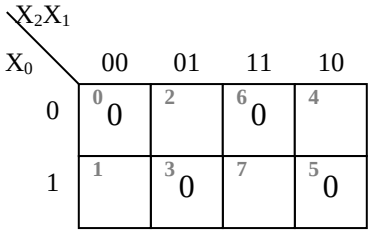
$Y4_{POS} = X_2(X_1 + X_0)$

K-Map of output "Y_3"



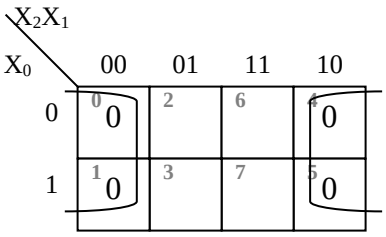
$Y3_{POS} = (X_2 + X_1)(X_1' + X_0)(X_2' + X_0')$

K-Map of output "Y_2"



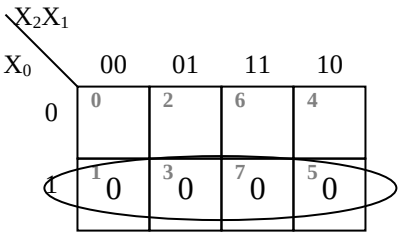
$Y2_{POS} = (X_2 + X_1 + X_0)(X_2 + X_1' + X_0')(X_2' + X_1' + X_0)(X_2' + X_1 + X_0')$

K-Map of output "Y_1"



$Y1_{POS} = X_1$

K-Map of output "Y_0"

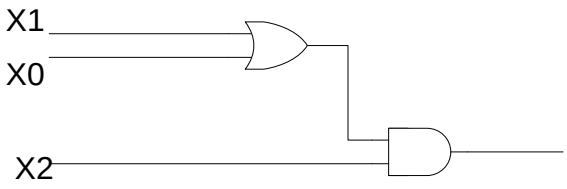


$Y0_{POS} = X_0$'

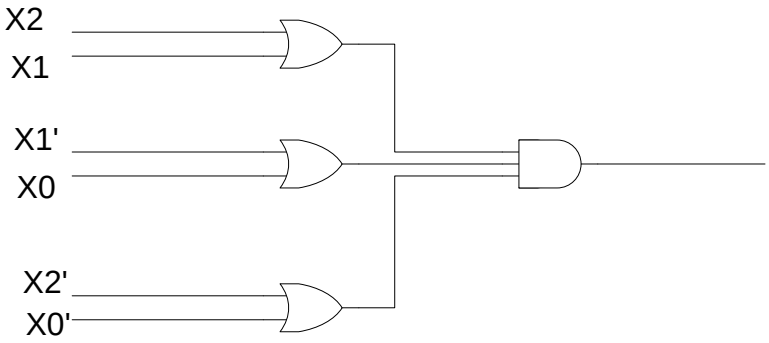
d.

- OR – AND Implementation

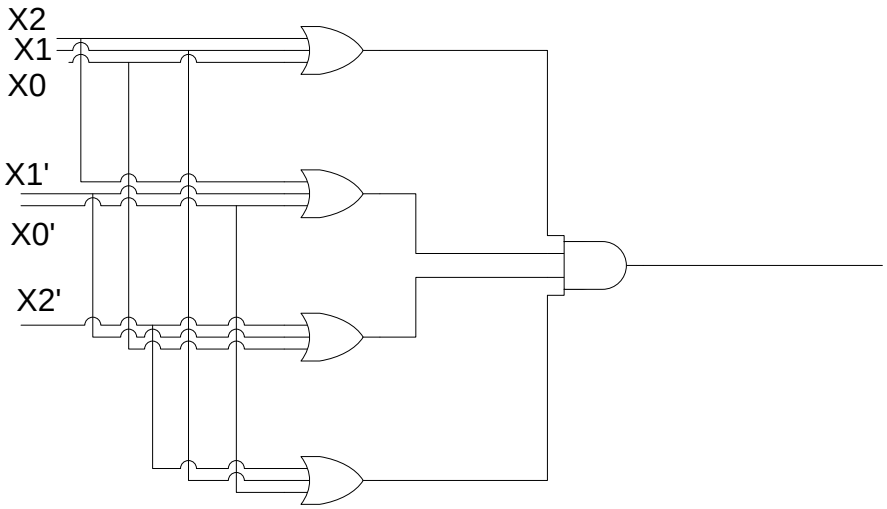
Implementation of Y_4



Implementation of Y_3

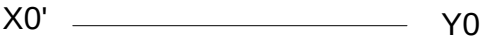


Implementation of Y_2



Implementation of Y_1 (straight wire connection)

Implementation of Y_0 (straight wire connection)



- NOR – NOR Implementation

Convert all gates in the above implementations to NOR gates. For Y1 and Y0 just use wires.

e. This will not be covered at this point of the class since we have not yet discussed adders. Ignore this question.

9. X is a 3-bit signed number. It ranges from -4 to +3.
Thus, $Z = X^2 + 2X + 1 = (X + 1)^2$ is ranging from 0 to +16. We need 6 bits to represent signed number Z.

a. The block diagram is below:

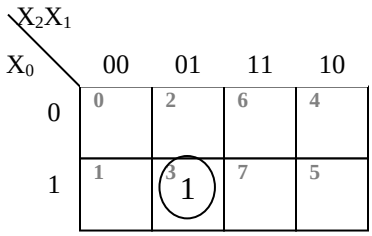
b. The truth table is below:

X	X_2	X_1	X_0	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0	Z
+0	0	0	0	0	0	0	0	0	1	+1
+1	0	0	1	0	0	0	1	0	0	+4
+2	0	1	0	0	0	1	0	0	1	+9
+3	0	1	1	0	1	0	0	0	0	+16
-1	1	1	1	0	0	0	0	0	0	+0
-2	1	1	0	0	0	0	0	0	0	+1
-3	1	0	1	0	0	0	0	0	0	+4
-4	1	0	0	0	0	1	0	0	1	+9

c.

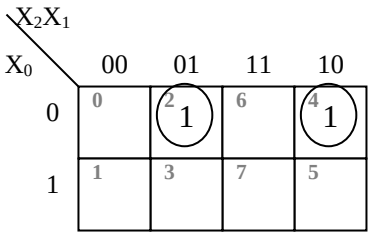
No K-Map needed for Z_5. "Z_5" = 0

K-Map of output "Z_4"



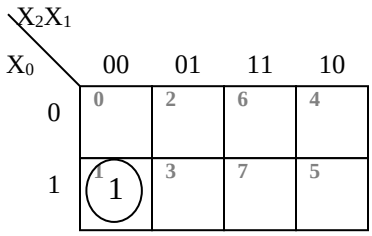
$Z4_{SOP} = X_2'X_1X_0$

K-Map of output "Z_3"



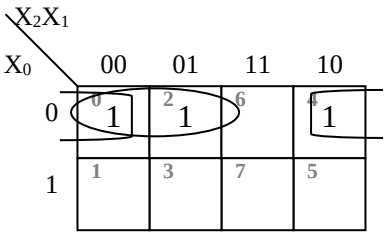
$Z3_{SOP} = X_2'X_1X_0' + X_2X_1'X_0'$

K-Map of output "Z_2"



$Z2_{SOP} = X_2'X_1'X_0$

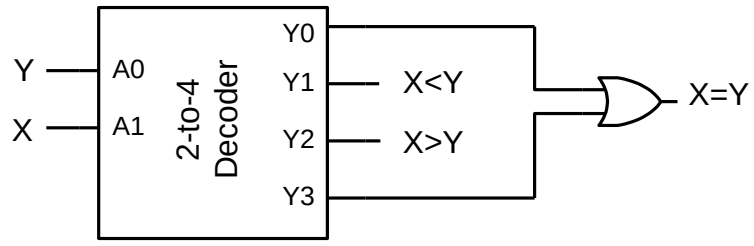
K-Map of output "Z_0"



$Z0_{SOP} = X_2'X_0' + X_1'X_0'$

Note: Z1 = 0 (constant)

10. Design a 1-bit comparator that takes in a bit X and a bit Y and outputs X<Y, X>Y, X=Y. Use a single 2-to-4 decoder and 1 single OR gate.



Consider the truth table of each of these simple functions. We can implement each as a sum of minterms. And since a decoder implements the minterms of the input variables, we can arrive at the circuit above.

X	Y	**X<Y**	**X>Y**	**X==Y**
0	0	**0**	**0**	**1**
0	1	**1**	**0**	**0**
1	0	**0**	**1**	**0**
1	1	**0**	**0**	**1**

HW5 - Signed Representations and Arithmetic

1.

a.

```
          111
          10100 =  20
      +   11101 =  29
Carry=1  110001 =  49
```

b.

```
  11100      28                    1111_
- 10100    - 20        2's comp    11100
  01000      8                     01011
                                 +      1
                                   01000
```

2.

b.

```
    111  1  _
  101.011
+ 011.010
 1000.101
```

3. For unsigned subtraction we still use take the 2's complement of the bottom number (but the check for overflow differs for unsigned vs. 2's complement)

a. $0110110_2 - 1000100_2$ cannot be performed as is because we cannot subtract a larger number from a smaller number in unsigned representation. For unsigned subtraction, after adding the 2's comp, check if the Cout=0, we can know there is overflow

```
  0110110      +54                  0111111_
- 1000100      -68       2's comp   0110110
                                     0111011
                                   +      1
                                     1110010
```

4.

a. Unsigned binary: $2^8 < 283_{10} < 2^9$. Therefore, 9 bits. $283_{10} = 100011011_2$

b. 2's complement: with n-bits we can make $-(2^{n-1})$ to $+(2^{n-1}-1)$. With n = 9, the range is -256 to +256. With n = 10, the range is -512 to +511. Thus 10 bits. $283_{10} = 100011011_2$

5. **Consider the following decimal numbers +21, +55, +121, -32, -99, -128**
   a. What are the corresponding *8-bit signed-magnitude representation*?.
   a. What are the corresponding *8-bit 2's complement representation*?.
   b. Which of the above numbers can be represented in 6-bit signed-magnitude, 6-bit 1's complement, 6-bit 2's complement representations, explain ?

	21	**55**	**121**	**-32**	**-99**	**-128**
A	16+4+1=21 00010101	00110111	01111001	10100000	11100011	NA
B	00010101	00110111	01111001	1110000	10011101	10000000
C	Y,Y,Y 010101 in all cases	N,N,N	N,N,N	N,N,Y 100000 in 2's comp	N,N,N	N,N,N

6. **What are the corresponding decimal representations for the following binary numbers: 01011011 , 11010010, if**
   a. The binary numbers are in *8-bit signed-magnitude format*?
   b. The binary numbers are in *8-bit 2's complement format*?

	01011011	**11010010**
a)	64+16+8+2+1=91	-(64+16+2)=-82
b)	64+16+8+2+1=91	-128+64+16+2=-46

7. **Perform the following addition problems for the following 2's complement numbers. State whether overflow does or does not occur for each problem. Justify your answer for why overflow does or does not occur. Check your work by converting each number to decimal.**

a
```
 1010 0111
+1110 0100
-1000 1011
```
n+n=n
cin=cout=1
No Overflow

b
```
 1001 0110
+1011 0011
-0100 1001
```
n+n=p
cin=0,cout=1
Overflow

c
```
 0101 1100
+1011 0101
-0001 0001
```
p+n=p
cin=cout=1
NoOverflow

d
```
 0101 1101
+0110 1001
 1100 0110
```
p+p=n
cin=1,cout=0
Overflow

8. **This is an exercise to help you remember what gates are used to create a full adder. Using a full adder as shown below and no other gates, can you produce the function Z = A•B (Hint: Write out the logical equations for S and Cout and see if you can hook up the inputs to produce Z).**

ANSWER:

Cin	X	Y	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = X \oplus Y \oplus Cin$$

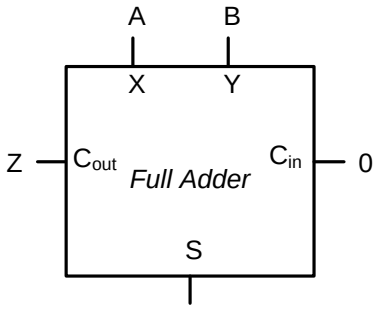
$$Cout = XY + XCin + YCin$$

We want Z=XY, and if we set Cin=0 then the formulas of S and Cout becomes as follows:

$$S = X \oplus Y$$

$$Cout = XY$$

Thus, Cout = Z = XY



9. **Using 1 half-adder and a minimal number of 4-bit binary adders, design a circuit to calculate Y = 25*X, where X is a 4-bit inputs.**

ANSWER:

If a number is multiplied by a number that is a power of 2, then the number rewritten using the above formula:

$$2^n X = X\,\underbrace{0\ldots0}_{n}$$

$$Y = 25*X = 16*X + 8*X + X = X0000+X000+X$$

Also adding the information that X is a 4-bit number above formula can be re-written

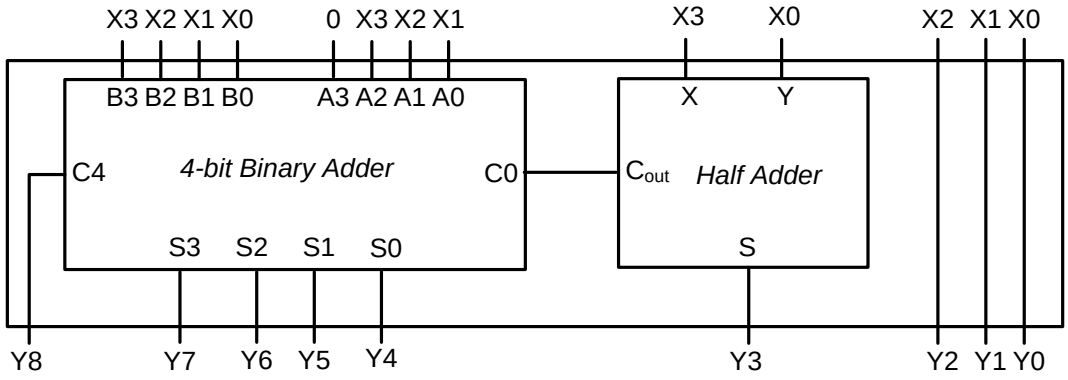
$X = X_3X_2X_1X_0$

$Y = X_3X_2X_1X_00000 + X_3X_2X_1X_0000 + X_3X_2X_1X_0$

This can be written as addition by 4 bit adders:

					X_3	X_2	X_1	X_0
		X_3	X_2	X_1	X_0	0	0	0
+	X_3	X_2	X_1	X_0	0	0	0	0

Since one row in each addition is all zeros, they can be eliminated and the other two numbers can be added using a 4-bit adder. However, the lower bits require only a single bit adder. The carry from the first addition is transferred to the next one.



10. Using a minimal number of 4-bit adders, design a circuit that implements Y=20*X+107, where X is a 3-bit unsigned number.

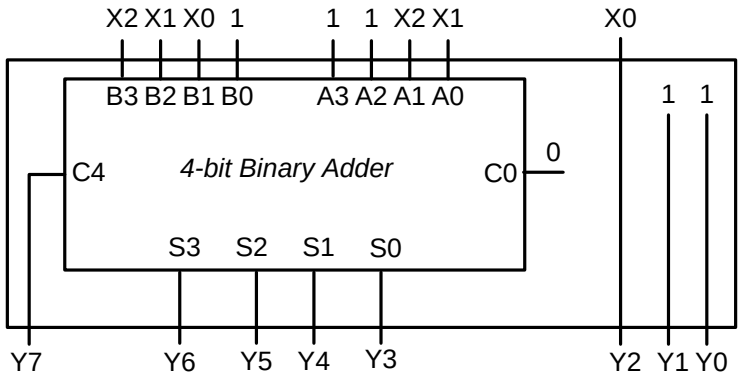
ANSWER:

$X = X_2X_1X_0$, and thus 20*X= 16*X+4*X. These numbers are achieved by inserting zeros at the end. Once we add these there numbers, we see that not all bits require addition. We actually need only a single 4-bit adder and nothing more.

	1	1	0	1	0	1	1
			X_2	X_1	X_0	0	0
+	X_2	X_1	X_0	0	0	0	0

Which can be collapsed to the following addition:

	1	1	X_2	X_1	X_0	0	0
+	X_2	X_1	X_0	1	0	1	1



11. **Design a minimal circuit using AND/OR/NOT gates to implement the comparison A > 10 where A is a 4-bit number.**
    a. Start by writing out the logical algorithm for when A > 10 then implement it using gates
    b. Check that your work is minimal by using a K-Map.

ANSWER:

Let's assume that the number A = $A_3A_2A_1A_0$.

Logical Algorithm: For any number A to be greater than a number B (in this case $10_{10} = 1010_2$), a more significant bit of A must be greater than B. However, in the case of A compared with 1010, we realize A3 and A1 can never be greater than the 1's in those places for 10. Similarly, A2 and A0 can never be less than the 0's in those places. Thus for A > 10, there are only 2 cases that need to be checked:

A3 = 1 and A2 > 0 (i.e. A2 = 1)

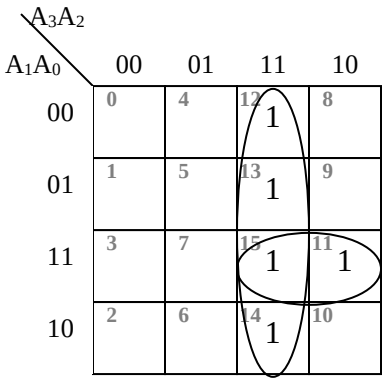
OR

A3=1 and A1 = 1 and A0 > 0 (i.e. A0 = 1).

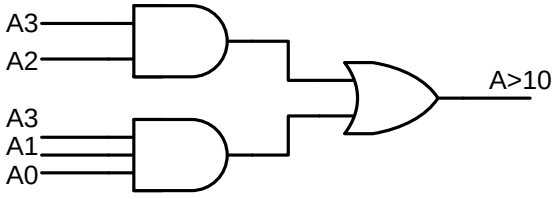
Thus A>10 = $A_3A_2 + A_3A_2A_0$.

Note that we don't have to check A2 in the second case because it is DEFINITELY greater than or equal to the 0 in that place of 1010_2.

A K-Map will also show the same equation
SUM(11,12,13,14,15).

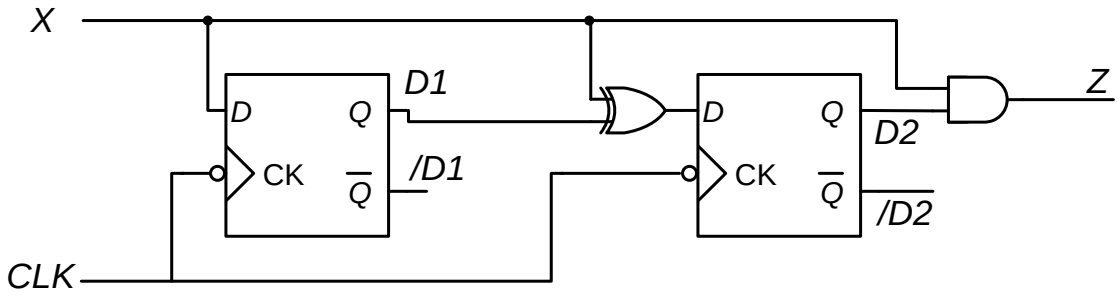


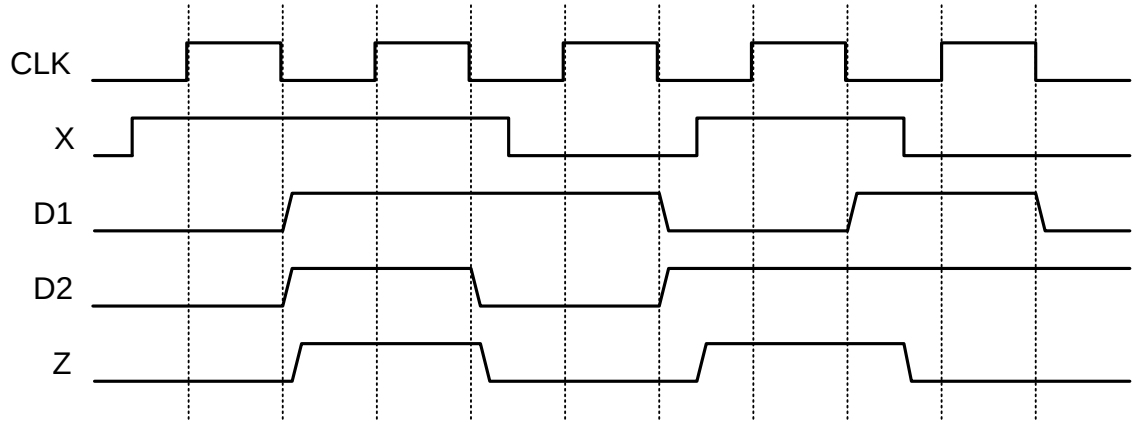
A>10=A_3A_2+ $A_3A_1A_0$



HW 6 – Latches, and Flip-Flops

1. Complete the following waveforms for negative edge-triggered D Flip-Flops.





2. We should find the state diagram following the steps in the class notes
   a. Convert the diagram to a table.

Current State		Next State								Output
		XY= 00		XY= 01		XY= 10		XY= 11		
Q_1	Q_0	Q_1^*	Q_0^*	Q_1^*	Q_0^*	Q_1^*	Q_0^*	Q_1^*	Q_0^*	Z
0	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	1	0	1	1
1	0	0	0	0	1	0	0	0	1	0
1	1	0	0	0	1	0	1	0	1	0

Convert Q* to D

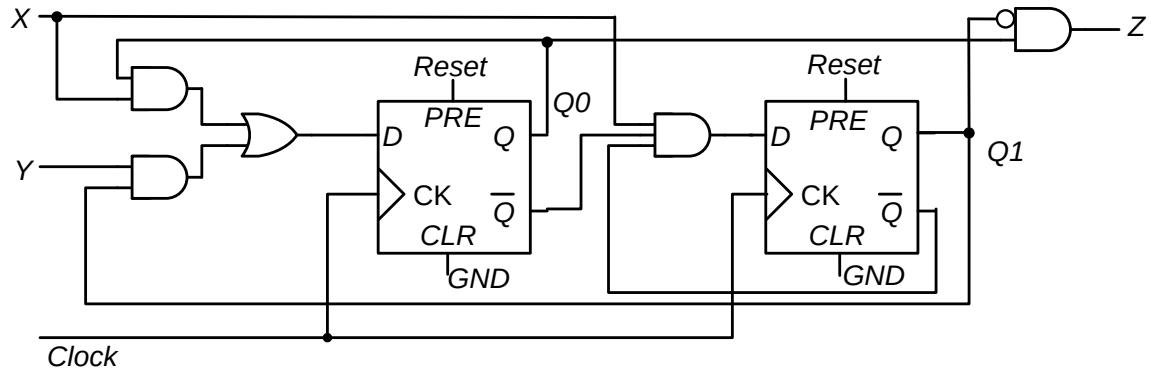
Current State		Next State								Output
		XY= 00		XY= 01		XY= 10		XY= 11		
Q_1	Q_0	D1	D0	D1	D0	D1	D0	D1	D0	Z
0	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	1	0	1	1
1	0	0	0	0	1	0	0	0	1	0
1	1	0	0	0	1	0	1	0	1	0

b. Perform K-maps to find equations for D1 and D0.

$D_0 = Q_1 \cdot Y + Q_0 \cdot X$

$D_1 = Q_1' \cdot Q_0' \cdot X$

c. Draw the circuit and implement the initial state (reset) condition.

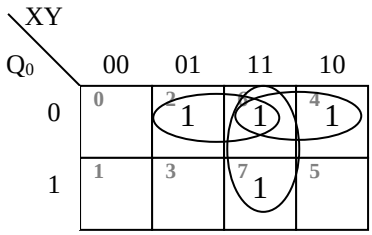


3. Because we have 2 states in the state diagram, we use one D flip-flop to implement state memory. We design the combinational logic for the next-state logic and the output function logic by building up the state/transition table.

Current State	Next State / Output								
	XY = 00		XY = 01		XY = 10		XY = 11		
Symbol/Code	Symbol/Code		Symbol/Code		Symbol/Code		Symbol/Code		H
F / 0	F / 0		G / 1		G / 1		G / 1		0
G / 1	F / 0		F / 0		F / 0		G / 1		1
Q0(t)	Q0* D0		Q0* D0		Q0* D0		Q0* D0		

After we get the state/transition table, we build the K-Maps to simplify expressions for D0, and H.

- D_0 K-Map:



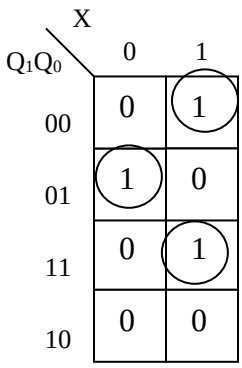
$D_0 = YQ_0' + XQ_0' + XY$ H = Q0

4.

Current State	Next State & Flop-Flop Inputs		Outputs
	Next State	Next State	
	X = 0	X = 1	
Symbol/Code	Symbol/Code	Symbol/Code	Z
A/00	B/01	D/11	0
B/01	C/10	B/01	0
C/10	B/01	A/00	1
D/11	B/01	C/10	0
Q1(t),Q0(t)	$Q1^*(t+1),Q0^*(t+1)$	$Q1^*(t+1),Q0^*(t+1)$	

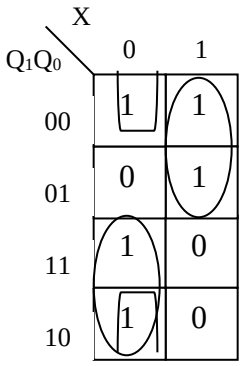
After we get the state/transition table, we build the K-Maps to simplify expressions for D1, D0, and Z.

- D1 K-Map:



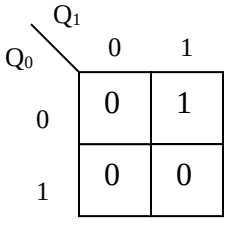
$D_1 = X'Q_1'Q_0 + XQ_1'Q_0' + XQ_1Q_0$

D0 K-Map



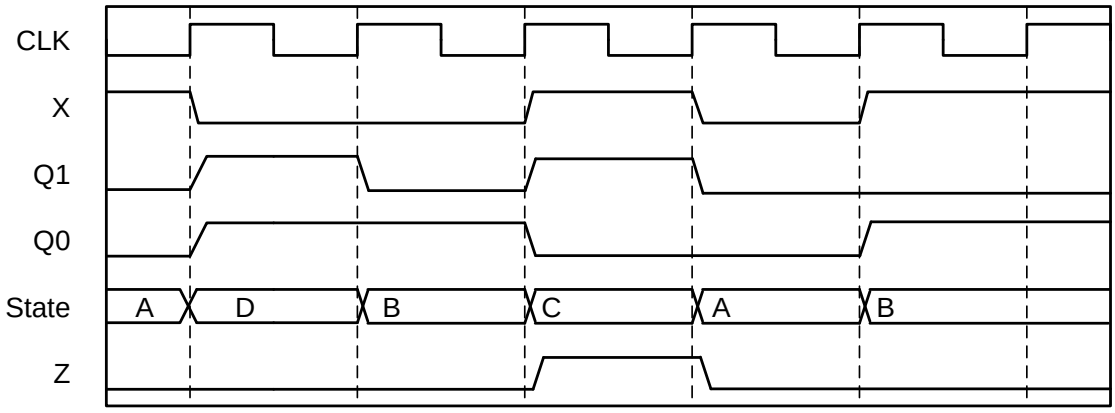
$D_0 = X'Q1+X'Q_0' + XQ_1'$

Z K-Map:



$Z = Q_1Q_0'$

a. The waveform is below:

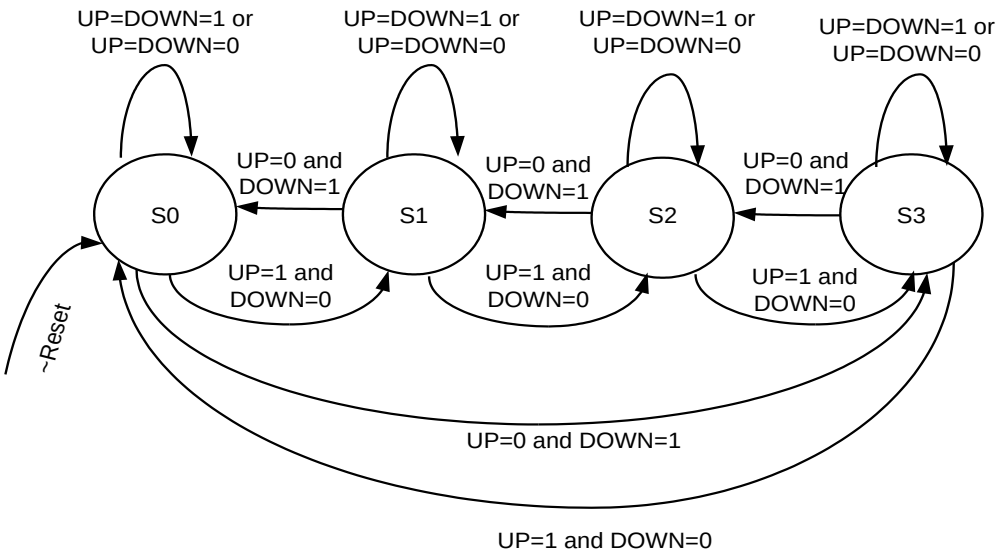


5.

a. We need 4 states and each of them stores one 2-bit binary number ranging from 00 to 11.

State	S0	S1	S2	S3
Assignment (Q_1 Q_0)	00	01	10	11

The state diagram is below:



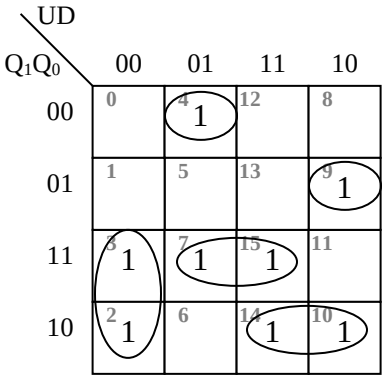
b. Because we have 4 states in the state diagram, we use two D flip-flops to implement state memory. We design the combinational logic for the next-state logic and the output function logic by building up the state/transition table.

U – UP, D – DOWN

Current State	Next State			
	U·D = 00	U·D = 01	U·D = 10	U·D = 11
Symbol/Code	Symbol/Code	Symbol/Code	Symbol/Code	Symbol/Code
S0/00	S0/00	S3/11	S1/01	S0/00
S1/01	S1/01	S0/00	S2/10	S1/01
S2/10	S2/10	S1/01	S3/11	S2/10
S3/11	S3/11	S2/10	S0/00	S3/11
Q1(t),Q0(t)	Q1(t+1),Q0(t+1) D1,D0	Q1(t+1),Q0(t+1) D1,D0	Q1(t+1),Q0(t+1) D1,D0	Q1(t+1),Q0(t+1) D1,D0

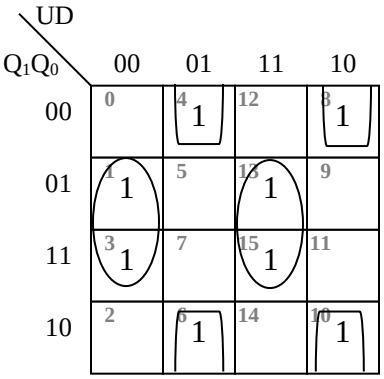
After we get the state/transition table, we build the K-Maps to simplify expressions for D_1, and D_0.

- D_1 K-Map:



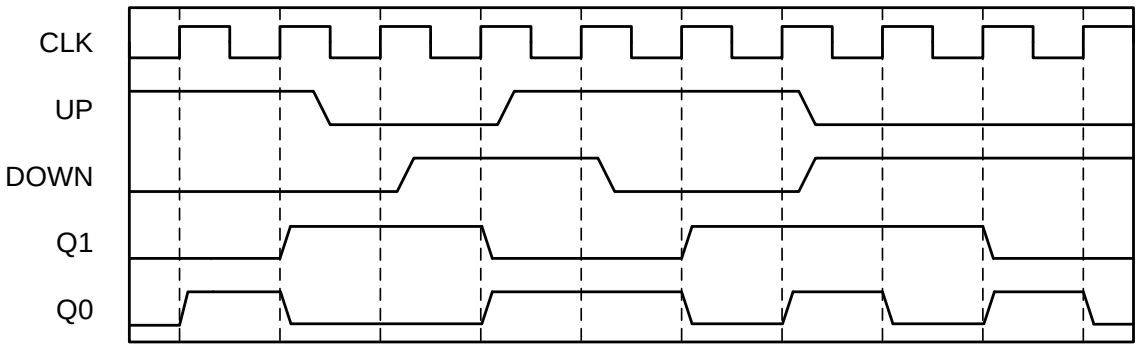
$$D_1 = U'\cdot D'Q_1 + D\cdot Q_1Q_0 + U\cdot Q_1Q_0' + U'\cdot DQ_1'Q_0' + U\cdot D'Q_1'Q_0$$

- D_0 K-Map:

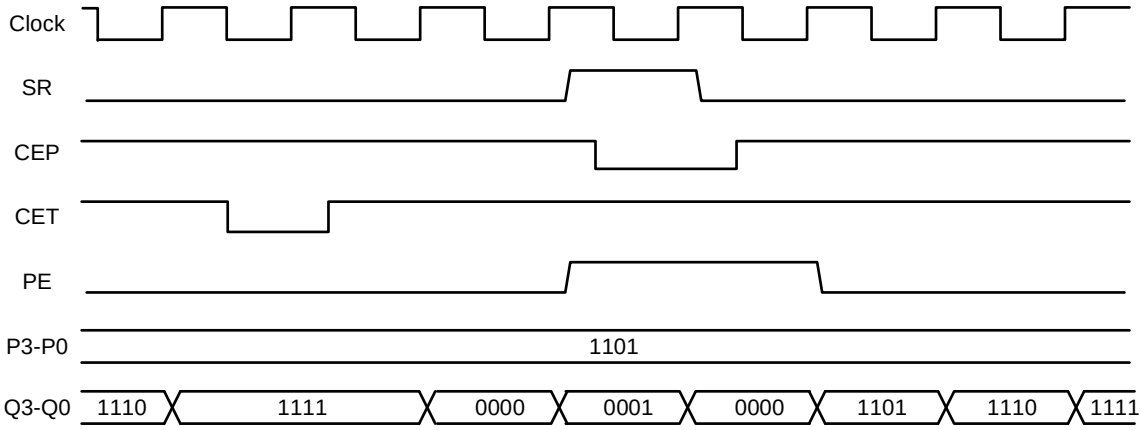


$$D_0 = U'\cdot D'Q_0 + U\cdot DQ_0 + U'\cdot DQ_0' + U\cdot D'Q_0'$$

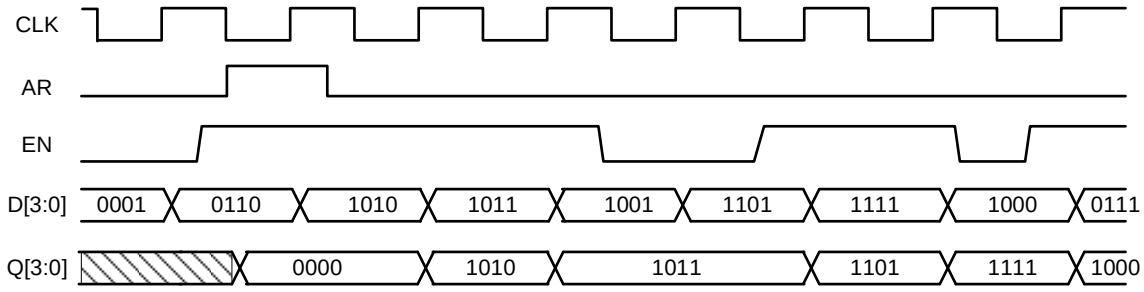
c. The waveform is below:



6. Complete the waveform for the 4-bit counter presented in class.



7. Complete the waveform for a 4-bit D-Register with Data (Load) Enable and asynchronous reset.



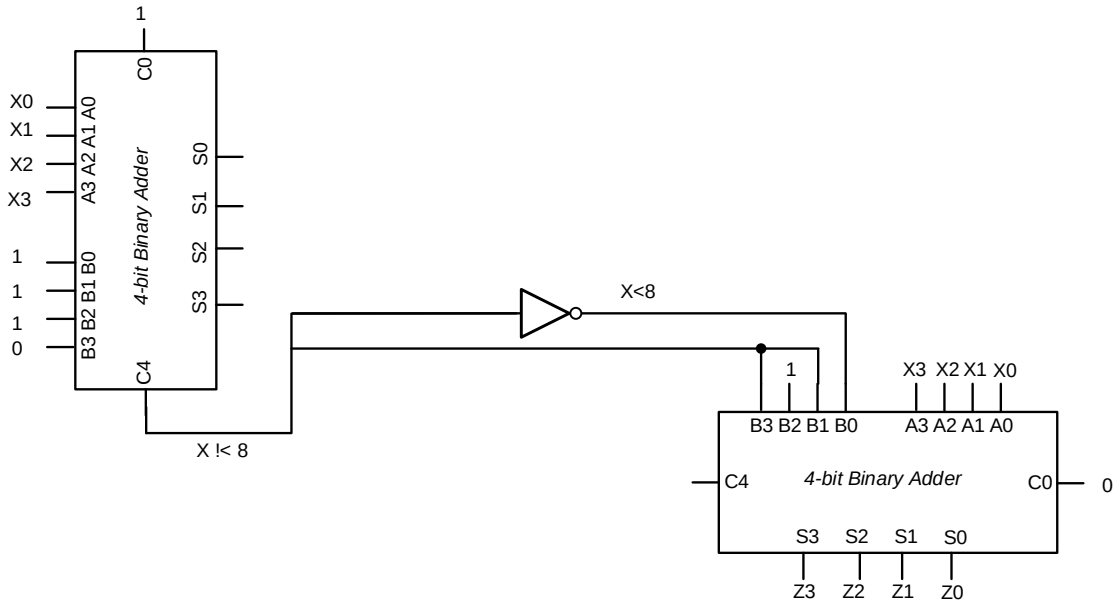
8.1. Unit 7 – Datapath Design

1. Implement a circuit that takes in a 4-bit number X[3:0] and produces a 4-bit value Z[3:0] according to the following function:

   if X < 8 then
   Z = X + 5;
   else
   Z = X – 2;

   Using the building blocks below and at most 1 inverter (no other gates), implement this function.

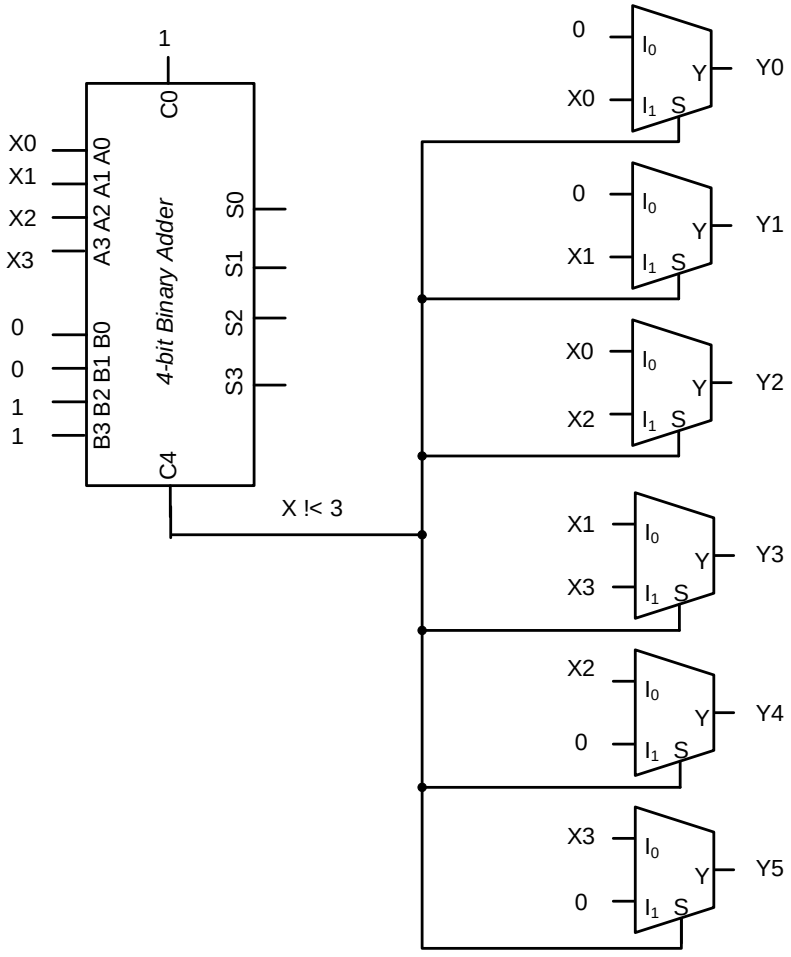
Note: To check X<8, subtract X-8 by taking the 2's complement of 8. The 2's comp. of 8 is 0111+1. Recall, a carry of 0 when subtracting unsigned numbers (A-B) indicates overflow which can only occur in unsigned subtraction if the result needed to be negative (i.e. A < B). So C4 will only be 1 if X is not less-than 8. We can invert it to get a signal that is true when X < 8. Use those signals to either add 5 (0101) or subtract 2 (add 1110).



2. Design a circuit that takes in an 4-bit number X[3:0] and outputs a number Y, and performs the operation below. Assume these numbers are unsigned. Use a 4-bit adder and 2-to-1 muxes. Hint: Multiplying by 4 (i.e. 2^2) in binary can be done with no gates just as multiplying by 100 (i.e. 10^2) in decimal can be done simply and easily.

if($X < 0011_2$)
Y = 4X
else
Y = X

4X would require 6 outputs and be X3 X2 X1 X0 0 0, thus even if we pass X we need to produce 6 outputs. Recall subtracting 3=0011 bin. is the same as adding the 2's complement. Next, a carry of 0 when subtracting unsigned numbers (A-B) indicates overflow which can only occur in unsigned subtraction if the result needed to be negative (i.e. A < B)



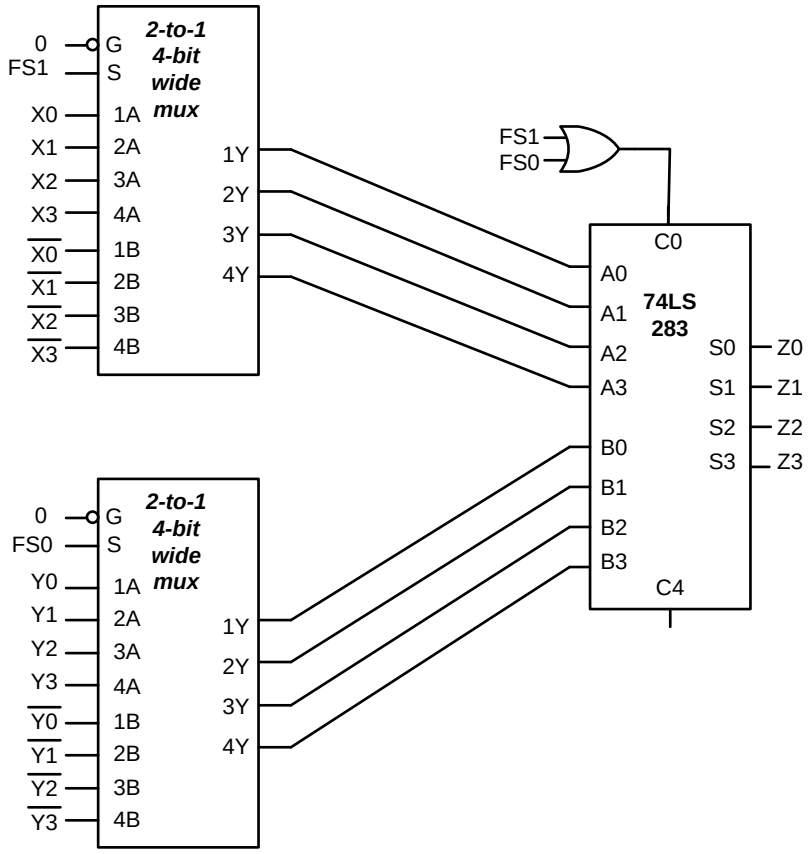
3. Design a circuit that takes in two 4-bit numbers, X[3:0] and Y[3:0] along with two function select bits, FS1 and FS0, and produces an output, Z[3:0], according to the following table:

FS1,FS0	Z
0,0	X+Y
0,1	X-Y
1,0	Y-X
1,1	don't care

Use (1) 4-bit adder, 4-bit wide 2-to-1 muxes, and any basic gates you desire.

Approach: Take a 4-bit adder and for each operation listed in the table above identify what should be passed to the A and B inputs of the adder. Then use muxes to perform that function. Design logic for the select bits of the muxes and the carry-in of the adder that use FS1 and FS0 as input.

Note: In the solution below the A and B inputs of the muxes are like the 0th and 1st inputs of 4 2-to-1 muxes, respectively. Ignore the G input.

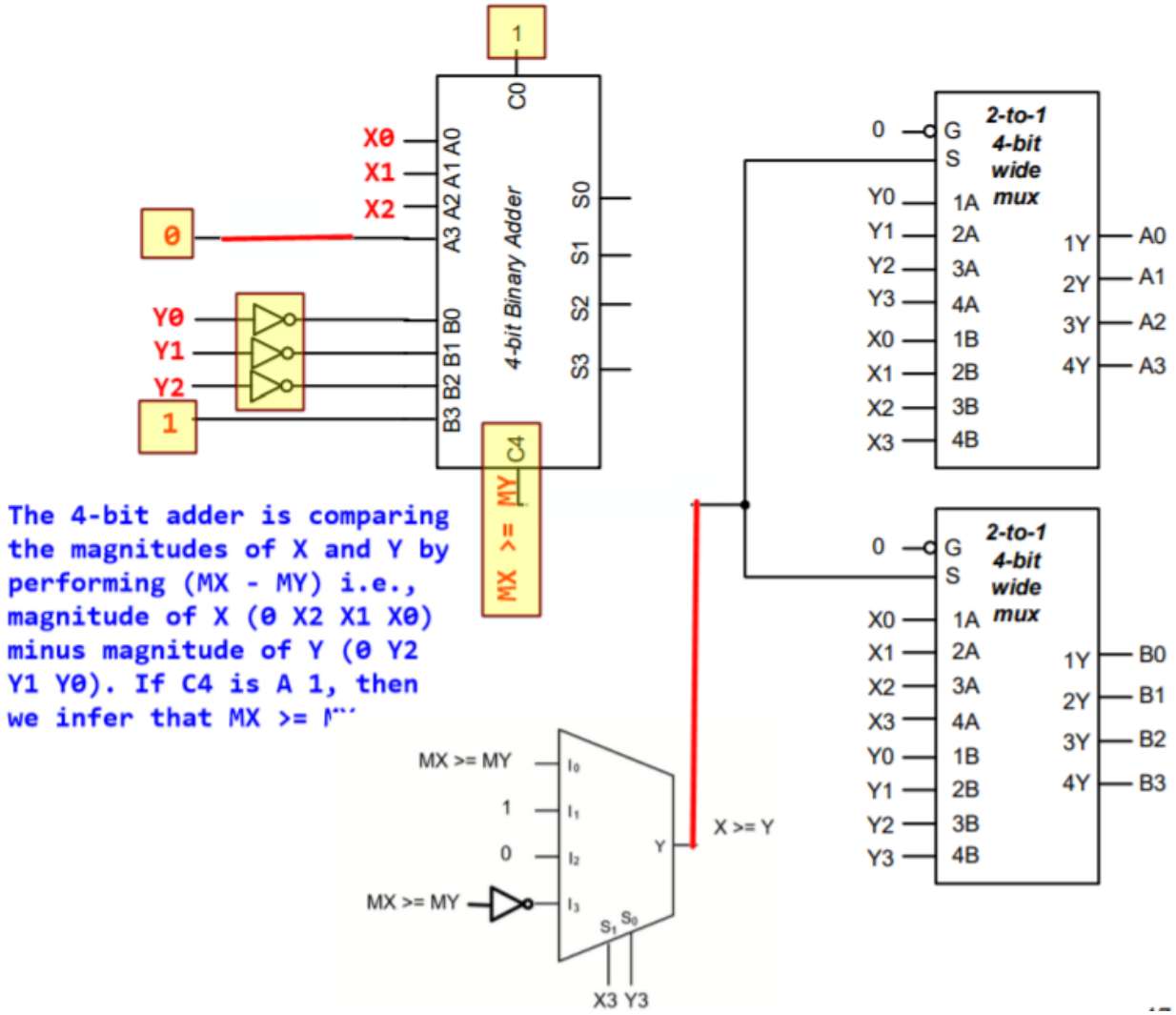


4. Design a circuit that takes in two 4-bit **signed magnitude** numbers, X[3:0] and Y[3:0] and produces two 4-bit outputs, A[3:0] and B[3:0] according to the following algorithm:

if X >= Y then
A = X and B = Y
else
A = Y and B = X

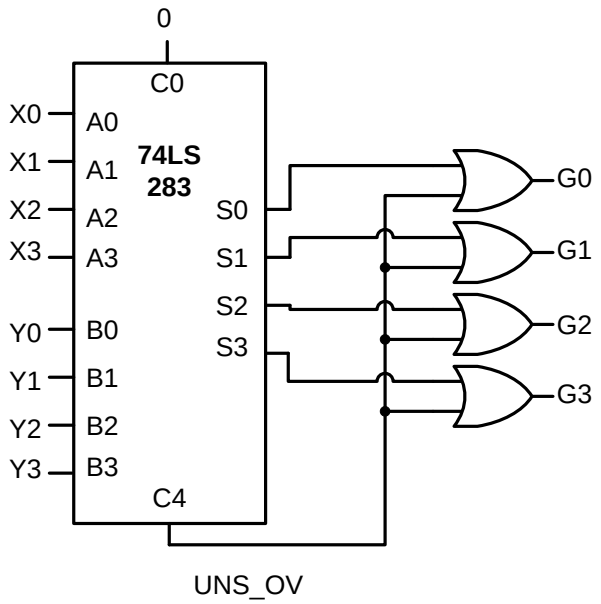
Use a 4-bit adder to subtract and perform the comparison, 4-bit wide 2-to-1 muxes, and basic logic gates. Assume the input 1000 = -0 will never occur on X or Y.

Approach: First think about how to compare signed magnitude numbers by using unsigned comparison techniques and how to precondition the inputs to allow the use of unsigned comparison. For example, 1001 = -1 in signed magnitude and 0100 = +4 in signed magnitude. So 1001 is really <u>less than</u> 0100. Use the comparison result to control the selects of the muxes.



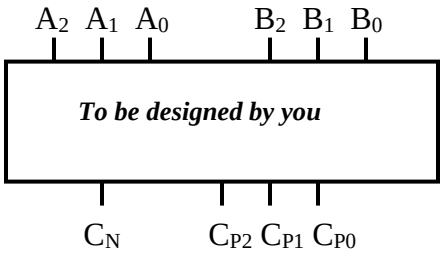
5. Design a circuit that adds two 4-bit, unsigned numbers: X[3:0] and Y[3:0] and outputs either the sum if there is no overflow or 1111 (=15_{10}, the maximum unsigned 4-bit number) if there is overflow.

Unsigned overflow is found by looking at Cout. You could use a mux to select the sum or 1111 but realize you can just use OR gates for this function.

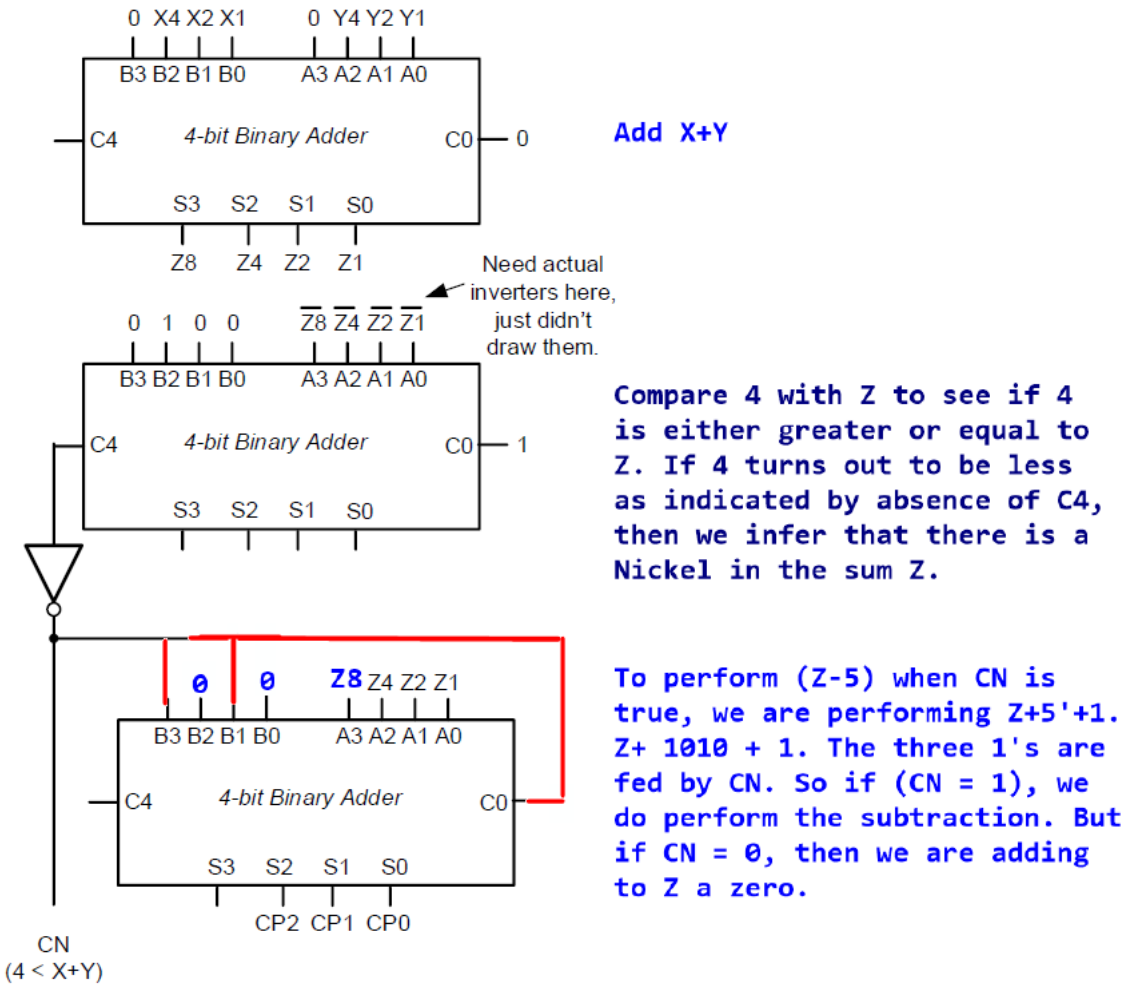


1. We would like to build an adder whose inputs are in units of pennies and produces sum in nickels and pennies. Essentially, this is a base-5 adder. The 2 inputs of this adder are A={$A_2A_1A_0$} and B={$B_2B_1B_0$}. Assume that the maximum input number that A and B can be is 4 pennies. Design an adder to produce the correct nickels/pennies sum, C={ [C_N] ; [$C_{P2}C_{P1}C_{P0}$] }. To implement this design you may assume you can use as many 4-bit adders as you like.

if X+Y > 4 dec. then
    CN = 1, CP[2:0] = X+Y-5
else
    CN = 0, CP[2:0] = X+Y

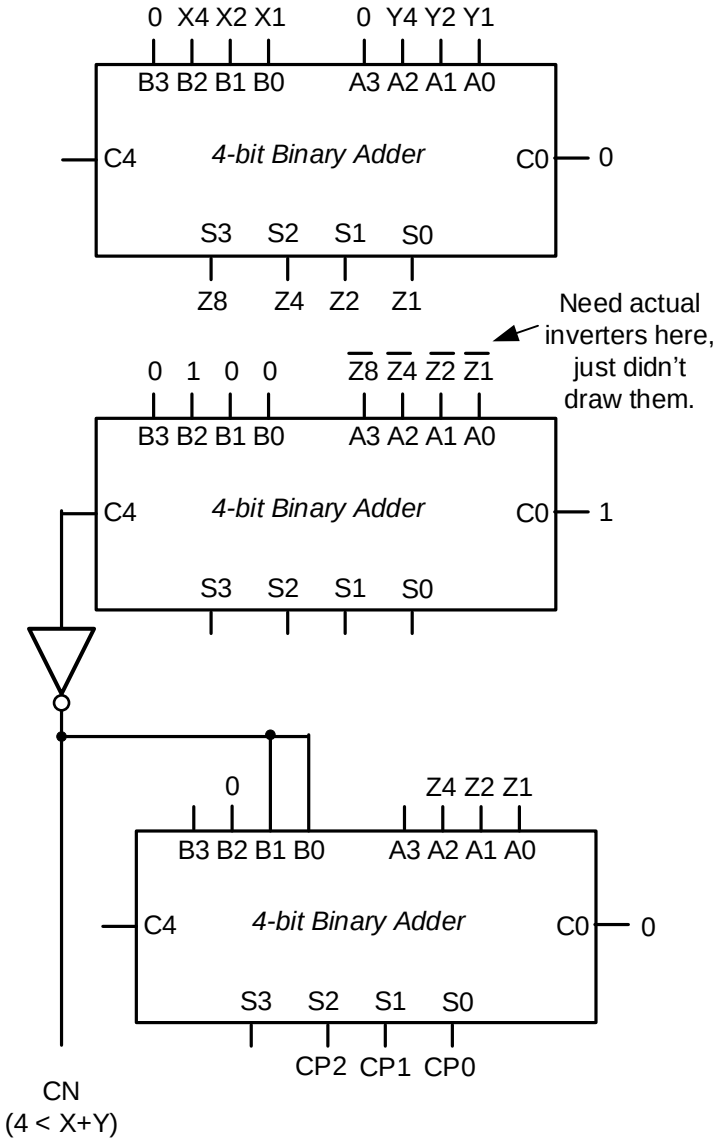


Block diagram for the nickel/penny adder you are going to design

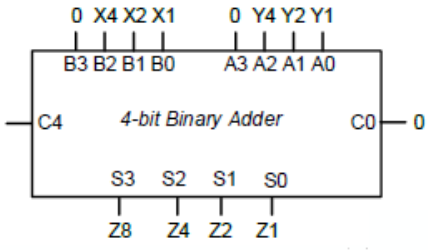


See below for some alternate implementations and simplifications.

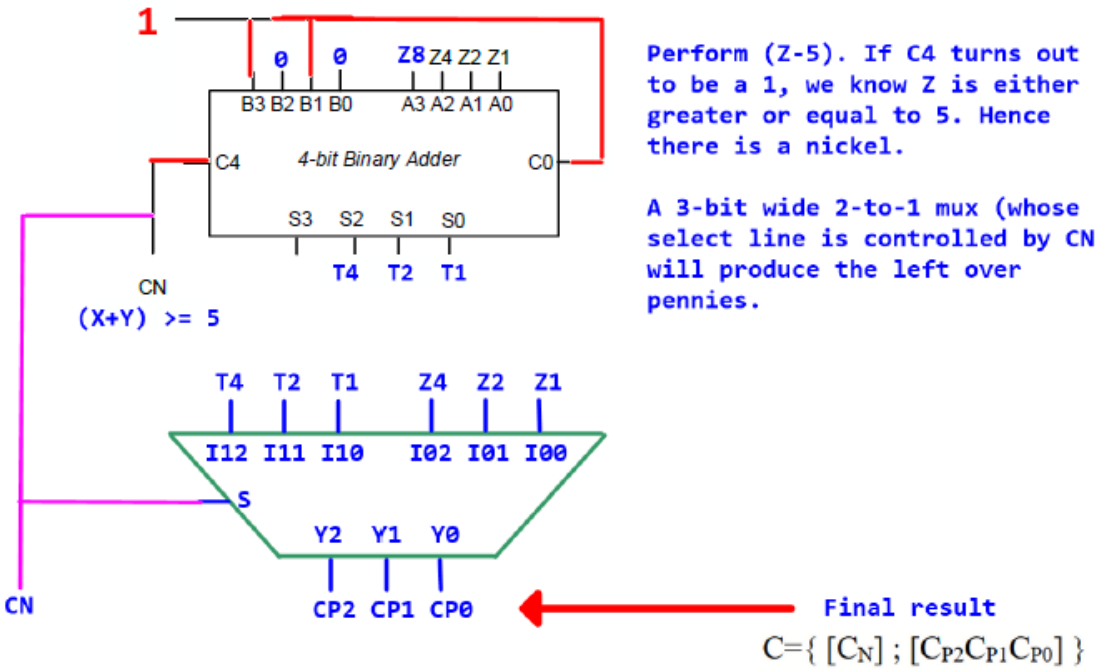
Now, noting that the final 4-bit adder really only produces a 3-bit output, we can substitute and use a 3-bit adder in its place…OR…simplify the 4-bit adder shown. Since the 4-bit adder does not utilize the S3 nor C4 output, we can simply and not connect A3, B3 since they will only affect S3 and C4 to arrive at the design below (which uses B0 rather than C0 for the +1 when taking the 2's complement.



Another approach using a mux rather than a 3rd adder is also shown (and also compares (X+Y) >= 5 rather than 4 < (X+Y)



Add X+Y



Perform (Z-5). If C4 turns out to be a 1, we know Z is either greater or equal to 5. Hence there is a nickel.

A 3-bit wide 2-to-1 mux (whose select line is controlled by CN will produce the left over pennies.