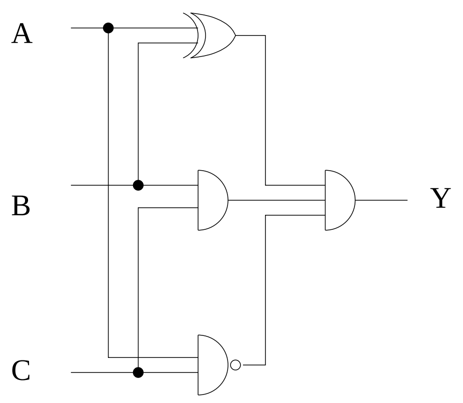
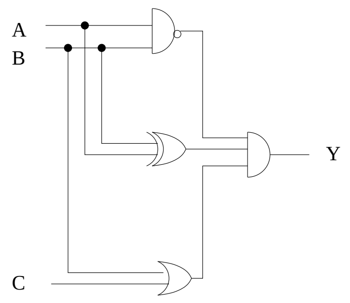
Questions

1. How many boolean functions can be defined on n input variables?
2. Consider the function:
$$Y = (A \cdot B) + \overline{(A \cdot C)} \cdot \overline{B}$$
   (a) Draw a combinational logic circuit that implements this function.
   (b) Draw a truth table for this function.
   (c) Write a sum-of-products representation of Y.
   (d) Write a product-of-sums representation of Y.
3. Write
$$Y = A \ + \ (\overline{B} \cdot C)$$
   (a) as a sum-of-products where each product depends on all three variables A, B, C.
   (b) as a product-of-sums where each sum depends on all three variables A, B, C.
4. Write a sum-of-products representation of the following circuit.



5. Write
$$Y = \overline{(A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + \overline{C})}$$
as a sum-of-products.
6. Write a sum-of-products representation of the output Y of the following circuit, in terms of the input variables A,B,C. The four gates below are NAND, XOR, OR, and AND.

   Explain how this circuit could be considered as a "read only memory".

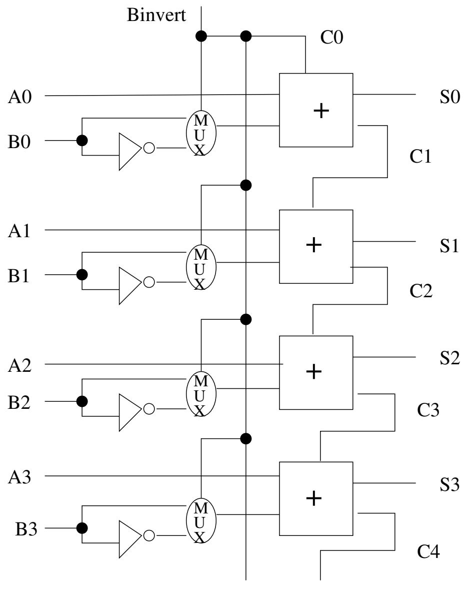


7. Draw a combinational logic circuit of an eight input multiplexor where the inputs

$$(D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0)$$

are each one bit variables. Label the values of inputs and outputs of each gate of the circuit, assuming the eight inputs have the values $(1, 0, 0, 1, 1, 1, 0, 0)$ respectively, and assuming D_1 is selected. [Do *not* draw the decoder circuit which is part of the multiplexor. Instead, merely label the inputs and outputs of this decoder as part of the multiplexor.]

8. Consider the low order four bits of the adder/subtractor circuit shown below. Label all wires (inputs, outputs) in the circuit by their value (0 or 1), assuming the circuit is computing $A - B$ where $A = 98$ and $B = -25$. [Do not draw the entire circuit. Only consider the four lowest order bits for this question.]



9. For the circuit in the previous question, give the values of (S3, S2, S1, S0) in the case that:

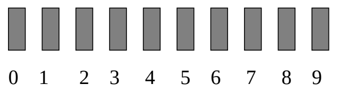
(a) $A = -25, \quad B = -39, \quad \text{Binvert} = 1$

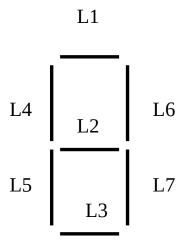
(b) $A = 25, \quad B = 39, \quad \text{Binvert} = 0$

10. Draw a combinational logic circuit of a four input multiplexor where the inputs (X_3, X_2, X_1, X_0) are each one bit variables. Label the values of inputs and outputs of each gate of the circuit, assuming the four inputs have the values $(X_3, X_2, X_1, X_0) = (1, 0, 0, 1)$ and assuming X_1 is selected. [Do *not* draw the decoder circuit which is part of the multiplexor.]

11. An example of an encoder that you should be familiar with is an output device (an LCD) that displays a digit from $\{0, 1, ..., 8, 9\}$. The digit is defined by "turning on" a subset of the following line segments. For example, when all seven of the line segments are on, the output digit is 8, whereas when all lines except L2 are turned on, the output digit is 0.

    Suppose that a user chooses a digit by pressing one of ten buttons. Construct a truth table from which you could build a combinational circuit. (Don't build the circuit. Just build the truth table.)





12. If we add two signed numbers using our n-bit adder circuit, then the result might give an error in the following sense: it could happen that we add two positive numbers and the result is a negative number, or we add two negative numbers and the result is a positive number, or we subtract a positive number from a negative number and the result is positive, or we subtract a negative number from a positive number and the result is negative. In all these cases, the error occurs because the adder defines addition on a circle, not on the (infinite) set of integers.

    For example, suppose $n = 8$ and consider adding 64 + 64, i.e. $64 = 01000000_2$ in binary. The result we wish is 128; however, if the result (10000000_2) is treated as signed then it is interpreted as -128 (not 128).

    Such errors are called *overflow*. We would like to detect such errors automatically.

    Define a binary variable $overflow$ which takes the value 1 when an overflow (error) occurs and 0 when no overflow occurs (no error). This overflow variable depends on the four variables *Binvert*, A_{n-1}, B_{n-1} and C_{n-1}.

    Fill in the truth table below, showing how overflow depends on the four variables *Binvert*, A_{n-1}, B_{n-1}, S_{n-1}. (*Binvert* says whether we are doing addition or subtraction.)

    Hint: Consider the cases of addition and subtraction separately.

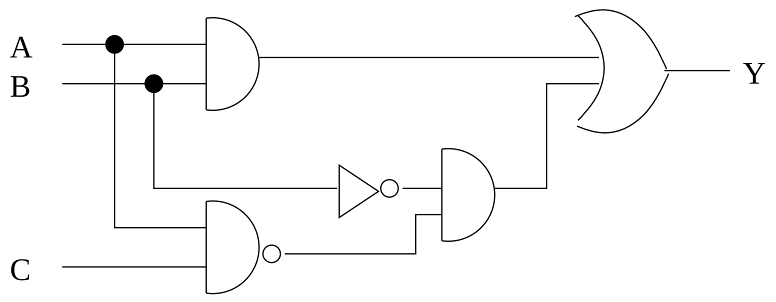
Binvert	A_{n-1}	B_{n-1}	S_{n-1}	$overflow$

Solutions

1. When we have two input variables, the truth table has four rows. A boolean function assigns one boolean value (Y) to each of these four rows. Thus, there are 2^4 possible boolean functions of two input variables.

   If there are n input variables, then the truth table has 2^n rows. We can thus define $2^{(2^n)}$ boolean functions on n variables.

2. (a)



(b)

A	B	C	$A \cdot B$	$A \cdot C$	$\overline{A \cdot C}$	$\overline{B}$	$\overline{(A \cdot C)} \cdot \overline{B}$	$Y = A \cdot B + \overline{(A \cdot C)} \cdot \overline{B}$	$\overline{Y}$
0	0	0	0	0	1	1	1	1	0
0	0	1	0	0	1	1	1	1	0
0	1	0	0	0	1	0	0	0	1
0	1	1	0	0	1	0	0	0	1
1	0	0	0	0	1	1	1	1	0
1	0	1	0	1	0	1	0	0	1
1	1	0	1	0	1	0	0	1	0
1	1	1	1	1	0	0	0	1	0

(c) From the truth table we have:

$$Y = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

(d) Again from the truth table we have:

$$\overline{Y} = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C$$

De Morgan's Laws now yields:

$$\overline{\overline{Y}} = Y = (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + \overline{C})$$

3. This question could be solved by building the truth table for the function $Y = A + (\overline{B} \cdot C)$ and extracting the correct rows to obtain each of the answers:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(a) Taking the rows of the truth table where $Y = 1$ we have that
$Y = (\overline{A} \cdot \overline{B} \cdot C) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$

(b) Taking the rows of the truth table with $Y = 0$ (or $\overline{Y} = 1$) we have that
$\overline{Y} = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot B \cdot C)$
Negating $\overline{Y}$ gives:
$Y = \overline{(\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot B \cdot C)}$
$Y = \overline{(\overline{A} \cdot \overline{B} \cdot \overline{C})} \cdot \overline{(\overline{A} \cdot B \cdot \overline{C})} \cdot \overline{(\overline{A} \cdot B \cdot C)}$
$Y = (A + B + C) \cdot (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C})$

4. Using a truth table and applying the sum-of-products technique gives (details omitted):

$$Y = \overline{A} \cdot B \cdot C$$

5.

$$\begin{aligned} Y &= \overline{(A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + \overline{C})} \\ &= \overline{(A + \overline{B} + \overline{C})} + \overline{(\overline{A} + B + \overline{C})} \\ &= \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C \end{aligned}$$

6.

A	B	C	$\overline{A \cdot B}$	$\oplus(A, B)$	B+C	Y
0	0	0	1	0	0	0
0	0	1	1	0	1	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	1	0	0	1	0

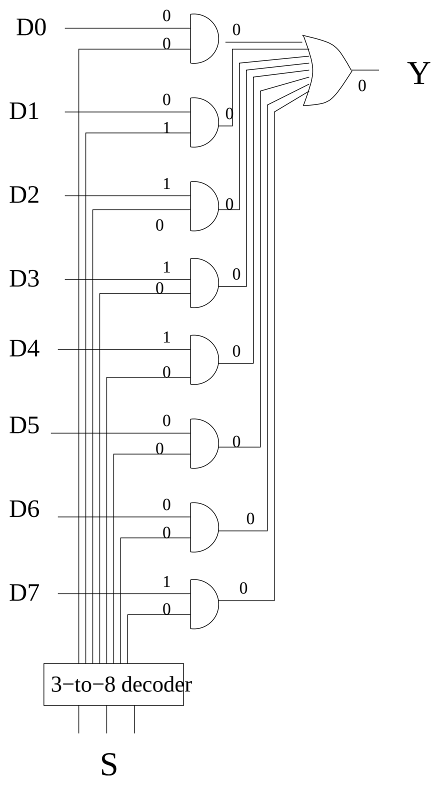
Thus,

$$Y = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C$$

To think of this circuit as a read only memory, the three input variables A, B, C need to encode an address (from 0 to 7) and so we need to specify an ordering of these three input variables. For example, if we associate (A, B, C) with (A_2, A_1, A_0) and treat $A_2A_1A_0$ as a binary number in base 2, then $(A, B, C) = (1, 0, 0)$ would encode the address 4, that is, $(100)_2$ in binary, whereas if we we associated (A, B, C) with (A_0, A_1, A_2) then $(A, B, C) = (1, 0, 0)$ would encode the address 1, that is, $(001)_2$ in binary.

Note that since there is only one output variable Y, only one bit of data is stored at each of the eight addresses in this "read only memory".

7.

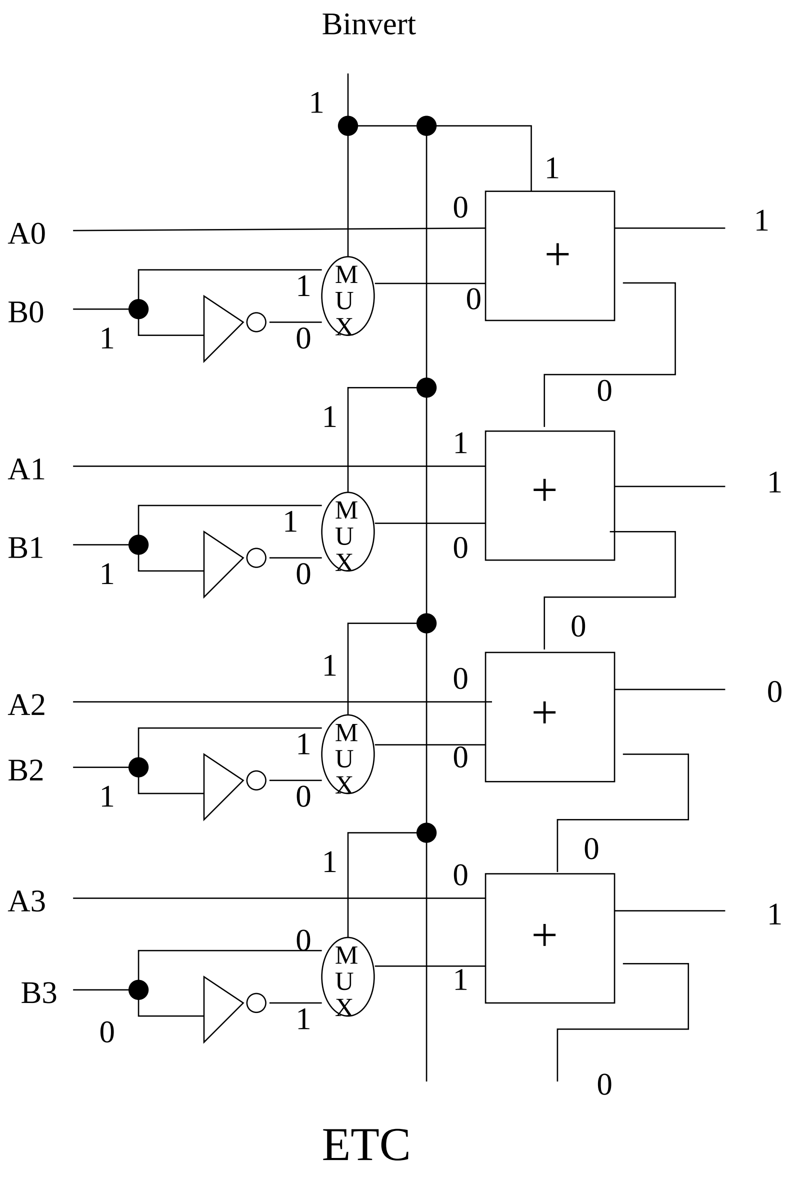


8. $A = 98$ in binary (signed) is $\ldots 0001100010$, so four low order bits of A are 0010

   $B = -25$ in binary (signed) is $\ldots 1111100111$, so four low order bits of A are 0111

   $A - B = 98 - (-25) = 123 = (001111011)_2$, so four low order bits are 1011.

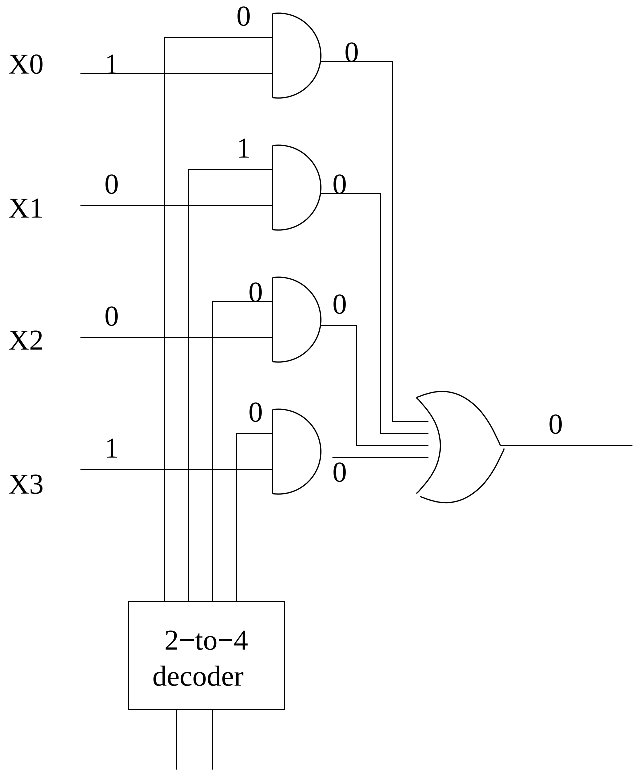
   Verify that this is consistent with the 0's and 1's on the wires (which you can compute indpendently by examining the sum and carry bits).



9. (a) Since Binvert = 1, we are performing subtraction: A - B = $-25 - (-39) = 39 - 25 = 14$. So the answer is 14 written in binary which is 00001110. The lower four bits are what we want, i.e. (S3,S2,S1,S0) = (1,1,1,0).

   (b) Since Binvert = 0, we are performing addition: A + B = $25 + 39 = 64$. In binary this is 01000000. The lower four bits are what we want, i.e. (S3,S2,S1,S0) = (0,0,0,0).

10.



11. Each of the 10 digits has a 7 bit code, namely a subset of 7 line segments that is "on".

0	1	2	3	4	5	6	7	8	9	L1	L2	L3	L4	L5	L6	L7
1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0
0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1	1
0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	1
0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	1
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1

The above truth table only considers combinations of input values such that exactly one of the inputs is 1. We could expand the truth table by adding another 1014 (that is, $2^{10} - 10$) rows to each of the 17 columns, and filling the right side ($L1 - L7$) of these new rows with 0's. If we then built a circuit based on this truth table, then the circuit would produce an output digit if and only if exactly one of the ten input variables were on. (If zero or more than input variables was 1, then it wouldn't light up any of the segments.)

From the above truth table, we could build a combinational logic circuit having 10 inputs and 7 outputs. For example, we could construct a sum-of-products expression for each of the line segments. L1 and L2 would be the sum of eight product terms, L3 and L4 would be the sum of seven product terms, etc. Each of these product terms has 10 variables.

12. First consider *addition* ($Binvert = 0$), namely $A + B$. There are several cases:

- if the two numbers are of opposite sign, then there can be no overflow (convince yourself).
- if the two numbers are positive ($A_{n-1} = 0,\ B_{n-1} = 0$) then there is overflow if and only if $S_{n-1} = 1$ which indicates a negative result.
- if the two numbers are negative ($A_{n-1} = 1,\ B_{n-1} = 1$), then there is overflow if and only if $S_{n-1} = 0$, i.e. a non-negative result.

Next consider *subtraction* ($Binvert = 1$), namely $A - B$.

We use similar reasoning as above:

- if A and B are of the same sign, then there is no overflow
- if $A > 0$ ($A_{n-1} = 0$) and $B < 0$ ($B_{n-1} = 1$), then there is overflow if and only if $S_{n-1} = 1$ (a negative result)
- if $A < 0$ and $B > 0$, then there is overflow if and only if $S_{n-1} = 0$ (a positive result).

$Binvert$	A_{n-1}	B_{n-1}	S_{n-1}	Overflow
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0