Logic Gates

CS 350: Computer Organization & Assembler Language Programming

[4/23: Act #3a solution: change OR gate to NOR gate]

A. Why?

- Logic gates are the lowest level of hardware that deal with logical values.

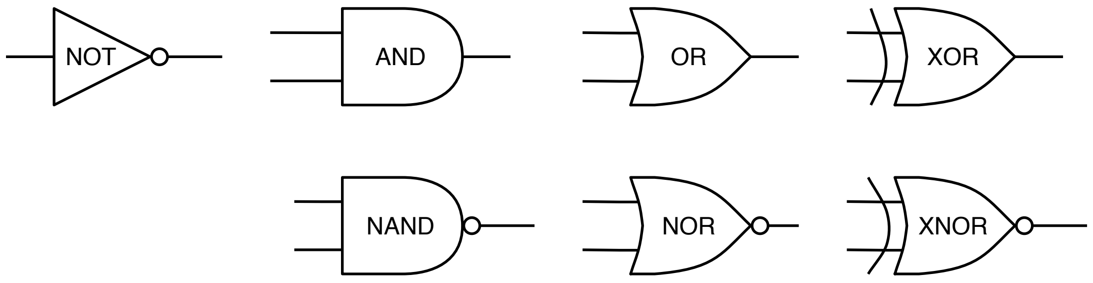
B. Outcomes

After this lecture, you should

- Know the symbols for the logic gates.
- Be able to trace the execution of a simple logic circuit.
- Be able to convert between truth tables, logical formulas, and loop-free logic circuits.

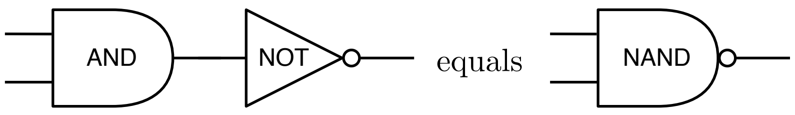
C. Logic Gates

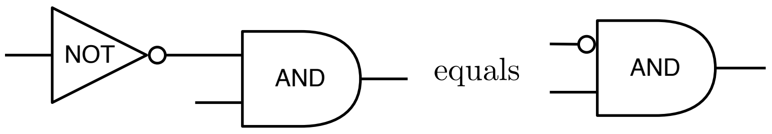
- A logic gate is a device that performs a logical calculation by taking logical inputs and producing logical output(s).
- (We're interested in electronic logic gates.)
- Symbols for gates



- Note little circle indicates negation.
    - Binary *XNOR* ("eks-nore") is logical equivalence (i.e., iff).
- Can extend binary gates to n-ary gates (n inputs, 1 output).
    - E.g., *AND(X, Y, Z, U)* = *((X AND Y) AND Z) AND U)*.
    - *OR(X, Y, Z, U)* = *((X OR Y) OR Z) OR U)*.

        - Since AND and OR are associative, parenthesization isn't important.
    - n-way *NAND/NOR/XNOR* are the *NOT* of n-way *AND/OR/XOR*
    - n-way *XOR* yields true iff an odd number of inputs are true.
    - n-way *XNOR* yields true iff an even number of inputs are true.
- Adding a circle to an output or input has the same effect as a *NOT* gate:

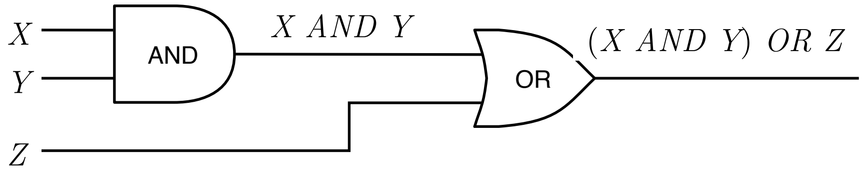




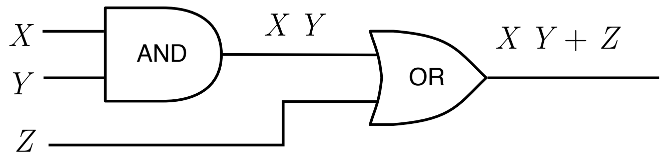
- It's straightforward to combine gates to calculate simple boolean expressions.
- Note the time it takes for a circuit to stabilize depends on the length of the longest path through the circuit. (The shorter the longest path, the faster the circuit, in general.)

D. Translating From a Simple Logic Circuit To a Propositional Formulas

- A simple logic circuit diagram can be translated to an equivalent propositional formula.
    - It must be a **combinatorial circuit**: It must have no loop.
- Start with the gate(s) that connect directly to the circuit's inputs and label the output of each gate with the proposition for that gate and its inputs.
    - Continue finding gates that have two labeled inputs and labeling their outputs. Here's an example:



or



E. Translating From a Truth Table to a Propositional Formula (review)

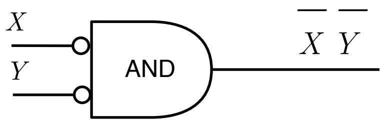
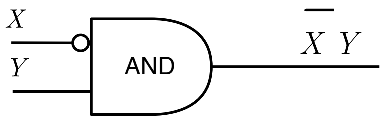
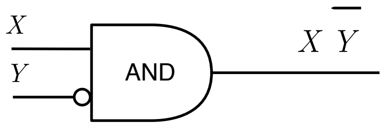
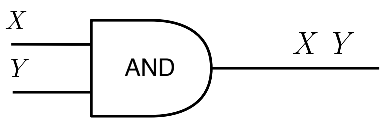
- Recall the technique we've been using to convert a truth table with m parameters to a corresponding proposition by using disjunctive normal form (the disjunction of conjunctions of literals).
- Each of the 2^m rows corresponds to the *AND* of the m literals. (E.g. $X = 0$ and $Y = 0$ corresponds to $\overline{X}\ \overline{Y}$.)
- We find the rows that have output 1 and *OR* the propositions for those rows.
- (In the trivial cases, all the outputs are 0 or 1 and we can use 0 or 1 as the expression.)
- **Example**: $\overline{X}\,\overline{Y} + X\ \overline{Y}$ describes the truth table below

X	***Y***	**Proposition**	**Out**
0	0	$\overline{X}\ \overline{Y}$	1
0	1	$\overline{X}\ Y$	0
1	0	$X\ \overline{Y}$	1
1	1	$X\,Y$	0

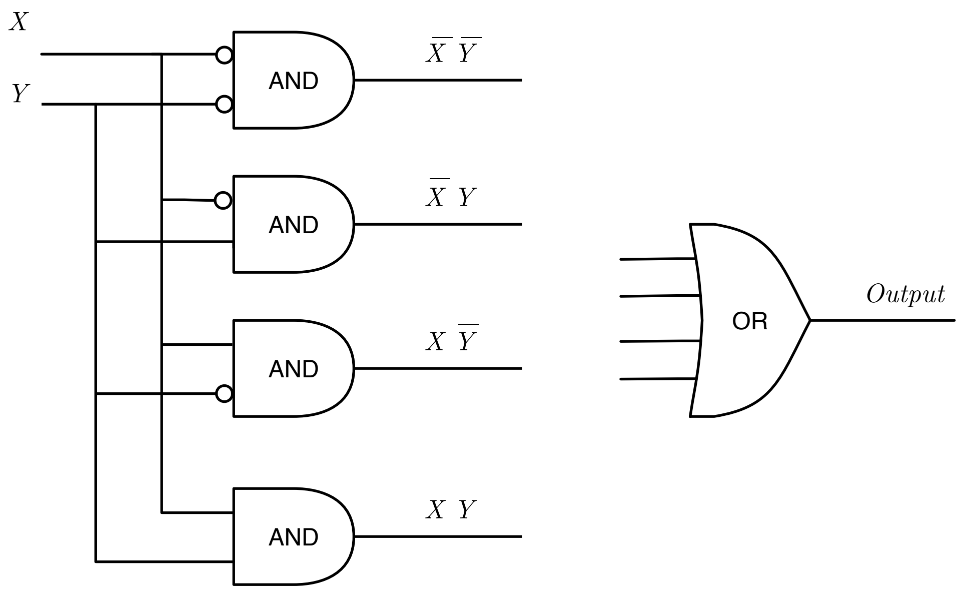
- This technique can produce long propositions that have shorter equivalents. Karnaugh maps and/or algebraic manipulations can be used to simplify the expression.

F. Translating From a Truth Table to a Simple Logic Circuit

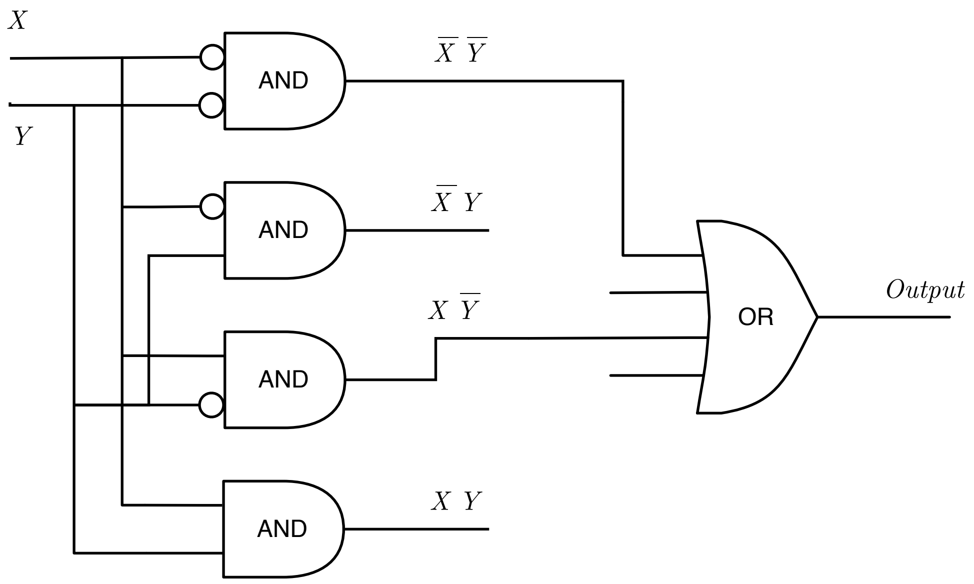
- A **Programming Logic Array** (**PLA**) is a general structure for implementing a truth table using logic gates. It relies on the disjunctive normal form representation of a truth table as a logical expression.
    - Each conjunct corresponds to an *AND* gate with literals as its inputs. The outputs of selected *AND* gates are all sent to an *OR* gate, and the output of the *OR* gate corresponds to the value in the truth table.
- A table with n logical inputs has 2^n rows (one for each gate). E.g., A table with inputs X and Y has $2^2 = 4$ rows. Note that for each input that is negated, we insert a *NOT* gate before the input goes to the *AND* gate.

X	***Y***	**Proposition**	**Gate**
0	0	$\overline{X}\ \overline{Y}$	
0	1	$\overline{X}\ Y$	
1	0	$X\ \overline{Y}$	
1	1	$X\ Y$	

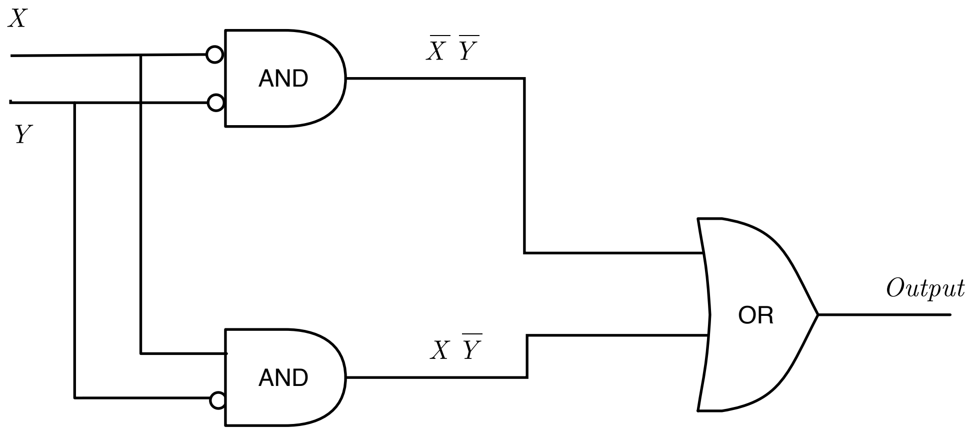
We begin with none of the *AND* gate outputs sent to the *OR* gate:



- To program a PLA, we connect the *AND* gates that correspond to output *T* to the *OR* gate; the *AND* gates that correspond to output *F* don't get connected. E.g., for the earlier truth table, we get:

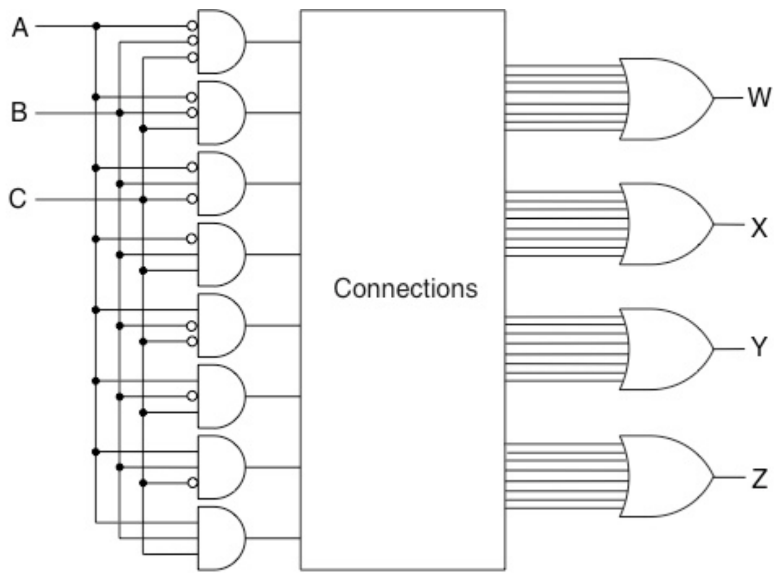


- If we're designing a circuit, we can simplify it tossing out the unused *AND* gates (and *OR* inputs). E.g., For the earlier truth table, we get



- (For a physical PLA, the unused *AND* gates and *OR* inputs don't get deleted, we just don't use them.)
- An actual PLA can have multiple outputs in addition to multiple inputs. (For an example, see Figure 3.17.)

Fig 3.17: 3-bit input, 4-bit output PLA



Logic Gates

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Logic gates are the lowest level of hardware that deal with logical values.

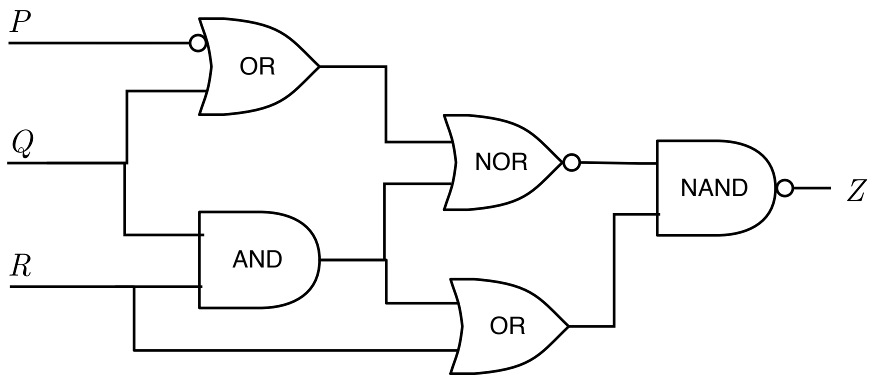
B. Outcomes

After this activity, you should be able to:

- Read and write the symbols for the logic gates.
- Trace the execution of a simple logic circuit.
- Convert between truth tables, logical formulas, and loop-free logic circuits.

C. Questions

1. Translate the logic gate circuit below to its equivalent boolean expression. Do a direct translation — don't simplify the expression.



2. Let E be the expression $\overline{X}\ \ Y + \neg(\overline{X} + \overline{Y}) + \neg(X + Y)$.

   (a) Write a logic gate implementation for E. Don't modify or simplify E.

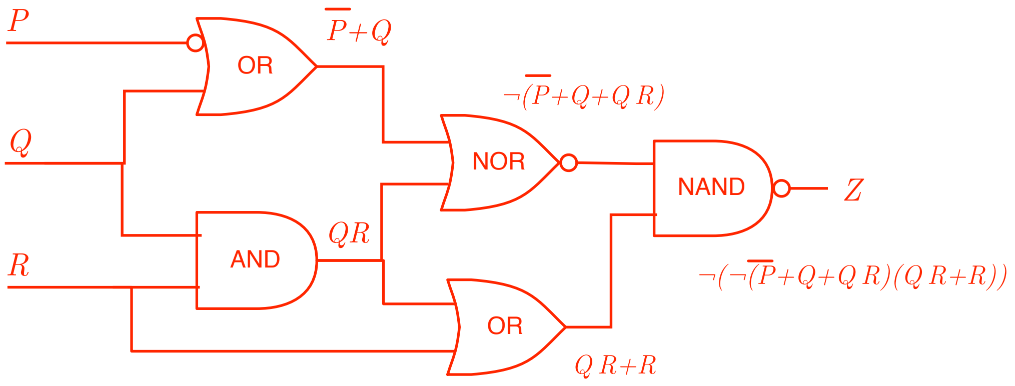
   (b) Use DeMorgan's laws (and possibly Pierce's law for removing double negations) to simplify E to full DNF. Write a logic gate implementation for this full DNF expression. (I.e., the PLA circuit for E.)

(c) Write out the Karnaugh map for E, develop an equivalent simpler expression, and write a logic gate implementation for this simpler expression.

3. Repeat Problem 2 on the expression $\overline{Y}\ (\overline{X}\,Y\,Z) + (X \oplus Y)\,Z + \neg(X \to \overline{Y}\,)\,Z$. (Read the $\oplus$ operator as XOR.)

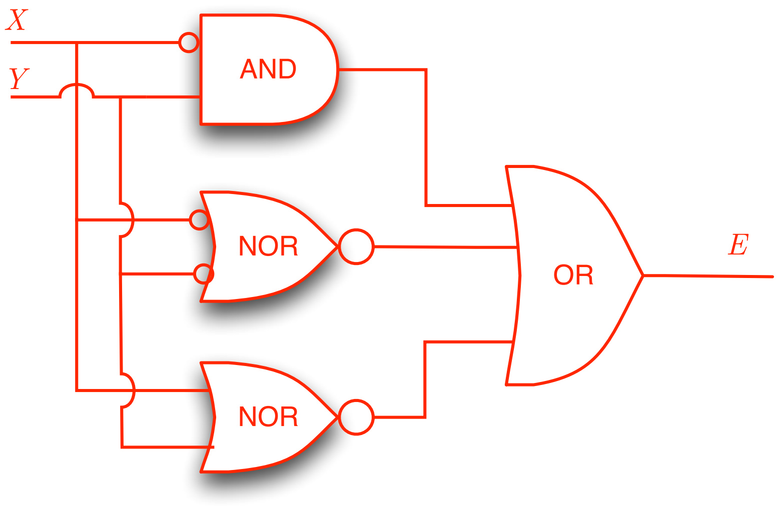
Solution

1. The expression is $\neg(\neg(\overline{P}+Q+Q\,R)(Q\,R+R))$. The expressions for the different parts of the diagram are shown in red below.

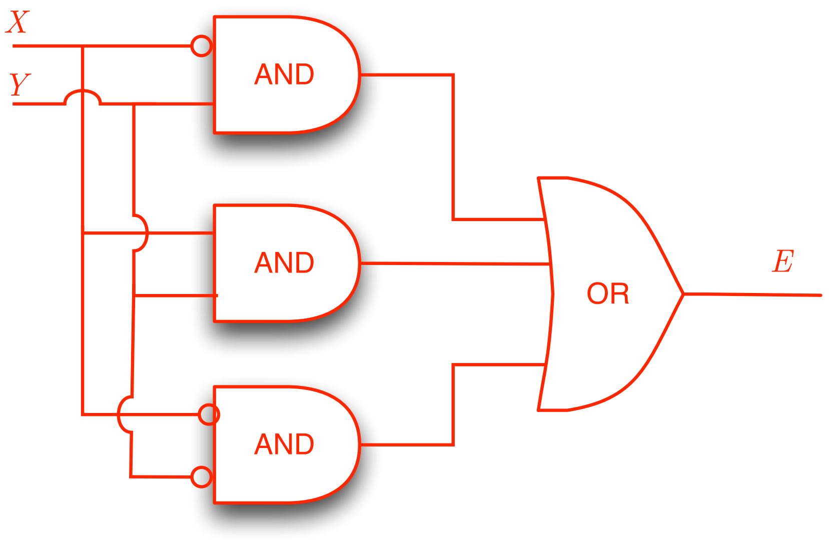


P
Q
R
OR
AND
NOR
NAND
OR
$\overline{P}+Q$
QR
$\neg(\overline{P}+Q+Q\,R)$
$Q\,R+R$
$\neg(\neg(\overline{P}+Q+Q\,R)(Q\,R+R))$
Z

2(a) Here's a direct translation of the expression to a logic circuit:



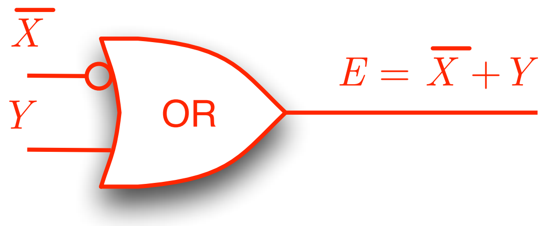
2(b) Using DeMorgan's laws $\overline{X}\,Y+\neg(\overline{X}+\overline{Y})+\neg(X+Y)$
$=\overline{X}Y+XY+\overline{X}\,\overline{Y}$. A circuit for this is:



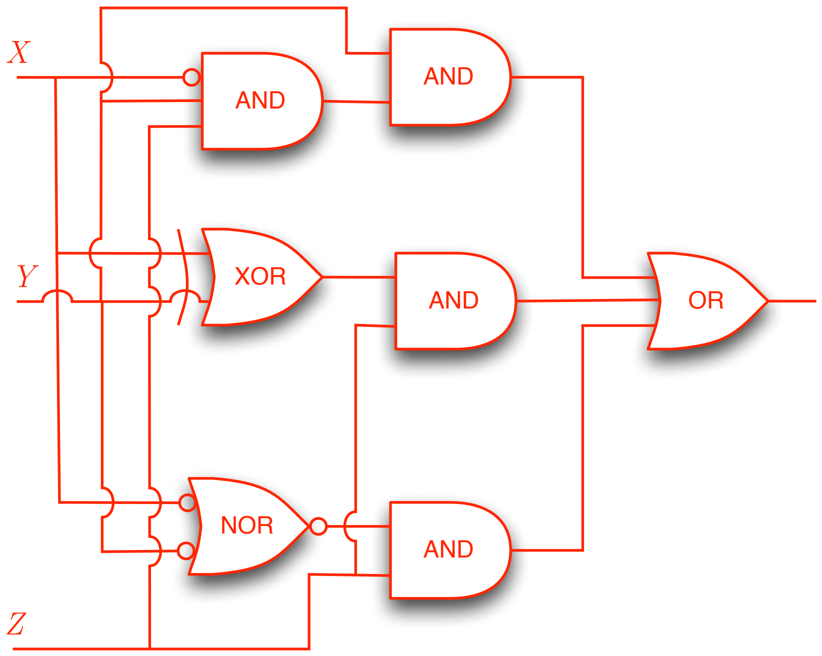
2(c) Here's a Karnaugh map; it indicates $E = \overline{X} + Y$ [updated 3/25]

	E
XY ***00***	$\boxed{1}$
01	1
11	1
10	0

2(d) Circuit for Karnaugh map:

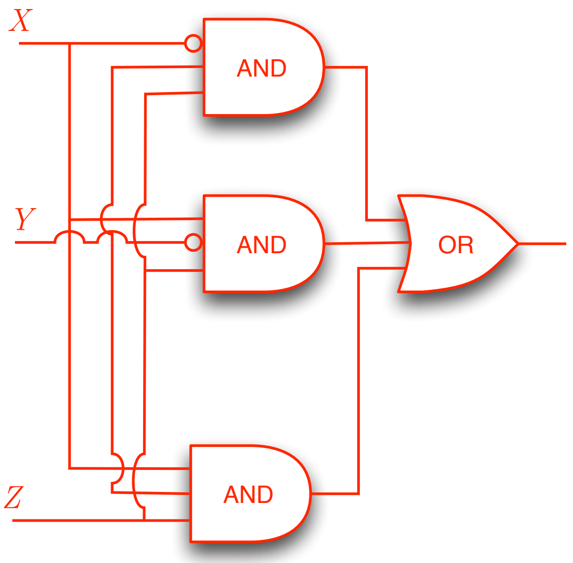


3(a) Logic gate implementation:



3(b) $\overline{Y}\,(\overline{X}\,Y\,Z) + (X \oplus Y)\,Z + \neg(X \rightarrow \overline{Y})\,Z = 0 + (X\,\overline{Y} + \overline{X}\,Y)Z + \neg(\overline{X} + \overline{Y})Z =$
$(X\,\overline{Y}\,Z + \overline{X}\,Y\,Z) + (X\,Y)Z = X\,\overline{Y}\,Z + \overline{X}\,Y\,Z + X\,Y\,Z.$

The PLA Circuit is

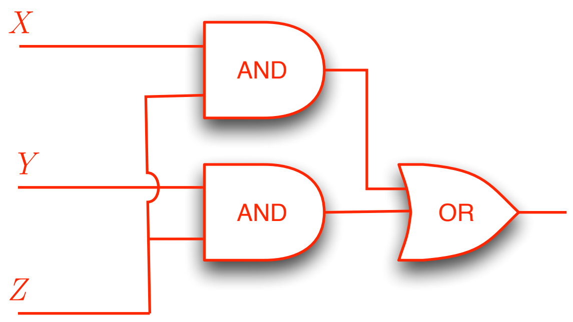


3(c) The Karnaugh map:

	***YZ* 00**	***01***	***11***	***10***
X* *0	0	0	1	0
1	0	1	1	0

The reduced expression is $XZ + YZ$.

3(d) A logic gate circuit for the reduced expression is:


X
Y
Z
AND
AND
OR