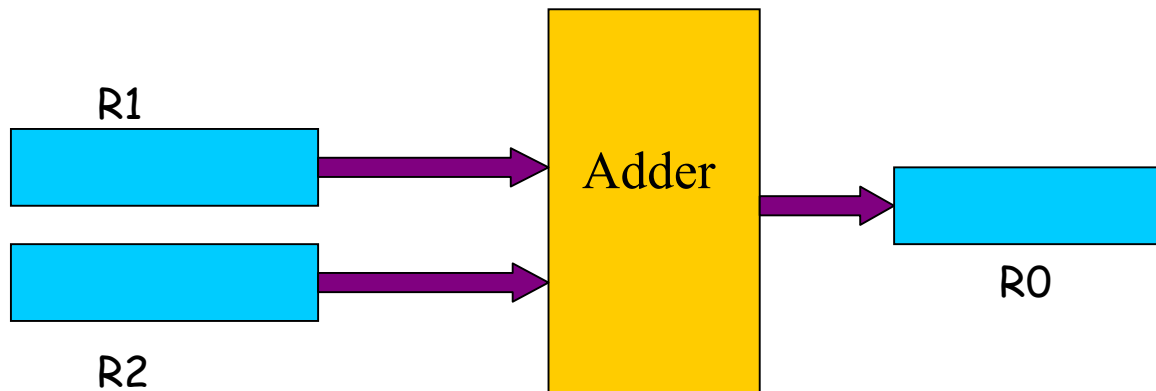


Understanding Logic Design

Appendix A of your Textbook does not have the needed background information. This document supplements it.

When you write add `ADD R0, R1, R2`, you imagine something like this:

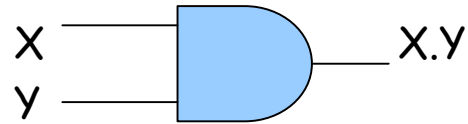


What kind of hardware can ADD two binary integers?

We need to learn about *GATES* and *BOOLEAN ALGEBRA* that are foundations of logic design.

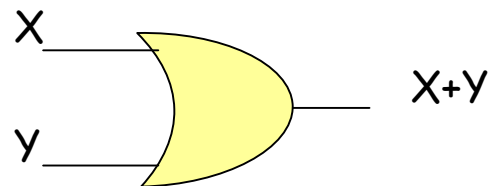
AND gate

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1



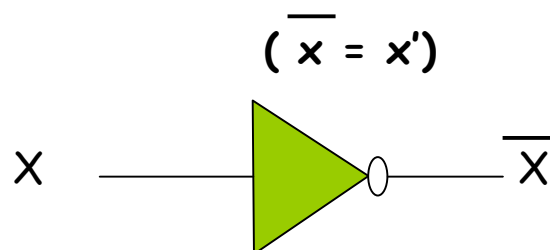
OR gate

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



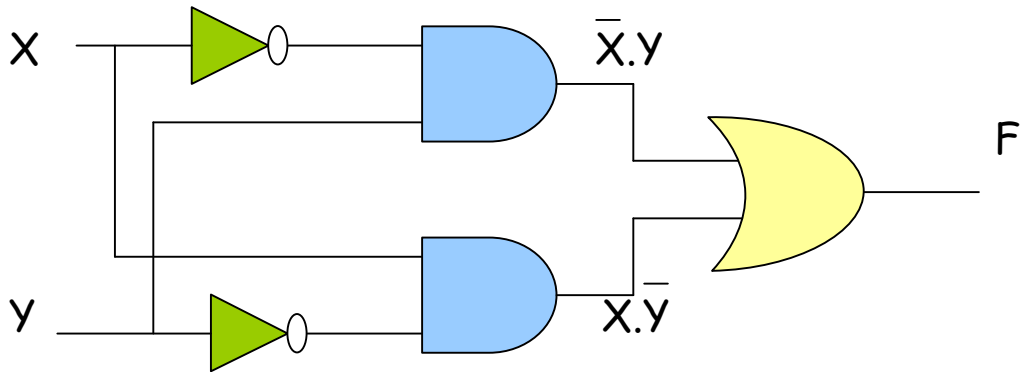
NOT gate

X	\overline{X}
0	1
1	0



Typically, logical 1 = +3.5 volt, and logical 0 = 0 volt. Other representations are possible.

Analysis of logical circuits



What is the value of F when X=0 and Y=1?

Draw a truth table.

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

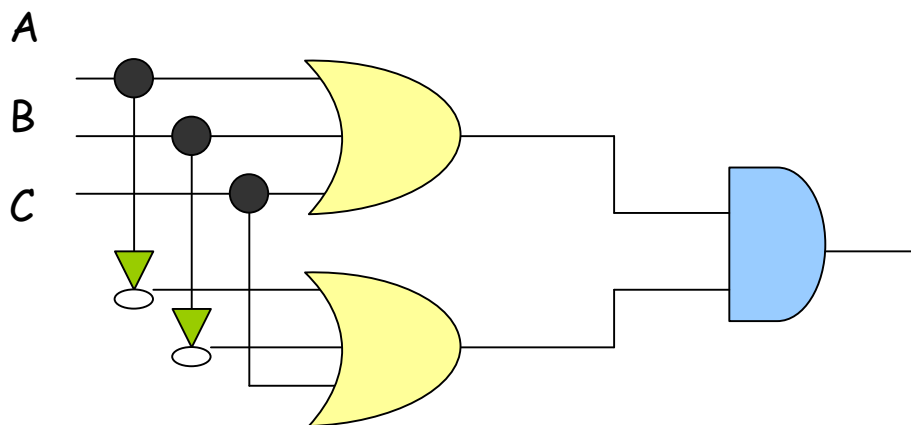
This is the **exclusive or** (XOR) function. In algebraic

form $F = \bar{X}.Y + X.\bar{Y}$

More practice

1. Let $\bar{A}.B + A.C = 0$. What are the values of A, B, C ?
2. Let $(A + B + C).(A + B + C) = 0$. What are the possible values of A, B, C ?

- Draw truth tables.
- Draw the logic circuits for the above two functions.



Boolean Algebra

$$\left. \begin{array}{l} A + 0 = A \\ A \cdot 1 = A \end{array} \right\}$$

$$A + A' = 1$$

$$A \cdot A' = 0$$

$$\left. \begin{array}{l} 1 + A = 1 \\ 0 \cdot A = 0 \end{array} \right\}$$

$$\left. \begin{array}{l} A + B = B + A \\ A \cdot B = B \cdot A \end{array} \right\}$$

$$\left. \begin{array}{l} A + (B + C) = (A + B) + C \\ A \cdot (B \cdot C) = (A \cdot B) \cdot C \end{array} \right\}$$

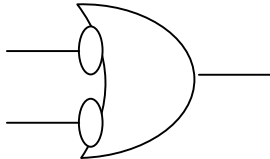
$$\left. \begin{array}{l} A + A = A \\ A \cdot A = A \end{array} \right\}$$

$$\left. \begin{array}{l} A \cdot (B + C) = A \cdot B + A \cdot C \\ A + B \cdot C = (A + B) \cdot (A + C) \end{array} \right\} \text{Distributive Law}$$

$$\left. \begin{array}{l} \overline{A \cdot B} = \overline{A} + \overline{B} \\ \overline{A + B} = \overline{A} \cdot \overline{B} \end{array} \right\} \text{De Morgan's theorem}$$

De Morgan's theorem

$$\left. \begin{array}{l} \overline{A \cdot B} = \bar{A} + \bar{B} \\ \overline{A + B} = \bar{A} \cdot \bar{B} \end{array} \right\}$$

Thus,  is equivalent to 

Verify it using truth tables. Similarly,

 is equivalent to 

These can be generalized to more than two variables: to

$$\left. \begin{array}{l} \overline{A \cdot B \cdot C} = \bar{A} + \bar{B} + \bar{C} \\ \overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C} \end{array} \right\}$$

Synthesis of logic circuits

Many problems of logic design can be specified using a truth table. Give such a table, can you design the logic circuit?

Design a logic circuit with three inputs A, B, C and one output F such that F=1 only when a majority of the inputs is equal to 1.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Sum of product form

$$F = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C$$

Draw a logic circuit to generate F

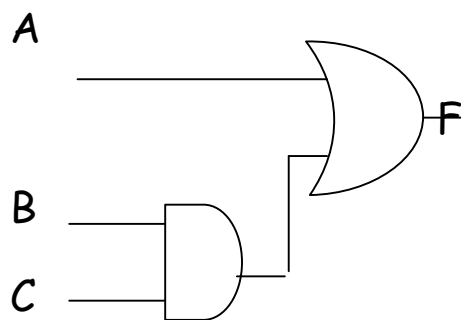
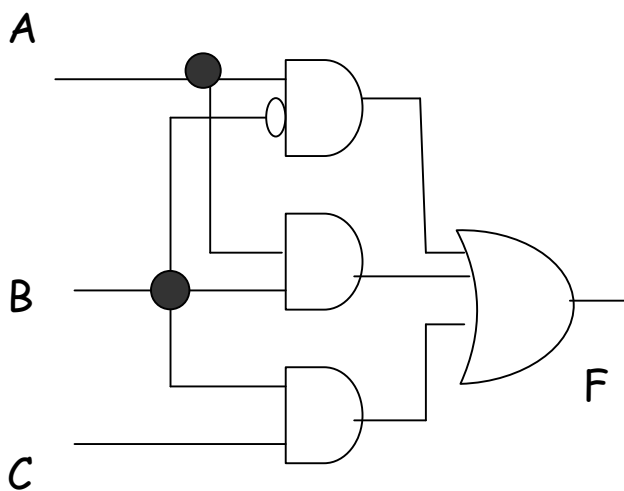
Simplification of Boolean functions

Using the theorems of Boolean Algebra, the algebraic forms of functions can often be simplified, which leads to simpler (and cheaper) implementations.

Example 1

$$\begin{aligned} F &= \overline{A} \cdot \overline{B} + A \cdot B + B \cdot C \\ &= \overline{A} \cdot (\overline{B} + B) + B \cdot C \\ &= \overline{A} \cdot 1 + B \cdot C \\ &= \overline{A} + B \cdot C \end{aligned}$$

How many gates do you save from this simplification?



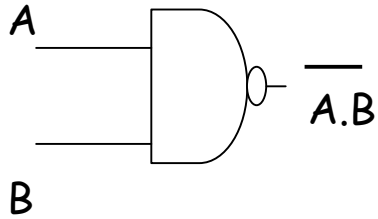
Example 2

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

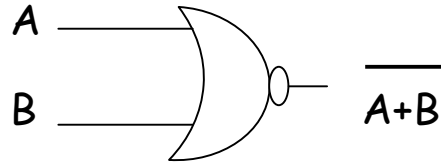
Example 3 Show that $A + A.B = A$

$$\begin{aligned} &A + AB \\ &= A.1 + A.B \\ &= A.(1 + B) \\ &= A.1 \\ &= A \end{aligned}$$

Other types of gates

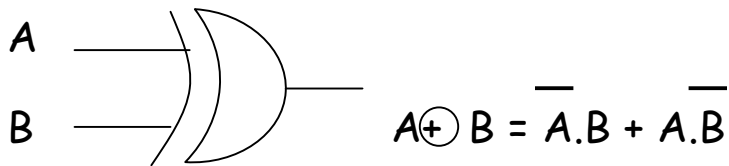


NAND gate



NOR gate

Be familiar with the truth tables of these gates.



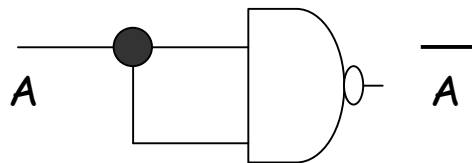
Exclusive OR (XOR) gate

NAND and NOR are universal gates

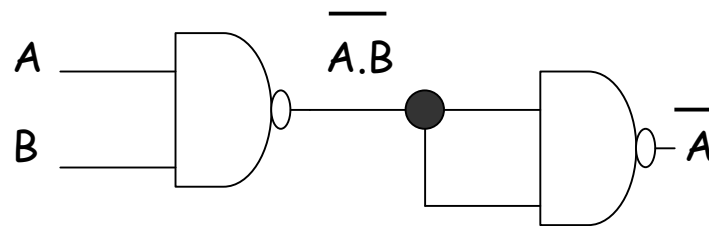
Any function can be implemented using **only NAND** or **only NOR** gates. How can we prove this?

(Proof for NAND gates) Any boolean function can be implemented using AND, OR and NOT gates. So if AND, OR and NOT gates can be implemented using NAND gates only, then we prove our point.

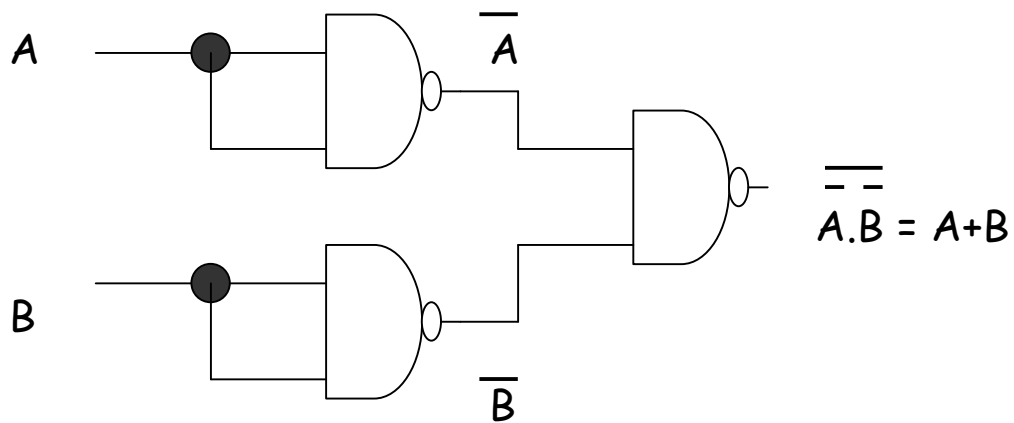
1. Implement NOT using NAND



2. Implementation of AND using NAND



3. Implementation of OR using NAND



Exercise. Prove that NOR is a universal gate.

Logic Design (continued)

XOR Revisited

XOR is also called **modulo-2** addition.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

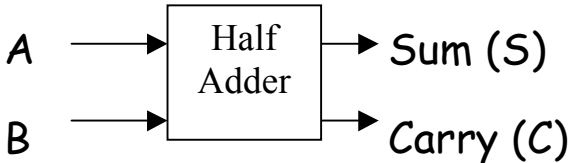
$A \oplus B = 1$ only when there are an odd number of 1's in (A,B). The same is true for $A \oplus B \oplus C$ also.

$$\left. \begin{array}{l} 1 \oplus A = \overline{A} \\ 0 \oplus A = A \end{array} \right\}$$

Why?

Logic Design Examples

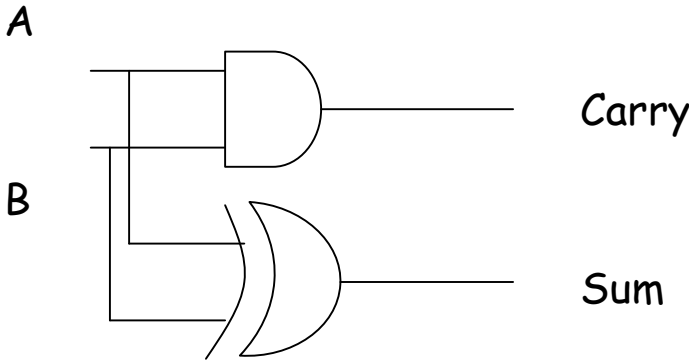
Half Adder



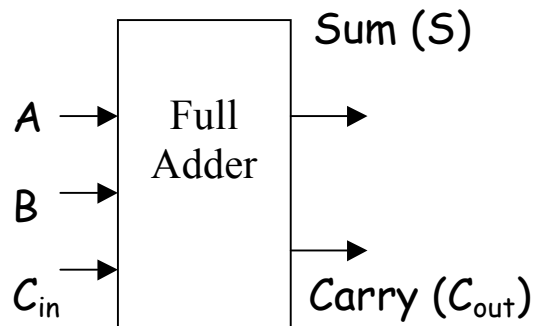
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C = A.B$$



Full Adder



A	B	C _{in}	S	C _{out}
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

$$S = A \oplus B \oplus C_{in}$$

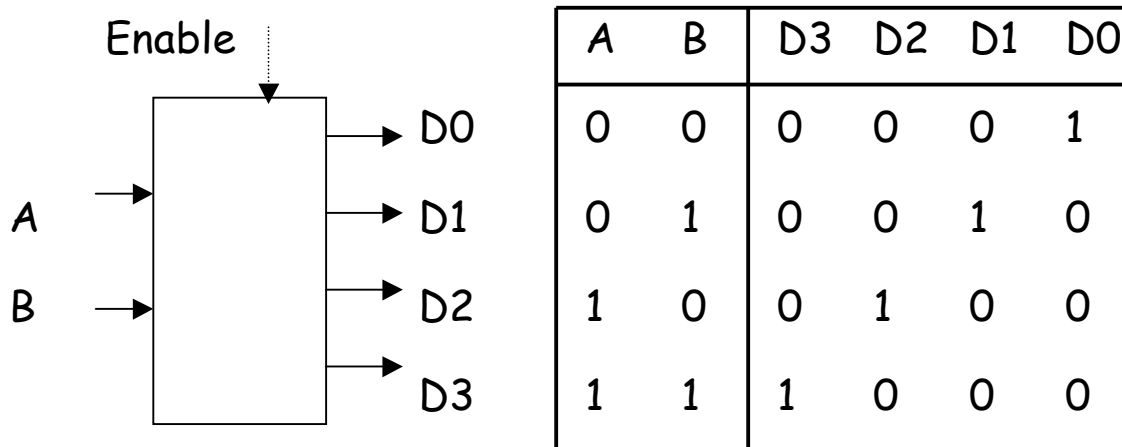
$$C_{out} = A.B + B.C_{in} + A.C_{in}$$

Design a full adder using two half-adders (and a few gates if necessary)

Can you design a 1-bit subtracter?

Decoders

A typical decoder has n inputs and 2^n outputs.



A 2-to-4 decoder and its truth table

$$\begin{aligned} D3 &= A.B \\ D2 &= A.\bar{B} \\ D1 &= \bar{A}.B \\ D0 &= \bar{A}.\bar{B} \end{aligned}$$

Draw the circuit of this decoder.

The decoder works per specs when (**Enable = 1**). When **Enable = 0**, all the outputs are 0.

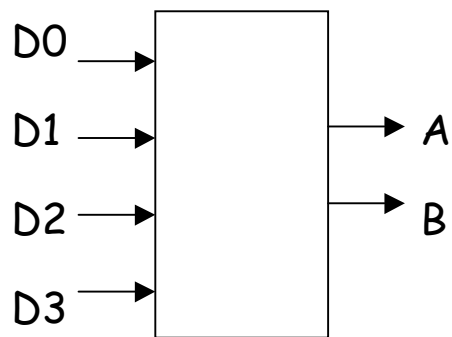
Exercise. Design a 3-to-8 decoder.

Question. Where are decoders used?

Can you design a 2-4 decoder using 1-2 decoders?

Encoders

A typical encoder has 2^n inputs and n outputs.



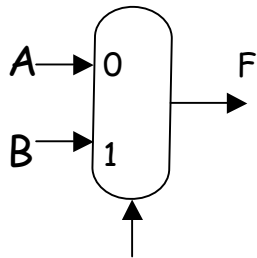
D0	D1	D2	D3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

A 4-to-2 encoder and its truth table

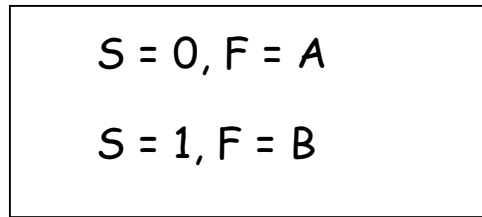
$$A = D1 + D3$$
$$B = D2 + D3$$

Multiplexor

It is a **many-to-one switch**, also called a **selector**.



Control S



Specifications of the mux

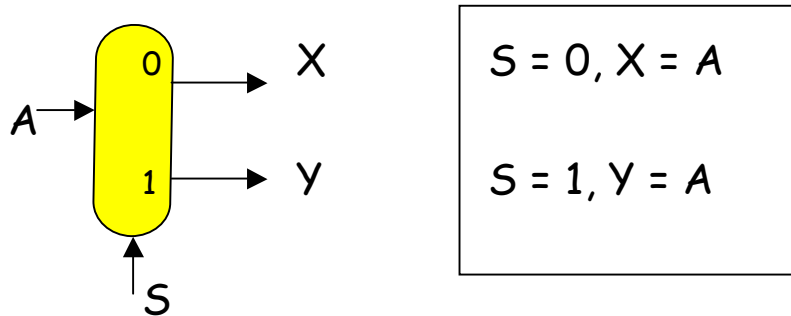
A 2-to-1 mux

$$F = \overline{S} \cdot A + S \cdot B$$

Exercise. Design a 4-to-1 multiplexor using two 2-to-1 multiplexors.

Demultiplexors

A demux is a **one-to-many** switch.



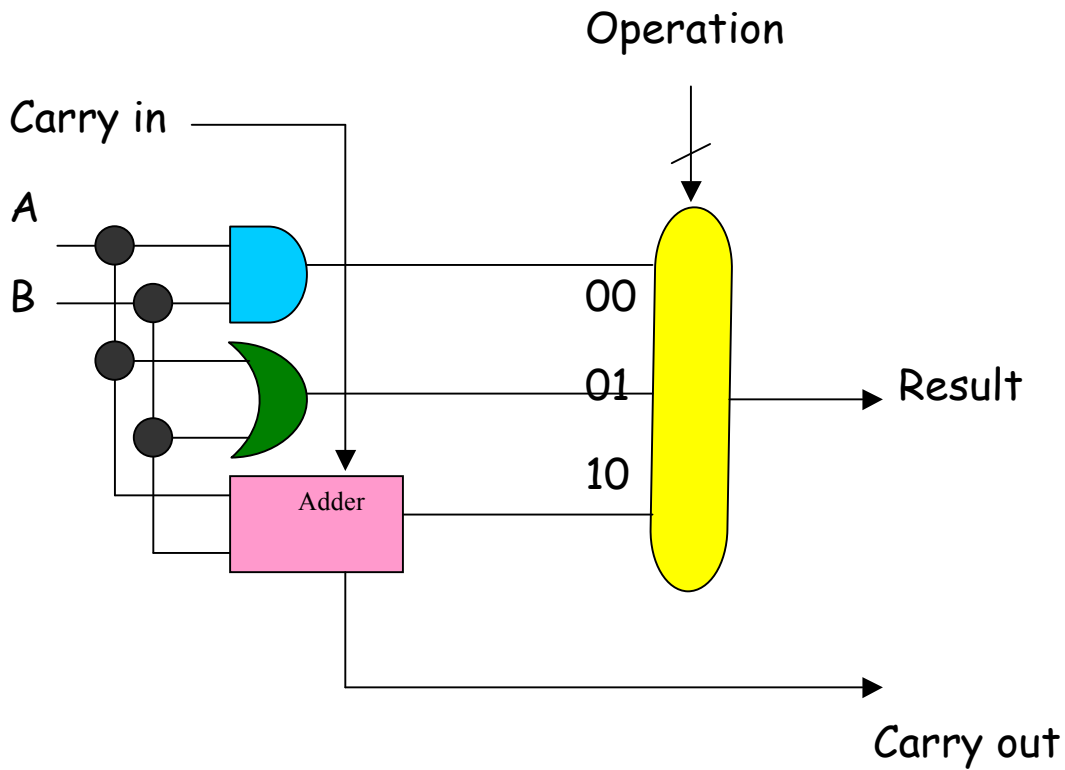
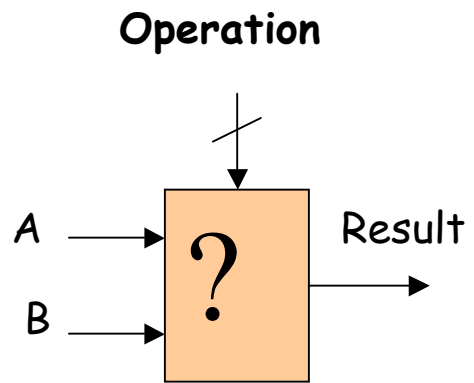
A 1-to-2 demux, and its specification.

So, $X = \overline{S} \cdot A$, and $Y = S \cdot A$

Exercise. Design a 1-4 demux using 1-2 demux.

A 1-bit ALU

Operation = 00 implies AND
Operation = 01 implies OR
Operation = 10 implies ADD



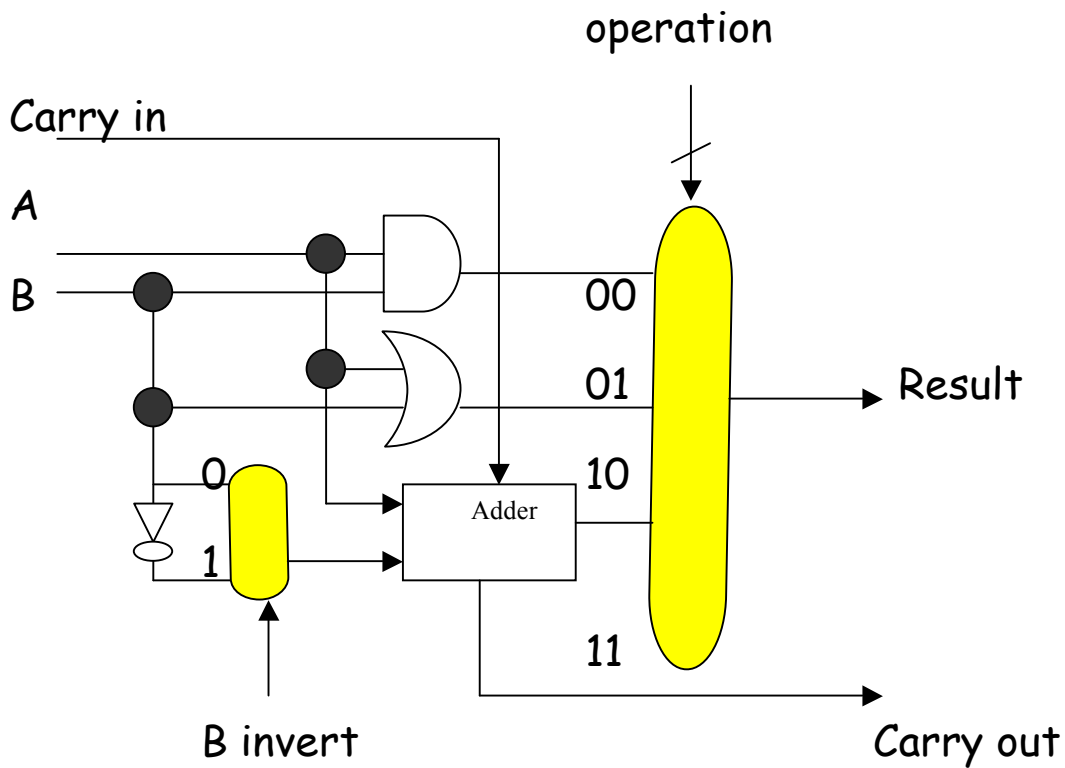
◆ Understand how this circuit works.

◆ Converting an adder into a subtractor

$A - B$ (here - means arithmetic subtraction)

= $A + 2$'s complement of B

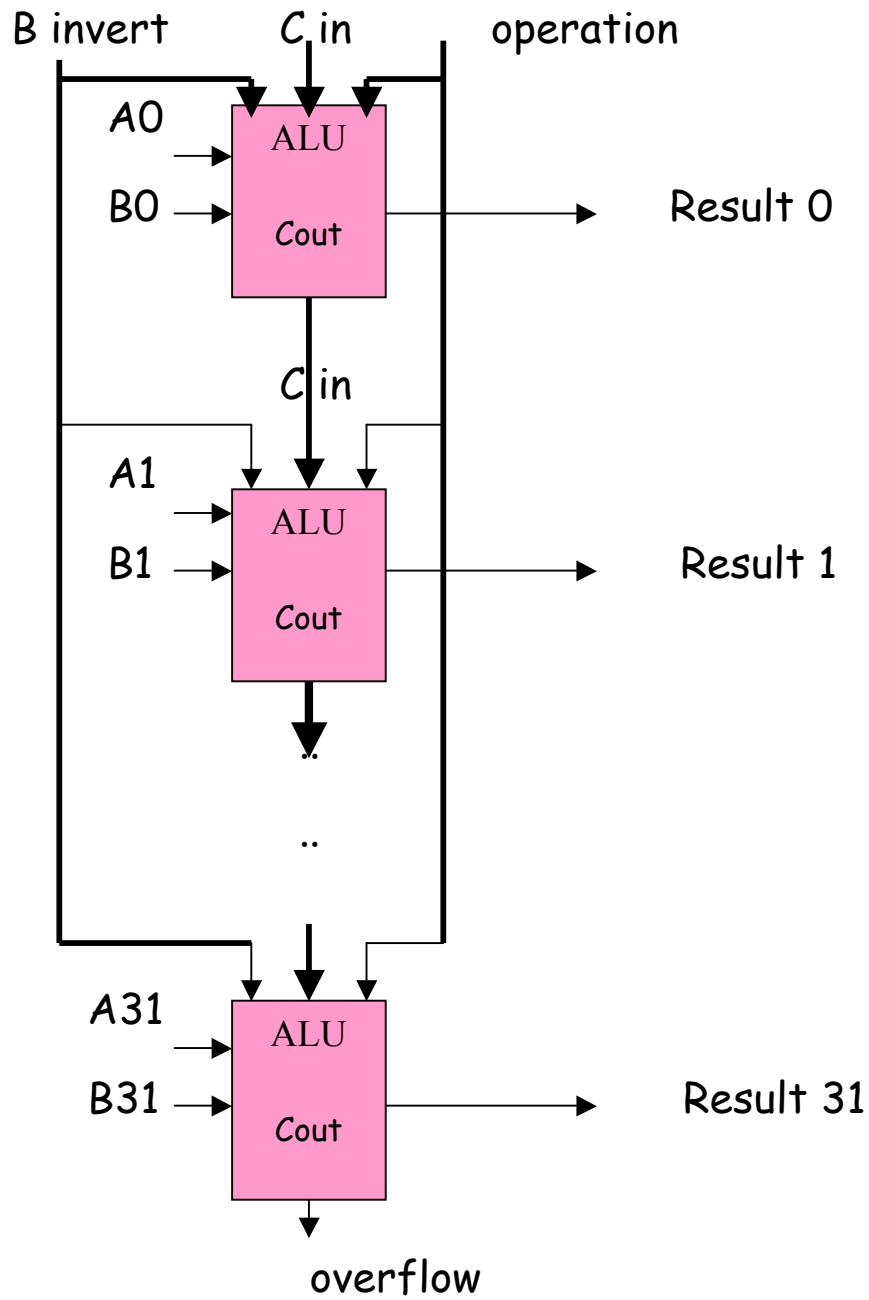
= $A + 1$'s complement of $B + 1$



1-bit adder/subtractor

For subtraction, **B invert = 1** and **Carry in = 1**

A 32-bit ALU



Combinational vs. Sequential Circuits

Combinational circuits

The output depends only on the current values of the inputs and not on the past values. Examples are adders, subtractors, and all the circuits that we have studied so far

Sequential circuits

The output depends not only on the current values of the inputs, but also on their past values. These hold the secret of how to memorize information. **We will study sequential circuits later.**