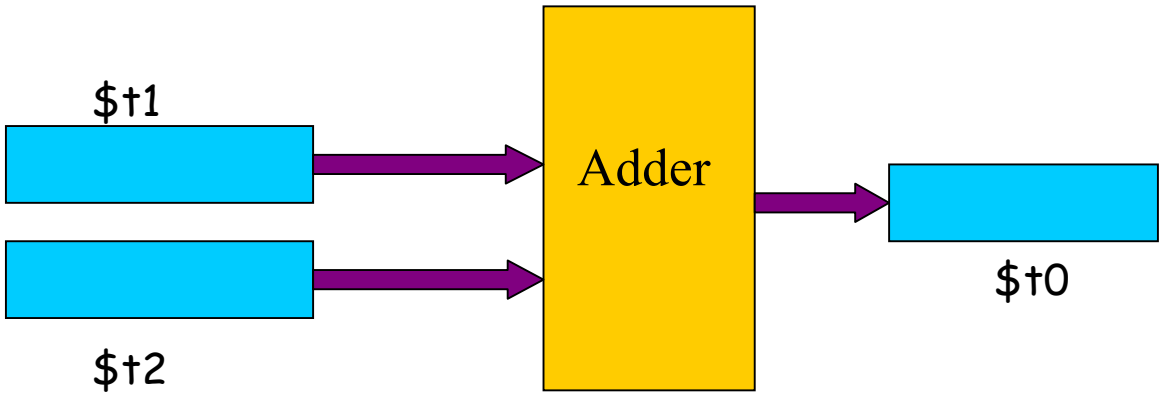
Logic Design

See Appendix B of your Textbook

When you write add $t0, $t1, $t2, you imagine something like this:

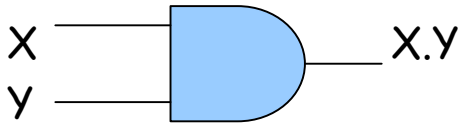


What kind of hardware can ADD two binary integers?

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

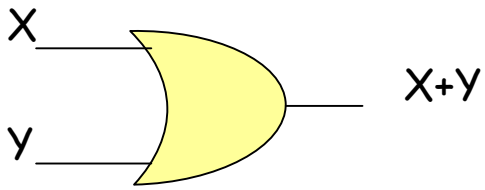
AND gate

X	y	X.y
0	0	0
0	1	0
1	0	0
1	1	1



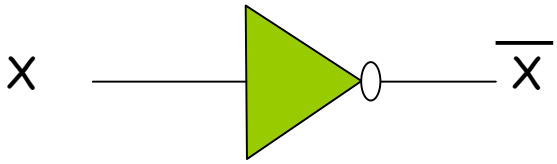
OR gate

X	y	X+y
0	0	0
0	1	1
1	0	1
1	1	1



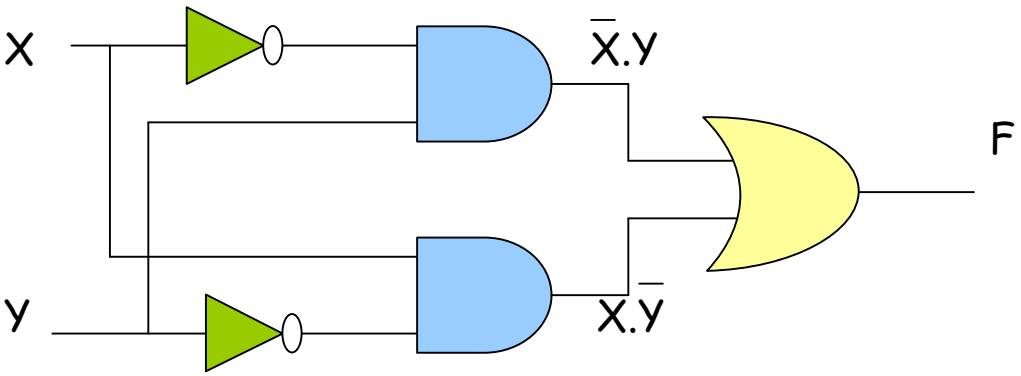
NOT gate

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.

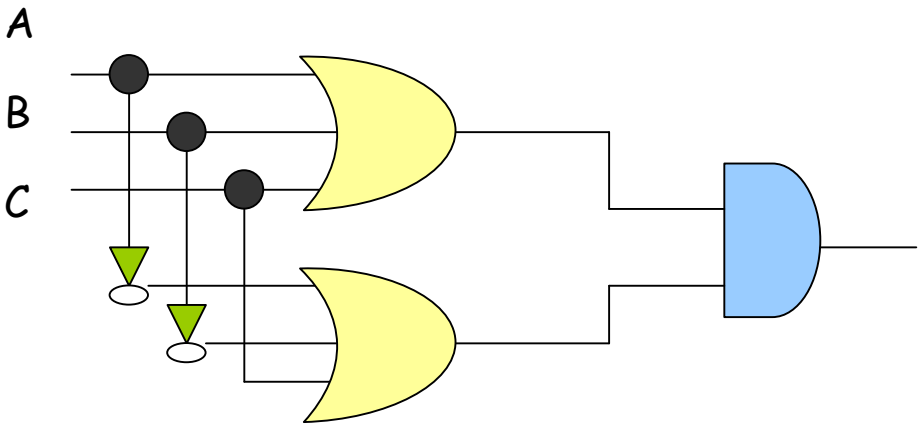
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

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

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



Boolean Algebra

$A + 0 = A$

$A \cdot 1 = A$

$A + A' = 1$

$A \cdot A' = 0$

$1 + A = 1$

$0 \cdot A = 0$

$A + B = B + A$

$A \cdot B = B \cdot A$

$A + (B + C) = (A + B) + C$

$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

$A + A = A$

$A \cdot A = A$

$A \cdot (B + C) = A.B + A.C$

$A + B.C = (A+B) \cdot (A+C)$

Distributive Law

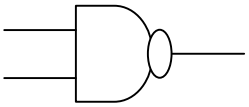
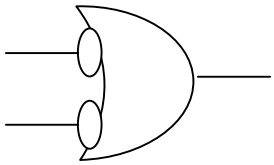
$\overline{A \cdot B} = \bar{A} + \bar{B}$

$\overline{A + B} = \bar{A} \cdot \bar{B}$

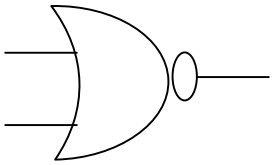
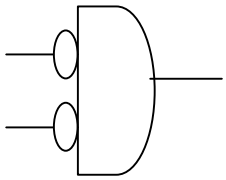
De Morgan's theorem

De Morgan's theorem

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$
$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

Thus,  **is equivalent to** 

Verify it using truth tables. Similarly,

 **is equivalent to** 

These can be generalized to more than two variables: to

$$\overline{A.\,B.\,C} = \bar{A} + \bar{B} + \bar{C}$$
$$\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

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

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

Draw a logic circuit to generate F

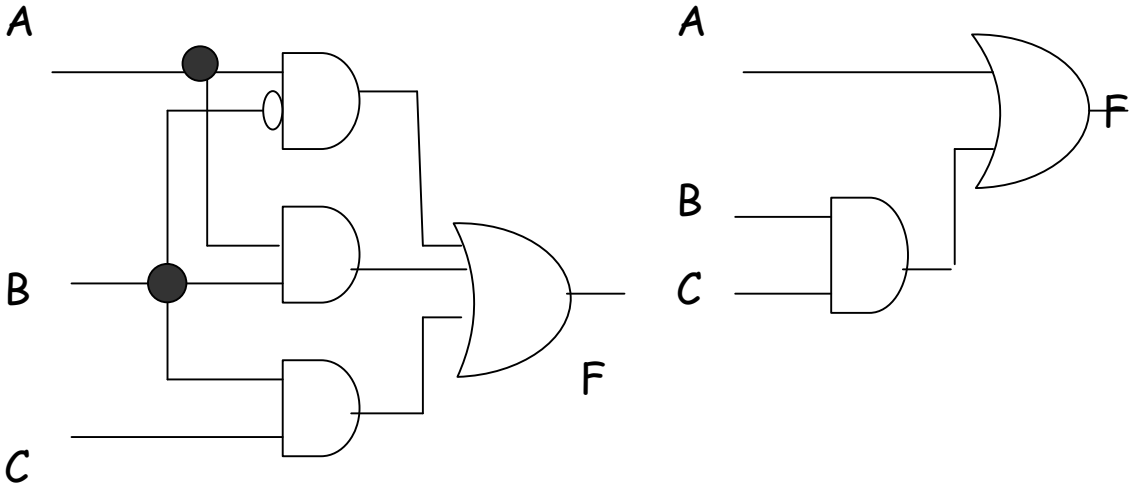
Simplification of Boolean functions

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

Example 1

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

How many gates do you save from this simplification?



Example 2

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

$$= \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C + A.B.C + A.B.C$$

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

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

$$= B.C + C.A + A.B$$

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

$$A + AB$$

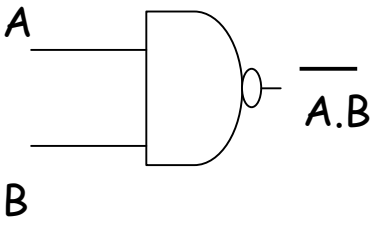
$$= A.1 + A.B$$

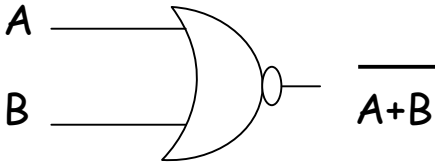
$$= A.\ (1 + B)$$

$$= A.\ 1$$

$$= A$$

Other types of gates

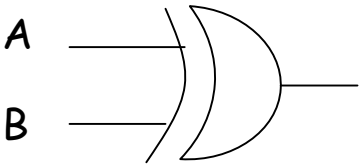




NAND gate

NOR gate

Be familiar with the truth tables of these gates.



$A \oplus B = \overline{A}.B + A.\overline{B}$

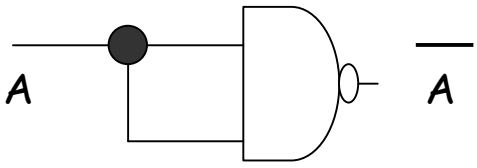
Exclusive OR (XOR) gate

NAND and NOR are universal gates

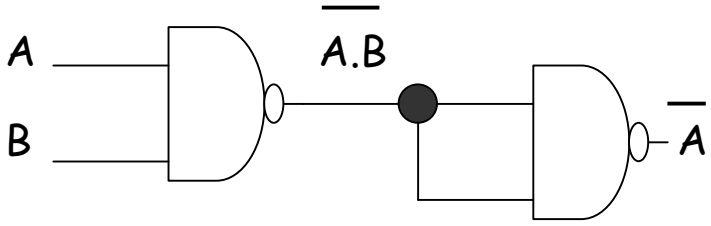
Any function can be implemented using **only NAND** or **only NOR** gates. How can we prove this?

(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.

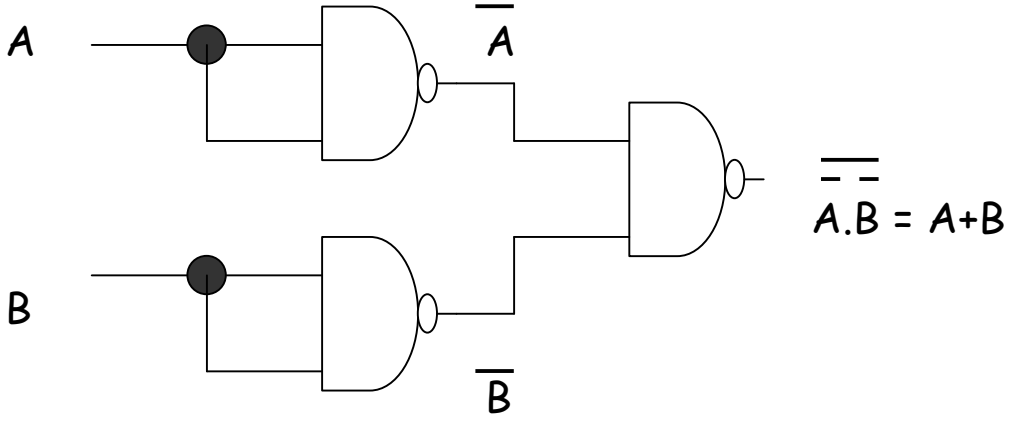
1. Implement NOT using NAND



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.

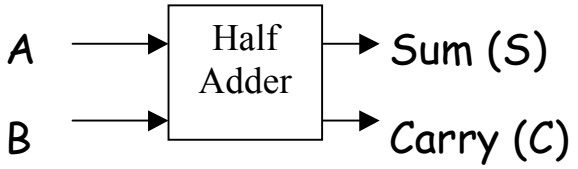
$1 \oplus A = \overline{A}$

$0 \oplus A = A$

Why?

Logic Design Examples

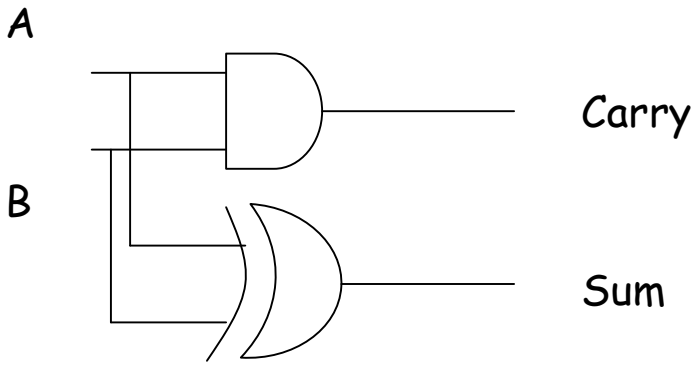
Half Adder



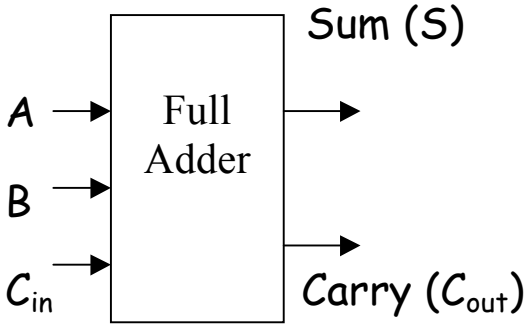
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = A \oplus B$

$C = A.B$



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

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

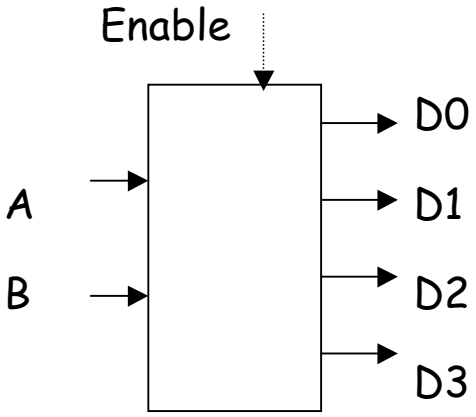
$C_{out} = A.B + B.C_{in} + A.C_{in}$

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

2. Can you design a 1-bit subtractor?

Decoders

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



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

A 2-to-4 decoder and its truth table

$D3 = A.B$

$D2 = A.\overline{B}$

$D1 = \overline{A}.B$

$D0 = \overline{A}.\overline{B}$

Draw the circuit of this decoder.

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

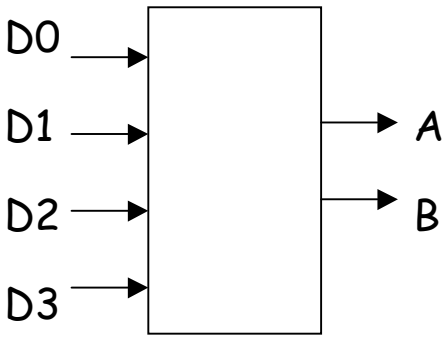
Exercise. Design a 3-to-8 decoder.

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

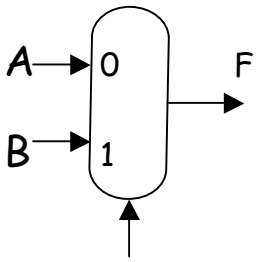
A 4-to-2 encoder and its truth table

$A = D1 + D3$

$B = D2 + D3$

Multiplexor

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



Control S

S = 0, F = A

S = 1, F = B

Specifications of the mux

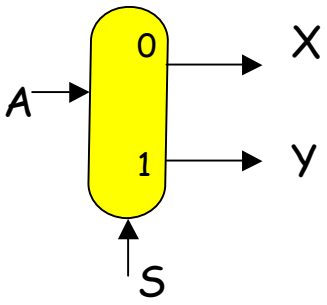
A 2-to-1 mux

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

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

Demultiplexors

A demux is a one-to-many switch.



S = 0, X = A

S = 1, Y = A

A 1-to-2 demux and its specification

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

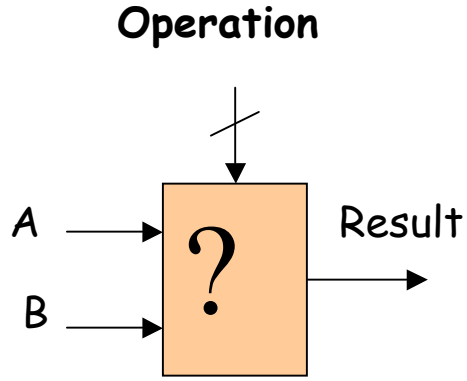
Exercise. Design a 1-4 demux using 1-2 demux.

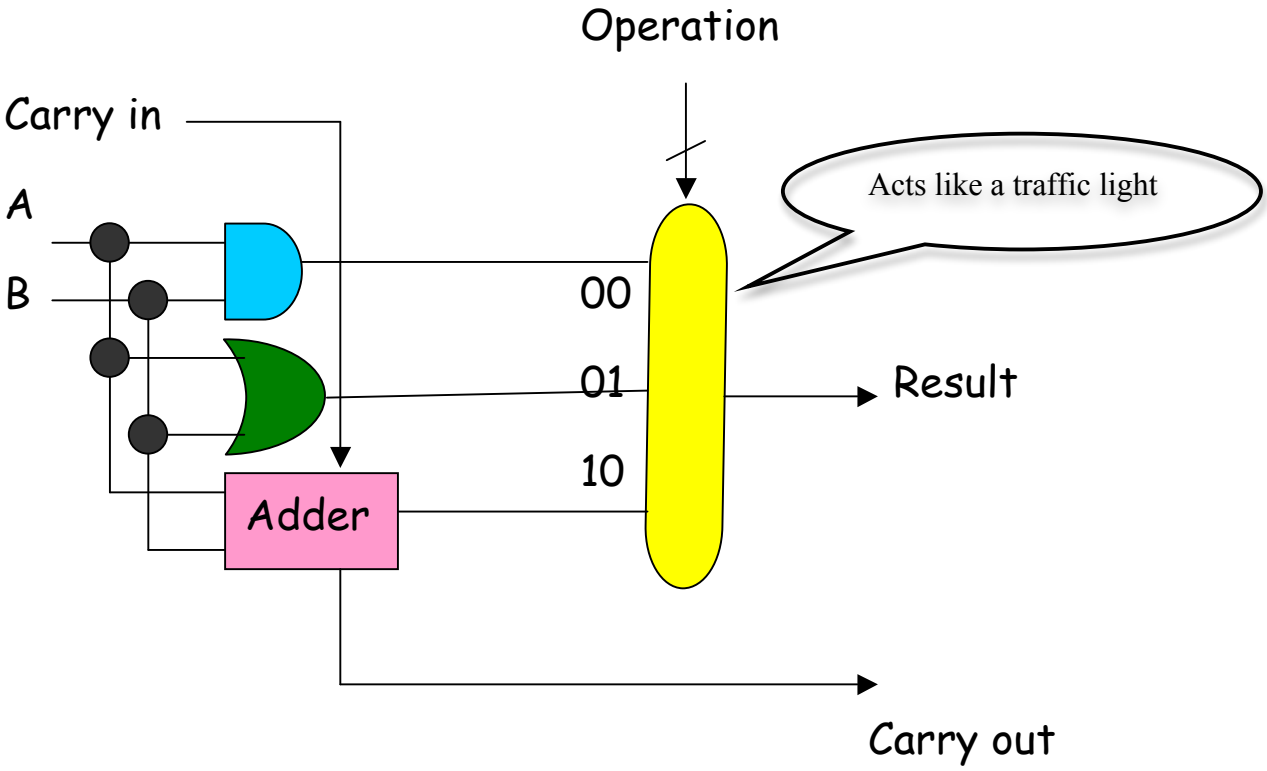
A 1-bit ALU

Operation = 00 implies AND

Operation = 01 implies OR

Operation = 10 implies ADD





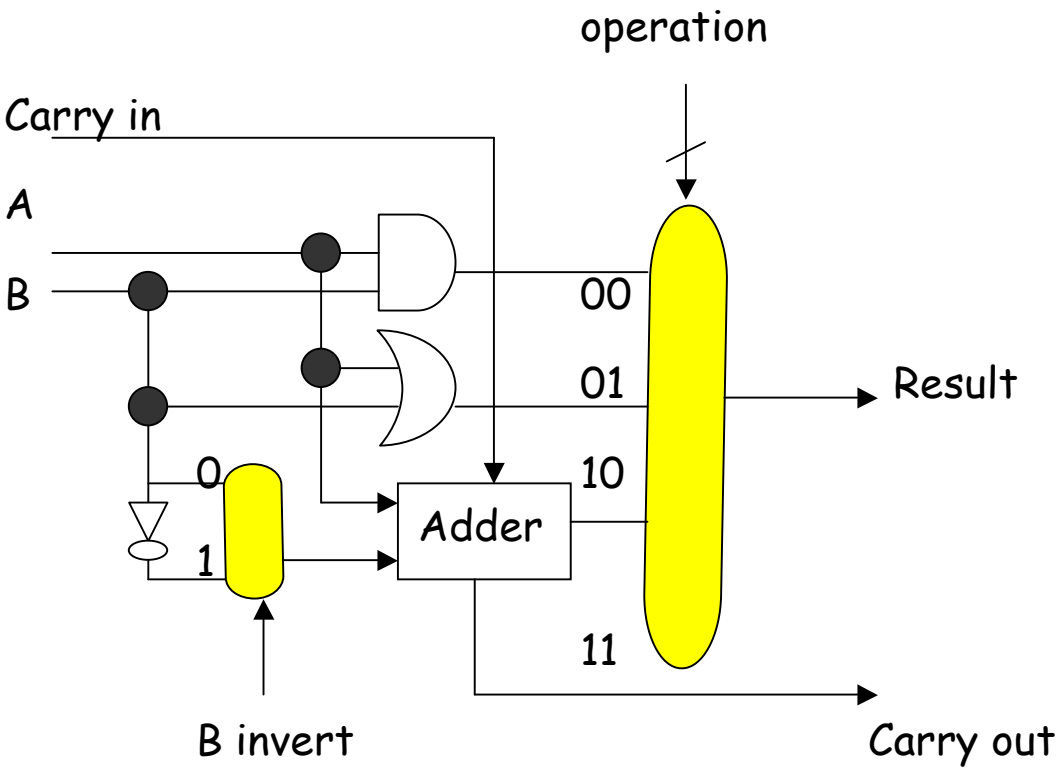
- Understand how this circuit works.
- Let us add one more input to the mux to implement **slt** when the Operation = 11

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

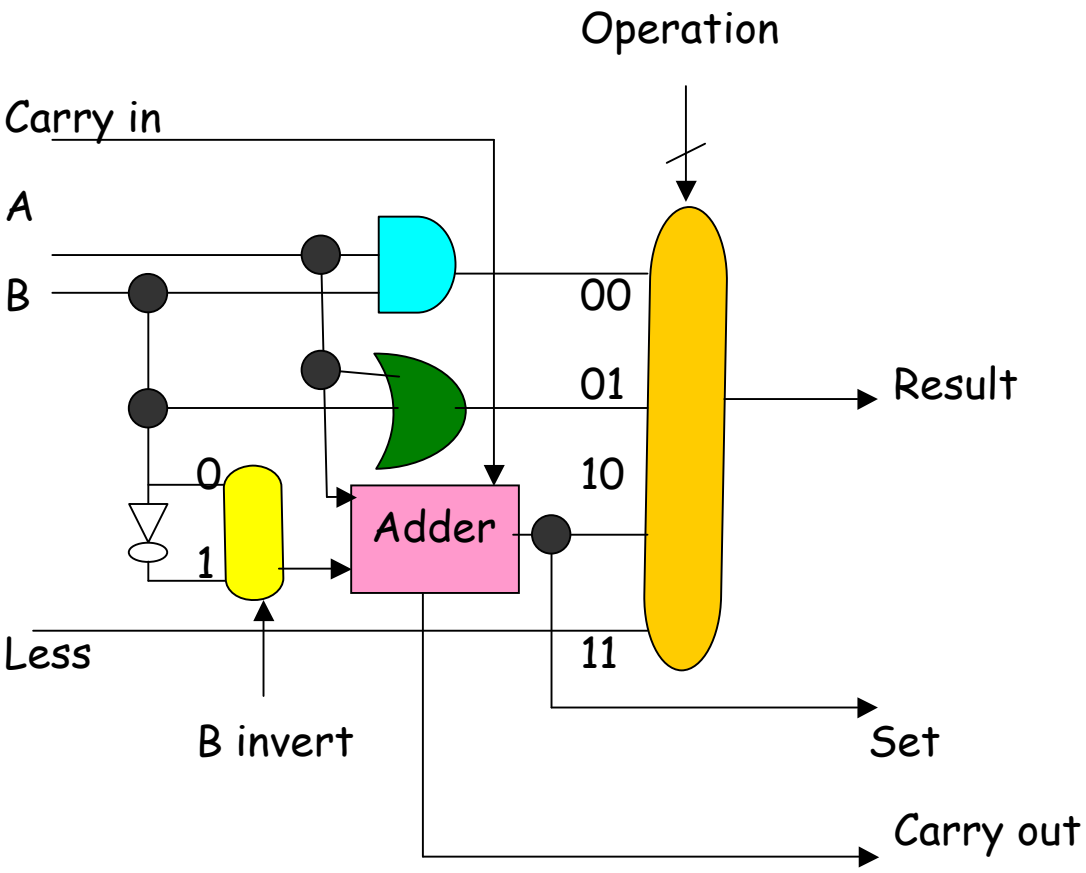


1-bit adder/subtractor

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

1-bit ALU for MIPS

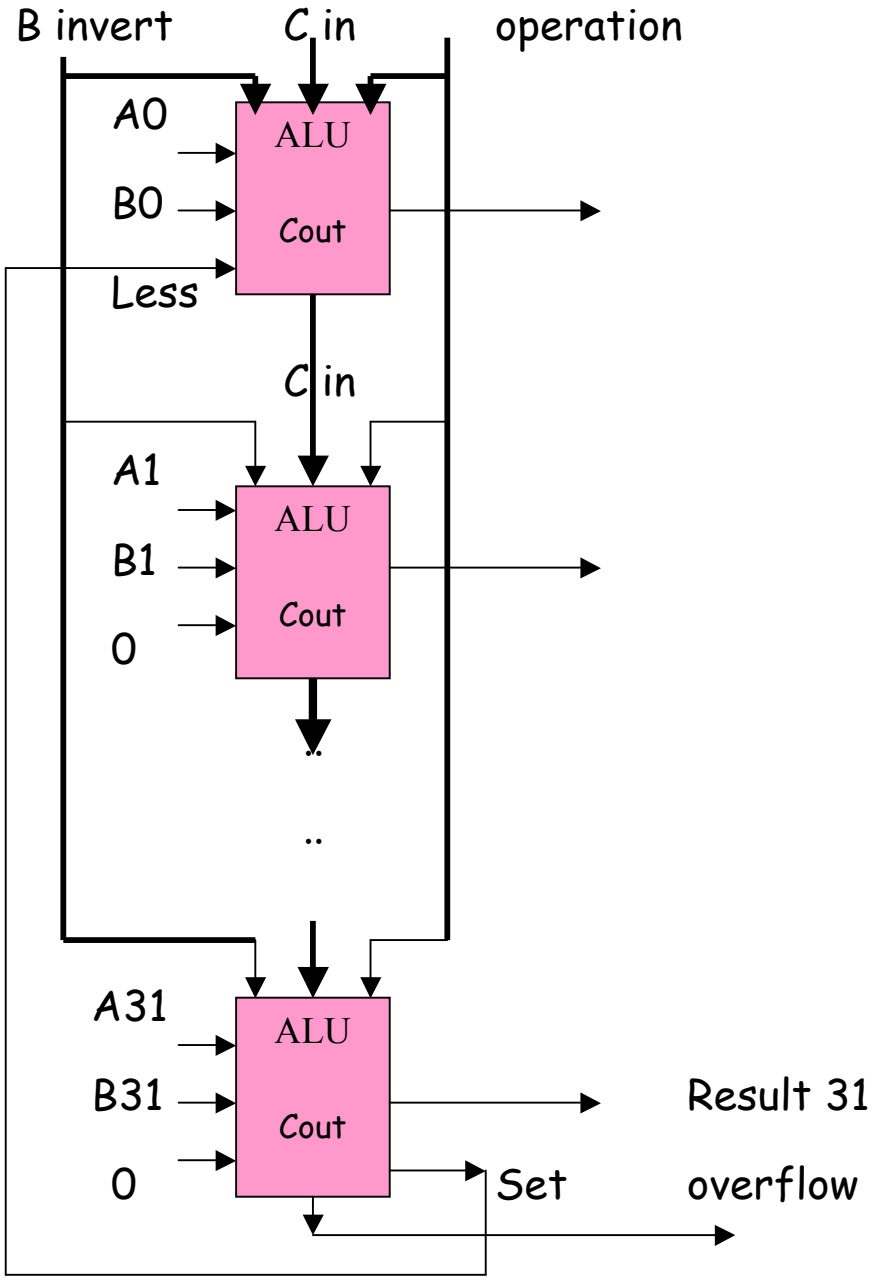
Assume that it has the instructions add, sub, and, or, slt.



Less = 1 if the 32-bit number A is less than the 32-bit number B. (Its use will be clear from the next page)

We now implement **slt** (If A < B then Set = 1 else Set = 0)

A 32-bit ALU for MIPS



Ripple Carry Adder

- **Addition**
  - most frequent operation
  - used also for multiplication and division
  - fast two-operand adder essential
- **Simple parallel adder**
  - for adding $x_{n-1},x_{n-2},\ldots,x_0$ and $y_{n-1},y_{n-2},\ldots,y_0$
  - using **n** full adders
- **Full adder**
  - combinational digital circuit with input bits x_i,y_i and incoming carry bit c_i, producing output sum bit s_i and outgoing carry bit c_{i+1}
  - incoming carry for next FA with input bits x_{i+1},y_{i+1}
  - $s_i = x_i \oplus y_i \oplus c_i$
  - $c_{i+1} = x_i \cdot y_i + c_i \cdot (x_i + y_i)$

Full-Adder (FA)

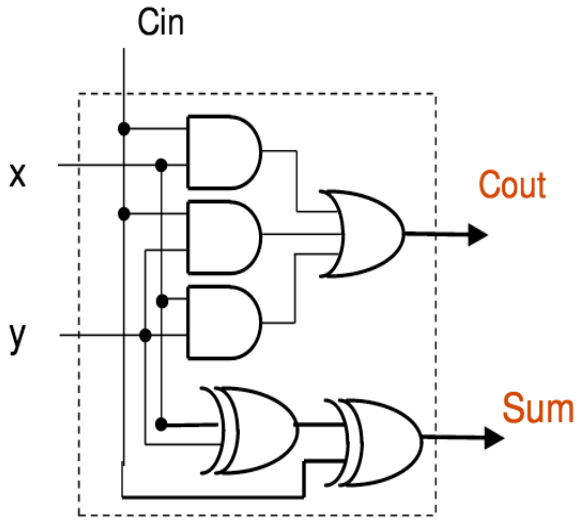
- Examine the Full Adder table

x	y	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$Cout = x \cdot y + Cin \cdot (x + y)$

$S = x'y'c + x'yc' + xy'c' + xyc$

$= x \oplus y \oplus c$

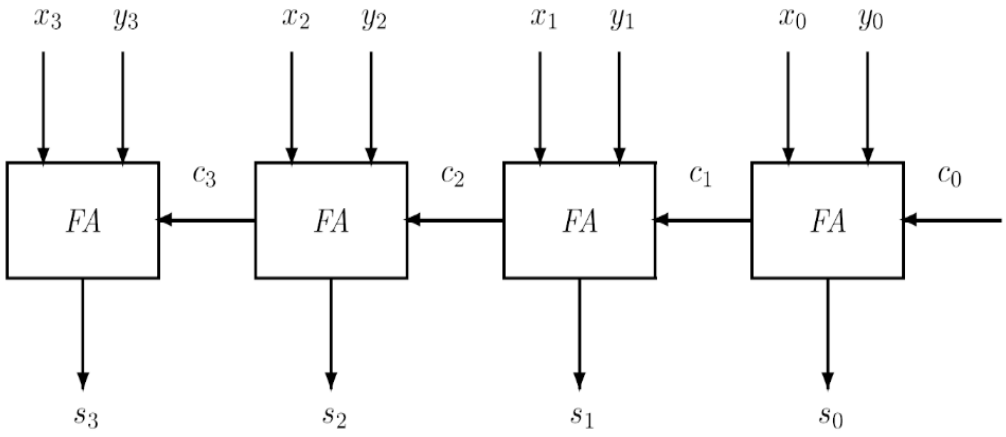


In general, for bit *i*:

$$c_{i+1} = x_i y_i + c_i (x_i + y_i)$$

where c_{i+1} = Cout, c_i = Cin

Parallel Adder: Ripple Carry



- In a parallel arithmetic unit
  - All **2n** input bits available at the same time
  - Carry propagates from the FA to the right to FA to the left
  - Carries ripple through all **n** FAs before we can claim that the sum outputs are correct and may be used in further calculations
- Each FA has a finite delay

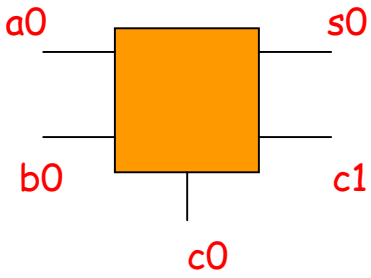
Fast Carry Propagation; Carry Look Ahead

During addition, the carry can trigger a "ripple" from the LSB to the MSB. This slows down the speed of addition.

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 +
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Calculate the max time it takes to complete a 32-bit addition if each stage takes 1 ns. How to overcome this?

Consider the following:



c1 = a0.b0 + a0.c0 + b0.c0

= a0.b0 + (a0 + b0).c0

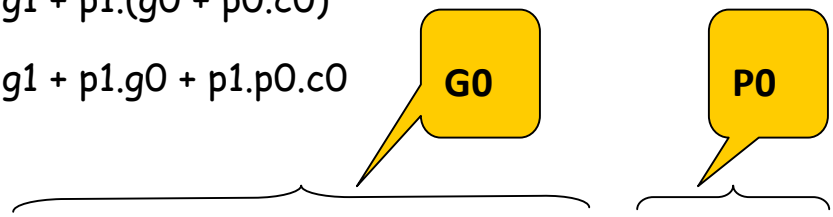
= g0 + p0.c0

(where **g0 = a0.b0**, **p0 = a0+b0**)

c2 = a1.b1 + (a1 + b1).c1

= g1 + p1.(g0 + p0.c0)

= g1 + p1.g0 + p1.p0.c0

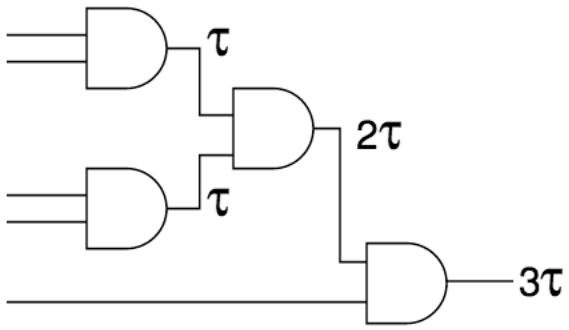


c4 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0 + p3.p2.p1.p0. c0

We could calculate c32 in this way.

Gates are limited to two inputs

- $C_4 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$



What if there were 6 inputs?
What if there were 7 inputs?
What if there were 8 inputs?
What if there were 9 inputs?

You can always use a two-level circuit to generate c32, which will speed-up addition (do 32-bit addition in 2 ns), but it is impractical due to the complexity.

Many practical circuits use a two-phase approach. Consider the example of a 16-bit adder, designed from four 4-bit adders. Let

$G0 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0$

$G1 = g7 + p7.g6 + p7.p6.g5 + p7.p6.p5.g4$

$G2 = g11 + p11.g10 + p11.p10.g9 + p11.p10.p9.g8$

$G3 = g15 + p15.g14 + p15.p14.g13 + p15.p14.p13.g12$

$P0 = p3.p2.p1.p0$

$P1 = p7.p6.p5.p4$

$P2 = p11.p10.p9.p8$

$P3 = p15.p14.p13.p12$

Then if C1, C2, C3, C4 are the output carry bit from the 1^{st}, 2^{nd}, 3^{rd}, 4^{th} 4-bit adders, then we can write

C1 = G0 + P0.c0

C2 = G1 + P1.C1 = G1 + P1.G0 + P1.P0.c0

C3 = G2 + P2.C2 = G2 + P2.G1 + P2.P1.G0 + P2.P1.P0.c0

C4 = G3 + P3.C3 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.G0 + P3.P2.P1.P0.c0

How does it help? **Count the number of levels**. The smaller is this number, the faster is the implementation

This is implemented in the **carry look-ahead adder**.

There are other implementations too.

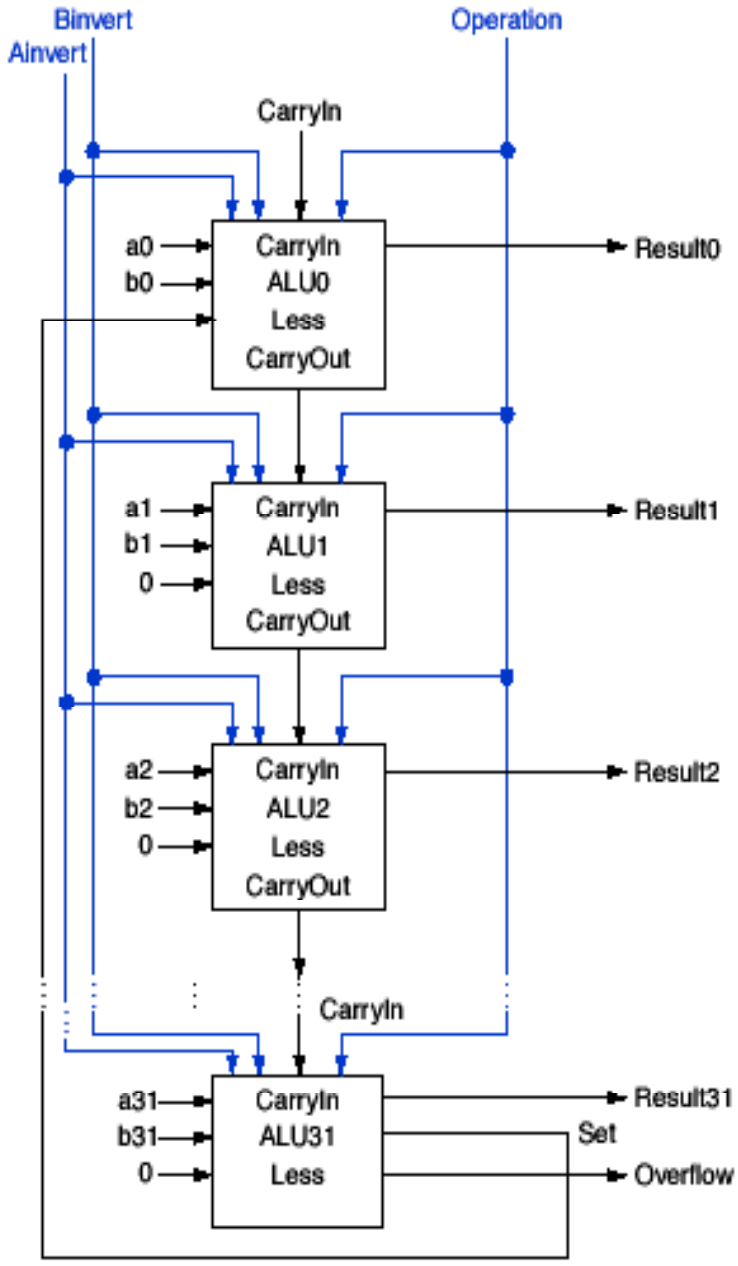
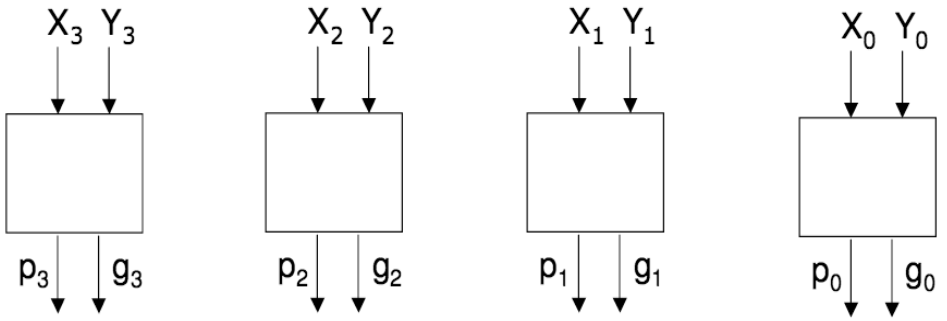


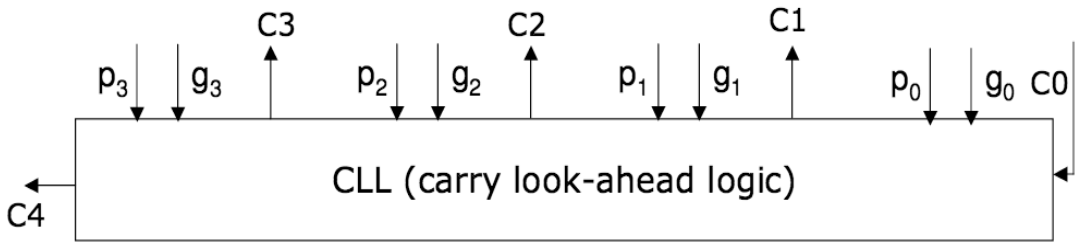
FIGURE B.5.11 A 32-bit ALU constructed from the 31 copies of the 1-bit ALU in the top of Figure B.5.10 and one 1-bit ALU in the bottom of that figure. The Less inputs are connected to 0 except for the least significant bit, which is connected to the Set output of the most significant bit. If the ALU performs a – b and we select the input 3 in the multiplexor in Figure B.5.10, then Result = 0 . . . 001 if a < b, and Result = 0 . . . 000 otherwise.

Delay of Carry Look Ahead Adders

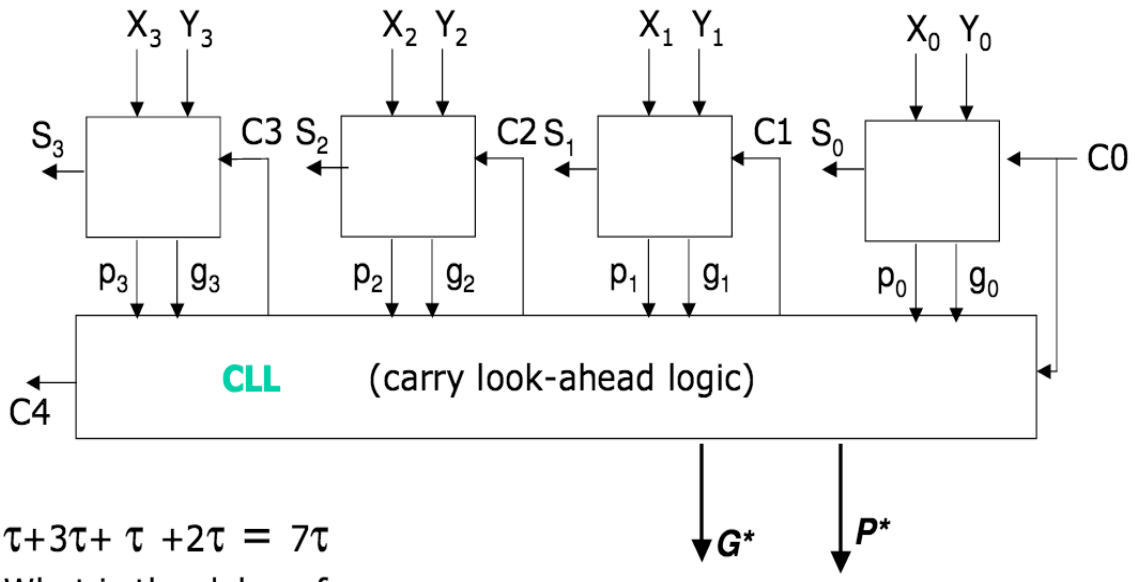
- Let τ be the delay of a gate



- If inputs are available at time t=0, when are p and g signals available?



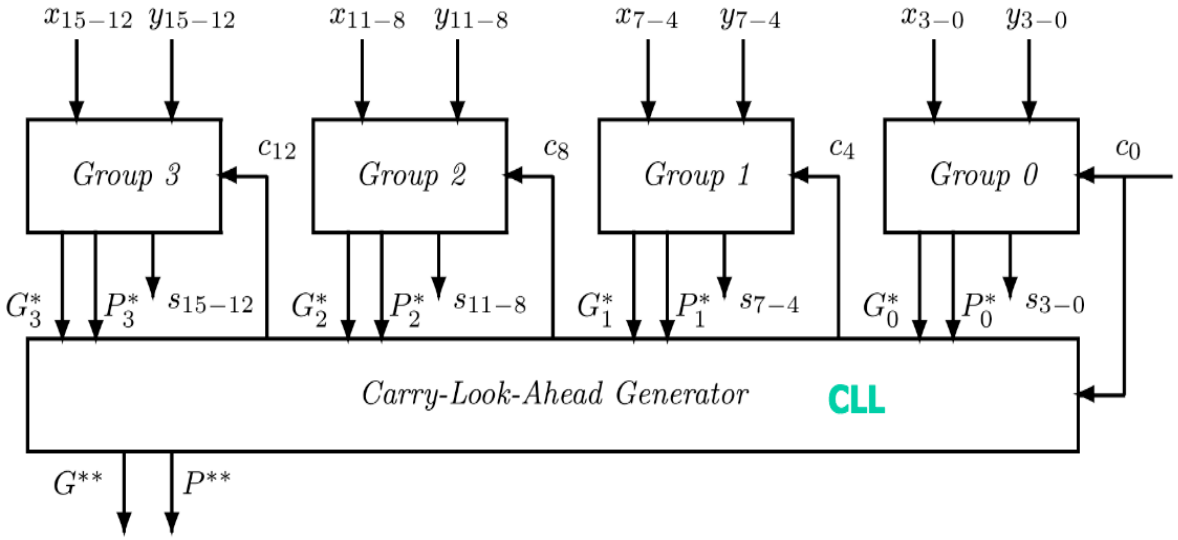
Total Delay



- τ+3τ+ τ +2τ = 7τ
- What is the delay of a 5 bit CLA?
- 6 bit CLA? 7 bit CLA?
- 8 bit CLA?

$$G^* = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$
$$P^* = P_0P_1P_2P_3.$$

2-level Carry Look Ahead (16-bit)



- n=16 - 4 groups, 4-bit each

$$
\begin{aligned}
c_4 &= G_0^* + c_0 P_0^*, \\
c_8 &= G_1^* + G_0^* P_1^* + c_0 P_0^* P_1^*, \\
c_{12} &= G_2^* + G_1^* P_2^* + G_0^* P_1^* P_2^* + c_0 P_0^* P_1^* P_2^*
\end{aligned}
$$

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.