

Digital Logic

'22H2

송 인 식

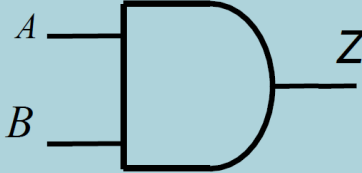
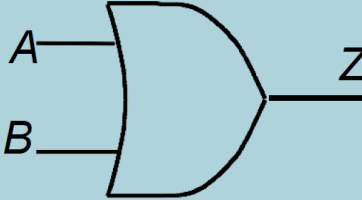
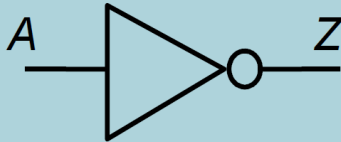
Outline

- Boolean Algebra
- Digital Logic

Boolean Algebra

- Boolean algebra is the basic math used in digital circuits and computers.
- A Boolean variable takes on only 2 values: $\{0,1\}$, $\{T,F\}$, $\{Yes, No\}$, etc.
- There are 3 fundamental Boolean operations:
 - AND, OR, NOT

Fundamental Boolean Operations

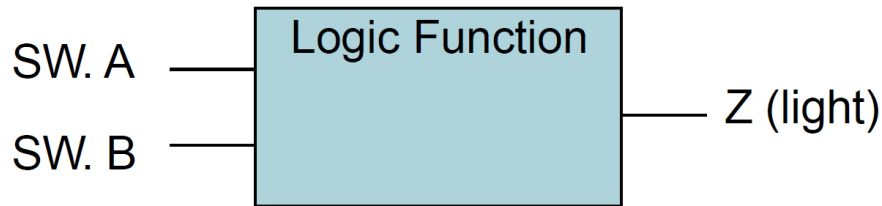
AND	OR	NOT																																				
																																						
$Z=A*B \text{ (AB)}$	$Z=A+B$	$Z=\bar{A}$																																				
Truth Table	Truth Table	Truth Table																																				
<table><tr><th>A</th><th>B</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Z	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>A</th><th>B</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Z	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>A</th><th>Z</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Z	0	1	1	0
A	B	Z																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
A	B	Z																																				
0	0	0																																				
0	1	1																																				
1	0	1																																				
1	1	1																																				
A	Z																																					
0	1																																					
1	0																																					

Boolean Algebra

- A *truth table* specifies output signal logic values for every possible combination of input signal logic values
- In evaluating Boolean expressions, the *Operation Hierarchy* is: 1)NOT 2)AND 3) OR. Order can be superseded using (...)
- Example: $A = T, B = F, C = T, D = T$
 - What is the value of $Z = (\bar{A} + B) \cdot (C + \bar{B} \cdot D)$?
$$Z = (\bar{T} + F) \cdot (C + \bar{B} \cdot D) = (F + F) \cdot (C + \bar{B} \cdot D)$$
$$= F \cdot (C + \bar{B} \cdot D) = F$$

Deriving Logic Expressions from Truth Tables

Light must be ON when both switches A and B are OFF, or when both of them are ON.



Truth Table:

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

- What is the Boolean expression for Z?

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

Minterms and Maxterms

- Minterms
 - AND term of all input variables
 - For variables with value 0, apply complements
- Maxterms
 - OR factor with all input variables
 - For variables with value 1, apply complements

A	B	Z	Minterms	Maxterms
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	1	AB	$\bar{A} + \bar{B}$

Minterms and Maxterms

- A function with n variables has 2^n minterms (and maxterms) – exactly equal to the number of rows in truth table
- Each minterm is true for exactly one combination of inputs
- Each Maxterm is false for exactly one combination of inputs

A	B	Z	Minterms	Maxterms
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	1	AB	$\bar{A} + \bar{B}$

Equivalent Logic Expressions

- Two equivalent logic expressions can be derived from Truth Tables:

1. *Sum-of-Products* (SOP) expressions:

- Several AND terms OR'd together, e.g.

$$ABC\bar{C} + \bar{A}BC + ABC$$

2. *Product-of-Sum* (POS) expressions:

- Several OR terms AND'd together, e.g.

$$(\bar{A} + \bar{B} + C)(A + \bar{B} + C)$$

Rules for Deriving SOP Expressions

1. Find each row in TT for which output is 1 (rows 1 & 4)
2. For those rows write a **minterm** of all input variables.
3. OR together all **minterms** found in (2): Such an expression is called a **Canonical SOP**

A	B	Z	Minterms	Maxterms
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	1	AB	$\bar{A} + \bar{B}$

$$Z = \bar{A} \bar{B} + AB$$

Rules for Deriving POS Expressions

1. Find each row in TT for which output is 0 (rows 2 & 3)
2. For those rows write a **maxterm**.
3. AND together all **maxterms** found in (2): Such an expression is called a **Canonical POS**

A	B	Z	Minterms	Maxterms
0	0	1	$\bar{A}.\bar{B}$	$A+B$
0	1	0	$\bar{A}.B$	$A+\bar{B}$
1	0	0	$A.\bar{B}$	$\bar{A}+B$
1	1	1	AB	$\bar{A}+\bar{B}$

$$Z = (A+\bar{B})(\bar{A}+B)$$

CSOP and CPOS

- Canonical SOP: $Z = \bar{A}\bar{B} + AB$
- Canonical POS: $Z = (A + \bar{B})(\bar{A} + B)$
- Since they represent the same truth table, they should be identical
 - Verify that $Z = \bar{A}\bar{B} + AB \equiv (A + \bar{B})(\bar{A} + B)$
- CPOS and CSOP expressions for the same TT are logically equivalent. Both represent the same information.

Boolean Identities

- Useful for simplifying logic equations.

	(a)	(b)
1	$\overline{\overline{A}} = A$	$\overline{\overline{A}} = A$
2	$A + \text{false} = A \quad (A + 0 = A)$	$A \cdot \text{true} = A \quad (A \cdot 1 = A)$
3	$A + \text{true} = \text{true} \quad (A + 1 = 1)$	$A \cdot \text{false} = \text{false} \quad (A \cdot 0 = 0)$
4	$A + A = A$	$A \cdot A = A$
5	$A + \overline{A} = \text{true} \quad (A + \overline{A} = 1)$	$A \cdot \overline{A} = \text{false} \quad (A \cdot \overline{A} = 0)$
6	$A + B = B + A$	$A \cdot B = B \cdot A$
7	$A + B + C = (A + B) + C = A + (B + C)$	$A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$
8	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + B \cdot C = (A + B)(A + C)$
9	$\overline{A + B} = \overline{A} \cdot \overline{B}$	$\overline{A \cdot B} = \overline{A} + \overline{B}$
10	$A \cdot B + A \cdot \overline{B} = A$	$(A + B)(A + \overline{B}) = A$
11	$A + A \cdot B = A$	$A(A + B) = A$
12	$A(\overline{A} + B) = A \cdot B$	$A + \overline{A} \cdot B = A + B$
13	$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$

Duals

Boolean Identities

- The right side is the dual of the left side
 1. Duals formed by replacing

AND \rightarrow OR

OR \rightarrow AND

0 \rightarrow 1

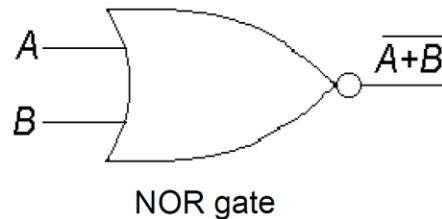
1 \rightarrow 0

2. The dual of any true statement in Boolean algebra is also a true statement.

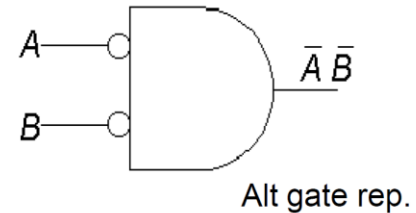
Boolean Identities

- De Morgan's laws very useful: 9a and 9b

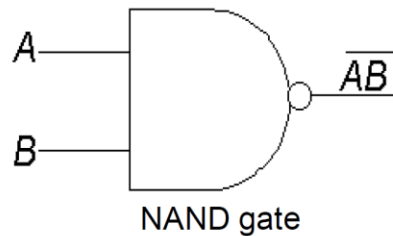
$$\overline{A+B} = \overline{A}.\overline{B}$$



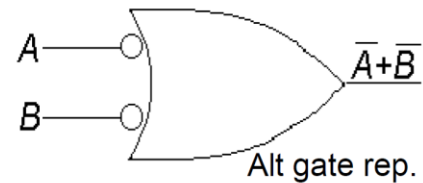
≡



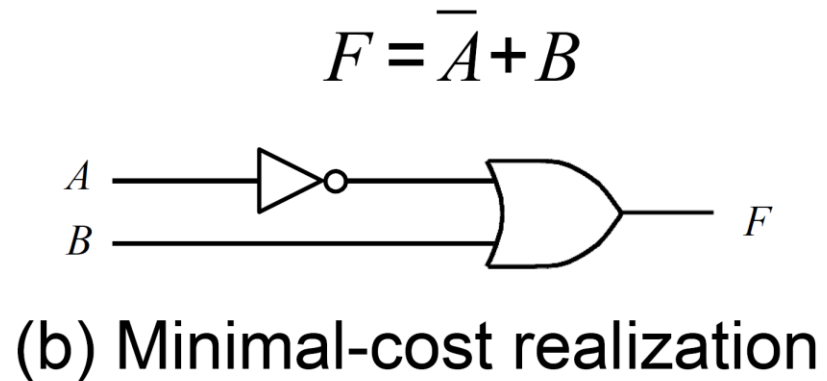
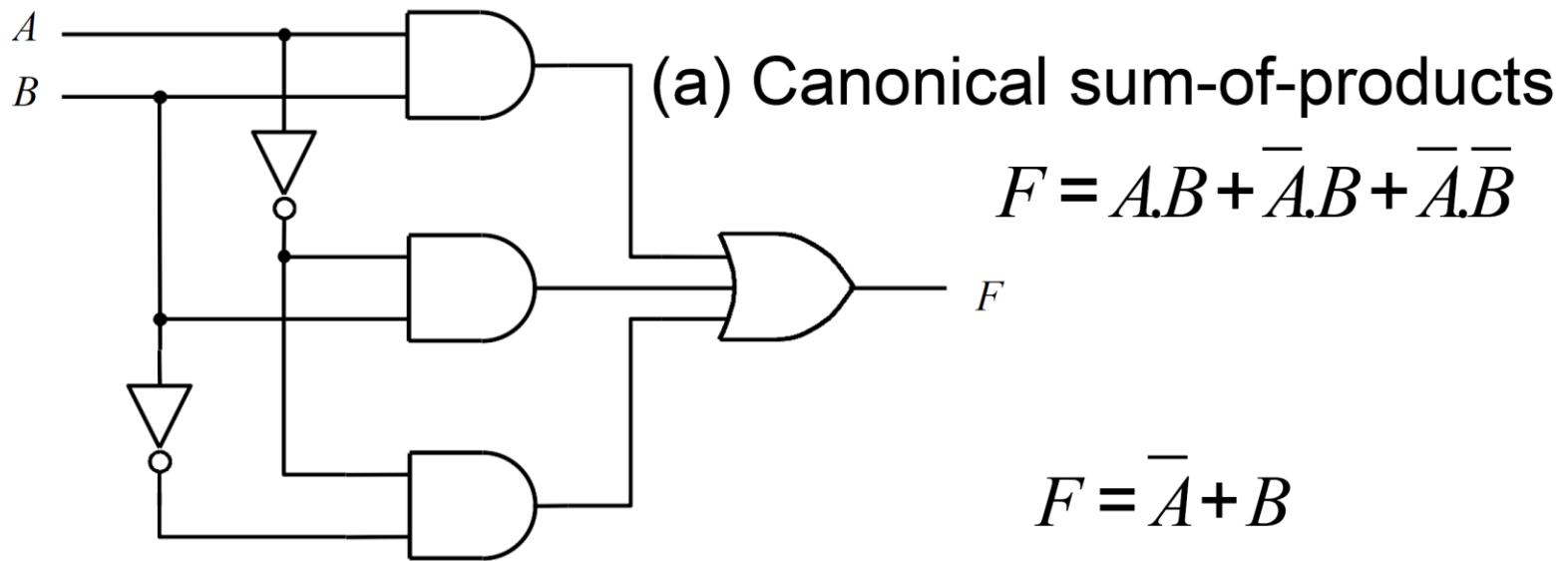
$$\overline{AB} = \overline{A} + \overline{B}$$



≡



Simplifying Logic Equations – Why?



Simplifying Logic Equations

- Simplifying logic expressions can lead to using smaller number of gates (parts) to implement the logic expression
- Can be done using
 - Boolean Identities (algebraic)
 - Karnaugh Maps (graphical)
- A minimum SOP (MSOP) expression is one that has no more AND terms or variables than any other equivalent SOP expression.
- A minimum POS (MPOS) expression is one that has no more OR factors or variables than any other equivalent POS expression.
- There may be several MSOPs of an expression

Example of Using Boolean Identities

- Find an MSOP for

$$\begin{aligned} F &= \overline{X}W + Y + \overline{Z}(Y + \overline{X}W) \\ &= \overline{X}W + Y + \overline{Z}Y + \overline{Z}\overline{X}W \\ &= \overline{X}W(1 + \overline{Z}) + Y(1 + \overline{Z}) \\ &= \overline{X}W + Y \end{aligned}$$

Outline

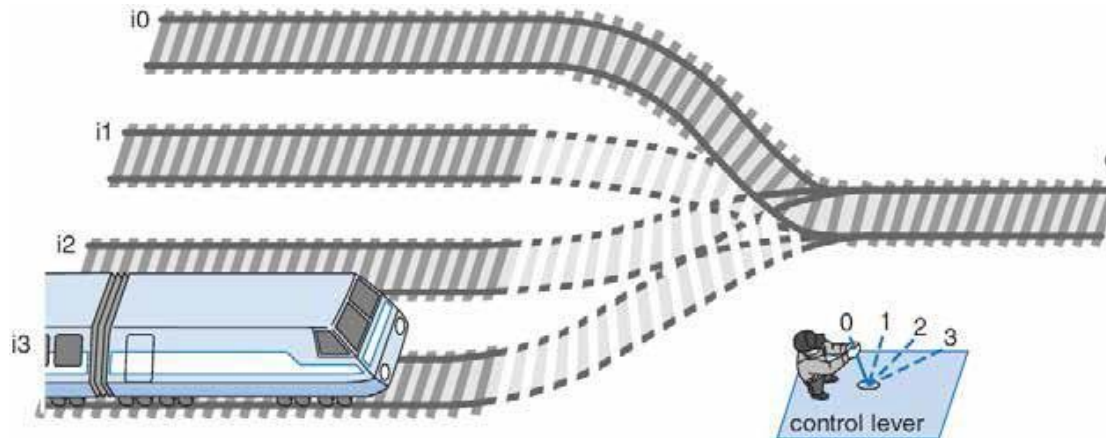
- Boolean Algebra
- Digital Logic

Digital Circuit Classification

- Combinational circuits
 - Output depends only solely on the current combination of circuit inputs
 - Same set of input will always produce the same outputs
 - Consists of AND, OR, NOR, NAND, and NOT gates
- Sequential circuits
 - Output depends on the current inputs and state of the circuit (or past sequence of inputs)
 - Memory elements such as flip-flops and registers are required to store the "state"
 - Same set of input can produce completely different outputs

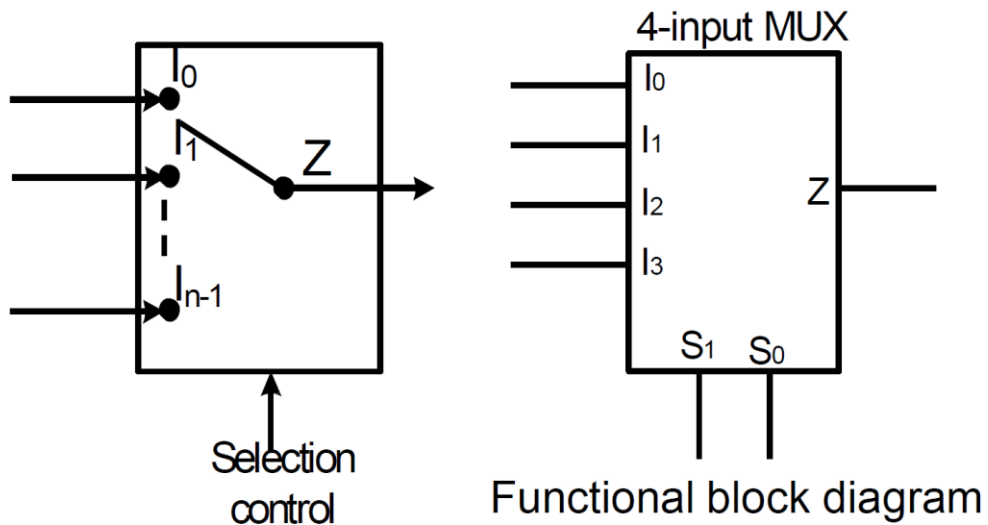
Multiplexer

- A multiplexer (MUX) selects data from one of N inputs and directs it to a single output, just like a railyard switch
 - 4-input Mux needs 2 select lines to indicate which input to route through
 - N-input Mux needs $\log_2(N)$ selection lines



Multiplexer (2)

- An example of 4-input Mux

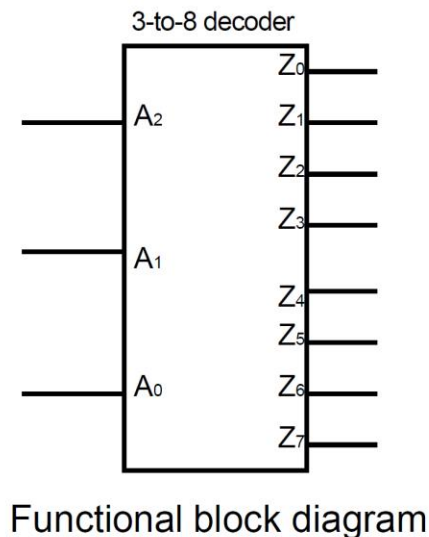


S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Truth Table

Decoder

- A decoder is a circuit element that will decode an N-bit code.
- It activates an appropriate output line as a function of the applied N-bit input code

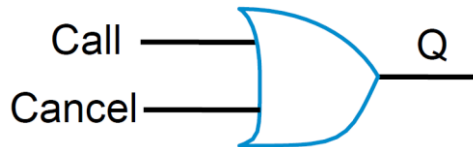


Truth Table

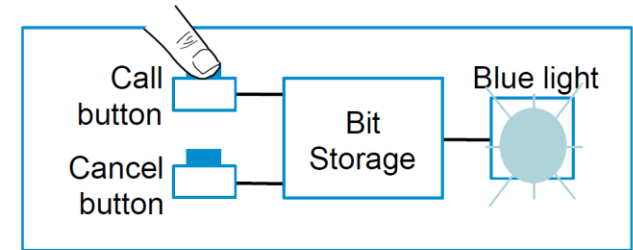
A ₂	A ₁	A ₀	Z ₀	Z ₁	Z ₂	Z ₃	Z ₄	Z ₅	Z ₆	Z ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Why Bit Storage ?

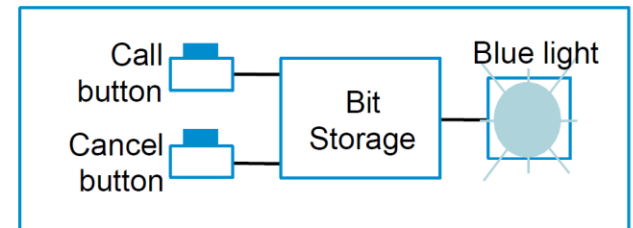
- Flight attendant call button
 - Press call: light turns on
 - **Stays on** after button released
 - Press cancel: light turns off
 - Logic gate circuit to implement this?



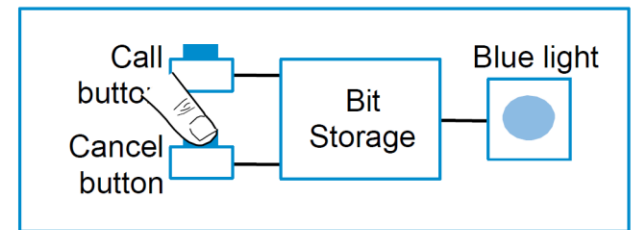
Doesn't work. $Q=1$ when $Call=1$, but doesn't stay 1 when $Call$ returns to 0
Need some form of "memory" in the circuit



1. Call button pressed – light turns on



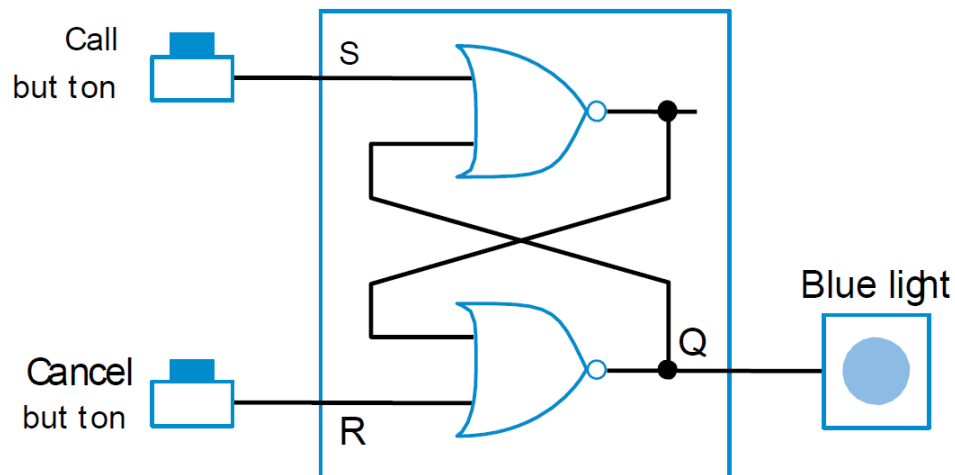
2. Call button released – light **stays on**



3. Cancel button pressed – light turns off

Bit Storage Using SR Latch

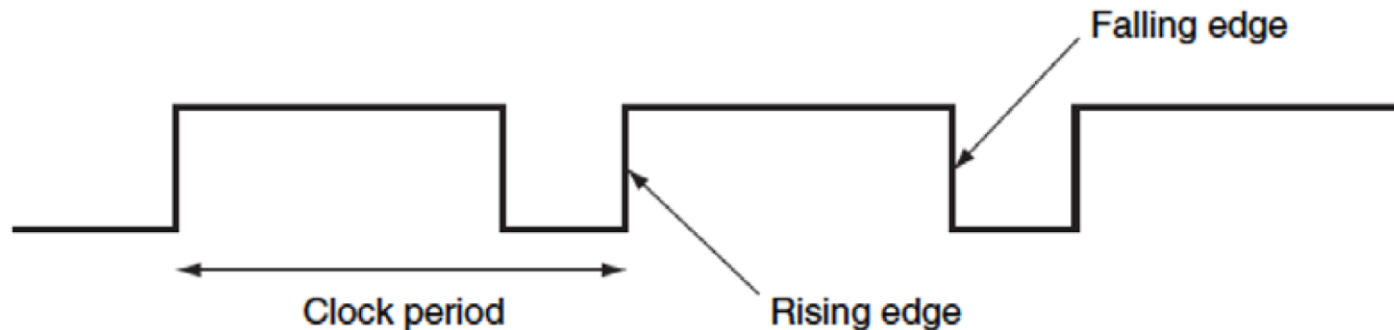
- Simplest memory elements are Latch and Flip-Flops
- SR (set-reset) latch is an ***un-clocked*** latch
 - Output $Q=1$ when $S=1, R=0$ (set condition)
 - Output $Q=0$ when $S=0, R=1$ (reset condition)
 - Problem: Q is undefined if $S=1$ and $R=1$



Clocks

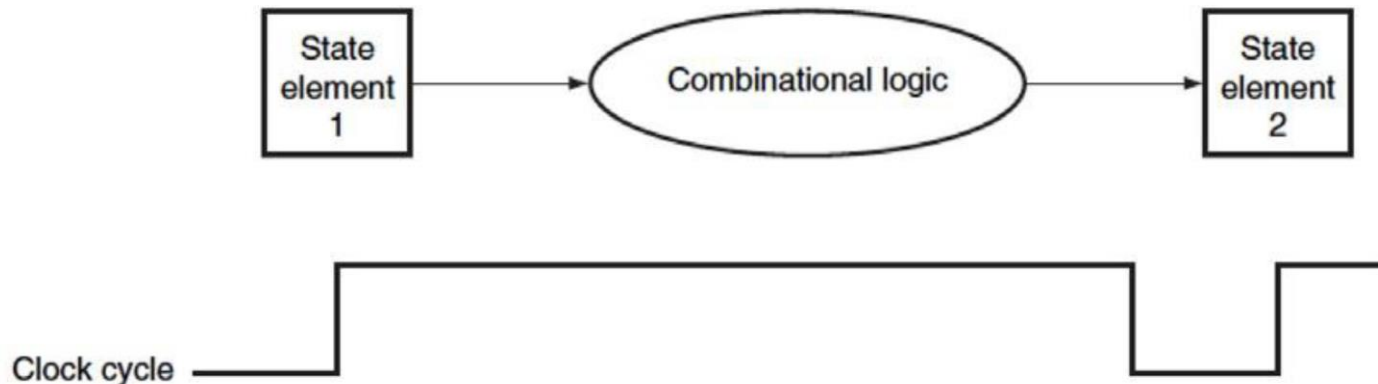
- Clock period: time interval between pulses
 - example: period = 20 ns
- Clock frequency: $1/\text{period}$
 - example: frequency = $1 / 20 \text{ ns} = 50 \text{ MHz}$
- Edge-triggered clocking: all state changes occur on a clock edge.

Freq	Period
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns



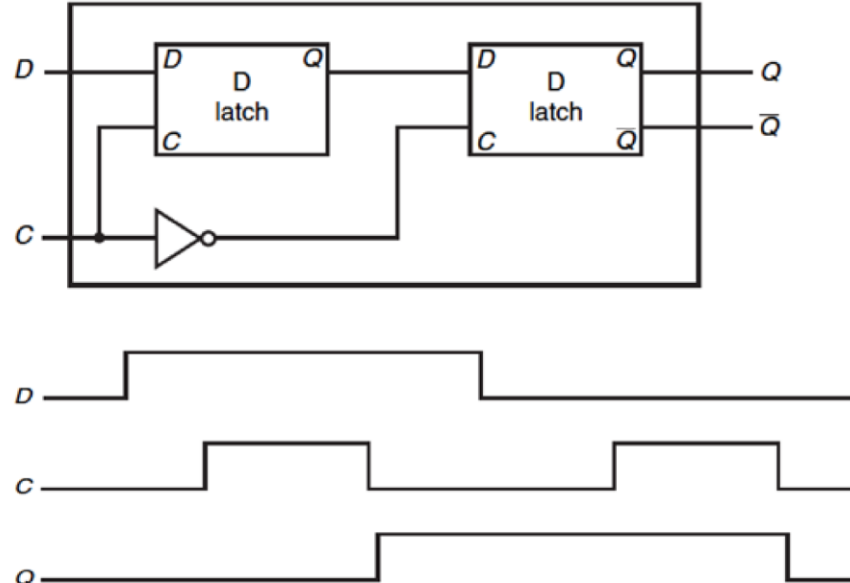
Clock and Change of State

- Clock controls when the state of a memory element changes
- ***Edge-triggered clocking***. all state changes occur on a clock edge.



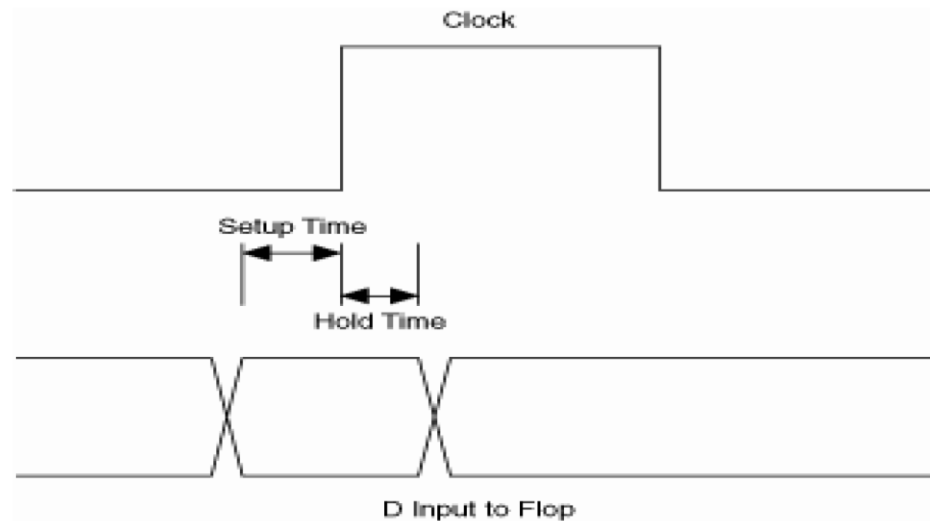
Clock Edge Triggered Bit Storage

- Flip-flop -Bit storage that stores on clock edge, not level
- D Flip-flop
 - Two latches, master and slave latches.
 - Output of the first goes to input of second, slave latch has inverted clock signal (falling-edge trigger)



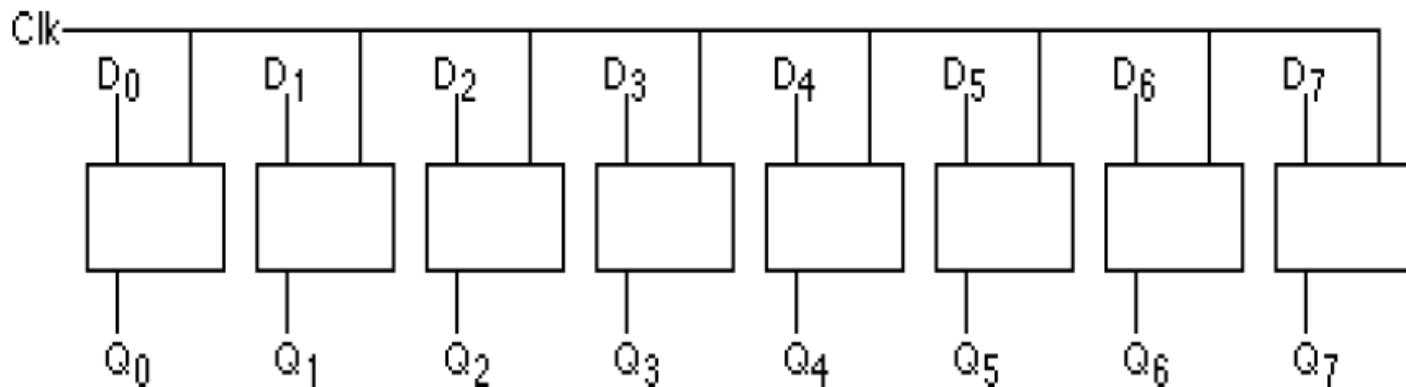
Setup and Hold Time

- Setup time
 - The minimum amount of time the data signal should be held steady before the clock edge arrives.
- Hold time
 - The minimum amount of time the data signal should be held steady after the clock edge.



N-Bit Register

- Cascade N number of D flip-flops to form a N -bit register
- An example of 8-bit register formed by 8 edge-triggered D flip-flops



Half Adders

- Need to add bits $\{0,1\}$ of A_i and B_i
- Associate

– binary bit 0 \leftrightarrow logic value F (0)

– binary bit 1 \leftrightarrow logic value T (1)

$$\begin{array}{l} C_{i+1} \\ A: A_n \dots A_{i+1} A_i \dots A_0 \\ B: B_n \dots B_{i+1} B_i \dots B_0 \\ S_i \end{array}$$

- This leads to the following truth table

A_i	B_i	Sum_i	$Carry_{i+1}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

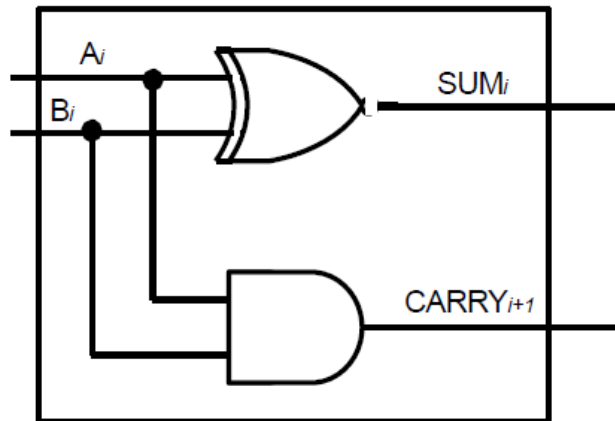
$$SUM_i = \bar{A}_i \bar{B}_i + A_i \bar{B}_i = A_i \oplus B_i$$

$$CARRY_{i+1} = A_i B_i$$

Half Adder Circuit

$$SUM_i = \overline{A_i}B_i + A_i\overline{B_i} = A_i \oplus B_i$$

$$CARRY_{i+1} = A_i B_i$$



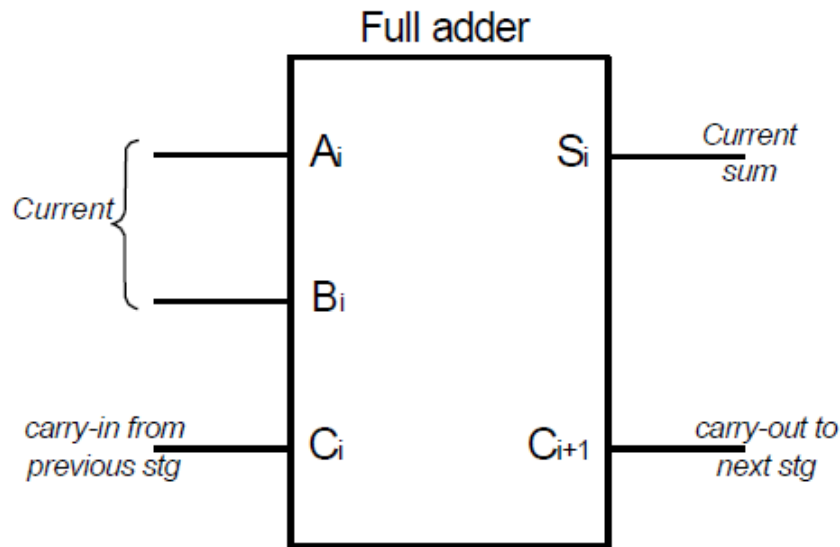
Half Adder Limitations

- Half adder circuits do not suffice for general addition because they do not include the carry bit from the previous stage of addition, e.g.

$$\begin{array}{rcccc} \textit{Carry} & & 0 & 1 & 1 & 0 \\ A & & 0 & 1 & 1 & 0 \\ B & + & 0 & 0 & 1 & 1 \\ \hline \textit{SUM} & & 1 & 0 & 0 & 1 \end{array}$$

Full Adders (1-Bit ALU)

- Full adders can use the carry bit from the previous stage of addition



A_i	B_i	C_i	S_i	C_{i+1}
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

Full Adder Logic Expressions

Sum

$$\begin{aligned}\text{SUM} &= \overline{A_i}\overline{B_i}C_i + \overline{A_i}B_i\overline{C_i} + A_i\overline{B_i}\overline{C_i} + A_iB_iC_i \\ &= \overline{A_i}(\overline{B_i}C_i + B_i\overline{C_i}) + A_i(\overline{B_i}\overline{C_i} + B_iC_i) \\ &= A_i(B_i \oplus C_i) + A_i(\overline{B_i \oplus C_i}) \\ &= A_i \oplus B_i \oplus C_i\end{aligned}$$

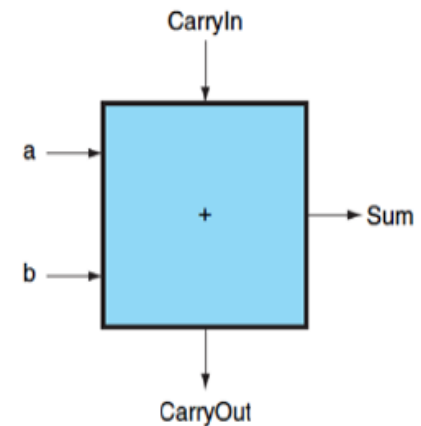
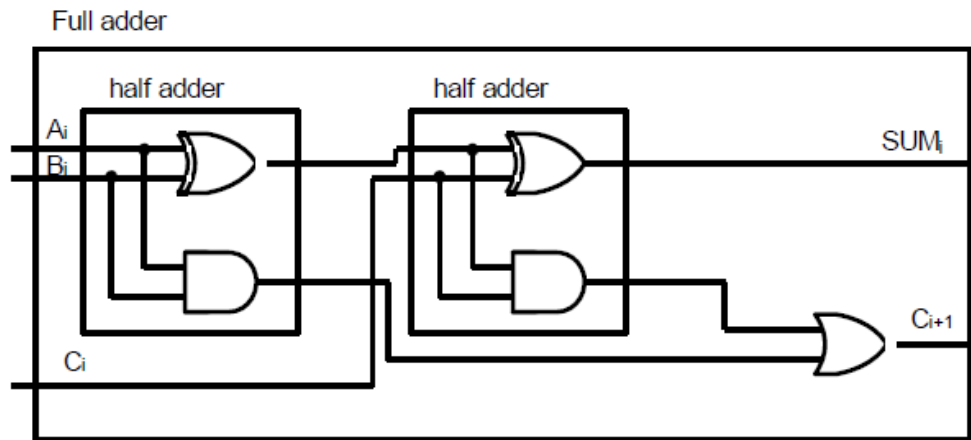
Carry

$$\begin{aligned}C_{i+1} &= A_iB_i + \overline{A_i}\overline{B_i}C_i + \overline{A_i}B_iC_i \\ &= A_iB_i + C_i(\overline{A_i}\overline{B_i} + \overline{A_i}B_i) \\ &= A_iB_i + C_i(A_i \oplus B_i)\end{aligned}$$

Full Adder Circuit

$$SUM = (A_i \oplus B_i) \oplus C_i$$

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$

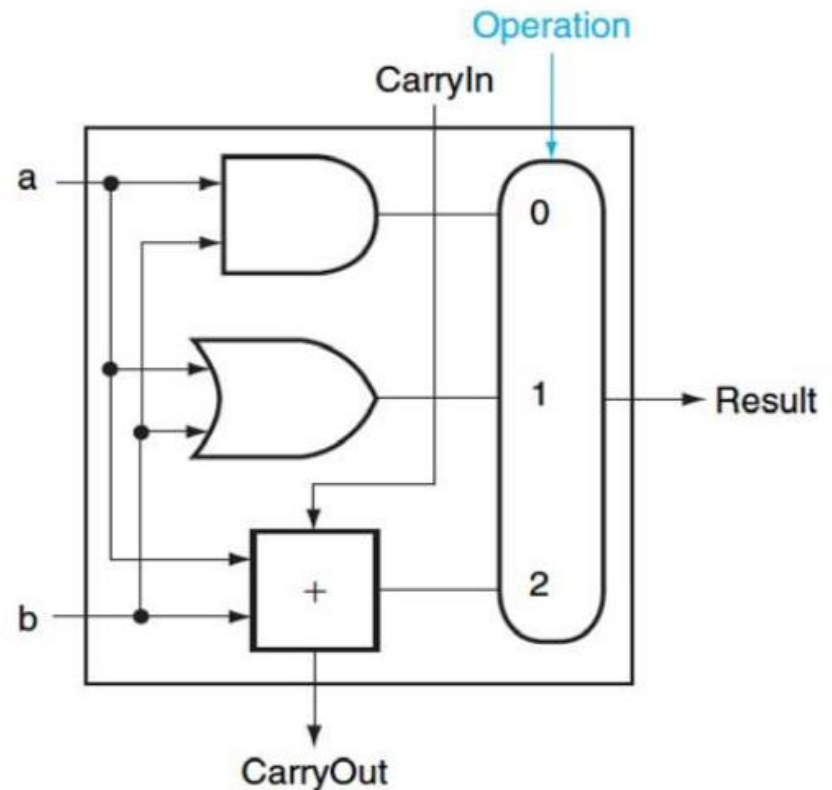


Note: A full adder adds 3 bits. Can also consider as first adding first two and then the result with the carry

Enhancement to 1-bit Adder(1)

- 1-bit ALU with AND, OR, and addition
 - Supplemented with AND and OR gates
 - A multiplexer controls which gate is connected to the output

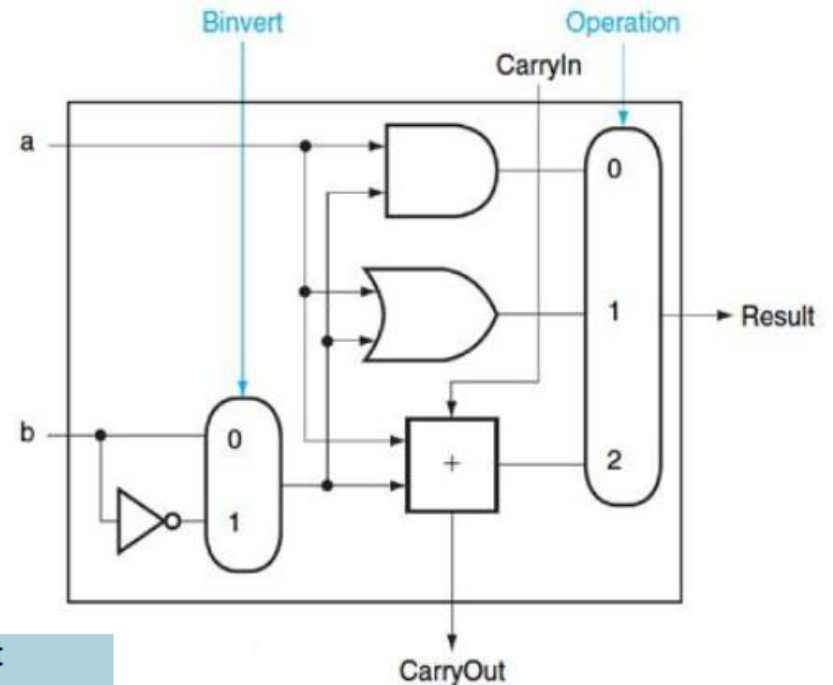
Operation	Result
00	AND
01	OR
10	Addition



Enhancement to 1-bit Adder(2)

- 1-bit ALU for subtraction
 - Subtraction is performed using 2's complement, i.e.

$$a - b = a + \bar{b} + 1$$

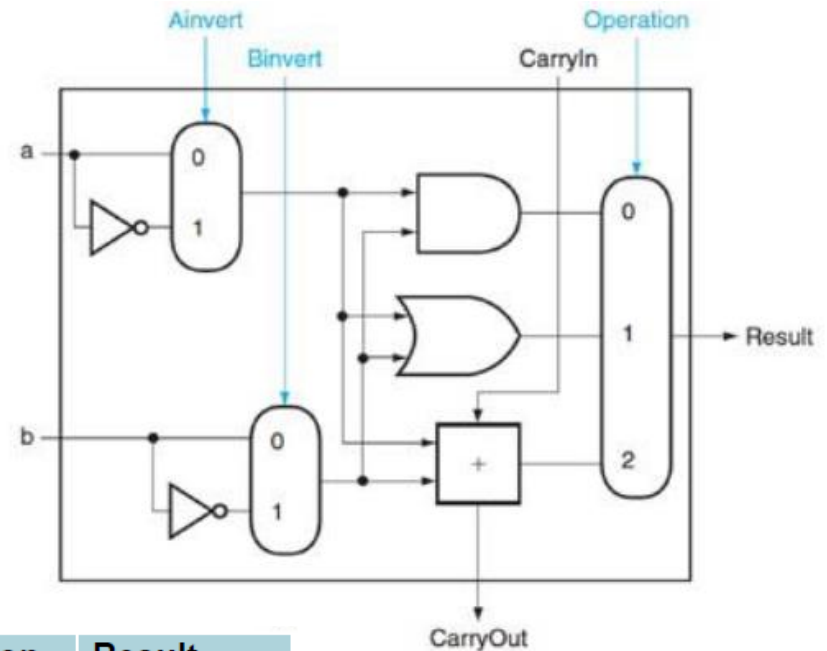


Binvert	CarryIn	Operation	Result
0	0	00	AND
0	0	01	OR
0	0	10	Addition
1	1	10	Subtraction

Enhancement to 1-bit Adder(3)

- 1-bit ALU for NOR operation
 - A MIPS ALU also needs a NOR function

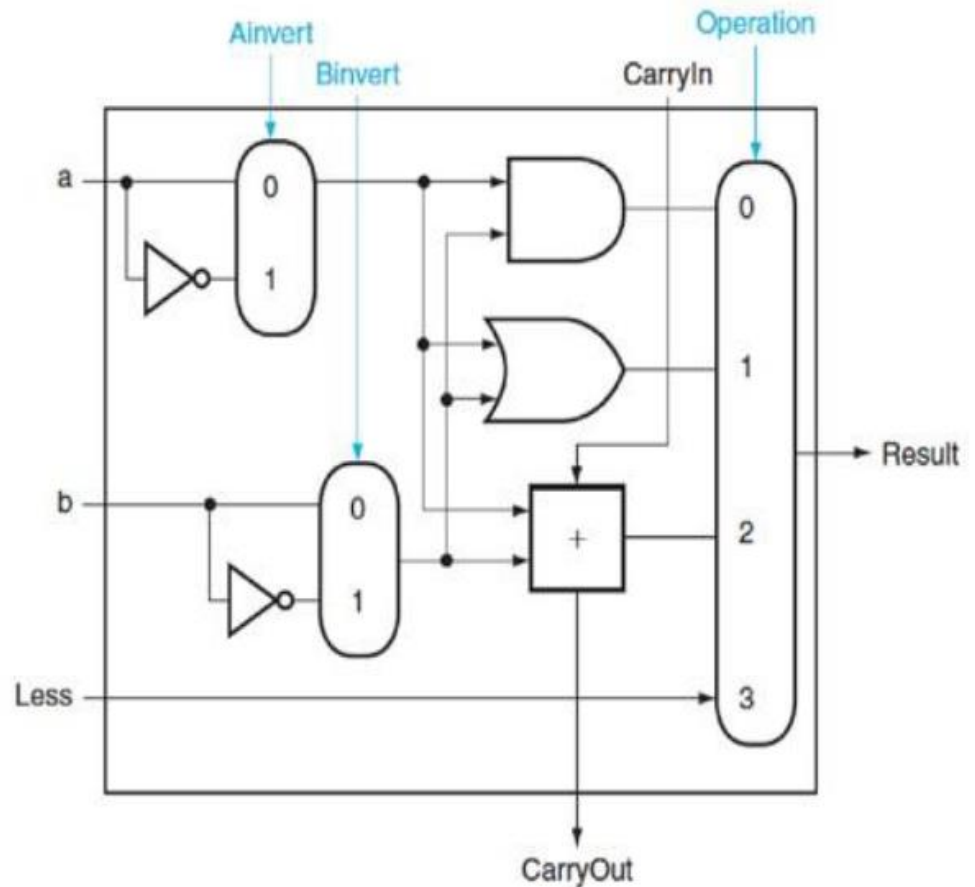
$$\overline{a + b} = \bar{a} \cdot \bar{b}$$



Ainvert	Binvert	CarryIn	Operation	Result
0	0	0	00	AND
1	1	0	00	NOR
0	0	0	01	OR
0	0	0	10	Addition
0	1	1	10	Subtraction

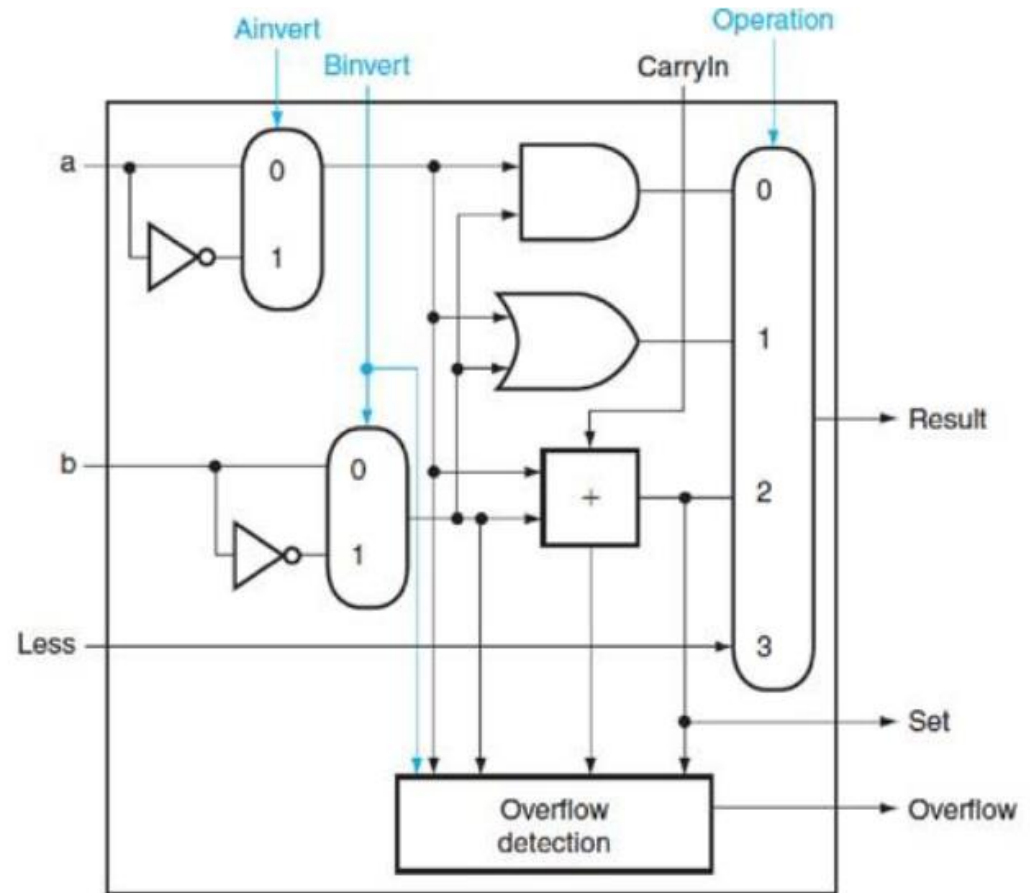
Enhancement to 1-bit Adder(4)

- 1-bit ALU for SLT operations
- `slt $s1, $s2, $s3`
 - If $(\$s2 < \$s3)$, $\$s1 = 1$,
else $\$s1 = 0$
- adding one input *less*
 - if $(a < b)$, set *less* to 1 or
if $(a - b) < 0$, set *less* to 1
 - If the result of
subtraction is negative,
set *less* to 1
- How to determine if
the result is negative?

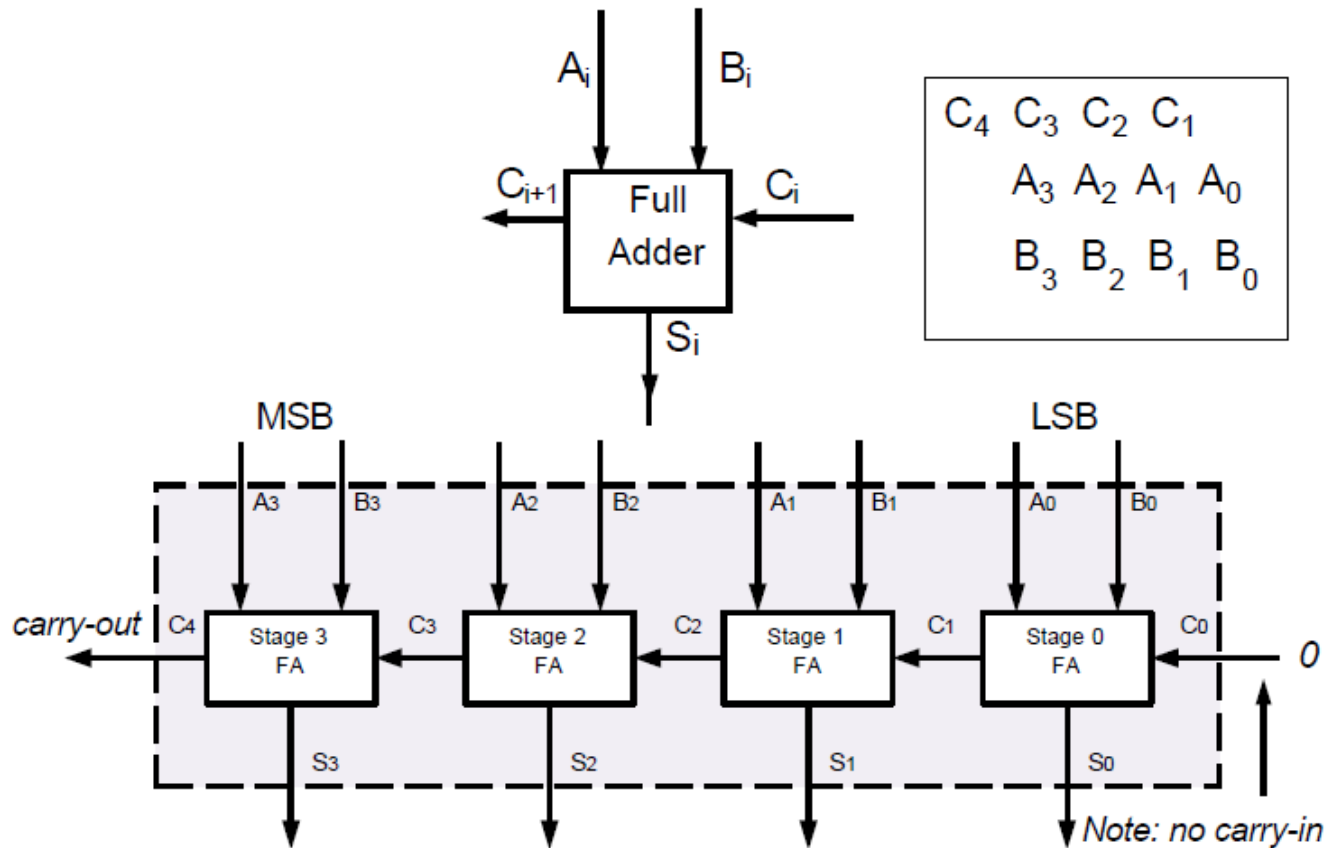


Enhancement to 1-bit Adder(5)

- How to determine if the result is negative?
 - Negative → Sign bit value=1
- Create a new output "Set" direct output from the adder and use only for slt
- An overflow detection is included for the most significant bit ALU



N-Bit Adders (Ripple Carry)



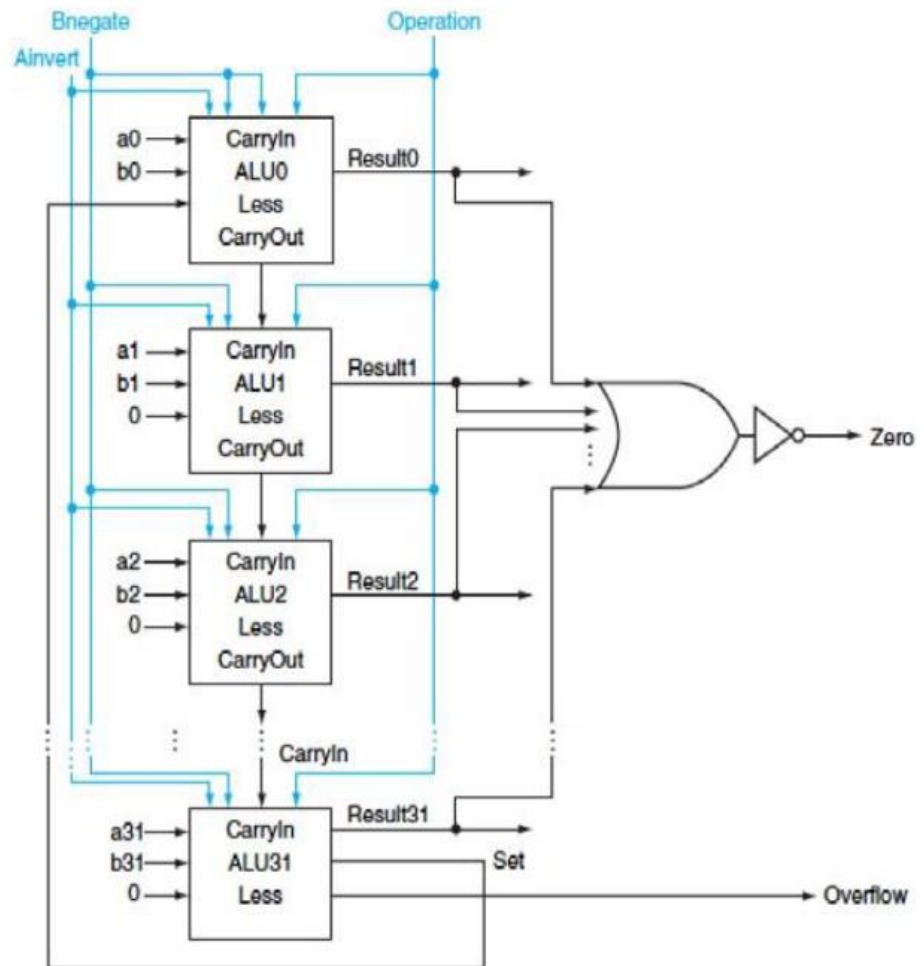
Ripple Carry Adders

- 4 FA's cascaded to form a 4-bit adder
- In general, N-FA's can be used to form a N-bit adder
- Carry bits have to propagate from one stage to the next. Inherent propagation delays associated with this
- Output of each FA is therefore not stable until the carry-in from the previous stage is calculated

32-Bit ALU

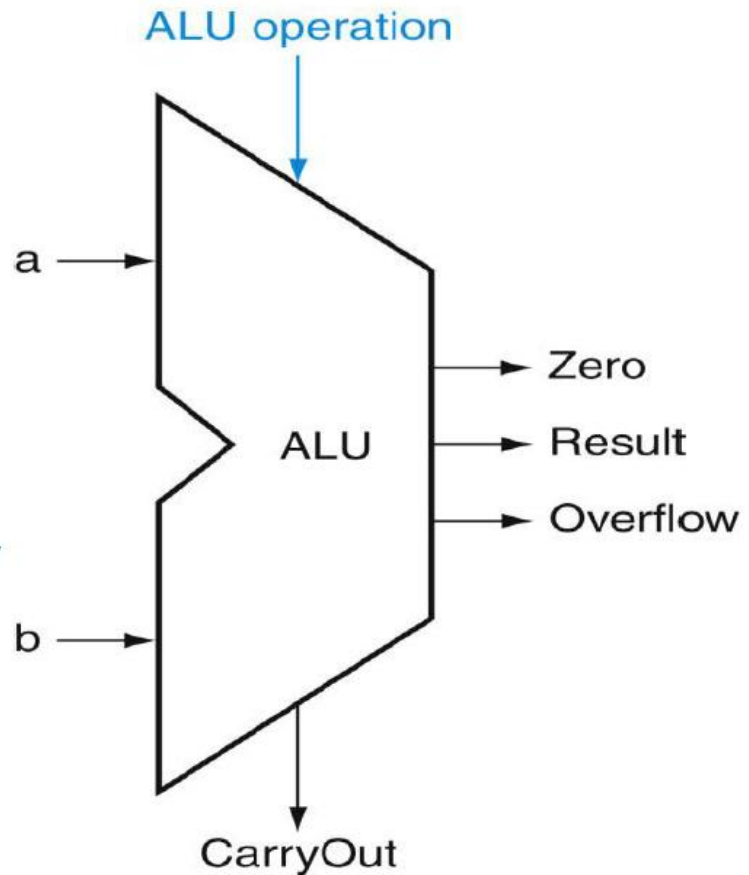
- OR and INV gates are added to support conditional branch instruction, i.e. test the result of $a-b$ if the result is 0.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



32-Bit ALU

- The symbol commonly used to represent an ALU
- This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder



Questions?