UNIT 3

Dr. S. Anbukkarasi
AP/CSE

ALU

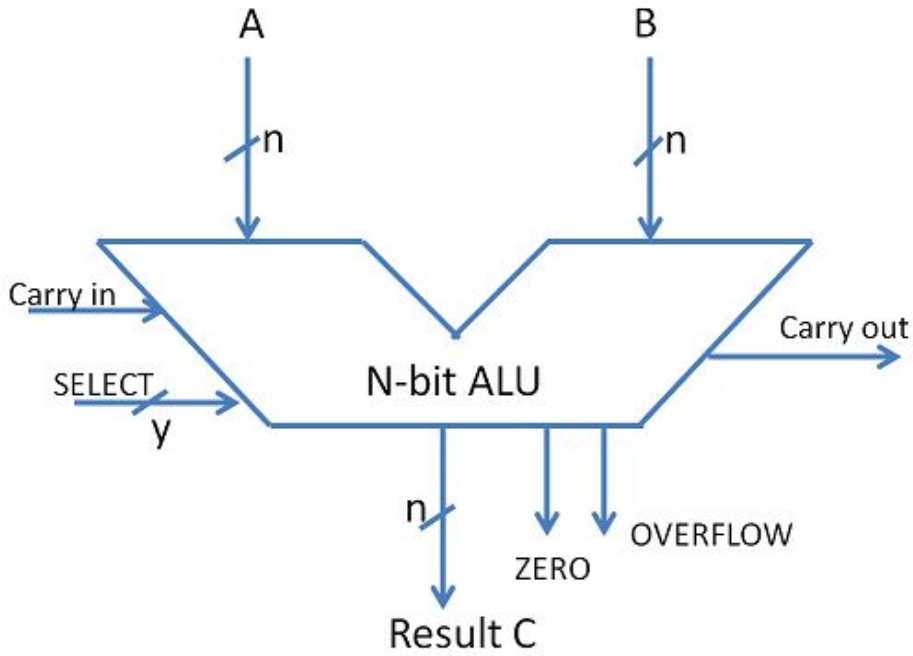
- ALU is the heart of any Central Processing Unit.
- A simple ALU is constructed with Combinational circuits.
- It is a digital circuit to do arithmetic operations like addition, subtraction, multiplication and division.
- Logical operations such as OR, AND, NOR etc.
- Data movement operations such as LOAD and STORE.
- Complex ALUs are designed for executing Floating point, decimal operations and other complex numerical operations. These are called as Co-processor and work in tandem with the main processor.
- The design specifications of ALU are derived from the Instruction set Architecture.
- The ALU must have the capabilities to execute the instructions of ISA.
- Modern CPUs have multiple ALU to improve the efficiency.

ALU includes the following Configurations :

- Instruction Set Architecture
- Accumulator
- Stack
- Register – Register architecture
- Register – Stack architecture
- Register – memory architecture

The size of input quantities of ALU is referred as **word length** of a computer

ALU Symbol


A
B
n
n
Carry in
Carry out
SELECT
y
N-bit ALU
n
ZERO
OVERFLOW
Result C

Gates Boolean Algebra

AND => C (output) = $A \cdot B$

OR => C (output) = $\overline{A + B}$

NAND => C (output) = $\overline{A \cdot B}$

NOR => C (output) = $\overline{A+B}$

XOR => C (Output) = $\overline{A}B + A\overline{B}$

XNOR => C (Output) = $\overline{AB} + AB$ (Complement of XOR)

ADDERS

The basic building blocks of an ALU in digital computers are Adders.

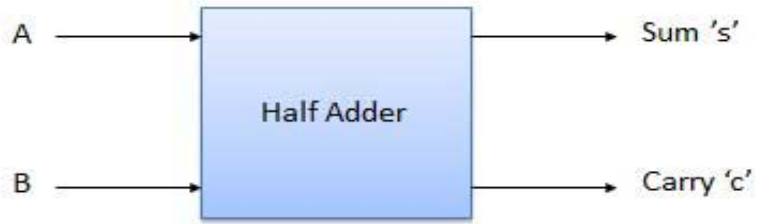
Types of Basic Adders are

- Half Adder
- Full Adder

- Parallel adders are nothing but cascade of full adders. The number of full adders used will depend on the number of bits in the binary digit which require to be added.
- Ripple carry adders are used when the input sequence is large. It is used to add two n-bit binary numbers.
- Carry look ahead adder is an improved version of Ripple carry adder.
- It generates the carry-in of each full adder simultaneously without causing any delay.

Half Adder

- Half adder is a combinational logic circuit with two input and two output. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs, carry and sum.



Truth Table and Circuit Diagram

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

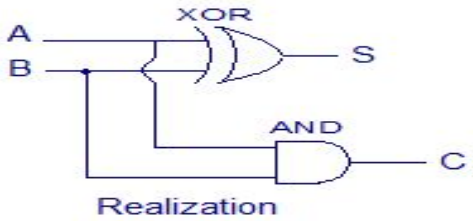
Truth table

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

$$\text{CARRY } C = A.B$$

Boolean Expression

Schematic



Realization

Full Adder

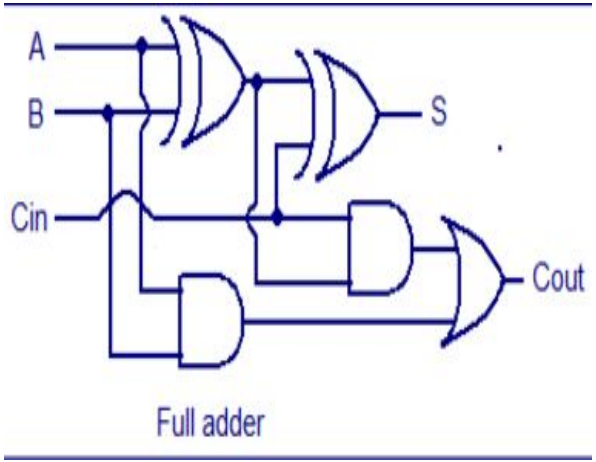
- Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

Truth Table and Circuit Diagram

$$S = A\overline{B}\overline{C} + \overline{A}\,\overline{B}C + ABC + \overline{A}B\overline{C}$$
$$= C\,(AB + \overline{A}\,\overline{B}) + \overline{C}\,(\overline{A}B + A\,\overline{B})$$
$$= C\,\overline{(\overline{A}B + A\,\overline{B})} + \overline{C}\,(\overline{A}B + A\,\overline{B})$$
$$= C\,\overline{(A \oplus B)} + \overline{C}\,(A \oplus B) \;= A \oplus B \oplus C$$

$$C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$
$$= (\overline{A}B + A\overline{B})\,C + AB\,(\overline{C} + C)$$
$$= (A \oplus B).C + AB.$$

Inputs			Output	
A	B	Cin	S	Co
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



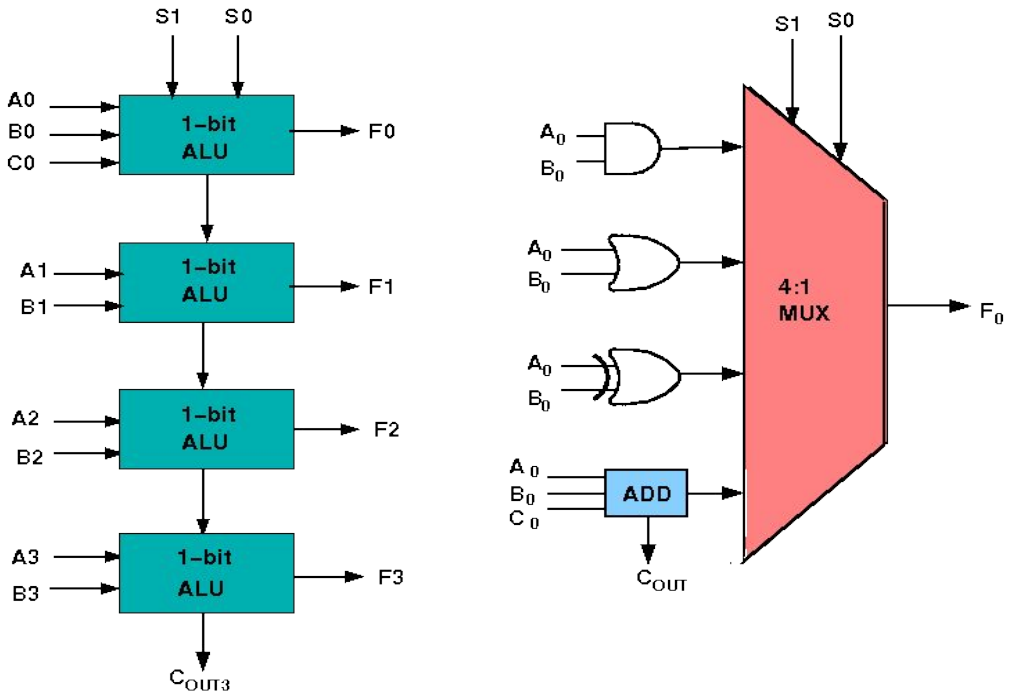
Full adder

Design of ALU

- ALU or Arithmetic Logical Unit is a digital circuit to do arithmetic operations like addition, subtraction, division, multiplication and logical operations like AND, OR, XOR, NAND, NOR etc.

- A simple block diagram of a 4 bit ALU for operations AND, OR,XOR and ADD is shown here :

The 4-bit ALU block is combined using 4 1-bit ALU block



Design Issues

The circuit functionality of a 1 bit ALU is shown here, depending upon the control signal S_1 and S_0 the circuit operates as follows:

for Control signal **S_1 = 0** , S_0 = 0, the output is **A And B**,
for Control signal **S_1 = 0** , S_0 = 1, the output is **A Or B**,
for Control signal **S_1 = 1** , S_0 = 0, the output is **A Xor B**,
for Control signal **S_1 = 1** , S_0 = 1, the output is **A Add B**.

16 bit ALU Design

Design the 16-bit Arithmetic Logic Unit (ALU) shown in Figure 1. The function table of the ALU is shown in Table 1.

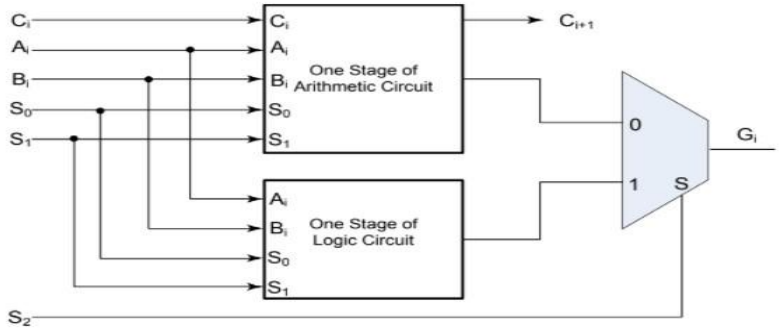


Figure 1: 16-bit ALU

Table 1: Function table of the ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + \underline{B} + 1$	Add with carry input of 1
0	1	0	0	$G = A + \underline{B}$	A plus 1's complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	0	0	X	$G = A \wedge B$	AND
1	0	1	X	$G = A \vee B$	OR
1	1	0	X	$G = \underline{A} \oplus B$	XOR
1	1	1	X	$G = \overline{A}$	NOT (1's complement)

Two Theorems

- Theorem1

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

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as **Bubbled OR**.

Theorem1 – Logic diagram

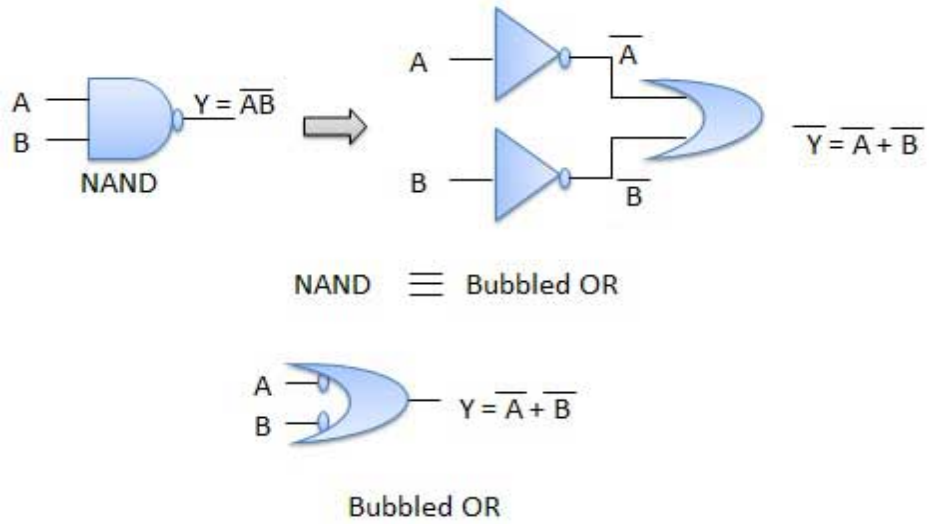


Table showing verification of the De Morgan's first theorem

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

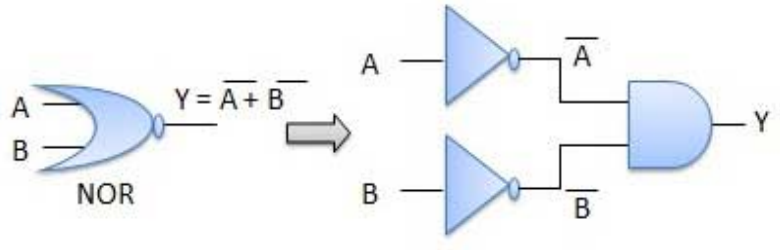
Theorem 2

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

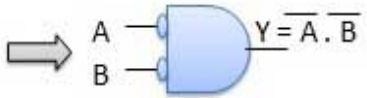
NOR = Bubbled AND

- The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
- This AND gate is called as **Bubbled AND**.

Theorem 2- Logic diagram



NOR ☰ Bubbled AND



Bubbled AND

Table showing verification of the De Morgan's second theorem

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Applications of De Morgan's Theorem

- In the domain of engineering, using De Morgan's laws, Boolean expressions can be built easily only through one gate which is usually NAND or NOR gates. This results in hardware design at a cheaper cost.
- Used in the verification of SAS code.
- Implemented in computer and electrical engineering domain.
- De Morgan's laws are also employed in Java programming.