

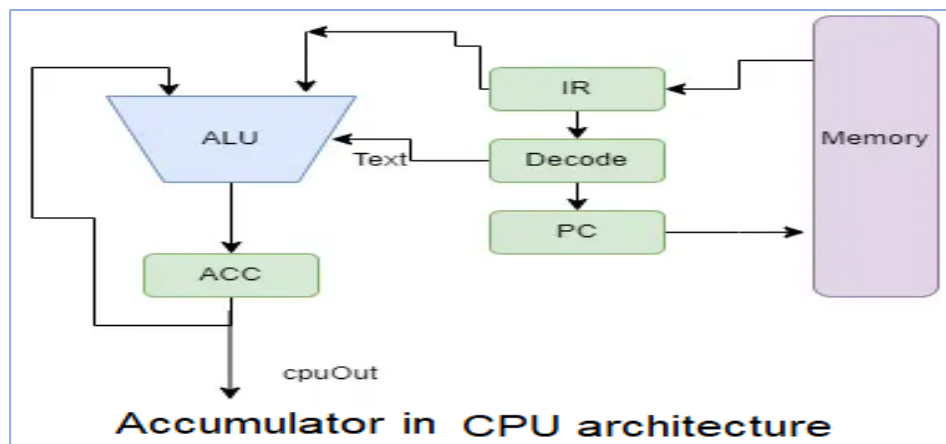
Arithmetic and Logic Unit Architecture

1- Arithmetic and Logic Unit (ALU)

The ALU is a component of the microprocessor or (CPU), ALU made of the electronic circuits and logic gates performs the operation. That is split into arithmetic operations (such as addition, subtraction, multiplication, division, increment and decrement) and logical operations (such as AND, OR, NOT, XOR, and comparisons) on binary data. The ALU consists of **transistor** organized into combinational logic circuit and control circuits that control the flow of data and direct the ALU's subunits to perform the correct operation: The ALU subunits are:

1. **Arithmetic Unit (AU)**: Performs arithmetic operations, it contains components like adders, subtractors, and sometimes multipliers.
2. **Logic Unit (LU)**: Performs logical operations including AND, OR and comparisons, by using logic gates designed for Boolean operations.
3. **Shift Unit**: Handles bit shifting operations like logical shift, arithmetic shift, and rotate operations, Shift is important for operations like multiplication, division, powers of two. Shift directions are:
 - Left Logical Shift (<<): bits move left, zeros enter from **right**.
 - Right Logical Shift (>>): bits move right, zeros enter from **left**.
4. **Temporary register**: are internal registers used by the ALU to hold intermediate data during computations. The ALU gets its input from the accumulator and temporary registers to perform its operations,
5. **Control and Sequencing Logic**: Controls how data flows between subunits and which unit to performs its operation

6. **Accumulator (ACC):** is special-purpose **register** work as temporarily **register** to stores the results of arithmetic and logical operations before they are moved to memory or other registers. The ALU uses data from memory, register and accumulator to perform operations and stores result in accumulator. **Example**, when adding multiple numbers, the accumulator first holds the result of adding the first two numbers. Then the next number is added to this stored result.



Note: ALU uses **combinational logic circuits**, its output dependent on the immediate inputs and does not have memory or depend on previous states.

ALU performs operations like addition, subtraction, AND, OR, XOR, etc., through interconnected logic gates like AND, OR, NOT, XOR. The main arithmetic operations are performed using combinational blocks **Inside the ALU:**

1. **Adders and Subtractors:** These are the core circuits for performing arithmetic operations. In some designs, a subtractor is implemented by using an adder and two's complement arithmetic.
2. **Multiplexers (MUX):** An ALU uses multiplexers to select which operation's result is sent to the output. For example, a MUX will select

between the result of the addition circuit or the result of the logic circuit, based on a control signal from the Control Unit.

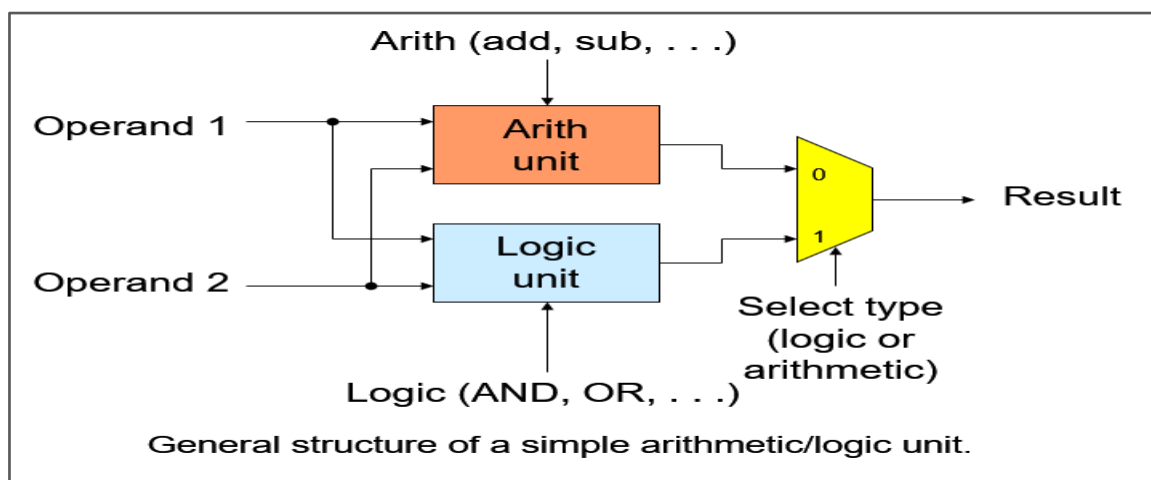
3. **Comparators:** These circuits compare two numbers and produce an output based on their relationship (e.g., whether one is greater than, less than, or equal to the other).

While the other circuits that perform basic arithmetic and logic functions of the that is in the **CPU, but outside the ALU** are:

- **Instruction Decoder** which generates the necessary control signals to execute instruction, is part of the Control Unit.
- **Encoder and Demultiplexers (DEMUX)** are within the CPU.

The ALU designed with different word size, modern computer have ALU with 32 or 64 bit word size, the speed of the CPU depends on the ALU.

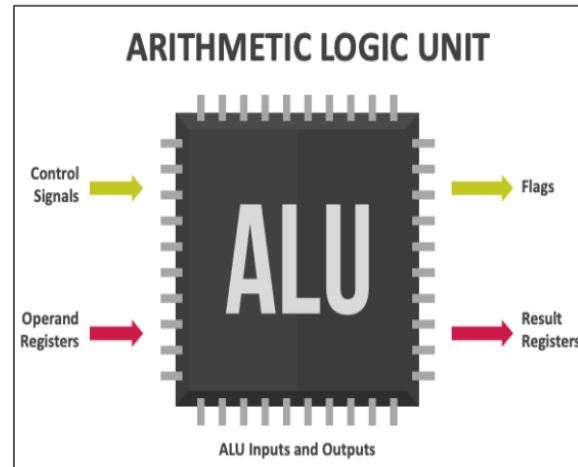
Modern CPUs and GPUs contain **Multiple ALUs** to increase processing speed and throughput by handling several operations in parallel. Currently CPUs have **four to sixteen** ALUs or even more, enabling them to execute many instructions simultaneously to speed up the execution process.



2 How the ALU Work

The CPU's **control unit** directs the ALU on what operation to perform, and the registers the move data between the ALU and other parts of the CPU and memory.

The ALU circuit has the integration of four key components **inputs, operands, outputs, and storage.**



The next steps explain how ALU works

when the **Control Unit** receives an instruction and sends signals to the ALU:

1. The **Control signals** tell the ALU which operation to perform and which inputs to use.
2. The **Operands** in the input registers are sent to the appropriate logic or arithmetic circuits.
3. The ALU performs the operation, and the **result** is sent to the **output** register.
4. **Status** flags also known as (**flag** register): The Flag register are special-purpose registers in the CPU that store status flags. The flag is updated according the outcome of the operation, such as the "zero" flag, sign flag, carry flag, and the "overflow" flag (if an operation resulted in an overflow). As in this [example](#) to add two numbers.

0 0 0 1	+	1 1 1 1
1 0 0 1		1 0 1 1
0 1 0 1		0 1 1 1
0 1 1 1 0		1 0 0 1 0
no overflow		overflow

In the ALU, the transistor work as switches to either open or closed (0,1), to perform logical operations such as bitwise operations, the ALUs apply bit shifting, to move bits in response to instructions example:

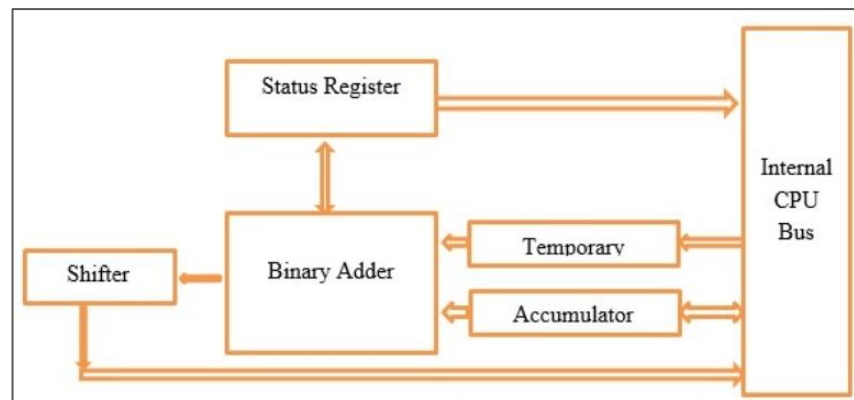
Example1 Logical Shift Left: an 8-bit binary: 00001011 (decimal 11).

- Logical shift left by 1: 00010110 (decimal 22) ←

Example2 Logical Shift Right: 8-bit binary number 01010011 (decimal 83).

- Logical shift right by 1: 00101001 (decimal 41) →

Shift operation
explained in this
figure:



3- Some Logic Gates Examples used by ALU unit:

1-Adder: a simple binary adder that adds two 1-bit binary numbers, a and b , to give a 2-bit sum. There are Two types of adder:

- **Half Adder** : The half-adder circuit has two inputs: A and B, which add **two** input digits and generate a carry and a sum as output. The combinational logic circuit designed by connecting one EX-OR gate and one AND gate

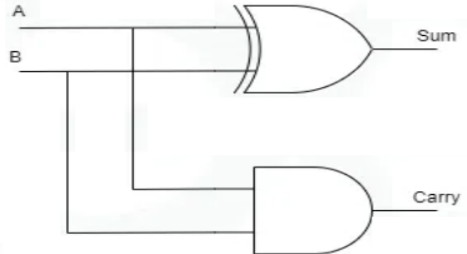
a	b	Sum
0	0	00 ($0 + 0 = 0$)
0	1	01 ($0 + 1 = 1$)
1	0	01 ($1 + 0 = 1$)
1	1	10 ($1 + 1 = 2$)

- **Full Adder**: is combinational circuit add **three** binary digits and produces two outputs (sum and carry out). A two inputs digits and the carry input is forwarded

from the previous addition making a chained, multi-bit addition. The circuit consist of two EX-OR gates, two AND gates, and one [OR gate](#).

- **The subtraction** logical gate design is the same as the Adder but the subtracter use **negation** gate on the second number.

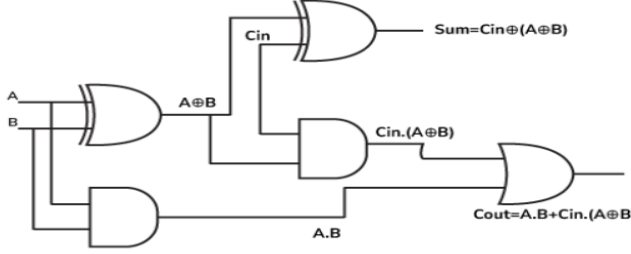
Half Adder gates and Truth Table



Truth Table

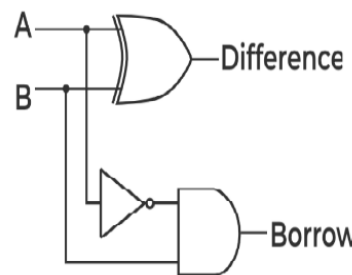
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder



Inputs			Outputs	
A	B	C	Sum	Carry
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

Half Subtractor



A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

H.W : Draw the Full subtractor circuit with truth table

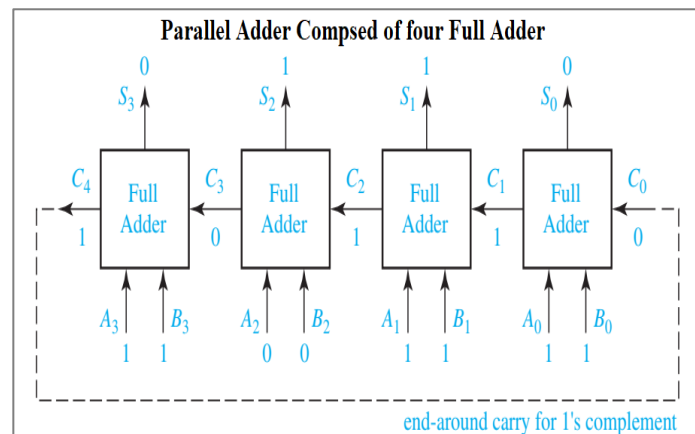
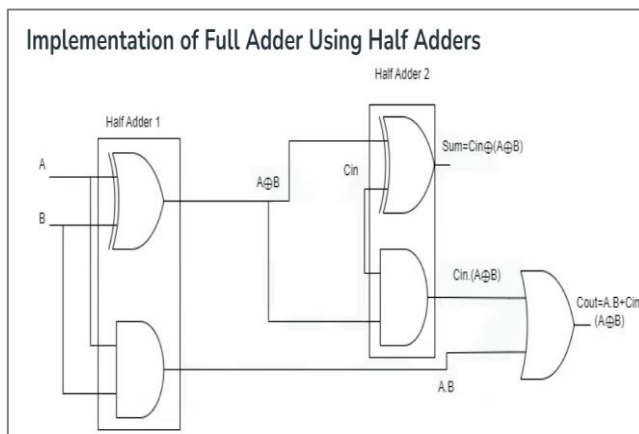
Note: The **Half adder** Only useful for the first bit of a multi-bit binary number where there is no carry-in. While Full adder two data bits and a carry-in bit from a previous addition stage. Example: Adding two 8-bit numbers: For the first (rightmost) bit, a half adder could be used, as there is no incoming carry, for

every subsequent bit, you need to add the two bits from the numbers plus the carry-out from the previous bit's sum. This requires a full adder.

- **A parallel adder** is a digital circuit within a computer's Arithmetic Logic Unit (ALU) that adds two or more multibit binary numbers simultaneously. Instead of processing bits one after the other (as a serial adder does), a parallel adder processes all pairs of corresponding bits at the same time, which results in much faster addition

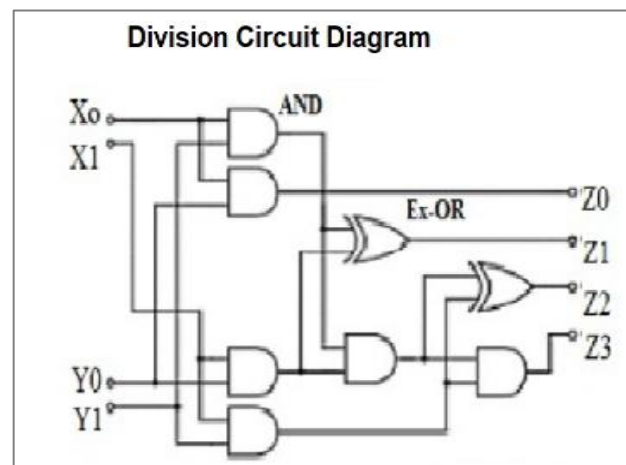
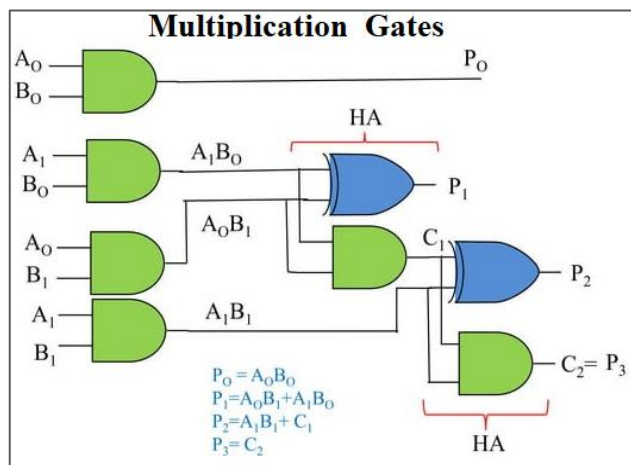
To add larger binary numbers, multiple full adders are chained together. Example: to add two 4-bit numbers, a parallel adder would use four full adders.

1. **Input:** The parallel adder takes in all the bits of two numbers, A and B, at once.
2. **Cascaded Full Adders:** The adders are connected in a cascade, where the carry-out from one full adder is connected to the carry-in of the next full adder in the chain.



2- Multiplier: Binary Multiplication contains only 0s and 1s. The four fundamental rules for binary multiplication are $0 \times 0 = 0$, $0 \times 1 = 0$, $1 \times 0 = 0$, $1 \times 1 = 1$. The multiplication of two binary numbers performed by using:

- **AND gates:** Four 2-input AND gates are required to generate the partial products.
- **Half adders:** One half adder is needed to add the second column of the partial products.
- **Full adder:** A single full adder is needed for the third column of the partial products, as it will have a carry-in from the previous addition stage.



Example: two binary multiplications: the partial products are generated for each digit in the multiplier. Then all these partial products are added to produce the final product value. In the partial product multiplication, when the multiplier bit zero, the partial product is zero, and when the multiplier bit is 1, the resulted partial product is the multiplicand.

The **division** circuit is more complex as explained in the figure.

1 0 1 0	→	Multiplicand
× 1 0 1 1	→	Multiplier
1 0 1 0	→	Partial product 1
1 0 1 0	→	Partial product 2
0 0 0 0	→	Partial product 3
1 0 1 0	→	Partial product 4
1 1 0 1 1 1 0		

3- Multiplexers : A multiplexer is a [combinational circuit](#) that has many data inputs and a single output, for input line 2^n , n selection lines are needed. Used in:

- **Memory addressing:** A MUX selects the correct memory location from a large memory bank when the CPU needs to read or write data. This reduces the number of connecting wires needed to access memory.
- **Peripheral communication:** In a keyboard, a MUX can combine the signals from many keys into a single data stream to send to the computer.
- **Arithmetic Logic Units (ALUs):** Inside a CPU's ALU, multiplexers are used to select which of many possible operations (addition, subtraction, etc.) should be performed based on the instruction provided
- **Used in Network:** A multiplexer for example combines many telephone conversations into a single wire, greatly reducing the cost and complexity of network infrastructure.

The Gates needed for a 2-to-1 MUX, for two data inputs, and one select line, and output.

- **One NOT gate:** Used to invert the single select line.
- **Two AND gates:** Used to enable one of the two input data lines based on the select line's state.
- **One OR gate:** Combines the output of the two AND gates to produce the single final output

