UNIT 3:

Register transfer and Micro operations

Bus and Memory Transfers

A digital system composed of many registers, and paths must be provided to transfer information from one register to another. The number of wires connecting all of the registers will be excessive if separate lines are used between each register and all other registers in the system.

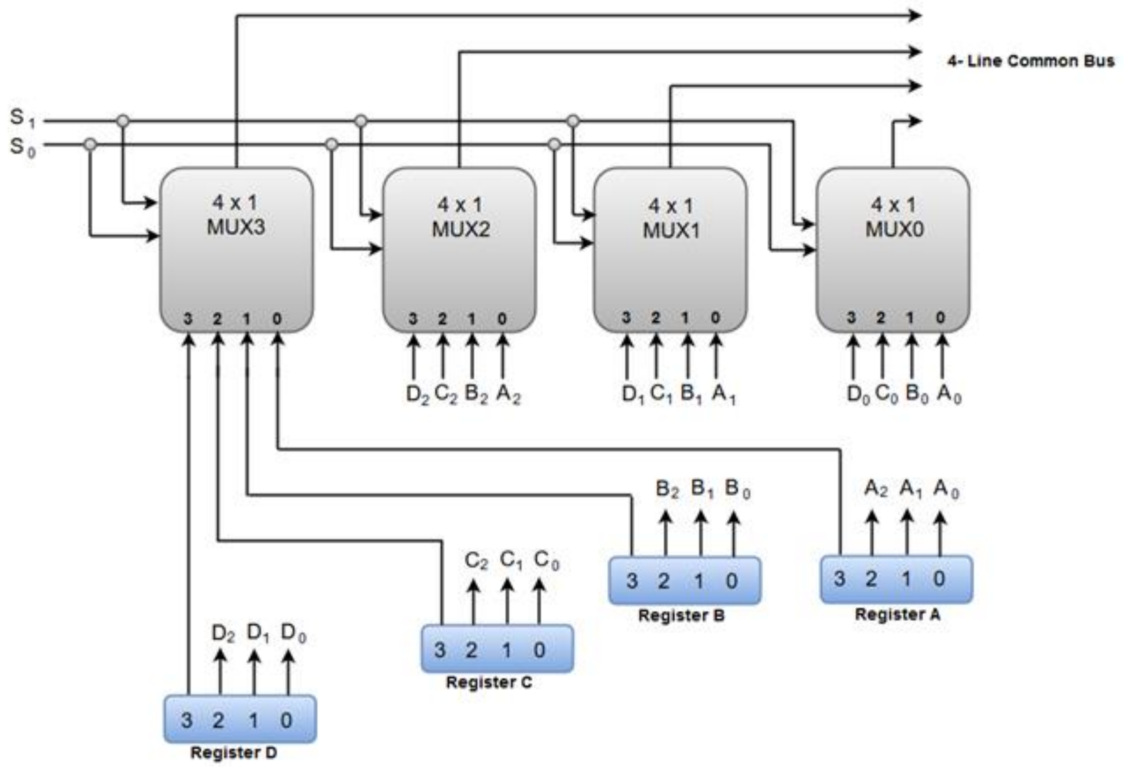
A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.

A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during a particular register transfer.

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

We have used labels to make it more convenient for you to understand the input-output configuration of a Bus system for four registers. For instance, output 1 of register A is connected to input 0 of MUX1.

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus. This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

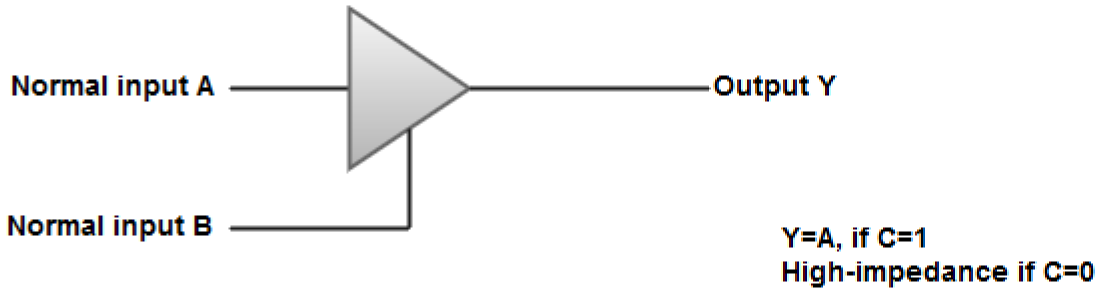
Note: The number of multiplexers needed to construct the bus is equal to the number of bits in each register. The size of each multiplexer must be 'k * 1' since it multiplexes 'k' data lines. For instance, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

A bus system can also be constructed using **three-state gates** instead of multiplexers.

The **three state gates** can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state.

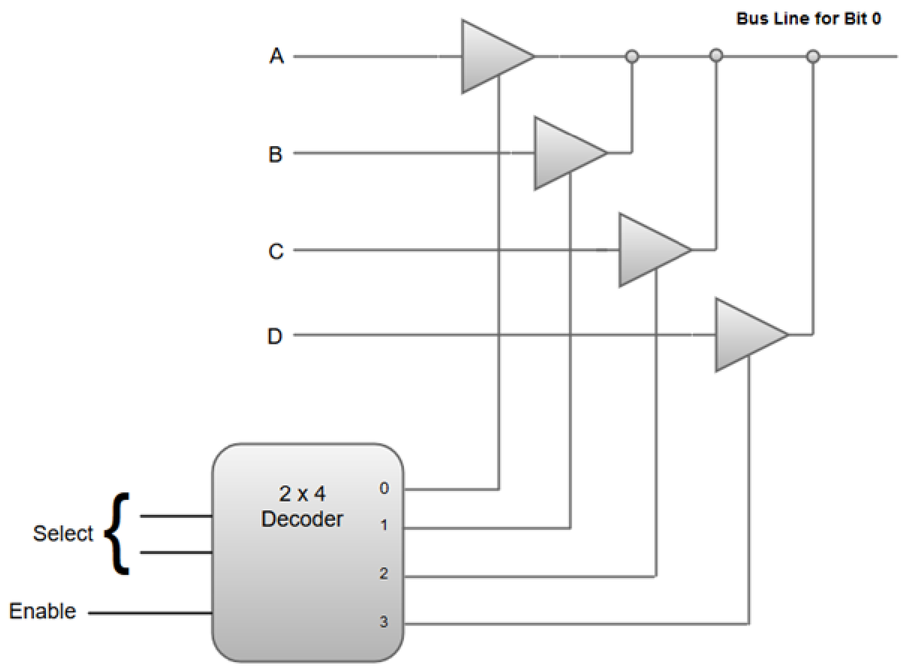
The most commonly used three state gates in case of the bus system is a **buffer gate**.

The graphical symbol of a three-state buffer gate can be represented as:



The following diagram demonstrates the construction of a bus system with three-state buffers.

Bus line with three state buffer:



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

Memory Transfer

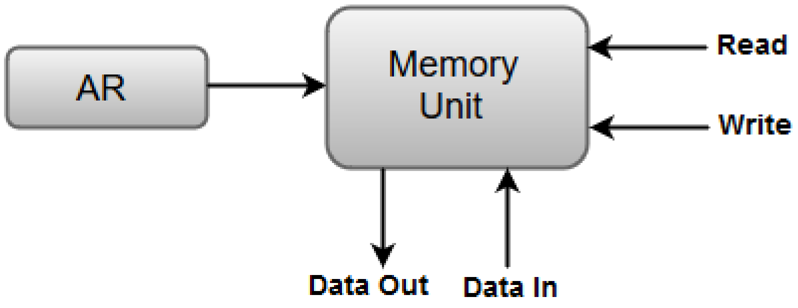
Most of the standard notations used for specifying operations on memory transfer are stated below.

- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.

- Thus, a read operation can be stated as:

1. Read: DR ← M [AR]
    - The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
    - And the corresponding write operation can be stated as:

1. Write: M [AR] ← R1
    - The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



Three-State Bus Buffers

A three-state bus buffer is an integrated circuit that connects multiple data sources to a single bus. The open drivers can be selected to be either a logical high, a logical low, or high impedance which allows other buffers to drive the bus.

Now, let's see the more detailed analysis of a 3-state bus buffer in points:

1. As in a conventional gate, 1 and 0 are two states.
2. The third state is a high impedance state.
3. The third state behaves like an open circuit.
4. If the output is not connected, then there is no logical significance.
5. It may perform any type of conventional logic operations such as AND, OR, NAND, etc.

Difference between normal buffer and three-state buffer:
It contains both normal input and control input. Here, the output state is determined by the control input.

- When the control input is 1, the output is enabled and the gate will behave like a conventional buffer.

- When the control input is 0, the output is disabled and the gate will be in a high impedance state.

More Points:

1. To form a single bus line, all the outputs of the 4 buffers are connected together.
2. The control input will now decide which of the 4 normal inputs will communicate with the bus line.
3. The decoder is used to ensure that only one control input is active at a time.
4. The diagram of a 3-state buffer can be seen below.

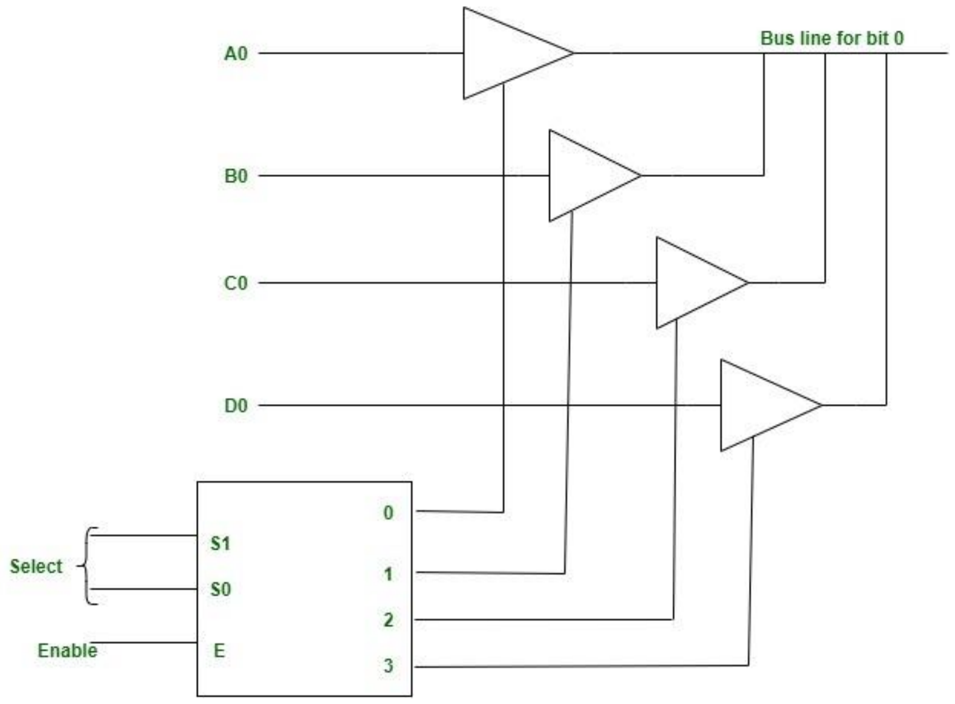


Figure – Bus Line with three-state -buffers

Binary Adder

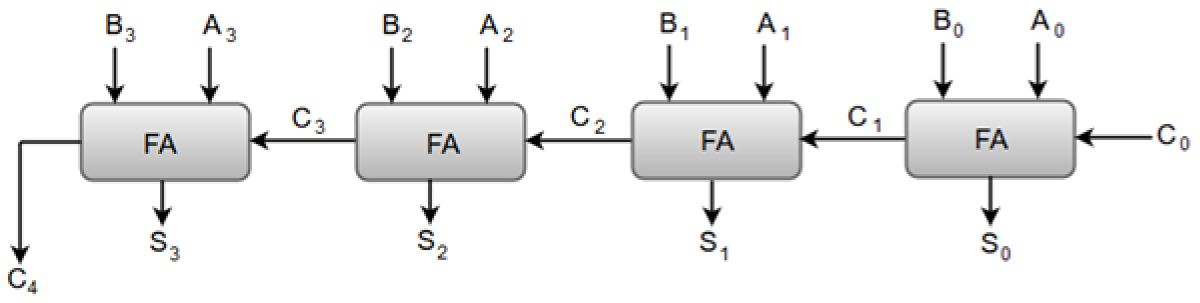
The Add micro-operation requires registers that can hold the data and the digital components that can perform the arithmetic addition.

A Binary Adder is a digital circuit that performs the arithmetic sum of two binary numbers provided with any length.

A Binary Adder is constructed using full-adder circuits connected in series, with the output carry from one full-adder connected to the input carry of the next full-adder.

The following block diagram shows the interconnections of four full-adder circuits to provide a 4-bit binary adder.

4 bit binary adder:



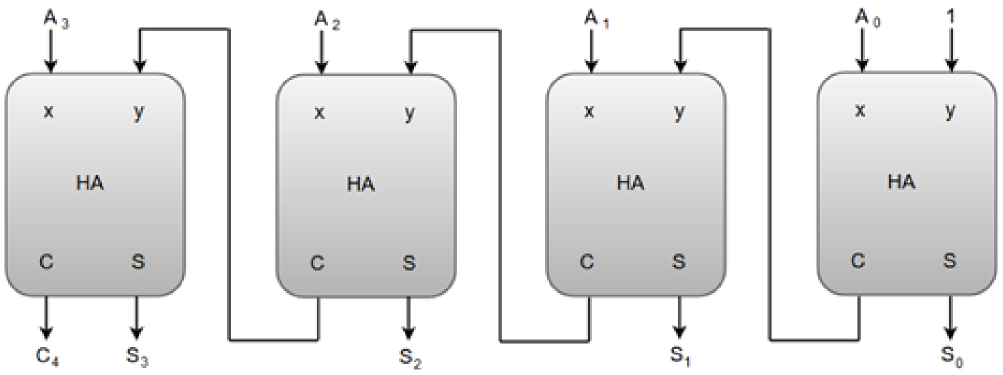
- The augend bits (A) and the addend bits (B) are designated by subscript numbers from right to left, with subscript '0' denoting the low-order bit.
- The carry inputs starts from C0 to C3 connected in a chain through the full-adders. C4 is the resultant output carry generated by the last full-adder circuit.
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The sum outputs (S0 to S3) generates the required arithmetic sum of augend and addend bits.
- The ***n*** data bits for the **A** and **B** inputs come from different source registers. For instance, data bits for **A** input comes from source register R1 and data bits for **B** input comes from source register R2.
- The arithmetic sum of the data inputs of A and B can be transferred to a third register or to one of the source registers (R1 or R2).

Binary Incrementer

The increment micro-operation adds one binary value to the value of binary variables stored in a register. For instance, a 4-bit register has a binary value 0110, when incremented by one the value becomes 0111.

The increment micro-operation is best implemented by a 4-bit combinational circuit incrementer. A 4-bit combinational circuit incrementer can be represented by the following block diagram.

4-bit binary incrementer:



- A logic-1 is applied to one of the inputs of least significant half-adder, and the other input is connected to the least significant bit of the number to be incremented.
- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.
- The binary incrementer circuit receives the four bits from A0 through A3, adds one to it, and generates the incremented output in S0 through S3.
- The output carry C4 will be 1 only after incrementing binary 1111.

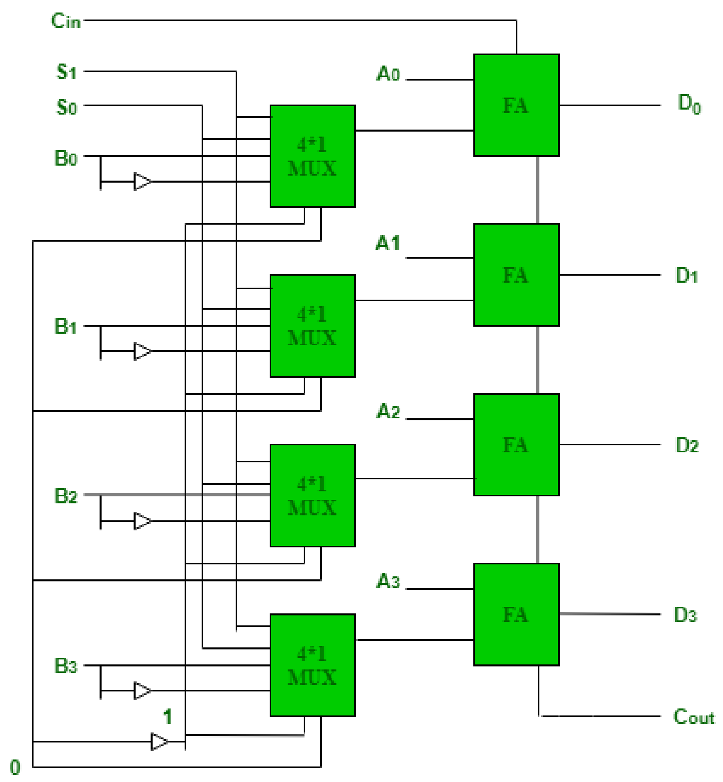
Note: The 4-bit binary incrementer circuit can be extended to an n-bit binary incrementer by extending the circuit to include n half-adders. The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

Arithmetic Circuits

Arithmetic circuits can perform seven different arithmetic operations using a single composite circuit.
It uses a full adder (FA) to perform these operations. A multiplexer (MUX) is used to provide different inputs to the circuit in order to obtain different arithmetic operations as outputs.

4-bit Arithmetic Circuit :

Consider the following 4-bit Arithmetic circuit with inputs A and B. It can perform seven different arithmetic operations by varying the inputs of the multiplexer and the carry (C_0).



Truth Table for the above Arithmetic Circuit :

S_0	S_1	C_0	MUX Output	Full Adder Output
0	0	0	B	A + B
0	0	1	B	A + B + 1
0	1	0	B’	A + B’
0	1	1	B’	A + B’ + 1 = A – B
1	0	0	0	A
1	0	1	0	A + 1

S_0	S_1	C_0	MUX Output	Full Adder Output
1	1	0	1	A – 1
1	1	1	1	A – 1 + 1 = A

Hence, the different operations for the inputs A and B are –

1. A + B (adder)
2. A + B + 1
3. A + B’
4. A – B (subtracter)
5. A
6. A + 1 (incrementer)
7. A – 1 (decrementer)

Logic Micro-Operations

Logic operations are binary micro-operations implemented on the bits saved in the registers. These operations treated each bit independently and create them as binary variables.

For example, the exclusive-OR micro-operation with the contents of two registers R1 and R2 is denoted by the statement

P: R1←R1⨁⨁R2

It determines a logic micro-operation to be implemented on the single bits of the registers supported that the control variable P = 1. Consider that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100.

The exclusive-OR micro-operation stated above represent the following logic computation −

1010	Content of R1
1100	Content of R2
0110	Content of R1 after P = 1

The content of R1, after the implementation of the micro-operation, is similar to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1.

Special Symbols

Special symbols will be approved for the logic micro-operations OR, AND, and complement, to categorize them from the matching symbols that can define Boolean functions. The symbol V can indicate an OR micro-operation and the symbol can indicate an AND micro-operation.

The complement micro-operation is similar to the 1's complement and supports a bar on the highest of the symbol that indicates the registered name. There are various symbols, and it will be applicable to differentiate between a logic micro-operation and a control (or Boolean) function.

There is another sense for supporting two sets of symbols can that recognize the symbol +, when can symbolize arithmetic plus, from a logic OR operation. Although the + symbol has two meanings, it will be available to determine between them by observing where the symbol appears.

When the symbol + appears in a micro-operation, it will indicate an arithmetic plus. When it appears in a control (or Boolean) function, it will indicate an OR operation. We cannot use it to symbolize an OR micro-operation.

For example, in the statement

P+Q: R1←R2+R3,R 4←R5V R6

The + between P and Q is an OR operation between two binary variables of a control function. The + between R2 and R3 determines an add micro-operation. The OR micro-operation is named by the symbol V between registers R5 and R6.

Shift Micro-Operations

Shift micro-operations are those micro-operations that are used for the serial transfer of information. These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations. There are three types of shift micro-operations: **1.**

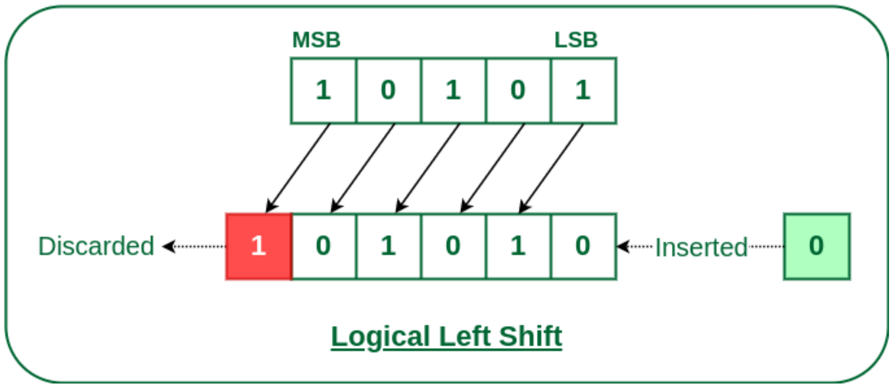
1. Logical Shift:

It transfers the 0 zero through the serial input. We use the symbols ‘**<<**‘ for the logical left shift and ‘**>>**‘ for the logical right shift.

Logical Left Shift:

In this shift, one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.

The left shift operator is denoted by the double left arrow key (<<). The general syntax for the left shift is shift-expression << k.



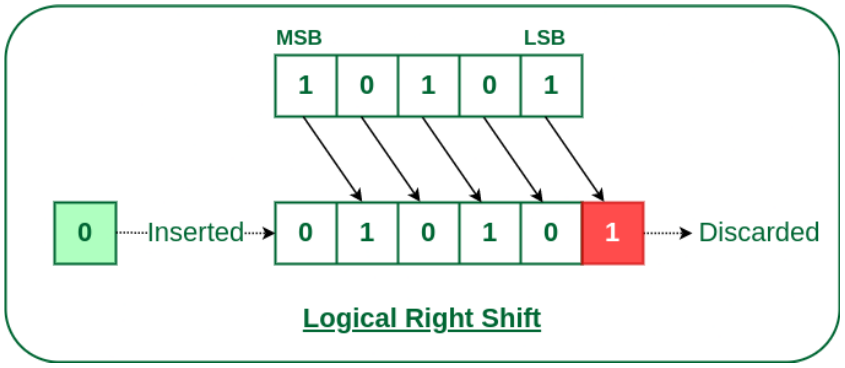
Logical Left Shift

Note: Every time we shift a number towards the left by 1 bit it multiplies that number by 2.

Logical Right Shift

In this shift, each bit moves to the right one by one and the least significant bit(LSB) is rejected and the empty MSB is filled with zero.

The right shift operator is denoted by the double right arrow key (>>). The general syntax for the right shift is “shift-expression >> k”.



Logical Right Shift

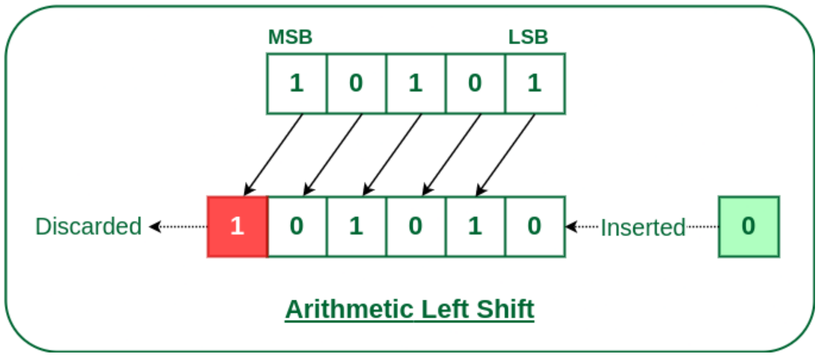
Note: Every time we shift a number towards the right by 1 bit it divides that number by 2.

<u>2. Arithmetic Shift:</u>
The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position.
Following are the two ways to perform the arithmetic shift.

1. Arithmetic Left Shift
2. Arithmetic Right Shift

Arithmetic Left Shift:

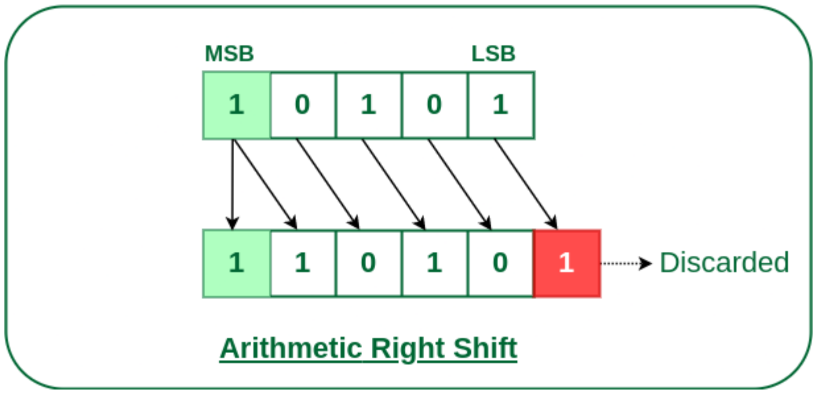
In this shift, each bit is moved to the left one by one. The empty least significant bit (LSB) is filled with zero and the most significant bit (MSB) is rejected. Same as the Left Logical Shift.



Arithmetic Left Shift

Arithmetic Right Shift:

In this shift, each bit is moved to the right one by one and the least significant(LSB) bit is rejected and the empty most significant bit(MSB) is filled with the value of the previous MSB.



Arithmetic Right Shift

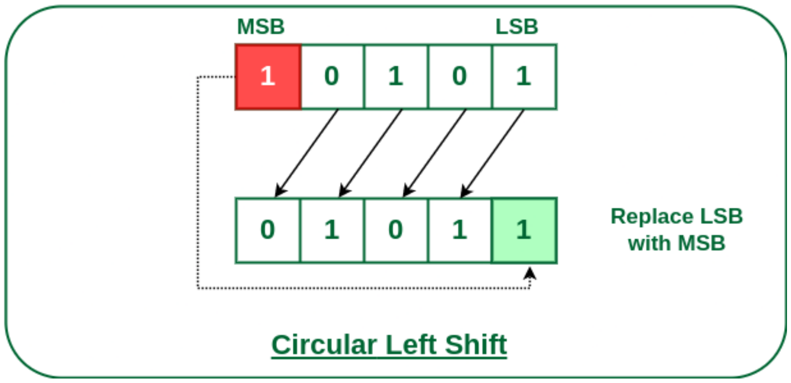
3. Circular Shift:

The circular shift circulates the bits in the sequence of the register around both ends without any loss of information.
Following are the two ways to perform the circular shift.

1. Circular Shift Left
2. Circular Shift Right

Circular Left Shift:

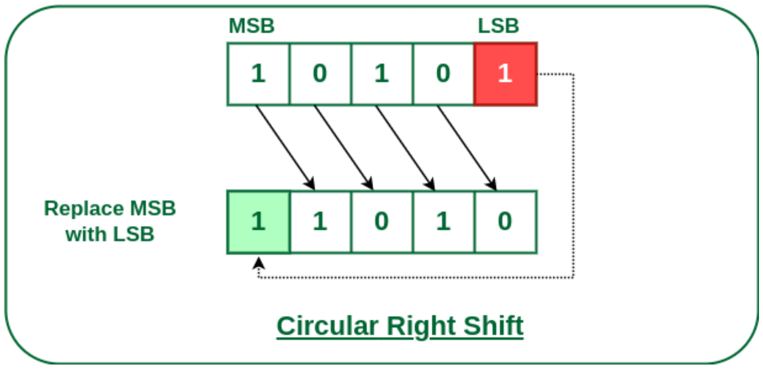
In this micro shift operation each bit in the register is shifted to the left one by one. After shifting, the LSB becomes empty, so the value of the MSB is filled in there.



Circular Left Shift

Circular Right Shift:

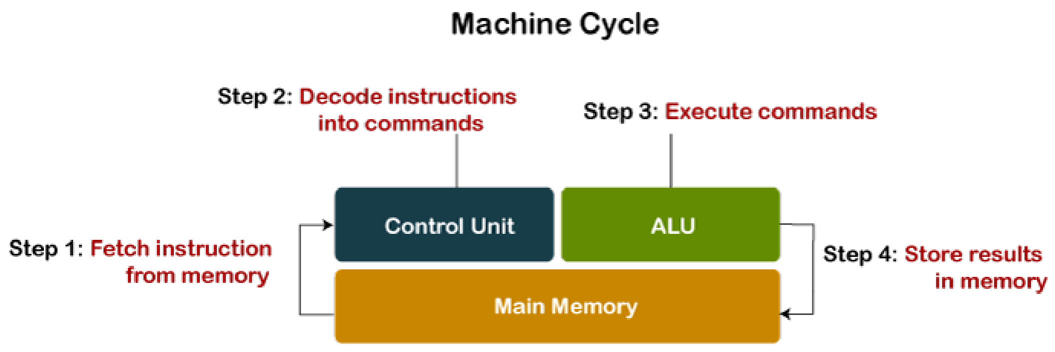
In this micro shift operation each bit in the register is shifted to the right one by one. After shifting, the MSB becomes empty, so the value of the LSB is filled in there.



Circular Right Shift

What is ALU (Arithmetic Logic Unit)?

In the computer system, ALU is a main component of the central processing unit, which stands for arithmetic logic unit and performs arithmetic and logic operations. It is also known as an integer unit (IU) that is an integrated circuit within a CPU or GPU, which is the last component to perform calculations in the processor. It has the ability to perform all processes related to arithmetic and logic operations such as addition, subtraction, and shifting operations, including Boolean comparisons (XOR, OR, AND, and NOT operations). Also, binary numbers can accomplish mathematical and bitwise operations. The arithmetic logic unit is split into AU (arithmetic unit) and LU (logic unit). The operands and code used by the ALU tell it which operations have to perform according to input data. When the ALU completes the processing of input, the information is sent to the computer's memory.


Machine Cycle
Step 2: Decode instructions into commands
Step 3: Execute commands
Control Unit
ALU
Step 1: Fetch instruction from memory
Step 4: Store results in memory
Main Memory

Except performing calculations related to addition and subtraction, ALUs handle the multiplication of two integers as they are designed to execute integer calculations; hence, its result is also an integer. However, division operations commonly may not be performed by ALU as division operations may produce a result in a floating-point number. Instead, the floating-point unit (FPU) usually handles the division operations; other non-integer calculations can also be performed by FPU.

Additionally, engineers can design the ALU to perform any type of operation. However, ALU becomes more costly as the operations become more complex because ALU destroys more heat and takes up more space in the CPU. This is the reason to make powerful ALU by engineers, which provides the surety that the CPU is fast and powerful as well.

The calculations needed by the CPU are handled by the arithmetic logic unit (ALU); most of the operations among them are logical in nature. If the CPU is made more powerful, which is made on the basis of the ALU is designed. Then it creates more heat and takes more power or energy. Therefore, it must be moderation between how complex and powerful ALU is and not be more costly. This is the main reason the faster CPUs are more costly; hence, they take much power and destroy more heat. Arithmetic and logic operations are the main operations that are performed by the ALU; it also performs bit-shifting operations.

Although the ALU is a major component in the processor, the ALU's design and function may be different in the different processors. For case, some ALUs are designed to perform only integer calculations, and some are for floating-point operations. Some processors include a single arithmetic logic unit to perform operations, and others may contain numerous ALUs to complete calculations. The operations performed by ALU are:

- **Logical Operations:** The logical operations consist of NOR, NOT, AND, NAND, OR, XOR, and more.
- **Bit-Shifting Operations:** It is responsible for displacement in the locations of the bits to the by right or left by a certain number of places that are known as a multiplication operation.
- **Arithmetic Operations:** Although it performs multiplication and division, this refers to bit addition and subtraction. But multiplication and division operations are more costly to make. In the place of multiplication, addition can be used as a substitute and subtraction for division.

Arithmetic Logic Unit (ALU) Signals

A variety of input and output electrical connections are contained by the ALU, which led to casting the digital signals between the external electronics and ALU.

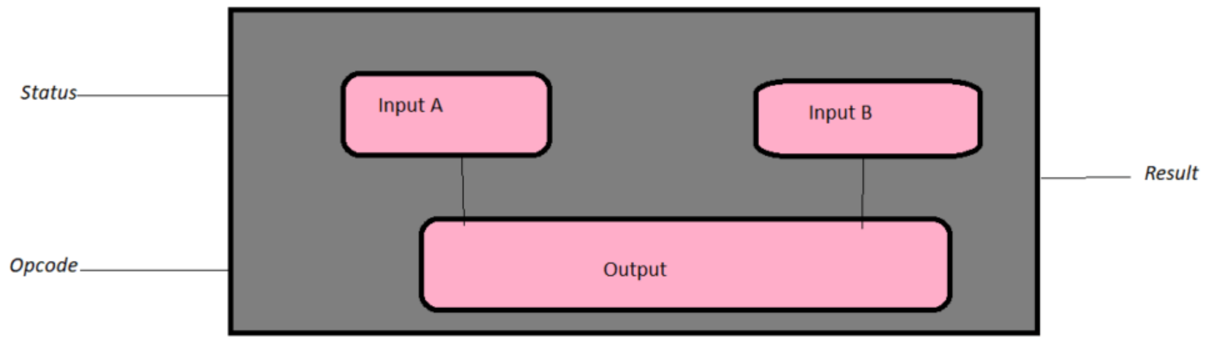
ALU input gets signals from the external circuits, and in response, external electronics get outputs signals from ALU.

Data: Three parallel buses are contained by the ALU, which include two input and output operand. These three buses handle the number of signals, which are the same.

Opcode: When the ALU is going to perform the operation, it is described by the operation selection code what type of operation an ALU is going to perform arithmetic or logic operation.

Status

- **Output:** The results of the ALU operations are provided by the status outputs in the form of supplemental data as they are multiple signals. Usually, status signals like overflow, zero, carry out, negative, and more are contained by general ALUs. When the ALU completes each operation, the external registers contained the status output signals. These signals are stored in the external registers that led to making them available for future ALU operations.
- **Input:** When ALU once performs the operation, the status inputs allow ALU to access further information to complete the operation successfully. Furthermore, stored carry-out from a previous ALU operation is known as a single "carry-in" bit.



Configurations of the ALU

The description of how ALU interacts with the processor is given below. Every arithmetic logic unit includes the following configurations:

- Instruction Set Architecture
- Accumulator
- Stack
- Register to Register
- Register Stack
- Register Memory

Accumulator

The intermediate result of every operation is contained by the accumulator, which means Instruction Set Architecture (ISA) is not more complex because there is only required to hold one bit.

Generally, they are much fast and less complex but to make Accumulator more stable; the additional codes need to be written to fill it with proper values. Unluckily, with a single processor, it is very difficult to find Accumulators to execute parallelism. An example of an Accumulator is the desktop calculator.

Stack

Whenever the latest operations are performed, these are stored on the stack that holds programs in top-down order, which is a small register. When the new programs are added to execute, they push to put the old programs.

Register-Register Architecture

It includes a place for 1 destination instruction and 2 source instructions, also known as a 3-register operation machine. This Instruction Set Architecture must be more in length for storing three operands, 1 destination and 2 sources. After the end of the operations, writing the results back to the Registers would be difficult, and also the length of the word should be longer. However, it can be caused to more issues with synchronization if write back rule would be followed at this place.

The MIPS component is an example of the register-to-register Architecture. For input, it uses two operands, and for output, it uses a third distinct component. The storage space is hard to maintain as each needs a distinct memory; therefore, it has to be premium at all times. Moreover, there might be difficult to perform some operations.

Register - Stack Architecture

Generally, the combination of Register and Accumulator operations is known as for Register - Stack Architecture. The operations that need to be performed in the register-stack Architecture are pushed onto the top of the stack. And its results are held at the top of the stack. With the help of using the Reverse polish method, more complex mathematical operations can be broken down. Some programmers, to represent operands, use the concept of a binary tree. It means that the reverse polish methodology can be easy for these programmers, whereas it can be difficult for other programmers. To carry out Push and Pop operations, there is a need to be new hardware created.

Register and Memory

In this architecture, one operand comes from the register, and the other comes from the external memory as it is one of the most complicated architectures. The reason behind it is that every program might be very long as they require to be held in full memory space. Generally, this technology is integrated with Register-Register Register technology and practically cannot be used separately.

Advantages of ALU

ALU has various advantages, which are as follows:

- It supports parallel architecture and applications with high performance.
- It has the ability to get the desired output simultaneously and combine integer and floating-point variables.
- It has the capability of performing instructions on a very large set and has a high range of accuracy.
- Two arithmetic operations in the same code like addition and multiplication or addition and subtraction, or any two operands can be combined by the ALU. For case, A+B*C.
- Through the whole program, they remain uniform, and they are spaced in a way that they cannot interrupt part in between.
- In general, it is very fast; hence, it provides results quickly.
- There are no sensitivity issues and no memory wastage with ALU.
- They are less expensive and minimize the logic gate requirements.

Disadvantages of ALU

The disadvantages of ALU are discussed below:

- With the ALU, floating variables have more delays, and the designed controller is not easy to understand.
- The bugs would occur in our result if memory space were definite.
- It is difficult to understand amateurs as their circuit is complex; also, the concept of pipelining is complex to understand.
- A proven disadvantage of ALU is that there are irregularities in latencies.
- Another demerit is rounding off, which impacts accuracy.