2.1 Register Transfer Language

Register transfer language is a symbolic notation for describing micro-operation transfers between registers.

The availability of hardware logic circuits that can perform a specified micro-operation and transfer the outcome of the operation to the same or another register is referred to as register transfer. The term “language” was coined by programmers to describe programming languages. This programming language is a method of expressing a computer process through symbols.

Following are some commonly used register transfer example with an example:

1. Accumulator: This is the most commonly used register for storing data read from memory.

2. General-Purpose Registers: These are used to store data on intermediate outcomes during the execution of a program. Assembly programming is required to access it.

3. Special Purpose Registers: Users do not have access to the Special Purpose Registers. These are computer system registers.

- **MAR:** Memory Address Registers are the registers that store the memory unit’s address
- **MBR:** This register stores instructions and data received from and sent from the memory
- **PC:** Program Counter indicates the next command to be executed
- **IR:** Instruction Register stores the to-be-executed instruction

Register Transfer

Information transferred from one register to another is designated in symbolic form by means of replacement operator.

$$R2 \leftarrow R1$$

It denotes the transfer of the data from register R1 into R2.

Normally we want the transfer to occur only in predetermined control condition. This can be shown by following **if-then** statement: if (P=1) then (R2 ← R1)

Here P is a control signal generated in the control section.

Control Function

A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:

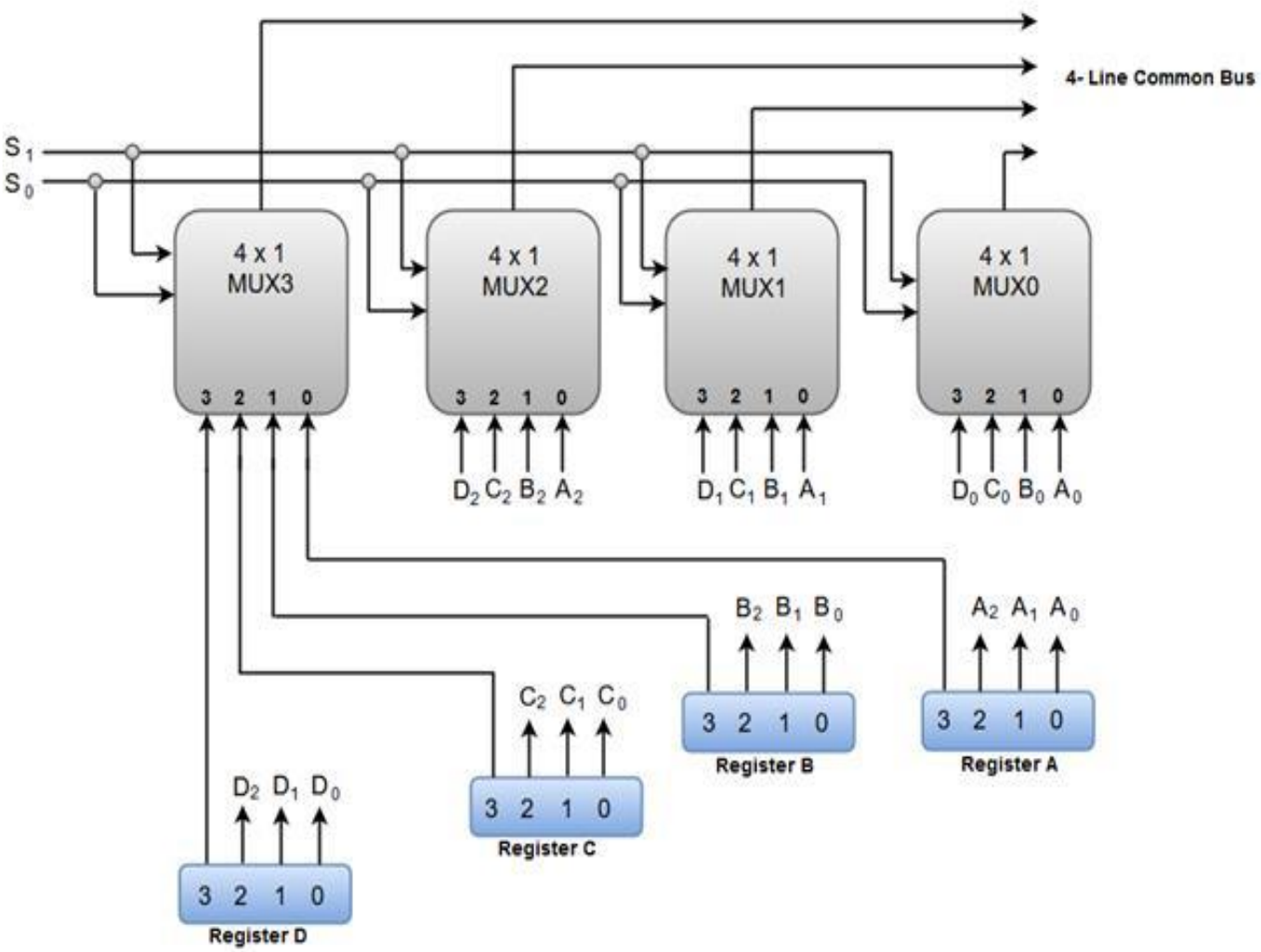
P: R2 ← R1

The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

2.2 Bus and Memory Transfer

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).

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

2.3 Arithmetic, Logic and Shift Micro-operations

Micro-operations are operations performed on data stored in registers. A micro-operation is a simple operation that is carried out on data contained in one or more registers.

Example: Load, Shift, count, and clear.

Types of Micro-Operations

The following are the different types of Micro-Operations:

1. Micro-operations that move binary data from one register to another are known as register transfers.

2. In registers, arithmetic micro-operations operate on numeric data stored.

3. Bit manipulation operations on non-numeric data are performed by logic micro-operations.

4. Shift micro-operations are data-based shift micro-operations.

1. Arithmetic Micro-Operations

- **Add Micro-Operation**

The following statement defines it:

$$R3 \rightarrow R1 + R2$$

The foregoing line tells the computer to add the data or contents of register R1 to the data or contents of register R2, then transfer the sum to register R3.

- **Subtract Micro-Operation**

Consider the following scenario:

$$R3 \rightarrow R1 + R2' + 1$$

Instead of using the minus operator, we use the complement of 1 and add one to the register being subtracted i.e. R1- R2 is equivalent to $R3 \rightarrow R1 + R2' + 1$

- **Increment/Decrement Micro-Operation**

In general, increment and decrement micro-operations are accomplished by adding and removing 1 from the register.

$$R1 \rightarrow R1 + 1$$
$$R1 \rightarrow R1 - 1$$

Symbolic Designation	Description
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 – 1	Decrement the contents of R1 by 1.

2. Logic Micro-Operations

- Logic micro-operation specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- For example, the exclusive-OR micro-operation with the contents of two registers R1 and R2 is symbolized by the statement.

$$P: R1 \oplus R2$$

- It specifies a logic micro-operation to be executed on the individual bits of the registers provided that the control variable P = 1.
- 1 0 1 0 contents of R1
  1 1 0 0 contents of R2
  0 1 1 0 contents of R1 after P=1.

The contents of R1 after execution of the micro-operation, is equal to bit-by-bit exclusive OR operation on each pair of bits in R2 and previous values of R1. Logic micro-operations are useful for bit manipulation of binary data and for making logical decisions.

3. Shift Micro-Operations

Shift micro-operations are used when the data is stored in registers. These micro-operations are used for the serial transmission of data. Here, the data bits are shifted from left to right. These micro-operations are also combined with arithmetic and logic micro-operations and data-processing operations.
There are three types of shift micro-operations-

1. Logical Shift
2. Arithmetic Shift
3. Circular Shift

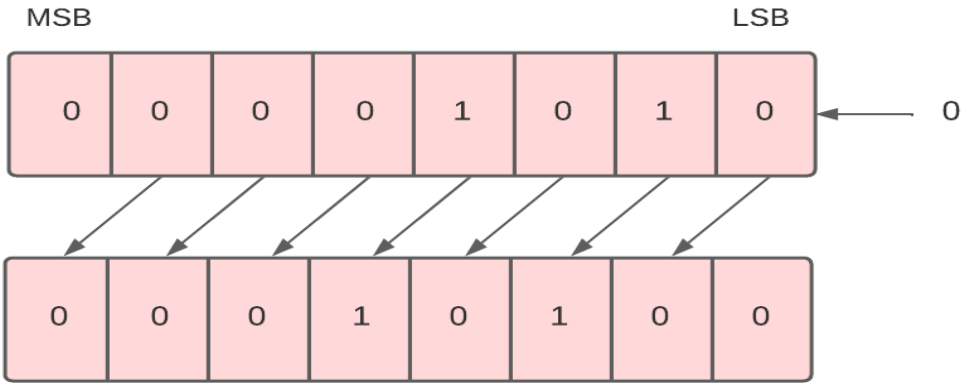
Logical Shift

The logical shift micro-operation moves the 0 through the serial input. There are two ways to implement the logical shift.

1. Logical Shift Left
2. Logical Shift Right

Logical Shift Left

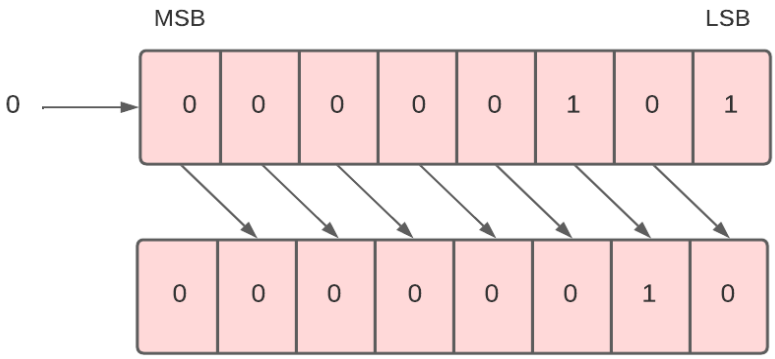
Each bit in the register is shifted to the left one by one in this shift micro-operation. The most significant bit (MSB) is moved outside the register, and the place of the least significant bit (LSB) is filled with 0.
For example, in the below data, there are 8 bits 00001010. When we perform a logical shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftmost bit i.e. 0 will be moved outside, and at the rightmost place or LSB, 0 will be inserted as shown below.



To implement the logical shift left micro-operation, we use the **shl** symbol.
For example, R1 -> shl R1.
This command means the 8 bits present in the R1 register will be logically shifted left, and the result will be stored in register R1.
Moreover, the logical shift left microoperation denotes the multiplication of 2. The example we've taken above when converted into decimal forms the number 10. And the result after the logical shift operation when converted to decimal forms the number 20.

Logical Shift Right

Each bit in the register is shifted to the right one by one in this shift micro-operation. The least significant bit (LSB) is moved outside the register, and the place of the most significant bit (MSB) is filled with 0.
For example, in the below data, there are 8 bits 00000101. When we perform a logical shift right on these bits, all these bits will be shifted towards the right. The LSB or the rightmost bit i.e. 1 will be moved outside, and at the leftmost place or MSB, 0 will be inserted as shown below.



To implement the logical shift right micro-operation, we use the **shr** symbol.
For example, R1 -> shr R1.
This command means the 8 bits present in the R1 register will be logically shifted right, and the result will be stored in register R1.
Logical right shift micro-operation generally denotes division by 2. The inputted bits when converted into decimal form the number 5. And the outcome when converted into decimal forms the number 2.

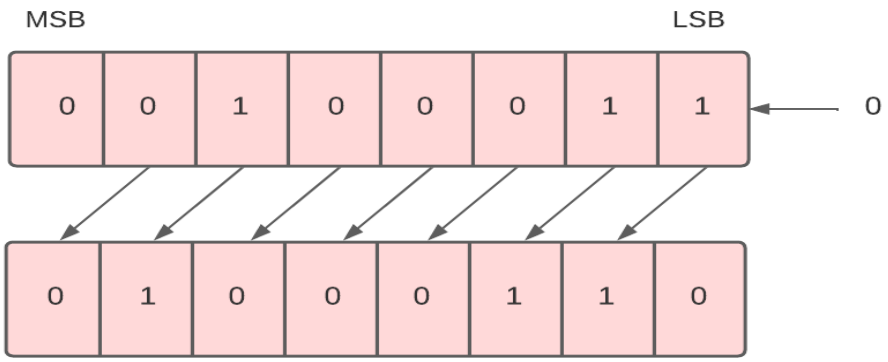
Arithmetic Shift

The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position. There are two ways to implement the arithmetic shift.

1. Arithmetic Shift Left
2. Arithmetic Shift Right

Arithmetic Shift Left

The arithmetic shift left micro-operation is the same as the logical shift left micro-operation. Each bit in the register is shifted to the left one by one in this shift micro-operation. The most significant bit (MSB) is moved outside the register, and the place of the least significant bit (LSB) is filled with 0.
For example, in the below data, there are 8 bits 00100011. When we perform the arithmetic shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftmost bit i.e. 0 will be moved outside, and at the rightmost place or LSB, 0 will be inserted as shown below.

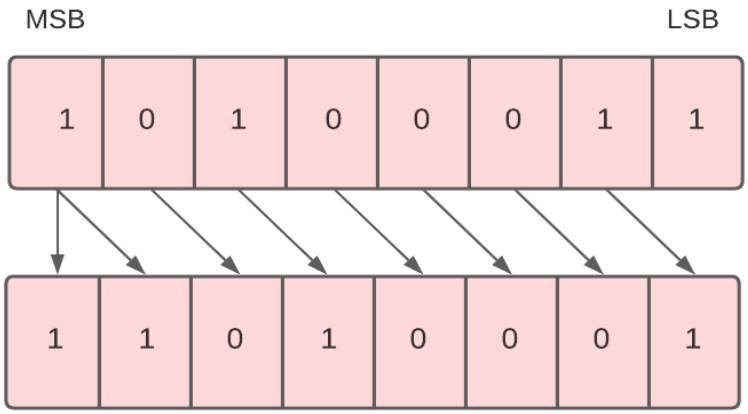


The given binary number (00100011) represents 35 in the decimal system. And the binary number after logical shift left (01000110) represents 70 in a decimal system. Since 35 * 2 = 70. Therefore, we can say that the arithmetic shift left multiplies the number by 2.
To implement the arithmetic shift left micro-operation, we use the **ashl** symbol.

For example, R1 -> ashl R1.
This command means the 8 bits present in the R1 register will be arithmetic shifted left, and the result will be stored in register R1.

Arithmetic Shift Right

Each bit in the register is shifted to the right one by one in this shift micro-operation. The least significant bit (LSB) is moved outside the register, and the place of the most significant bit (MSB) is filled with the previous value of MSB. For example, in the below data, there are 8 bits 10100011. When we perform an arithmetic shift right on these bits, all these bits will be shifted towards the right. The LSB or the rightmost bit i.e. 1 will be moved outside, and at the leftmost place or MSB, the previous MSB value, i.e. 1, will be inserted as shown below.



Arithmetic right shift divides the number by 2.
To implement the arithmetic shift right micro-operation, we use the **ashr** symbol.
For example, R1 -> ashr R1.
This command means the 8 bits present in the R1 register will be arithmetic shifted right, and the result will be stored in register R1.

Circular Shift

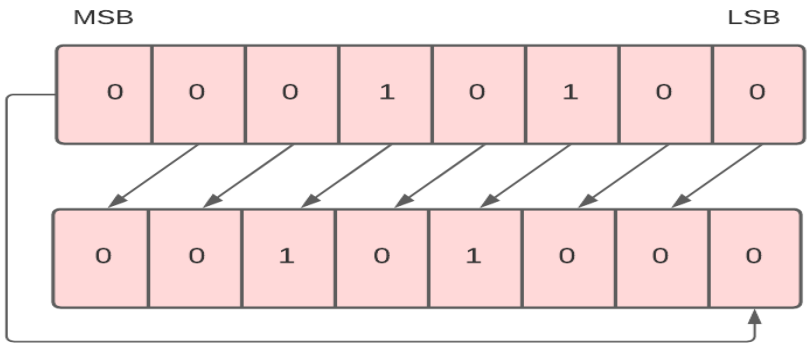
The circular shift, also known as the rotate shift, moves the bits in the register's sequence around both ends, thus ensuring no loss of information. There are two ways to implement the circular shift.

1. Circular Shift Left
2. Circular Shift Right

Circular Shift Left

Each bit in the register is shifted to the left one by one in this shift micro-operation. After shifting, the least significant bit (LSB) place becomes empty, so it is filled with the value at the most significant bit (MSB).

For example, in the below data, there are 8 bits 00010100. When we perform a circular shift left on these bits, all these bits will be shifted towards the left. The MSB or the leftmost bit i.e. 0 will be placed at the rightmost place or LSB as shown below.



To implement the circular shift left micro-operation, we use the **cil** symbol.

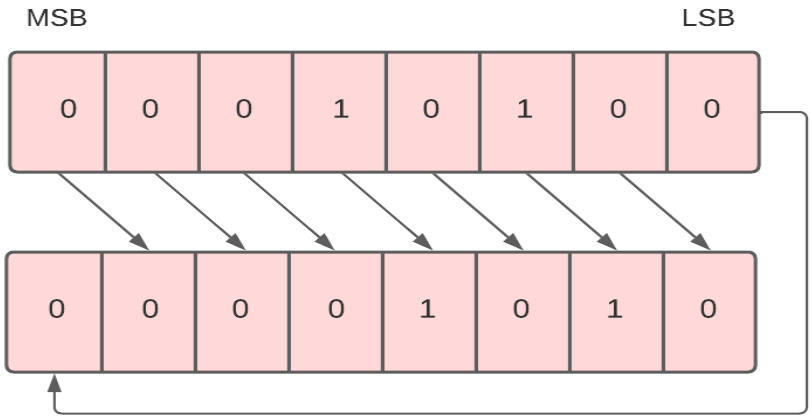
For example, R1 -> cil R1.

This command means the 8 bits present in the R1 register will be circular shifted left, and the result will be stored in register R1.

Circular Shift Right

Each bit in the register is shifted to the right one by one in this shift micro-operation. After shifting, the most significant bit (MSB) place becomes empty, so it is filled with the value at the least significant bit (LSB).

For example, in the below data, there are 8 bits 00010100. When we perform a circular shift right on these bits, all these bits will be shifted towards the right. The LSB or the leftmost bit, i.e. 0, will be placed at the rightmost place or MSB.



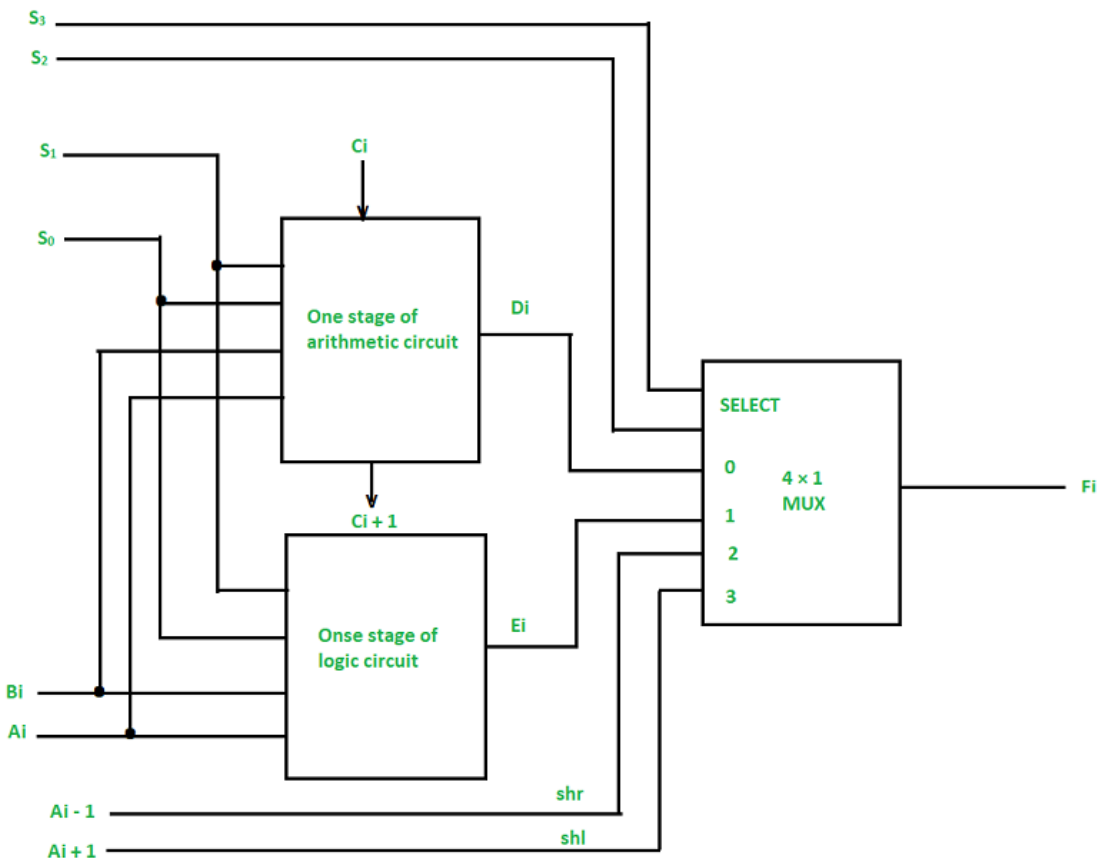
To implement the circular shift right micro-operation, we use the **cir** symbol.
For example, R1 -> cir R1.
This command means the 8 bits present in the R1 register will be circular shifted right, and the result will be stored in register R1.

2.4 Arithmetic Logic shift Unit

The Arithmetic Logic Shift Unit (ALSU) is part of a computer system's Arithmetic Logic Unit(ALU). It can be defined as a digital circuit that performs arithmetic, logical, and shift operations. Instead of having individual registers performing micro-operations directly, computer systems employ many storage registers connected to a common operational unit ALU.

The major characteristic of ALSU is to perform all the logical, arithmetic, and shift operations. Operations like addition, subtraction, multiplication, and division are called arithmetic operations. Logical operation refers to operation on numbers and special character operations, and it connects two or more phrases of information (expressions). Shift micro-operations are operations for serial transfer of information.

The arithmetic, logic, and shift circuits are combined to one ALU with common selection variables. The diagram below shows one stage of an arithmetic logic shift unit. The subscript i designates a typical stage.



With inputs S1 and S0, a specific micro-operation is chosen. At the output, a 4 x 1 multiplexer selects between an arithmetic output and a logic output. Inputs S3 and S2 select the data in the multiplexer. The multiplexer's other two data inputs receive inputs Ai — 1 for the shift-right operation(**shr**) and Ai + 1 for the shift-left operation (**shl**).

It should be noted that the circuit diagram only depicts one typical stage. For an n-bit ALU, the circuit shown above must be repeated n times. The output that carries C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in the sequence. The input carried to the first stage is the input carry Cin, which provides a selection variable for the arithmetic operations.

The circuit whose one stage is specified in the diagram given above provides eight arithmetic, four logical, and two-shift operations. Each operation is selected with five variables S3, S2, S1, S0, and C_{in}. Here C_{in} is used for selecting an arithmetic operation

only. The table given below is the function table of the Arithmetic Logic Shift Unit.

S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B’	Subtract with borrow
0	0	1	0	1	F = A + B’ + 1	Subtraction
0	0	1	1	0	F = A – 1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = A’	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift Left A into F

The above table shows 14 operations of **ALU**(Arithmetic Logical Unit).

1. The first eight are arithmetic operations(where S3S2=00).

2. The next four operations are logical and are selected with S3S2=01.
3. The final two operations are shift operations, selected with S3S2=10 and 11.