

## REGISTER TRANSFER AND MICROOPERATIONS

### Register Transfer Language

**Digital System** : An interconnection of digital hardware modules that accomplish specific information-processing task.

### How Digital systems are built?

Digital systems are built with digital modules . Those modules are connected with control and data paths.

### How Digital modules are built ?

Digital modules are built with digital components .  
Ex: registers, decoders, arithmetic elements and control logic.

### How Digital modules are defined?

Digital modules are best defined by  
    registers they contain  
    and  
    operations they perform on stored data.

### Micro operations:

The operations executed on data stored in registers are called Micro-operations.  
An elementary operation performed in information stored in register(s).  
Ex: shift, count, clear, increment, load etc.,

### How internal hardware organization of a digital computer is defined ?

Internal hardware organization of a digital computer is best defined by:

    Registers it contains and their functions  
    Sequence of micro operations performed on data inside registers  
    Control that ignites the sequence of micro operations

### Register transfer language:

RTL is set of symbolic notations used to describe micro operations, transfer among registers.

RTL is a system for expressing in symbolic form the micro operation sequences among registers in a digital module

## Designation of Registers:

Registers are designated by capital letters; sometimes followed by numbers to denote the function of a register.

Examples:

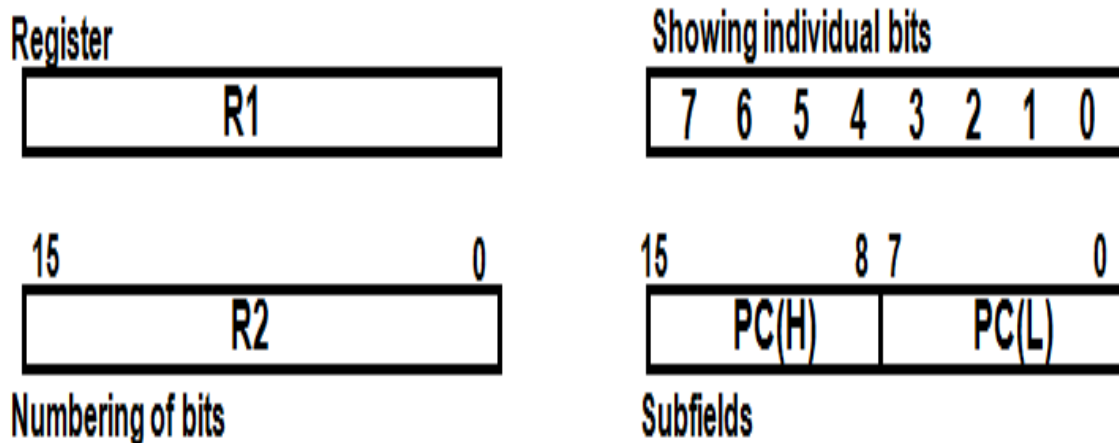
MAR for Memory Address Register

PC for Program Counter

IR for Instruction register

The individual flip flops in n-bit register is numbered from 0 in right most to n-1 in left most . A register can be viewed as a single entity or may also be represented showing the bits of data they contain.

Registers can be designated by a whole register, portion of a register, or a bit of a register.



## Register transfer:

Basic symbols for register transfer :

Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A ← B, B ← A

Register Transfer is defined as copying the content of one register to another.

For register transfers, the data transfer from one register to another is designated in symbolic form by replacement operator.

$R2 \leftarrow R1$

the contents of register R1 are copied (loaded) into register R2

A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse


This is a **non-destructive**; i.e. the contents of R1 are not altered by copying (loading) them to R2 .

**That register transfer also implies that:**

The **data lines** extend from the source register (R1) to the destination register (R2) with lines equal to the bit numbers of R1 and R2.

**Parallel load** occurs in the destination register (R2).

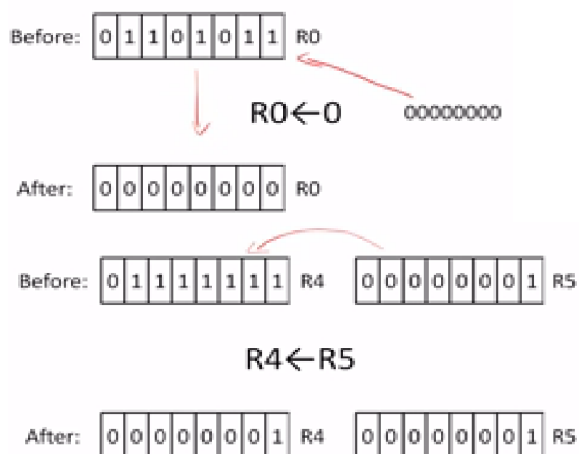
**Control lines** are needed to perform this action .



## Register Transfer

- Data can move from register to register.
- Digital logic used to process data

for example:  $C \leftarrow A + B$



### Control Function:

Transfer has to happen under a certain condition by means of a control signal, called a control function.

This looks if-then statement.

If the signal is “1” then action will take place.

P also could be a combination of Boolean variables which yields a single Boolean output.

P:  $R2 \leftarrow R1$

Which means “if  $P = 1$ , then load the contents of register R1 into register R2”, i.e., if  $(P = 1)$  then  $(R2 \leftarrow R1)$ .

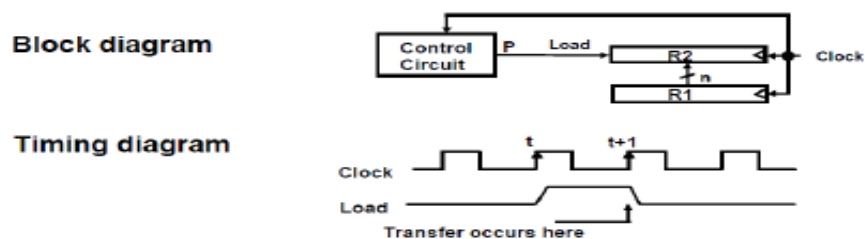
## Hardware Implementation for control function:

R1 transfer to R2.

Output of register R1 is connected to n input of register R2.

Register R2 has a load control activated by P control function and the whole operation is synchronized with the central clock.

The rising edge of the CLK input triggers activates P at t time and at t+1 time the transfer takes place.



Assume : Registers are comprised of D FF that acts on rising edge clocks.

## Simultaneous Operations:

In cases where two or more operations are to occur simultaneously, they are separated with commas as shown next:

P: R3  $\leftarrow$  R5, MAR  $\leftarrow$  IR

If the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR .

## Bus and Memory Transfer

If we need to move data from and to multiple registers , problem arises.

The number of wires will be so large if separate lines are used to connect all registers with each other.

To completely connect n registers we need n(n-1) lines.

So the cost is in order of O(n<sup>2</sup>).

This is not a realistic approach to be used in a large digital system.

The solution is to use a common "Bus".

## New approach:

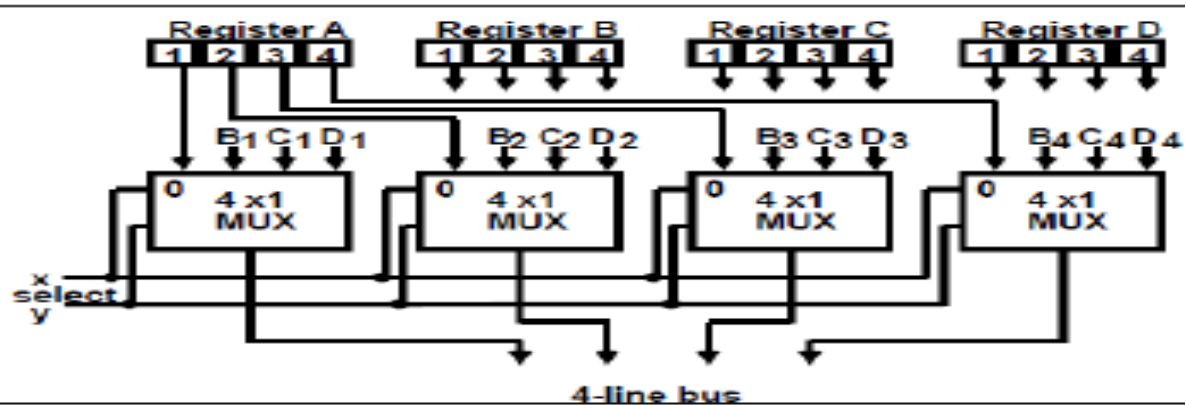
Use one centralized set of circuits for data transfer the "bus".

Also use control circuits to select which register is the source, and which is the destination.

## Definition of a bus:

Bus is a path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

One way of constructing a bus is by using multiplexers.



The bus system will multiplex k registers of n bits each to produce an n-line common bus.  
 The number of multiplexers need is n  
 The size of each multiplexer is K x 1

Another way of constructing a bus is by using buffers or 3-state gates.

### Bus Transfer in RTL:

The transfer of information from a bus into one of many destination registers can be accomplished by connecting bus lines to the inputs of all registers and activating load control of selected destination.

The symbolic statement for a bus transfer may mention the bus or may be implied in the statement.

$R2 \leftarrow R1$

OR

$BUS \leftarrow R1, R2 \leftarrow BUS$

### Memory Transfer:

Memory (RAM) can be thought as a sequential circuits containing some number of registers.

### Registers

0	1	1	0	1	0	1	1
0	0	1	0	1	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1

These registers hold the words of memory.

Each of the  $r$  registers is indicated by an address.

These addresses range from 0 to  $r-1$ .

Each register (word) can hold  $n$  bits of data.

Assume: RAM contains  $r = 2^k$  words.

It needs the following

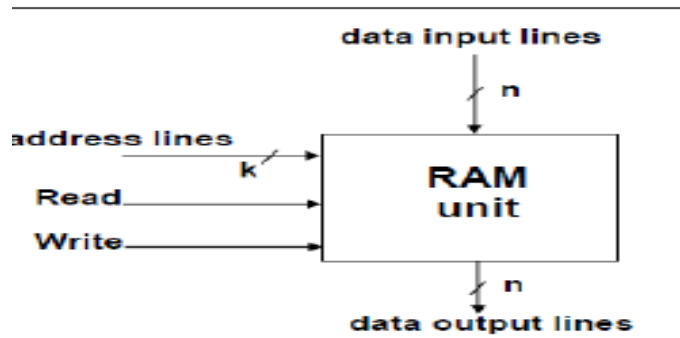
$n$  data input lines

data output lines

$k$  address lines

a Read control line

a Write control line



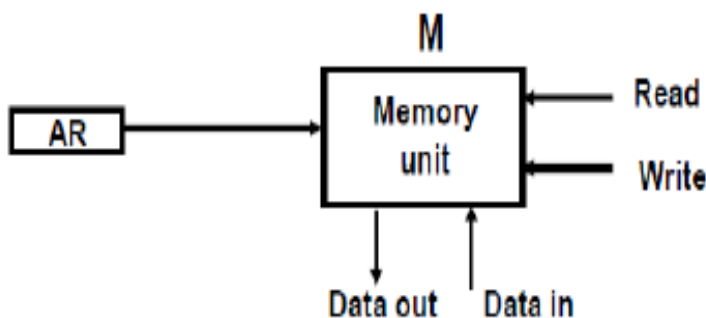
The memory can be viewed at the register level as a device,  $M$ .

We must specify the address in memory we will be using.

Address	← 8 bit →							
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								

Memory is usually accessed in computer systems by putting the desired address in a special register, the Memory Address Register (MAR, or AR).

When memory is accessed, the contents of the MAR get sent to the memory unit's address lines.



**Read Operation:**

when address of required location is transferred into address register AR then the content is loaded into data register DR.

$$DR \leftarrow M[AR]$$

**Write operation:**

The content of data register DR is transferred into memory location addressed by address register AR.

$$M[AR] \leftarrow DR$$

**Arithmetic Micro operation**

A micro operation is an elementary operation performed with data stored in register.

**Micro operations** are classified into:

- Register transfer micro operation
- Arithmetic micro operation
- Logic micro operation
- Shift micro operation

**Basic arithmetic micro operations:**

- Addition
- Subtraction
- Increment
- Decrement
- Arithmetic Shift

The Add micro operation is specified as:

$$R3 \leftarrow R1 + R2.$$

add content of R1 to R2 and store result of addition in R3.

Usually it is implemented using hardware full adders.

The Subtraction is usually implemented using complementation and addition

$$R3 \leftarrow R1 + R2 + 1 (R1 - R2)$$

i.e subtract R2 from R1 by adding the complement of R2 plus 1 to R1.

It is implemented using a full adder a complement circuit.

Increment and decrement are implemented using Up and Down Counter.

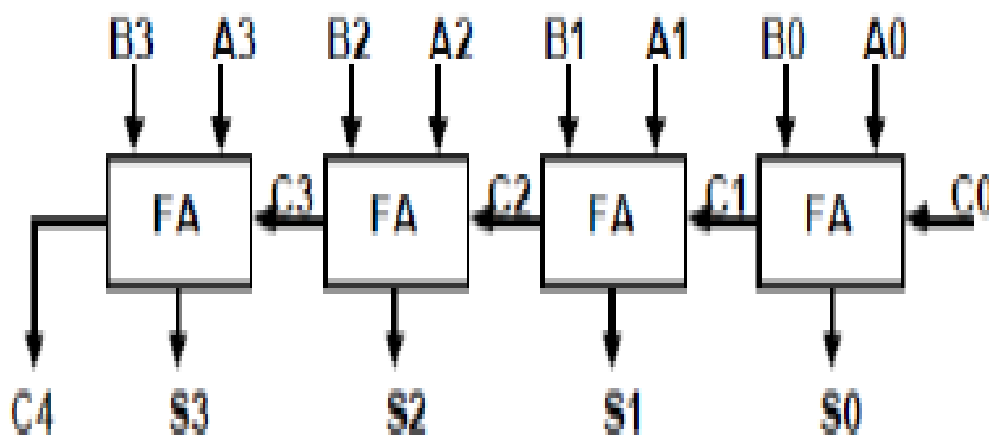
Multiplication and division are implemented using sequence of additions and subtractions respectively.

$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow R2'$	Complement the contents of R2
$R2 \leftarrow R2'+1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + R2'+1$	subtraction
$R1 \leftarrow R1 + 1$	Increment
$R1 \leftarrow R1 - 1$	Decrement

### Binary Adder:

Binary adders are constructed from full adders connected in cascade.

N-bit binary adder circuit need n number of full adders.



### Binary Adder-Subtractor:

Subtraction of  $A - B$  can be done by taking 2's complement of B and added to A.

The 2's complement can be done by taking 1's complement then adding "1" to the result.

And finally the 1's complement is the binary inversion.

The addition and subtraction operations can be combined into one common circuit by including ExOR with each full adder.

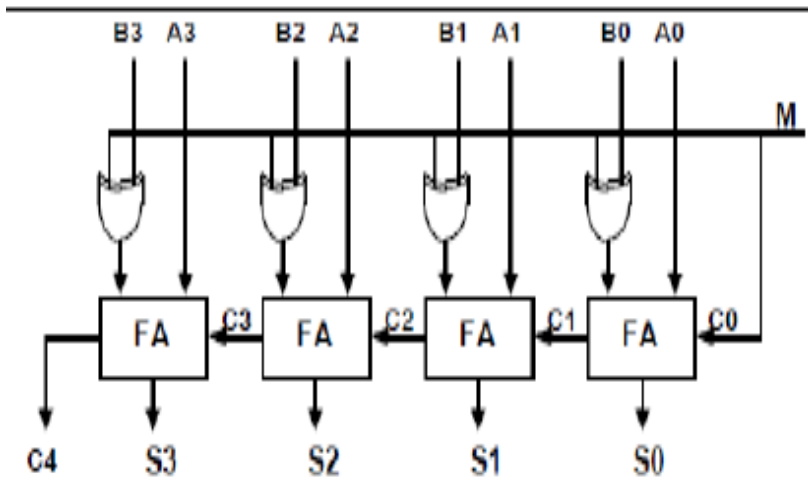
M control addition or subtraction operations.

If  $M=0$  then it is an adder

And

if  $M=1$  then it is a subtractor .





### Binary Incrementor:

The binary incrementor always adds one to the number in a register.

### How can we implement incrementor?

1) using a counter.

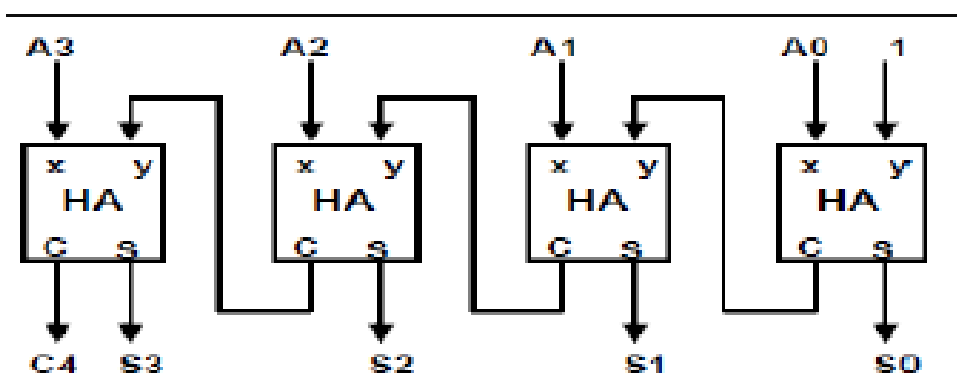
When clock transition arrives the count is incremented.

2) using half adders.

incrementor of 4 bits:

It can be extended to n bits easily.

least significant adder always have one of its input as “1” while its carry is cascaded to other half adders



### Arithmetic Circuit :

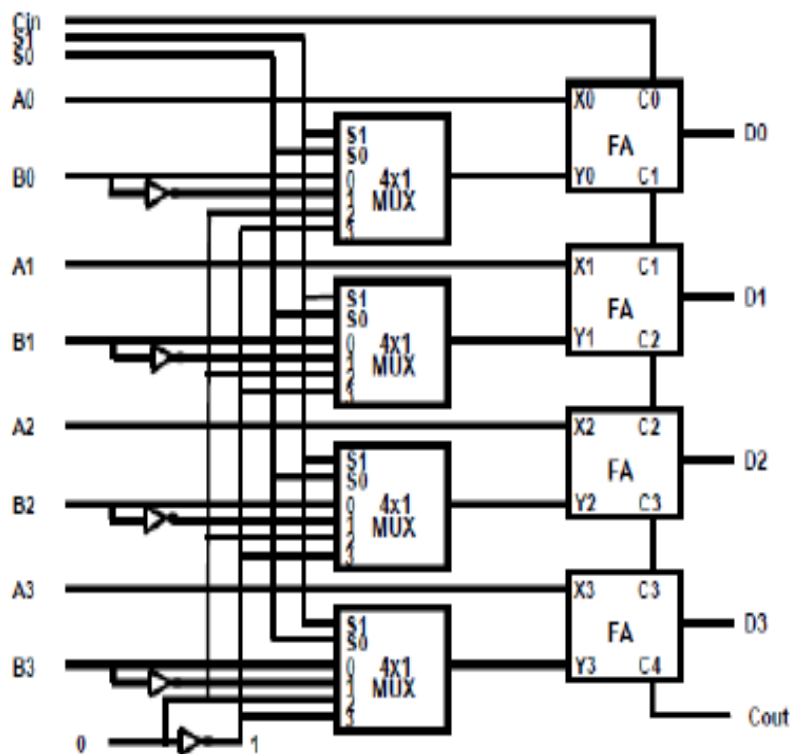
The arithmetic micro operations can be implemented in one composite arithmetic circuit.

This circuit comprised of full adders and multiplexers.

The multiplexer controls which data is fed into Y input of the adder.

The output of the binary adder is computed from  $D = A + Y + C_{in}$

The Y input can have one of 4 different values: B, B', always “1”, or always “0”.



S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B'	$D = A + B'$	Subtract with borrow
0	1	1	B'	$D = A + B' + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

### Logic Micro operation :

Logic micro operation specify binary operations on the strings of bits in registers.

Logic micro operations are bit-wise operations, i.e., they work on the individual bits of data. These are useful for bit manipulations on binary data and also useful for making logical decisions based on the bit value.

There are, in principle, 16 different logic functions that can be defined over two binary input variables.

However, most systems only implement four of these:  
AND ( ), OR ( ), XOR ( ), Complement/NOT

The others can be created from combination of these.

x	y	Boolean Function	Micro-Operations	Name
0	0	$F_0 = 0$	$F \leftarrow 0$	Clear
0	0	$F_1 = xy$	$F \leftarrow A \wedge B$	AND
0	0	$F_2 = xy'$	$F \leftarrow A \wedge B'$	Transfer A
0	0	$F_3 = x$	$F \leftarrow A$	
0	1	$F_4 = x'y$	$F \leftarrow A' \wedge B$	Transfer B
0	1	$F_5 = y$	$F \leftarrow B$	
0	1	$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
0	1	$F_7 = x + y$	$F \leftarrow A \vee B$	OR
1	0	$F_8 = (x + y)'$	$F \leftarrow (A \vee B)'$	NOR
1	0	$F_9 = (x \oplus y)'$	$F \leftarrow (A \oplus B)'$	Exclusive-NOR
1	0	$F_{10} = y'$	$F \leftarrow B'$	Complement B
1	0	$F_{11} = x + y'$	$F \leftarrow A \vee B'$	Complement A
1	0	$F_{12} = x'$	$F \leftarrow A'$	
1	0	$F_{13} = x' + y$	$F \leftarrow A' \vee B$	NAND
1	1	$F_{14} = (xy)'$	$F \leftarrow (A \wedge B)'$	
1	1	$F_{15} = 1$	$F \leftarrow \text{all 1's}$	Set to all 1's

The hardware implementation of logic micro operation requires the insertion of the most important gates like AND, OR, EXOR, and NOT for each bit or pair of bits in the registers.

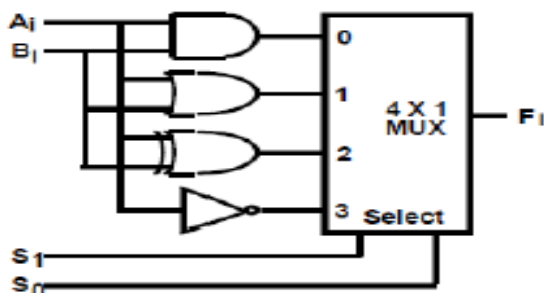
one stage of a circuit:

This circuit generates the four basic logic micro operations.

It consists of four gates and a multiplexer.

The two selection lines of the multiplexer selects one of the four logic operations available at one time.

The circuit shows one stage for bit “i” but for logic circuit of n bits the circuit should be repeated n times but with one remark; the selection pins will be shared with all stages.



**Function table**

$S_1$	$S_0$	Output	$\mu$ -operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	Complement

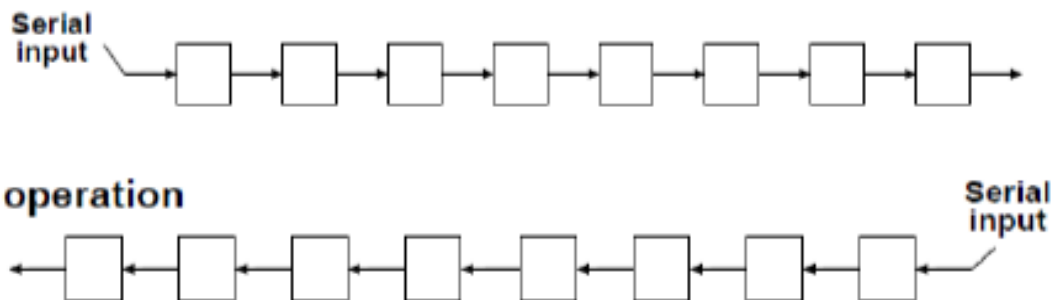
**Shift Micro operations :**

Shift micro-operations are used for serial transfer of data beside they are used in conjunction with arithmetic, logic, and other data processing operations.

There are 3 types of shift micro operations.

What differentiates them is the information that goes into the serial input:

- Logical shift
- Circular shift
- Arithmetic shift



**Logical Shift:**

Logical shift is one that transfers 0 through the serial input.

In a Register Transfer Language, the following notation is used

- shl for a logical shift left
- shr for a logical shift right

Examples:

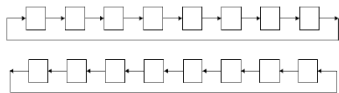
```
R2 ← shr R2
R3 ← shl R3
```

**Circular Shift:**

The circular shift rotates of the register around the two ends without loss of information. This is accomplished by connecting the two ends of the shift register to each other.

- cil indicates circular shift left
- cir indicates circular shift right

Circular shift right and left :



Examples:

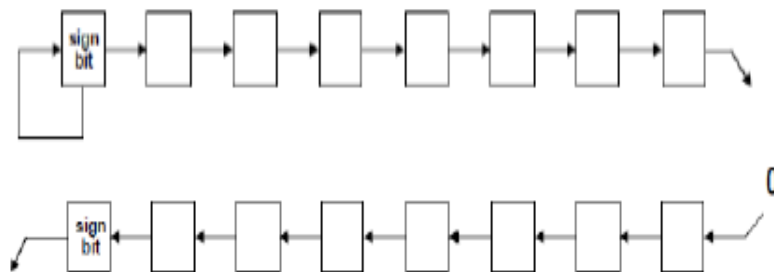
$R2 \leftarrow \text{cir } R2$

$R3 \leftarrow \text{cil } R3$

### Arithmetic Shift:

Arithmetic shift is a micro-operation that shifts a signed binary number to the left or right. Arithmetic shift must leave sign bit unchanged.

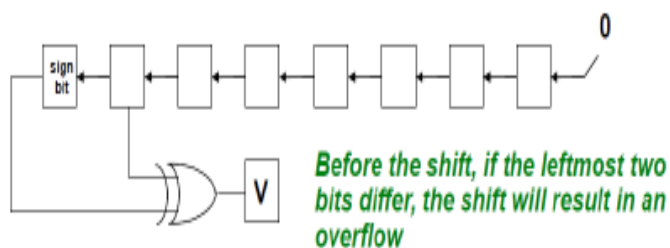
The arithmetic shift right is considered **divide by 2** and left shift is considered **multiply by 2**. Arithmetic Shift Right & Left.



Arithmetic shifts must leave the sign bit unchanged just to preserve the sign of the resulted number.

If that case happened then it will be an overflow.

An overflow flip flop will be used to detect arithmetic shift left overflow .



In a RTL, the following notation is used for arithmetic shifts:

ashl for an arithmetic shift left

ashr for an arithmetic shift right

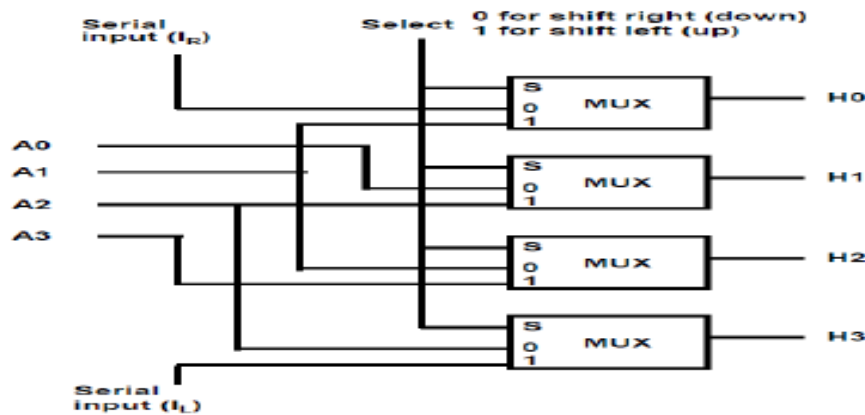
Examples:

$R2 \leftarrow \text{ashr } R2$

$R3 \leftarrow \text{ashl } R3$

## Hardware Implementation:

- 1) Use bidirectional shift register with parallel load.
- 2) another solution can be constructed from multiplexers .



The figure shows four 2 by 1 multiplexers with 4 input lines A0 to A3 and 4 output lines H0 to H3.

The upper (left) multiplexer can take its inputs from serial in (IR) or A0.

The last multiplexer (bottom or right) can take its inputs from A3 or serial input (IL).  
the single line select will select for shift right or left operations.

## Arithmetic Logic Shift Unit

Instead of having individual registers performing micro-operations directly, computer systems employ a number of storage registers connected to a unit called Arithmetic Logic Unit (ALU).

This unit has 2 operands input ports and one output port and a number of select lines to help in selecting different operations.

The ALU is made of combinational circuit so that the entire register transfer operation from the sources to the destination is performed in one clock cycle.

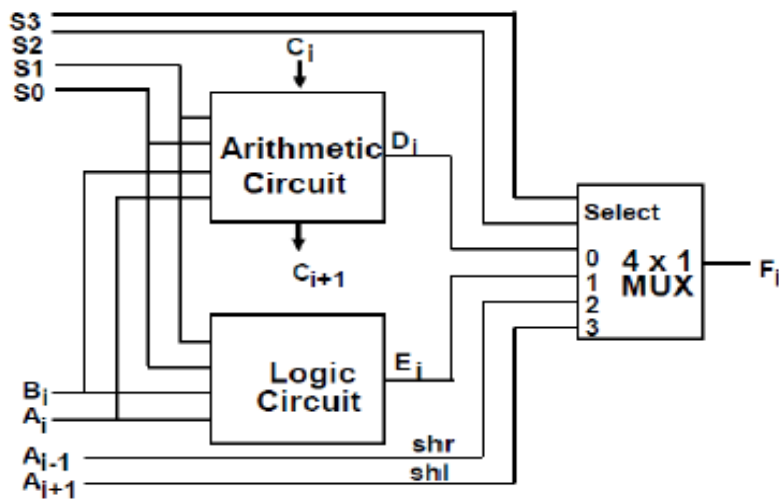
The arithmetic, logic, and shift circuits known previously will be combined in one ALU with common selection inputs.

One stage (bit) of ALSU with its table is shown here.

the arithmetic and logic units will select their operations simultaneously when S0 and S1 are applied;

while S2 and S3 will select one of those unit outputs or a shift left bit stage or shift right bit stage.

The circuit shown provides 8 arithmetic operations, 4 logic operations, and 2 shift operations



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	$F = A \wedge B$	AND
0	1	0	1	X	$F = A \vee B$	OR
0	1	1	0	X	$F = A \oplus B$	XOR
0	1	1	1	X	$F = A'$	Complement A
1	0	X	X	X	$F = \text{shr } A$	Shift right A into F
1	1	X	X	X	$F = \text{shl } A$	Shift left A into F