5.1. REGISTER TRANSFER LANGUAGE

Digital modules contain the registers and perform the operations on the data stored in them. The operations performed or executed on the data stored in registers are called microoperations. A microoperation is a primary or elementary operations which are performed on the on the data stored in one or more registers of the digital modules. The result of the operation may replace the previous information of a register or may be transferred to another register. For example of the microoperations are shift, count, clear, and load. Some of the digital components introduced here in this chapter are registers that implement microoperations.

When you want to specify the internal hardware organization of any digital computer you must define the following:

1. The internal registers set of the computer and their functions.
2. The sequence of microoperations.
3. The control that begins the above sequence.

The register transfer language (RTL) is used to describe the microoperations. This language uses some predefined symbolic notations for each microoperations. The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register. The word "language" is borrowed from programmers, who apply this term to programming languages.

Small Concept

The RTL is used for Microperation.

Any programming language is a procedure for writing notations to specify a given computational and logical process. A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module. It is a convenient tool for describing the internal organization of digital computers in concise and precise manner. It can also be used to facilitate the design process of digital systems.

For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations

The basic features of register transfer language are:

- RTL is a symbolic language,
- RTL is a convenient tool for describing the internal organization of digital computers
- RTL can also be used to facilitate the design process of digital systems

5.2. REGISTER TRANSFER

The computer registers are always designated by capital letters to denote its function for example, The register that holds an address for the memory unit is called MAR, the program counter register is called PC, the instruction register is IR and R1 is a processor register.

The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1, see Fig. 5.1 for the different representations of a registers.

Figure 5.1 shows the different types representation of registers in block diagram form. The most common way to represent a register is by a rectangular box with the name of the

register inside, as in Fig. 5.1(*a*). The individual bits can be distinguished as in (*b*). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (*c*). A 16-bit register is partitioned into two parts in (*d*). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC. The symbol PC (0 – 7) or PC(L) refers to the low-order byte and PC(8 – 15) or PC(H) to the high-order byte.

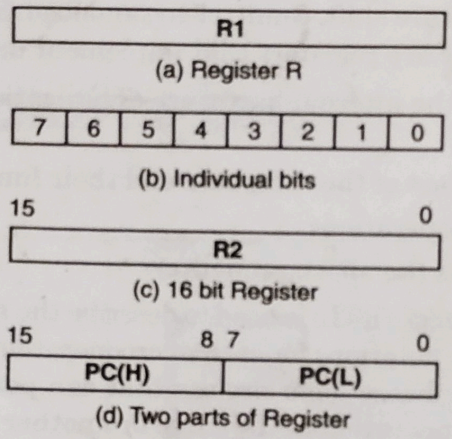

Fig. 5.1. The block diagram of different types of registers

Register Transfer:

- Copying the contents of one register to another is a register transfer
- A register transfer is indicated as

$$R2 \rightarrow R1$$

- In this case 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
- Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2

Control Function :

- Often actions need to only occur if a certain condition is true
- This is similar to an "if" statement in a programming language
- In digital systems, this is often done via a control signal, called a control function, If the signal is 1, the action takes place
- This is represented as:

$$P: R2 \leftarrow R1$$

or,

$$If\ (P = 1)\ then\ (R2 \rightarrow R1)$$

Where *P* is a control function that can be either 0 or 1, which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

Hardware implementation of controlled transfers

$$P: R2 \leftarrow R1$$

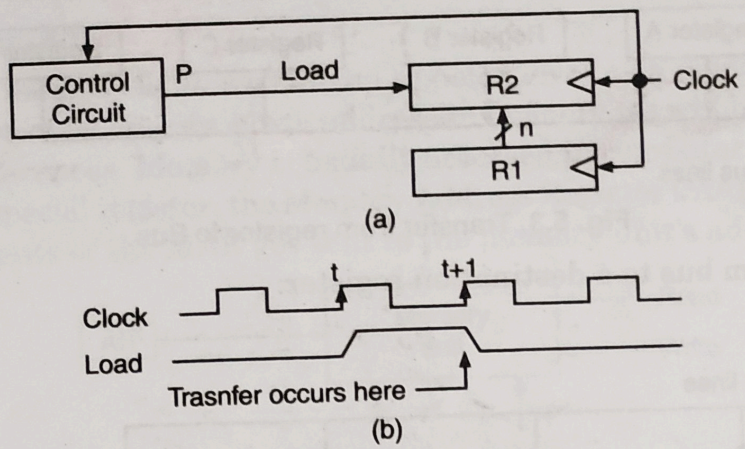

Fig. 5.2. (a) Block Diagram (b) Timining Diagram

The same clock controls the circuits that generate the control function and the destination register. Registers are assumed to use positive-edge-triggered flip-flops.

Simultaneous Operations :

- If two or more operations are to occur simultaneously, they are separated with commas

P: R3 ← R5, MAR ← IR

- Here, 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

Table 5.1. Basic Symbols for Register Transfers

Symbols	Description	Example
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

5.3. BUS AND MEMORY TRANSFERS

In the typical digital computer registers uses the physical paths to transfer information form one register to another. The computer contains huge number of registers therefore number of wires will be excessive if separate lines are used between each register.

A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the onus during each particular register transfer.

Bus Transfer in RTL

- To transfer from a register to bus:

BUS ← R

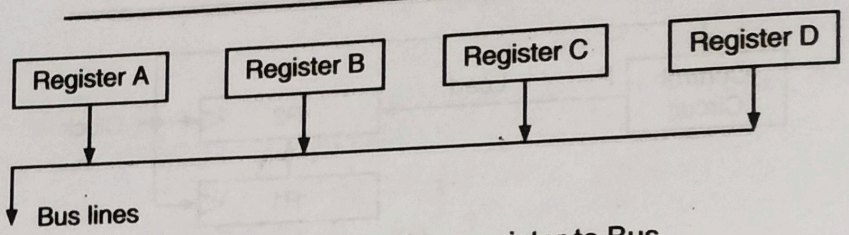

Fig. 5.3. Transfer from register to Bus

- To transfer from bus to a destination register:

  R2 ← BUS

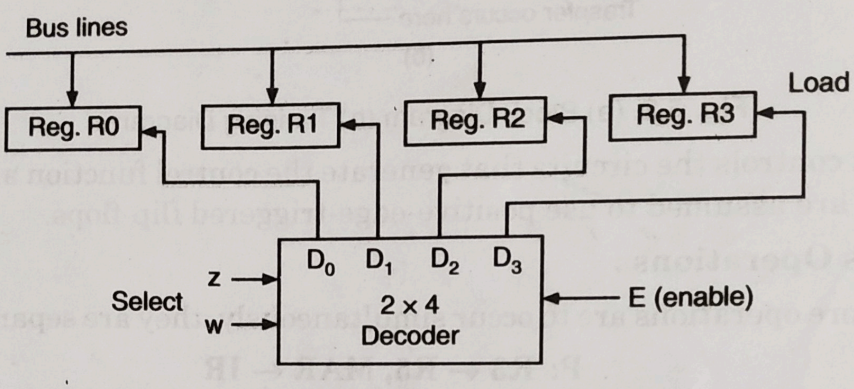

Fig. 5.4. Transformation from Bus to destination register

- Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either

  R2 ← R1 or BUS ← R1, R2 ← BUS

In the former case the bus is implicit, but in the latter, it is explicitly indicated.

Memory (RAM) is also a sequential circuit containing some number of registers and these registers hold the words of memory and following features:

- 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 the RAM contains r = 2k words. It needs the following
  - n data input lines
  - n data output lines
  - k address lines
  - A Read control line
  - A Write control line

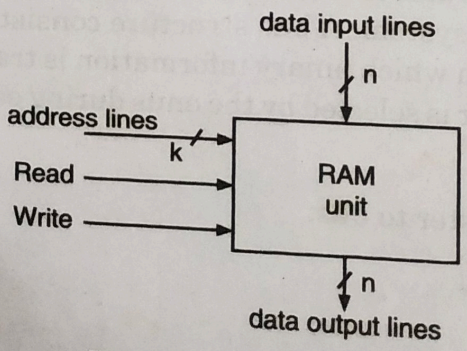

Fig. 5.5. Memory unit

Memory Transfer in RTL

Collectively, the memory is viewed at the register level as a device, M. Since it contains multiple locations, we must specify which address in memory we will be using. This is done by indexing memory references. 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.

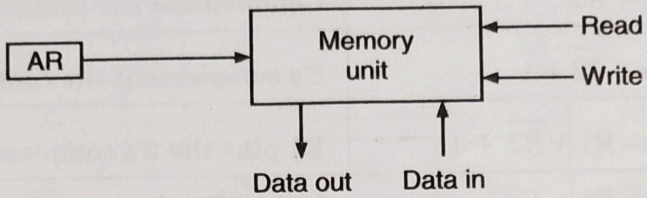

Fig. 5.6.

- **Memory Read Operations**
  To read a value from a location in memory and load it into a register, the register transfer language notation looks like this:

  R1 ← M[MAR]

- This causes the following to occur
  - The contents of the MAR get sent to the memory address lines
  - A Read (= 1) gets sent to the memory unit
  - The contents of the specified address are put on the memory's output data lines
  - These get sent over the bus to be loaded into register R1

5.4. MICRO OPERATIONS

A micro-operation is an elementary operation performed with the data stored in registers. The micro-operations most often encountered in digital computers are classified into four categories:

1. **Register Transfer Micro operations:** transfer binary information from one register to another
2. **Arithmetic Micro operations:** perform arithmetic operations on numeric data stored in registers
3. **Logic Micro operations:** perform bit manipulation operations on non-numeric data stored in registers
4. **Shift Micro operations:** perform shift operations on data stored in registers

The register transfer micro operation does not change the content at the time of transfer from source to destination register. Unlike register transfer, other three micro operations change the information content during the operations. We have already discussed the register transfer micro operation in the section 5.3. The remaining three micro operations are

Small Concept

Micro operations always used on registers.

5.4.1. Arithmetic Micro operations

The basic arithmetic micro operations are addition, subtraction, increment, decrement, and shift and these are shown in the table with examples:

Table 5.2: Arithmetic Micro operations

Micro operations.	Symbolic designation	Description
Addition	R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
Subtraction	R3 ← R1 – R2	Contents of R1 minus R2 transferred to R3
1's Complement	R2 ← $\overline{R2}$	Complement the contents of R2 (1's complement)
2's Complement	R2 ← $\overline{R2}$ + 1	2's complement the contents of R2 (negate)
Subtraction	R3 ← R1 + $\overline{R2}$ + 1	R1 plus the 2's complement of R2 (subtraction)
Increment by 1	R1 ← R1 + 1	Increment the contents of R1 by one
Decrement by 1	R1 ← R1 – 1	Decrement the contents of R1 by one

The basic set of arithmetic micro operations not includes the multiplication and division arithmetic. The addition and subtraction can be implemented by digital circuits which performs the addition and subtraction such as half-adder, full-adder, binary adder, binary subtractor and binary adder-subtractor.

The arithmetic micro operations listed in above table 5.2 can be implemented in one composite arithmetic circuit. The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations. Multiplexers are used to choose between the different operations. The output of the binary adder is calculated from the following sum:

$$D = A + Y + C_{in}$$

where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. Cin is the input carry, which can be equal to 0 or 1. By controlling the value of Y with the two selection inputs S1 and S0 ad making Cin equal to 0 or 1, it is possible to generate the eight arithmetic micro operations listed in Table 5.3.

Table 5.3: Truth table for 4-bit Arithmetic Circuit

Micro operation	Select			Input	Output
	S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$
Addition	0	0	0	B	$D = A + B$
Add with Carry	0	0	1	B	$D = A + B + 1$
Subtract with Borrow	0	1	0	$\overline{B}$	$D = A + \overline{B}$
Subtraction	0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$
Transfer A	1	0	0	0	$D = A$
Increment A	1	0	1	0	$D = A + 1$
Decrement A	1	1	0	1	$D = A - 1$
Transfer A	1	1	1	1	$D = A$

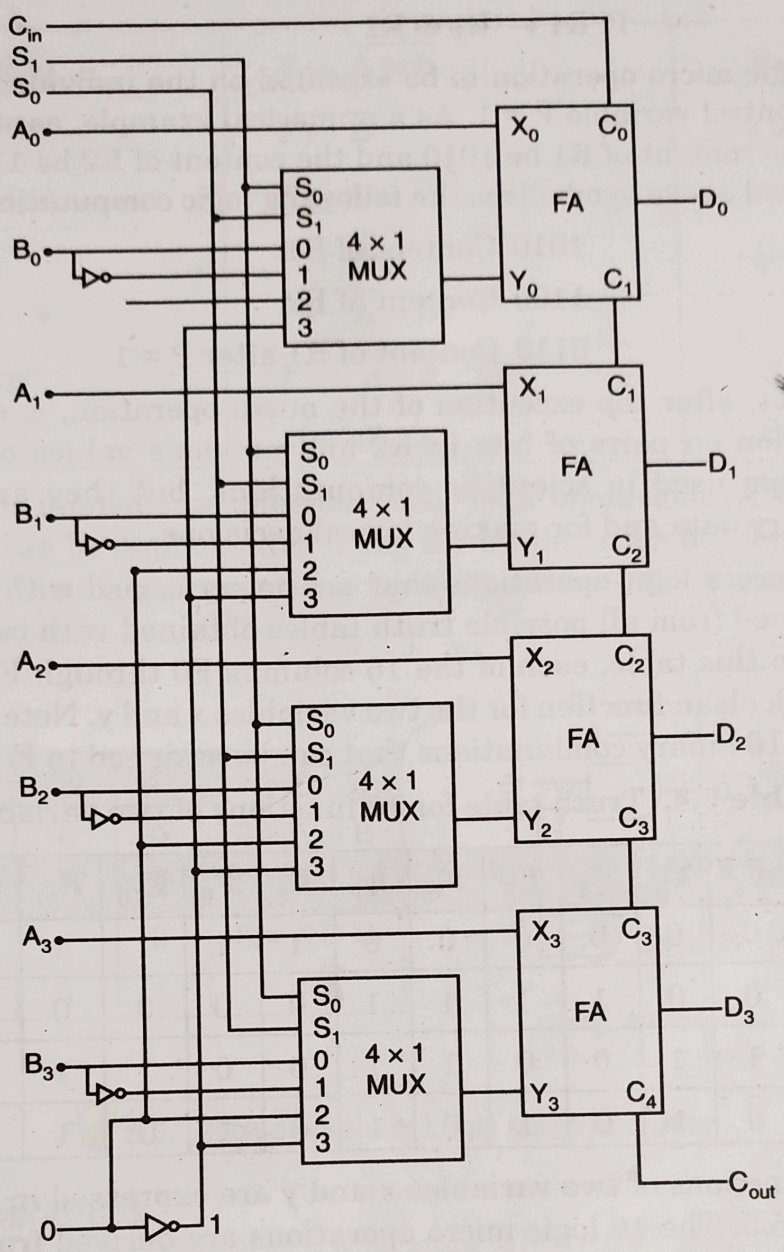

Fig. 5.7. 4-bit arithmetic circuit

When $S_1S_0 = 10$, the input from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes $D = A + 0 + C_{in}$. This gives $D = A$ when $C_{in} = 0$ and $D = A + 1$ when $C_{in} = 1$. In the first case we have a direct transfer from input A to output D. In the second case, the value of A is incremented by 1.

When $S_1S_0 = 11$, all 1's are inserted into the Y inputs of the adder to produce the decrement operation $D = A - 1$ when Cin. This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A +2's complement of 1 = A - 1. When $C_{in} = 1$, then $D = A - 1 + 1 = A$, which causes a direct transfer from input A to output D. Note that the microoperation $D = A$ is generated twice, so there are only seven distinct micro operations in the arithmetic circuit.

5.4.2. Logic Micro Operations

Logic micro operations specify binary operations for strings of bits stored in registers. These operations consider each it 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

$$P: R1 \leftarrow R1 \oplus R2$$

It specifies a logic micro operation to be executed on the individual bits of the registers provided that the control variable P = 1. As a numerical example, assume 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 symbolizes 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 execution of the micro operation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1. The logic micro operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables as shown in Table 5.4 in this table, each of the 16 columns F0 through F15 represents a truth table of one possible Boolean function for the two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

Table 5.4. Truth table for 16 functions of two variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table 5.5. The 16 logic micro operations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B. As shown in the table 5.5:

Table 5.5: Logic Micro operations

S.No.	Boolean Function	Symbolic Operation	Micro operations
1	F0 = 0	$F \leftarrow 0$	Clear
2	F1 = xy	$F \leftarrow A \cap B$	AND
3	F2 = xy'	$F \leftarrow A \cap B$	
4	F3 = x	$F \leftarrow A$	Transfer A
5	F4 = x'y	$F \leftarrow A \cap B$	
6	F5 = y	$F \leftarrow B$	Transfer B
7	F6 = x ⊕ y	$F \leftarrow A \oplus B$	Exclusive-or
8	F7 = x + y	$F \leftarrow A \cup B$	OR
9	F8 = (x + y)'	$F \leftarrow \overline{A \cup B}$	NOR

10	F9 = (x ⊕ y)'	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
11	F10 = y'	$F \leftarrow \overline{B}$	Complement B
12	F11 = x + y'	$F \leftarrow A \cup \overline{B}$	
13	F12 = x'	$F \leftarrow \overline{A}$	Complement A
14	F13 = x' + y	$F \leftarrow \overline{A} \cup B$	
15	F14 = (xy)'	$F \leftarrow \overline{A \cap B}$	NAND
16	F15 = 1	F ← all 1's	Set to all 1's

Most of the systems implements only four basic logic operations (AND, OR , XOR , NOT) because all the others can be implemented using the combination of these four operations. To implement these four basic logic operations figure 5.x shows the digital circuit logic micro operations and truth table.

Micro operation	S0	S1	Output
AND	0	0	E = A ∩ B
OR	0	1	E = A ∪ B
XOR	1	0	E = A ⊕ B
NOT (Complement)	1	1	$E = \overline{A}$

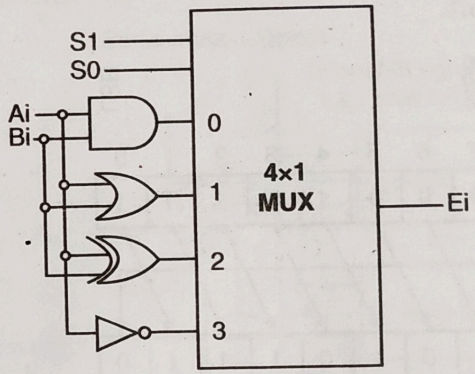

Fig. 5.8. Block diagram of Logic Micro operation

5.4.3. Shift Micro operations

Shift Micro operations are used to serial transfer data content of a register into either left or right. They are also used in the arithmetic operations, logic operations and other data processing operations.

In the shift operations only flip-flop of the register exchanges their content either from right or left flip-flop according to the input received. Shift operations act on bytes, words and longwords in data registers, but only words in memory.

There are three types of shift micro operations:

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

Logical Shift

In a logical shift all the bits are moved left or right and zero enters at the input of the shifter. A logical shift left is indicated by SHL and a shift right by SHR. In the Fig. 5.9 logical shift left and logical shift right are shown for one bit, where MSB and LSB are the most and least significant bits respectively.

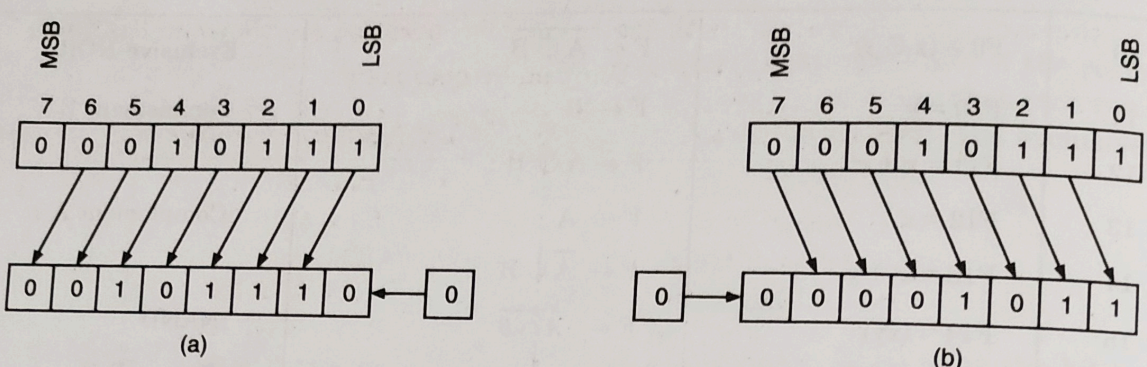

Fig. 5.9. Logical Shift Micro operations (a) Left Shift (b) Right shift

Circular Shift

In a circular shift the bit shifted out is moved to the position of the bit shifted in. The bits are shifted left by CIL, circulate left, and right by CIR, circulate right. No bit is lost during a circular shift. The bit copied from one end of the register to the other is also copied into the carry bit.

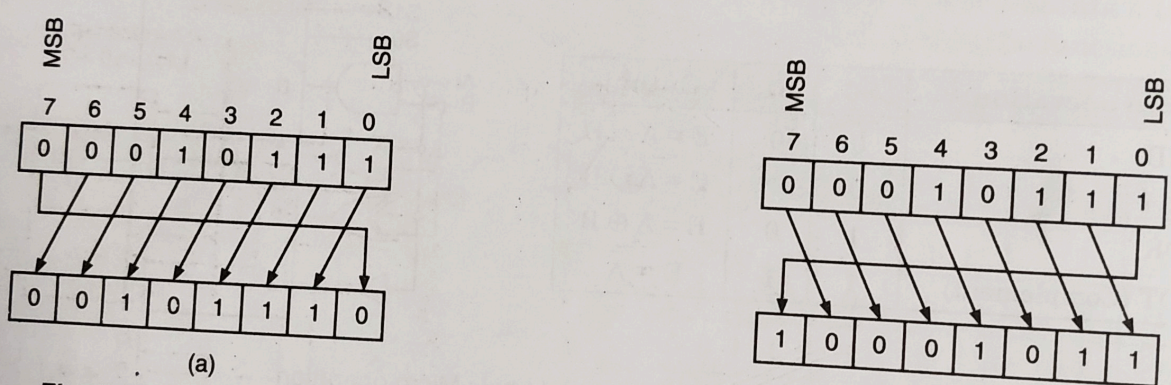

Fig. 5.10. Circular Shift Micro operations (a) Circulate Left (CIL) (b) Circulate Right (CIR)

Arithmetic Shift

An arithmetic shift left, ASL, is almost identical to a logical shift left. An arithmetic shift right, ASR, causes the most significant bit, the sign-bit, to be propagated right and, therefore, preserves the correct sign of a two's complement value. For example, if the bytes $00101010 = 42_{10}$ and $10101010 = -86_{10}$ are shifted one place right by the instruction ASR, the results of the arithmetic shift are $0001010 = 21_{10}$ and $11010101 = -43_{10}$ respectively. Arithmetic shifts are intended to be used in conjunction with two's complement arithmetic.

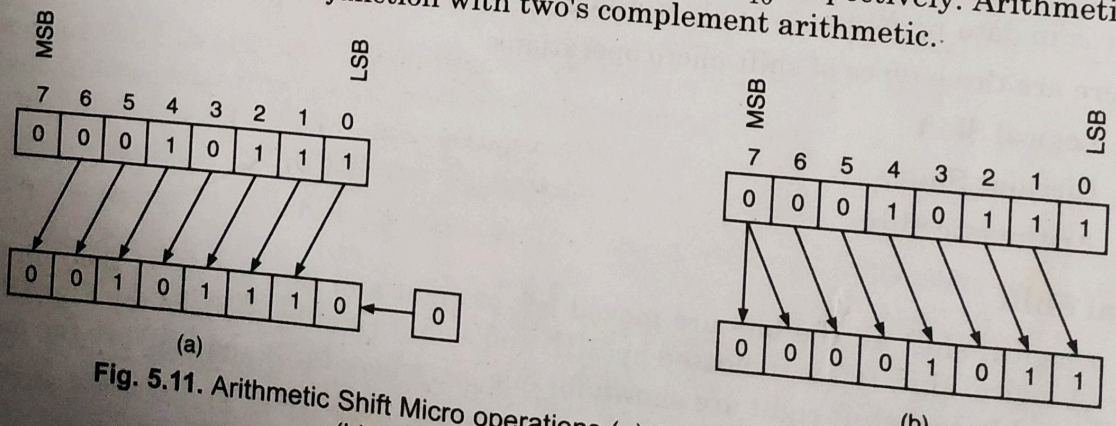

Fig. 5.11. Arithmetic Shift Micro operations (a) Arithmetic Left shift (ASHL) (b) Arithmetic Right shift (ASHR)

All the shift micro operations with their symbolic operation are shown in the Table 5.6.

Table 5.6: Shift micro operations

Micro operation	Symbolic Operation
Logical Shift Left	R2 ← SHL R2
Logical Shift Right	R2 ← SHR R2
Circular Shift Left	R2 ← CIL R2
Circular Shift Right	R2 ← CIR R2
Arithmetic Shift Left	R2 ← ASHL R2
Arithmetic Shift Right	R2 ← ASHR R2

These micro operations (shift, logical and arithmetic) can be implemented either by a bidirectional shift-register with parallel load or by a combinational circuit shifter with multiplexers. A combinational circuit shifter can be constructed with help of multiplexers. A 4-bit shifter circuit shown in Fig. 5.12, which has four data inputs, A_0 through A_3 and four outputs H_0 through H_3. Two serial inputs are used IL and IR.

Select	Output			
S	H_0	H_1	H_2	H_3
0	IR	A_0	A_1	A_2
1	A_1	A_2	A_3	IL

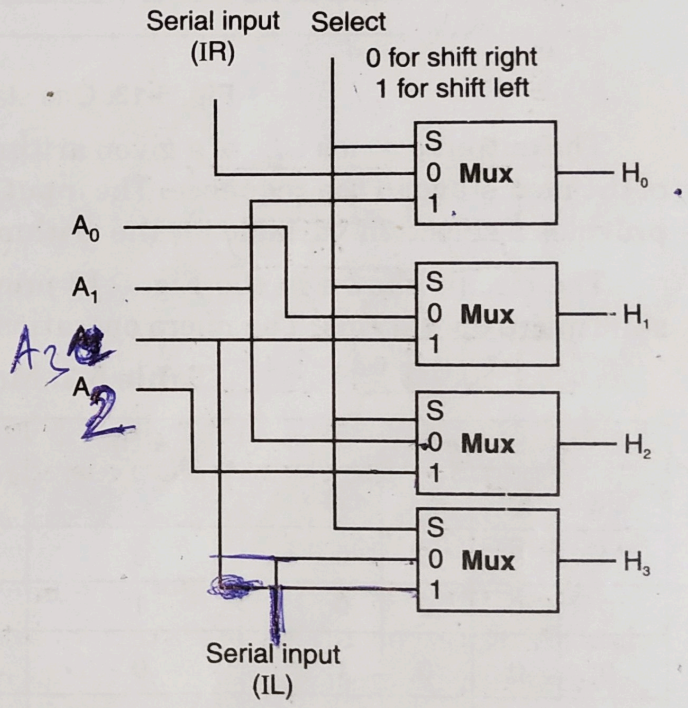

Fig. 5.12. Block diagram of Shift Micro operation

5.5. ALU (ARITHMETIC + LOGIC + SHIFT)

As we discussed the design of various circuits for arithmetic, logic and shift micro operations. Instead of having separate registers performing the each micro operations, a combined circuit can be design for all micro operation. The combination of arithmetic, logic and shift circuit into a single operational unit is called an arithmetic logic unit (ALU).

The arithmetic, logic and shift circuits can be combined into one ALU with common selection variables as shown in the Fig. 5.13.

One stage of arithmetic logic unit is shown in Fig. 5.13. the subscript i indicate a typical stage. Input Ai and Bi are applied to both arithmetic and logic units. A particular micro-operation is selected with S1 and S0. The choice between an arithmetic output E_i and a logic output Hi can be achieve through a 4 x1 multiplexer at the output. The data are selected with

inputs S3 and S2. The other two data inputs to the multiplexer are A_{i-1} for the shift right operation and A_{i+1} for the shift-left operation.

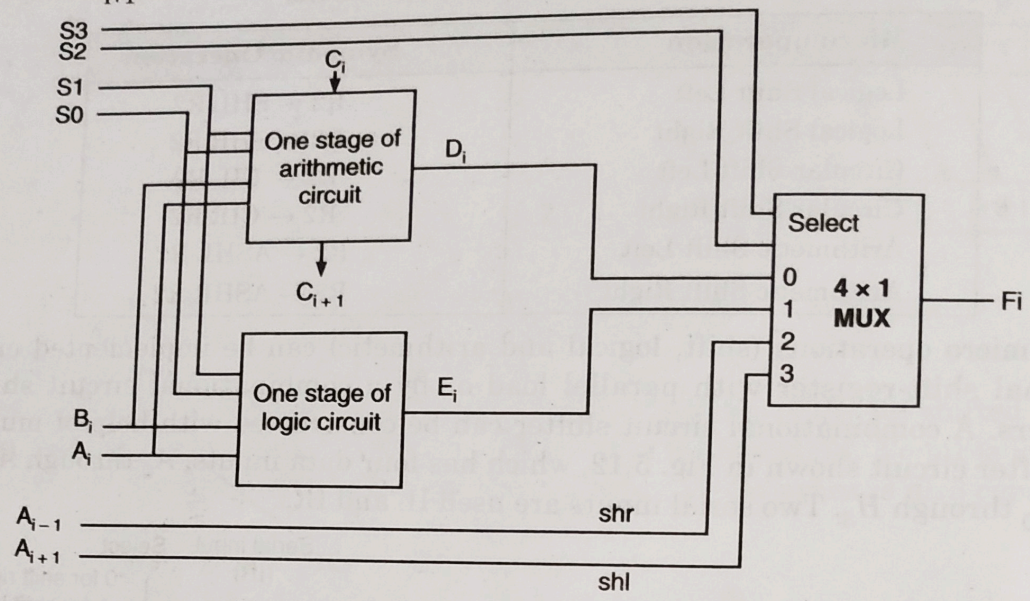

Fig. 5.13. One stage of arithmetic logic unit

The output 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 carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations.

The circuit shown in the Fig. 5.13 provides all eight arithmetic, four logic and only two shift micro operations. The micro operations are shown in the Table 5.7.

Table 5.7: Micro operation of ALU

Select				Cin	Symbolic Operation	Micro operation
S3	S2	S1	S0			
0	0	0	0	0	F = A + B	Addition
0	0	0	0	1	F = A + B + 1	Add with Carry
0	0	0	1	0	F = A + $\overline{B}$	Subtract with Borrow
0	0	0	1	1	F = A + $\overline{B}$ + 1	Subtraction
0	0	1	0	0	F = A	Transfer A
0	0	1	0	1	F = A + 1	Increment A
0	0	1	1	0	F = A – 1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	×	F = A ∩ B	AND
0	1	0	1	×	F = A ∪ B	OR
0	1	1	0	×	F = A ⊕ B	XOR
0	1	1	1	×	F = $\overline{A}$	NOT (Complement)
1	0	×	×	×	F = SHL A	Shift left A into F
1	1	×	×	×	F = SHR A	Shift right A into F