UNIT-I

**Register Transfer and Micro-Operations**:

**CONTENTS:**

1. Register Transfer Language,
2. Register Transfer,
3. Memory Transfers-Bus transfer,
4. Bus construction with Multiplexers,
5. Arithmetic Micro-operations – Binary Adder, Binary Adder – Subtractor, Binary Incrementer, Arithmetic circuit.
6. Logic Micro-operations- List of Micro operations,
7. Hardware Implementation, Some Applications.
8. Shift Micro-operations - Hardware Implementation,
9. Arithmetic Logic Shift Unit.

**Register Transfer language:**

* Combinational and sequential circuits can be used to create simple digital systems
* Logic gates are the low-level building blocks of a digital computer
* Simple digital systems are frequently characterized in terms of
* 1) The registers they contain,
* 2) The operations that are performed on data stored in them
* The operations executed on the data in registers are called micro-operations ex: shift, count, clear, load

**The internal hardware organization of a digital system is best defined by specifying:**

* The set of registers and the flow of data between them.
* The sequence of micro-operations performed on the data which are stored in the registers.
* The control paths that initiates the sequence of micro-operation

Rather than specifying a digital system in words, a specific notation is used , Register Transfer Language

The symbolic notation used to describe the micro-operation transfer among register is called **register transfer language**

**Register Transfer language**

* A symbolic language
* A convenient tool for describing the internal organization of digital computer
* Can also be used to facilitate the design process of digital systems
* Registers are designated by capital letters sometimes followed by numbers(ex: A,R13)

Often the names indicate function

MAR: memory address register

PC: Program Counter

IR: Instruction Register

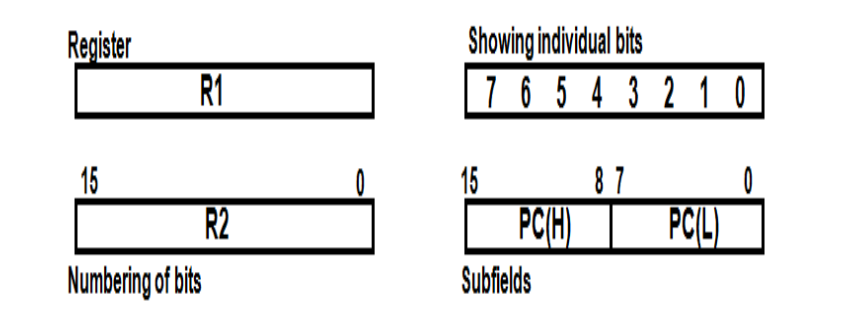
Registers and their contents can be viewed and represented in different ways

A register can be viewed as a single entity

MAR

**Designation of a Register**

* a register
* Portion of a register
* A bit of a register



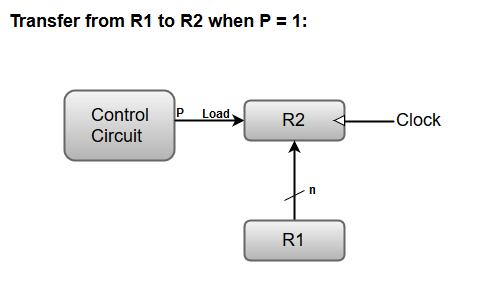
**Register Transfer**

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

R2🡨R1

* In this case contents of R1 are copied into R2
* A simultaneous transfer of all bits from 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 loading them to R2
* Typically, most of the users want the transfer to occur only in a 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.
* It is more convenient to specify a control function (P) by separating the control variables from the register transfer operation. For instance, the following statement defines the data transfer operation under a specific control function (P).
* P:  R2 ← R1

**Hardware Implementation**

****

Here, the letter 'n' indicates the number of bits for the register. The 'n' outputs of the register R1 are connected to the 'n' inputs of register R2.

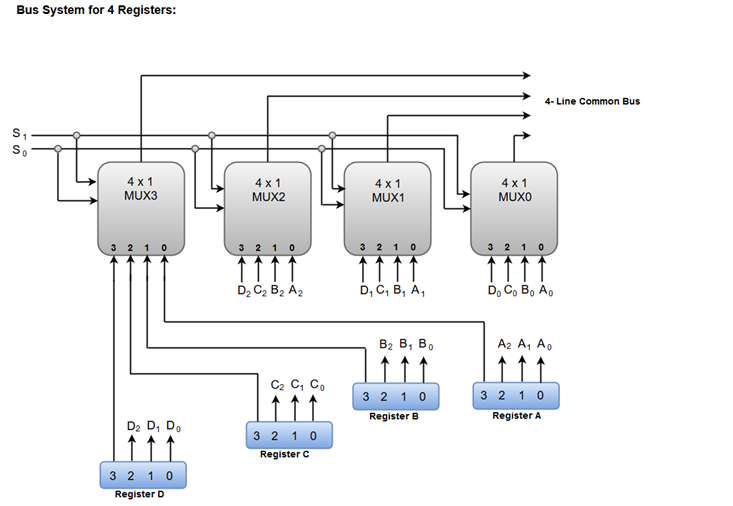
A load input is activated by the control variable 'P' which is transferred to the register R2.

**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.

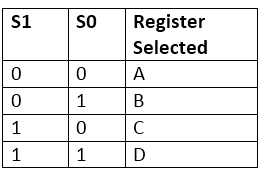
****

The 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.

****

**Memory Transfer**

* 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:

**Read:  DR ← M [AR]**

**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)

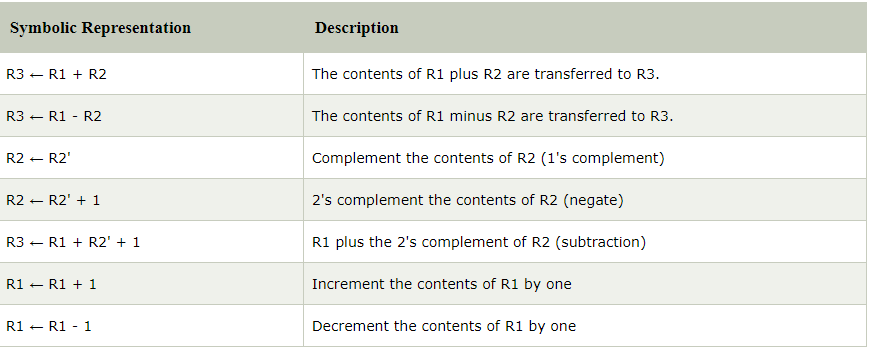
**Arithmetic microoperations**

In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

The basic Arithmetic Micro-operations are classified in the following categories:

1. Addition
2. Subtraction
3. Increment
4. Decrement

The following table shows the symbolic representation of various Arithmetic Micro-operations.

****

**Binary Adder**

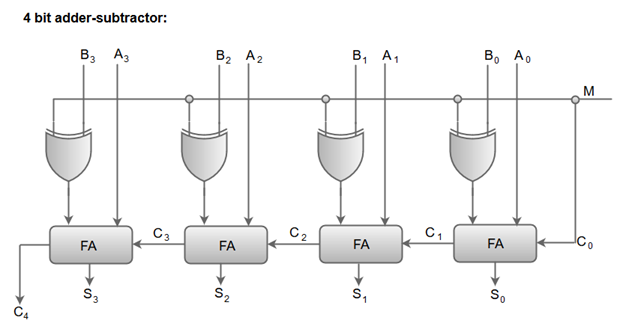
* The Add micro-operation requires registers that can hold the data and the digital components that can perform the arithmetic addition.
* A Binary Adder is a digital circuit that performs the arithmetic sum of two binary numbers provided with any length.
* A Binary Adder is constructed using full-adder 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.

****

* 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.

# Binary Adder-Subtractor

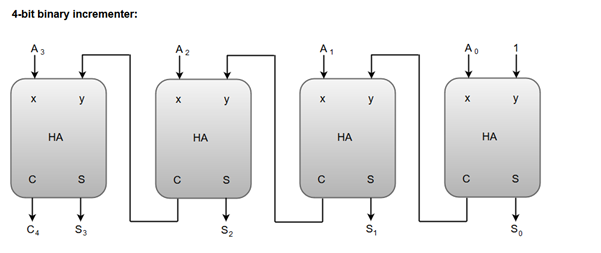
The Subtraction micro-operation can be done easily by taking the 2's compliment of addend bits and adding it to the augend bits.

****

* When the mode input (M) is at a low logic, i.e. '0', the circuit act as an adder and when the mode input is at a high logic, i.e. '1', the circuit act as a subtractor.
* The exclusive-OR gate connected in series receives input M and one of the inputs B.
* When M is at a low logic, we have B⊕ 0 = B.  
  The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
* When M is at a high logic, we have B⊕ 1 = B' and C0 = 1.  
  The B inputs are complemented, and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B.

# Binary Incrementer

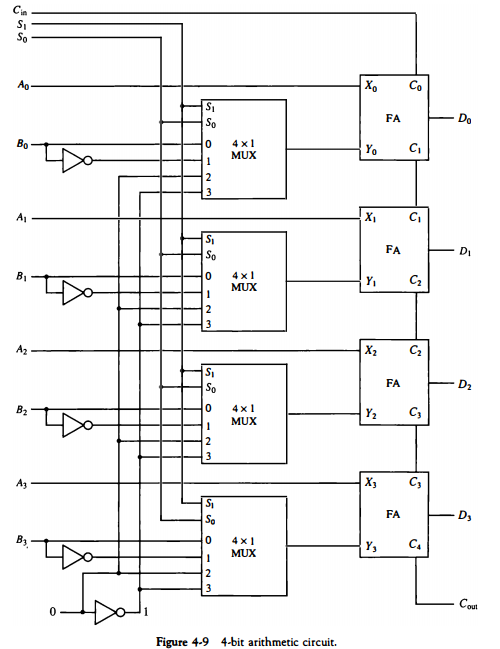
The increment micro-operation adds one binary value to the value of binary variables stored in a register. For instance, a 4-bit register has a binary value 0110, when incremented by one the value becomes 0111.

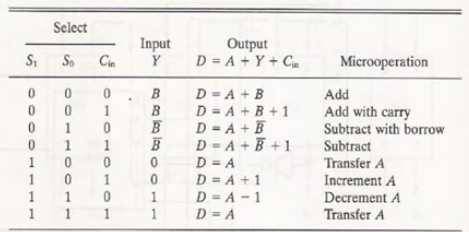
****

* 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

**Arithmetic Circuit**

* 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



****

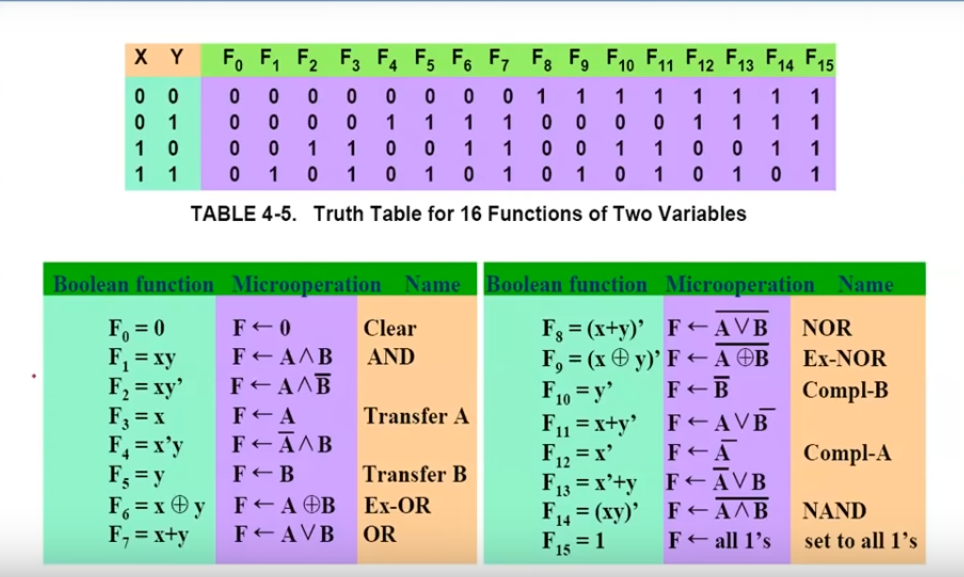
**Logic Microoperations**

* **Logic microoperations** 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 microoperation with the contents of two registers R 1 and R2 is symbolized by the statement P: R1 ← R1 ⊕ R2

* **It specifies a logic microoperation** 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.
* **Special symbols** will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions.
* **The symbol V** will be used to denote an OR microoperation and the symbol ∧ to denote an AND microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.
* in the statement P + Q: R1 ← R2 + R3, R4 ← R5 ∨ R6 the + between P and Q is an OR operation between two binary variables of a control function.

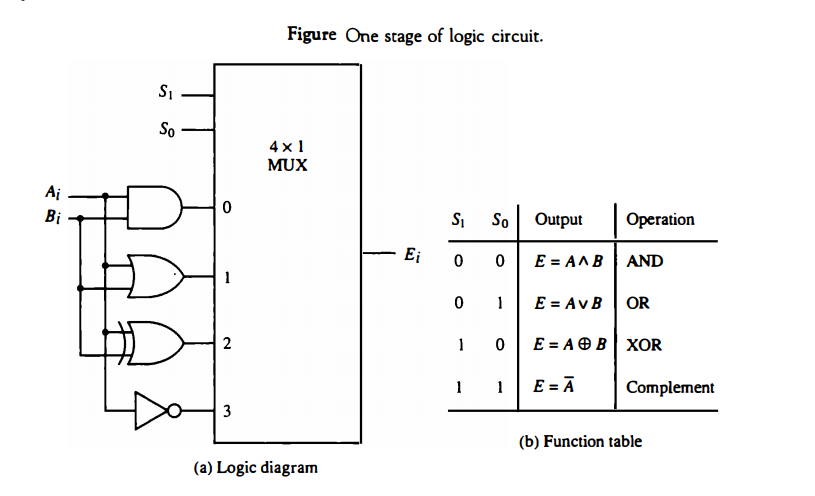
**The + between R2 and R3** specifies an add microoperation. The OR microoperation is designated by the symbol V between registers R5 and R6.

### List of Logic Microoperations

****

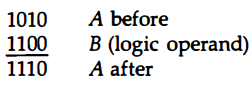
### Hardware Implementation

* **The hardware implementation** of logic rnicrooperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
* **Although there are 16 logic rnicrooperations**, most computers use only four-AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.
* **Figure below** shows one stage of a circuit that generates the four basic logic rnicrooperations .

****

**Applications of Logic Microoperations**

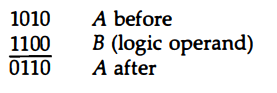
* The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B.

**The following numerical example** clarifies this operation:  


**From the truth table** we note that the bits of A after the operation are obtained from the logic-OR operation of bits in B and previous values of A

**Selective Complement:**

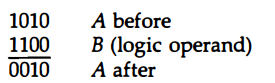
**The selective-complement** operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B. For example:

****

**Therefore**, the exclusive-OR rnicrooperation can be used to selectively complement bits of a register.

**Selective clear**

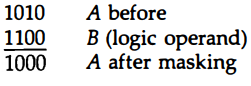
**The selective-clear** operation clears to 0 the bits in A only where there are corresponding 1's in B. For example:

****

**One can deduce** that the Boolean operation performed on the individual bits is AB'. The corresponding logic microoperation is 

**Mask**

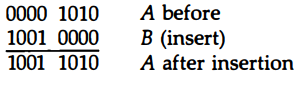
**The mask operation** is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. The mask operation is an AND micro operation as seen from the following numerical example:



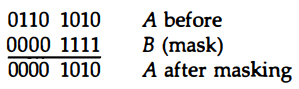
**Insert**

The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value. For example, suppose that an A register contains eight bits, 0110 1010.

**To replace the four leftmost bits** by the value 1001 we first mask the four unwanted bits:

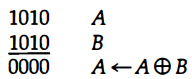


**and then** insert the new value:



**CLEAR**

**The clear operation** compares the words in A and B and produces an all 0' s result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation as shown by the following example:



### Shift Microoperations

**Shift microoperations** are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations.

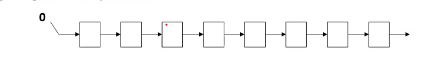
**The contents of a register** can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input.

* **During a shift-left** operation the serial input transfers a bit into the rightmost position.
* **During a shift-right** operation the serial input transfers a bit into the leftmost position.

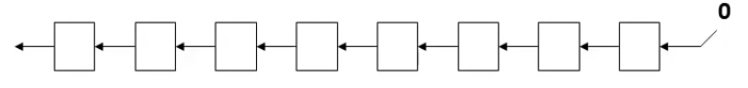
**There are three types of shifts:** logical, circular, and arithmetic.

**Logical Shift**

Right Logical shift

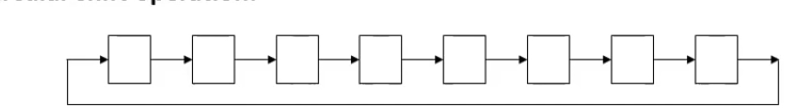


Left Logical Shift

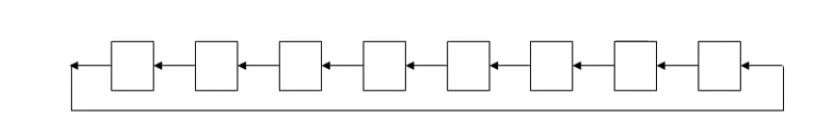


**Circular Shift**

**Right Circular Shift**

****

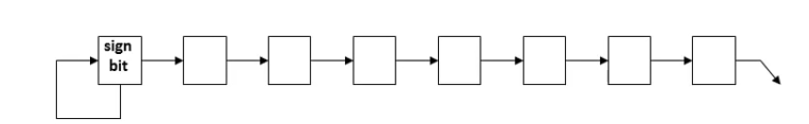
**Left Circular Shift**

****

**Arithmetic shift**

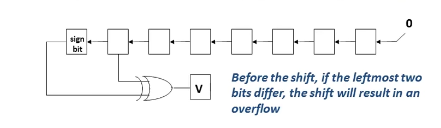
* An arithmetic shift is meant for signed binary numbers
* An arithmetic left shift multiplies a signed number by 2
* An arithmetic right shift divides a signed by number by 2
* Sign bit : 0 for positive and 1 for negative
* The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division

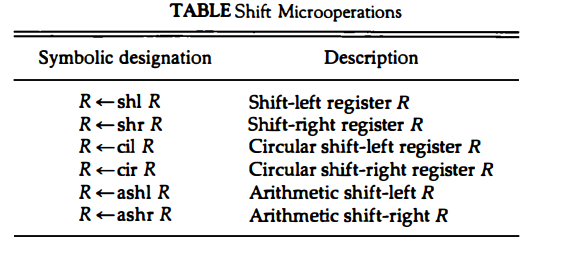
**Right Arithmetic shift**

****

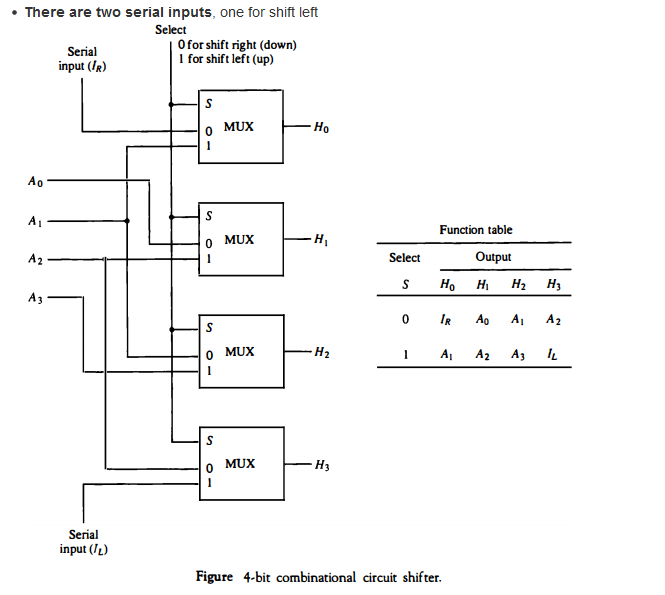
**Left Arithmetic shift**

A left arithmetic shift left must be checked for the overflow

****

****

### Hardware Implementation



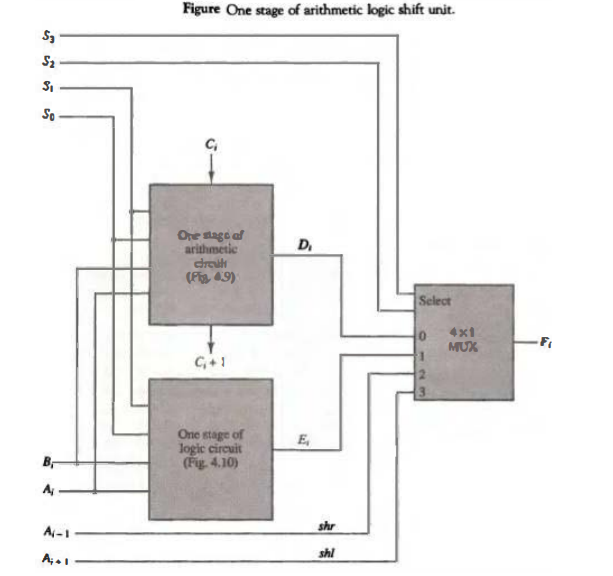
**(IL) and the other for shift right (IL)**. When the selection input S = 0, the input data are shifted right (down in the diagram). When S = 1, the input data are shifted left (up in the diagram). The function table in Fig. above shows which input goes to each output after the shift.

**Arithmetic Logic Shift Unit**

**Instead of having individual registers** performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.

**To perform a microoperation**, the contents of specified registers are placed in the inputs of the common ALU.

**The ALU performs** an operation and the result of the operation is then transferred to a destination register.

****

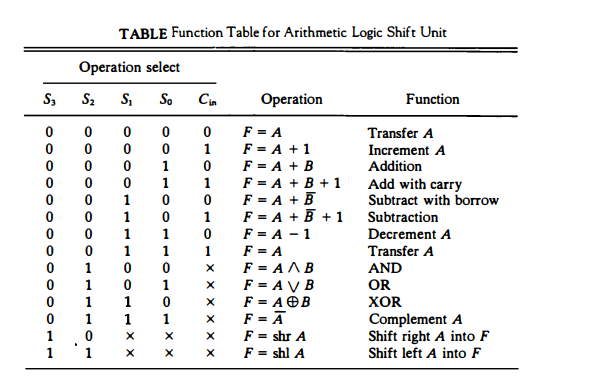
**A particular microoperation** is selected with inputs S1 and S0.

**The data in the multiplexer** are selected with inputs S3 and S2.

**The other two data inputs** to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1 for the shift-left operation. Note that the diagram shows just one typical stage.

**The circuit of Fig. above**  must be repeated n times for an n-bit ALU. The output carry Ci+1 of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence.

**The circuit** whose one stage is specified in Fig. above provides eight arithmetic operation, four logic operations, and two shift operations.

****