

## Unit-1: REGISTER TRANSFER AND MICROOPERATIONS

### CONTENTS:

- ✓ Register Transfer Language
- ✓ Register Transfer
- ✓ Bus And Memory Transfers
- ✓ Types of Micro-operations
- ✓ Arithmetic Micro-operations
- ✓ Logic Micro-operations
- ✓ Shift Micro-operations
- ✓ Arithmetic Logic Shift Unit

### BASIC DEFINITIONS:

- A digital system is an interconnection of digital hardware modules.
- The modules are registers, decoders, arithmetic elements, and control logic.
- The various modules are interconnected with common data and control paths to form a digital computer system.
- Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them.
- The operations executed on data stored in registers are called *microoperations*.
- A *microoperation* is an elementary operation performed on the information stored in one or more registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- Examples of microoperations are shift, count, clear, and load.
- The internal hardware organization of a digital computer is best defined by specifying:
  1. The set of registers it contains and their function.
  2. The sequence of microoperations performed on the binary information stored in the registers.
  3. The control that initiates the sequence of microoperations.

### REGISTER TRANSFER LANGUAGE:

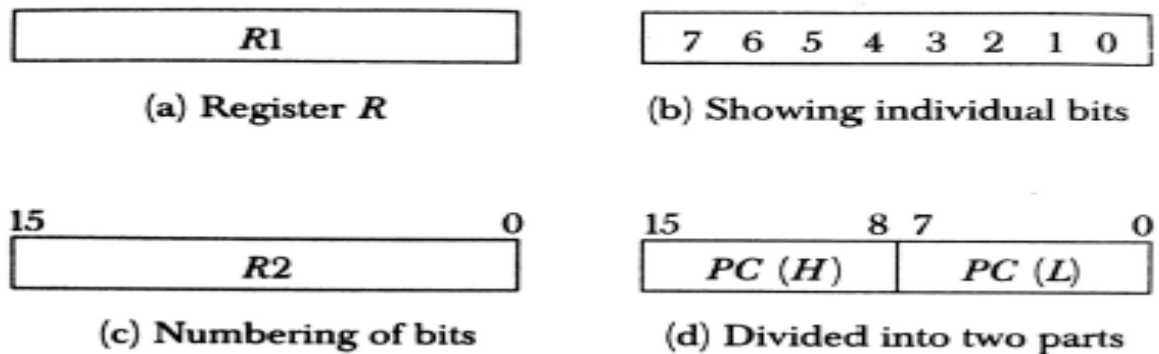
- The symbolic notation used to describe the micro-operation transfer among registers is called RTL (Register Transfer Language).
- The use of **symbols** instead of a **narrative explanation** provides an organized and concise manner for listing the micro-operation sequences in registers and the control functions that initiate them.

- 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.

### Registers:

- Computer registers are designated by upper case letters (and optionally followed by digits or letters) to denote the function of the register.
- For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name **MAR**.
- Other designations for registers are **PC** (for program counter), **IR** (for instruction register, and **R1** (for processor register).
- The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.
- Figure 4-1 shows the representation of registers in block diagram form.

**Figure 4-1** Block diagram of register.



- The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 4-1(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).
- 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.

### Register Transfer:

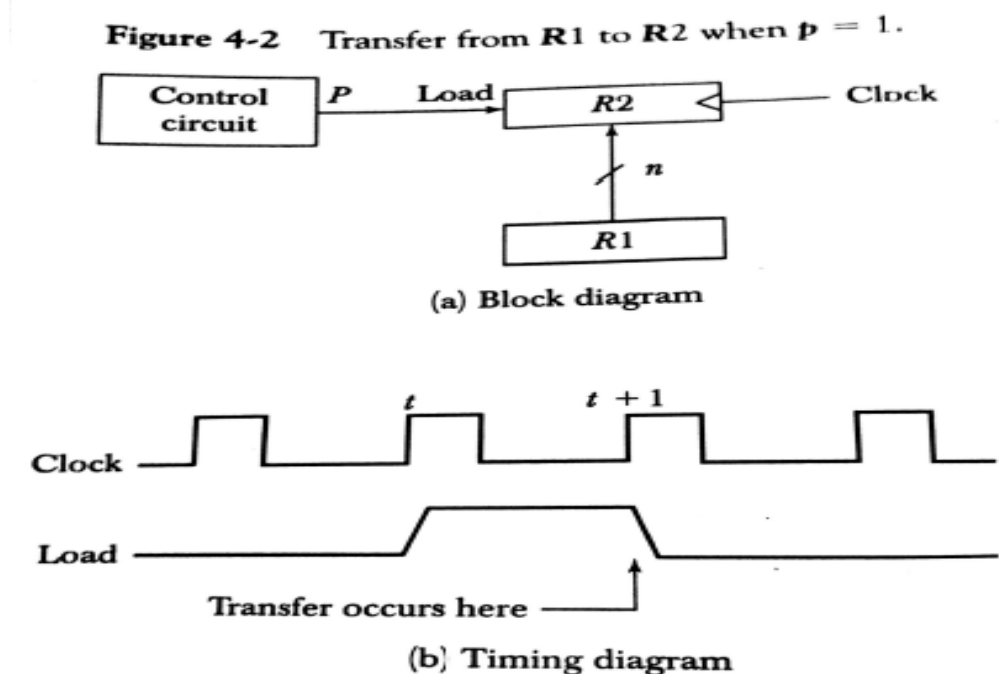
- Information transfer from one register to another is designated in symbolic form by means of a *replacement operator*.
- The statement **R2 ← R1** denotes a transfer of the content of register R1 into register R2.
- It designates a replacement of the content of R2 by the content of R1.
- By definition, the content of the source register R 1 does not change after the transfer.
- If we want the transfer to occur only under a predetermined control condition then it can be shown by an if-then statement.

**if (P=1) then R2 ← R1**

- P is the control signal generated by a control section.
- We can separate the control variables from the register transfer operation by specifying a **Control Function**.
- Control function is a Boolean variable that is equal to 0 or 1.
- control function is included in the statement as

**P:  $R2 \leftarrow R1$**

- Control condition is terminated by a colon implies transfer operation be executed by the hardware only if  $P=1$ .
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
- Figure 4-2 shows the block diagram that depicts the transfer from R1 to R2.



- The  $n$  outputs of register  $R1$  are connected to the  $n$  inputs of register  $R2$ .
- The letter  $n$  will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
- Register  $R2$  has a load input that is activated by the control variable  $P$ .
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
- As shown in the timing diagram,  $P$  is activated in the control section by the rising edge of a clock pulse at time  $t$ .
- The next positive transition of the clock at time  $t + 1$  finds the load input active and the data inputs of  $R2$  are then loaded into the register in parallel.

- P may go back to 0 at time  $t+1$ ; otherwise, the transfer will occur with every clock pulse transition while P remains active.
- Even though the control condition such as P becomes active just after time  $t$ , the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time  $t+1$ .
- The basic symbols of the register transfer notation are listed in below table

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1, R1 \leftarrow R2$

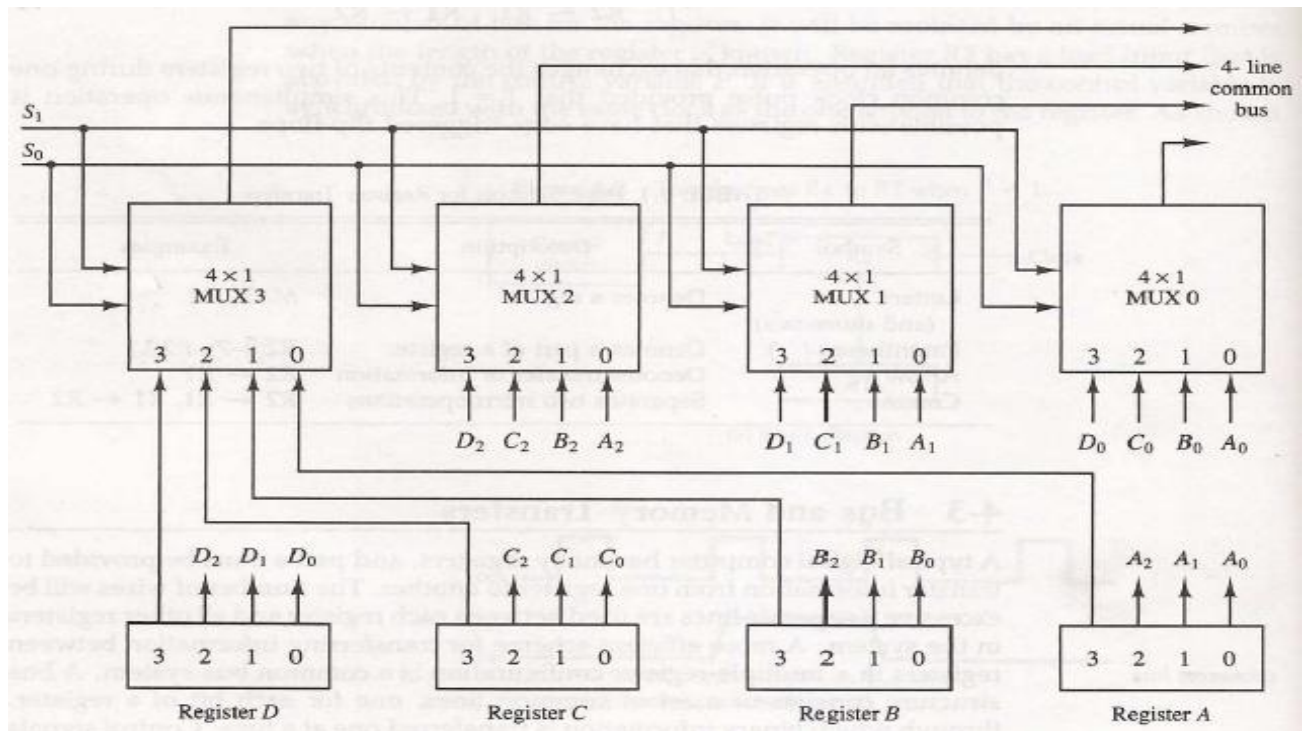
- A comma is used to separate two or more operations that are executed at the same time.
- The statement  
 $T : R2 \leftarrow R1, R1 \leftarrow R2$  (exchange operation)  
denotes an operation that exchanges the contents of two registers during one common clock pulse provided that  $T=1$ .

### **Bus and Memory Transfers:**

- A more efficient scheme for transferring information between registers in **a multiple-register configuration** is a **Common Bus System**.
- A common bus consists of a set of common lines, one for each bit of a register.
- Control signals determine which register is selected by the bus during each particular register transfer.
- Different ways of constructing a Common Bus System
  - ✓ Using Multiplexers
  - ✓ Using Tri-state Buffers

### **Common bus system is with multiplexers:**

- The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in below Figure.



- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs,  $S_1$  and  $S_0$ .
- For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labelled  $A_1$ .
- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.
- Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.
- The two selection lines  $S_1$  and  $S_0$  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  $S_1S_0 = 00$ , the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
- This 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, register B is selected if  $S_1S_0 = 01$ , and so on.
- Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

$S_1$	$S_0$	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

- In general a bus system has
  - ✓ multiplex "k" Registers

- ✓ each register of “n” bits
  - ✓ to produce “n-line bus”
  - ✓ no. of multiplexers required = n
  - ✓ size of each multiplexer = k x 1
- When the bus is included in the statement, the register transfer is symbolized as follows:
- BUS ← C, R1 ← BUS**
- The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

**R1 ← C**

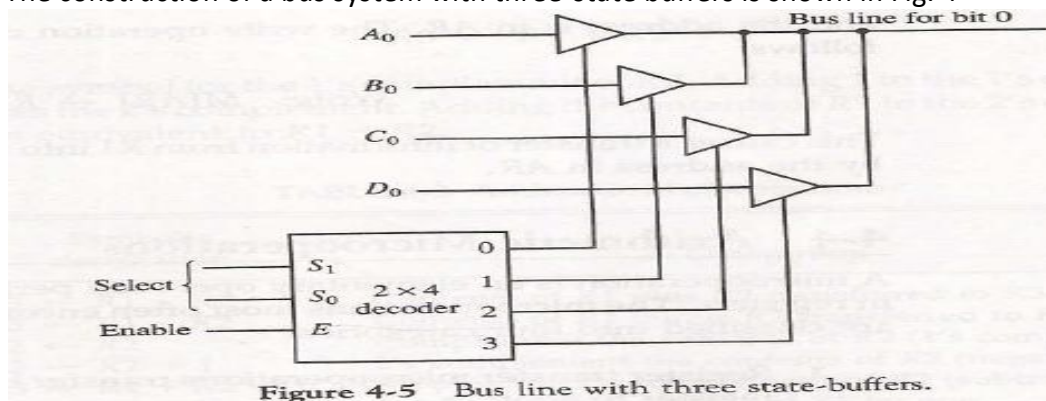
### Three-State Bus Buffers:

- A bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is a *high-impedance state*.
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.
- The graphic symbol of a three-state buffer gate is shown in Fig. 4-4.

**Figure 4-4** Graphic symbols for three-state buffer.



- It is distinguished from a normal buffer by having both a normal input and a control input.
- The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.
- When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.
- The construction of a bus system with three-state buffers is shown in Fig. 4



- The outputs of four buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.
- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

### Memory Transfer:

- The transfer of information from a memory word to the outside environment is called a *read* operation.
- The transfer of new information to be stored into the memory is called a *write* operation.
- A memory word will be symbolized by the letter M.
- The particular memory word among the many available is selected by the memory address during the transfer.
- It is necessary to specify the address of M when writing memory transfer operations.
- This will be done by enclosing the address in square brackets following the letter M.
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data are transferred to another register, called the data register, symbolized by DR.
- The read operation can be stated as follows:

**Read: DR ← M [AR]**

- This causes a transfer of information into DR from the memory word M selected by the address in AR.
- The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.
- The write operation can be stated as follows:

**Write: M [AR] ← R1**

### Types of Micro-operations:

- Register Transfer Micro-operations: Transfer binary information from one register to another.
- Arithmetic Micro-operations: Perform arithmetic operation on numeric data stored in registers.
- Logical Micro-operations: Perform bit manipulation operations on data stored in registers.
- Shift Micro-operations: Perform shift operations on data stored in registers.
- Register Transfer Micro-operation doesn't change the information content when the binary information moves from source register to destination register.



- There are two 4-bit inputs  $A$  and  $B$  and a 4-bit output  $D$ .
- The four inputs from  $A$  go directly to the  $X$  inputs of the binary adder.
- Each of the four inputs from  $B$  are connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of  $B$ .
- The other two data inputs are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs  $S_1$  and  $S_0$ . The input carry  $C_{in}$ , goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- By controlling the value of  $Y$  with the two selection inputs  $S_1$  and  $S_0$  and making  $C_{in}$  equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in Table 44.

**TABLE 4-4 Arithmetic Circuit Function Table**

Select			Input $Y$	Output $D = A + Y + C_{in}$	Microoperation
$S_1$	$S_0$	$C_{in}$			
0	0	0	$B$	$D = A + B$	Add
0	0	1	$B$	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	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 $A$

#### **Addition:**

- When  $S_1S_0 = 00$ , the value of  $B$  is applied to the  $Y$  inputs of the adder.
  - ✓ If  $C_{in} = 0$ , the output  $D = A + B$ .
  - ✓ If  $C_{in} = 1$ , output  $D = A + B + 1$ .
- Both cases perform the add microoperation with or without adding the input carry.

#### **Subtraction:**

- When  $S_1S_0 = 01$ , the complement of  $B$  is applied to the  $Y$  inputs of the adder.
  - ✓ If  $C_{in} = 1$ , then  $D = A + \bar{B} + 1$ . This produces  $A$  plus the 2's complement of  $B$ , which is equivalent to a subtraction of  $A - B$ .
  - ✓ When  $C_{in} = 0$  then  $D = A + \bar{B}$ . This is equivalent to a subtract with borrow, that is,  $A - B - 1$ .

#### **Increment:**

- When  $S_1S_0 = 10$ , the inputs 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.



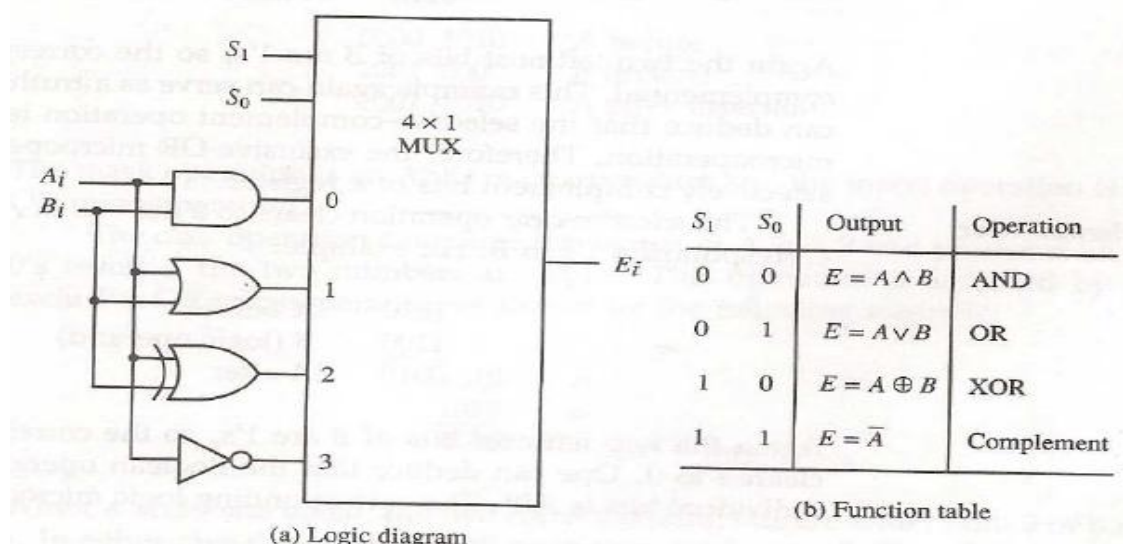
TABLE 4-6 Sixteen Logic Microoperations

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

#### Hardware Implementation:

- The hardware implementation of logic microoperations 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 microoperations, most computers use only four--AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.
- Figure 4-10 shows one stage of a circuit that generates the four basic logic microoperations.
- It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic.
- The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs  $S_1$  and  $S_0$  choose one of the data inputs of the multiplexer and direct its value to the output.

Figure 4-10 One stage of logic circuit.



- ✓ 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:

0110 1010	A before
0000 1111	B (mask)
0000 1010	A after masking

and then insert the new value:

0000 1010	A before
1001 0000	B (insert)
1001 1010	A after insertion

- ✓ The mask operation is an AND microoperation and the insert operation is an OR microoperation.

#### ➤ 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

1010	A
1010	B
0000	$A \leftarrow A \oplus B$

### Shift Microoperations:

- Shift microoperations are used for serial transfer of data.
- The contents of a register can be shifted to the left or the right.
- 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.
- The symbolic notation for the shift microoperations is shown in Table 4-7.

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow shl R$	Shift-left register <i>R</i>
$R \leftarrow shr R$	Shift-right register <i>R</i>
$R \leftarrow cil R$	Circular shift-left register <i>R</i>
$R \leftarrow cir R$	Circular shift-right register <i>R</i>
$R \leftarrow ashl R$	Arithmetic shift-left <i>R</i>
$R \leftarrow ashr R$	Arithmetic shift-right <i>R</i>

#### ➤ **Logical Shift:**

- A *logical* shift is one that transfers 0 through the serial input.
- The symbols *shl* and *shr* for logical shift-left and shift-right microoperations.