

## Module1–Register Transfer and Microoperations:

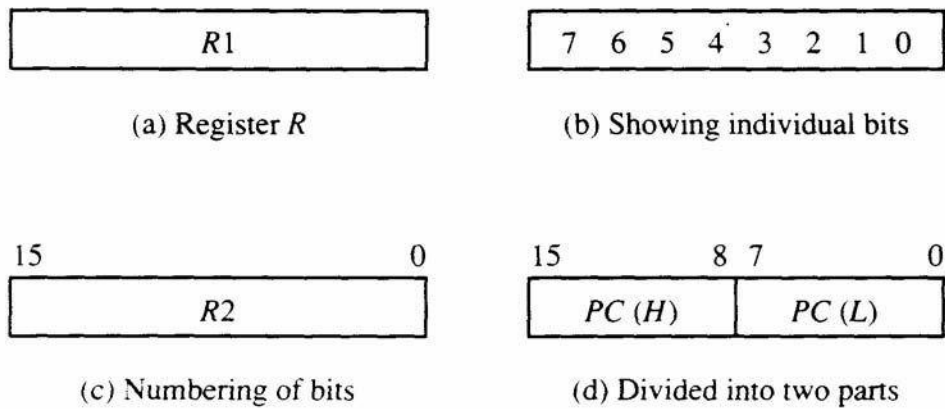
### Register Transfer Language

- Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements, and control logic
- The modules are interconnected with common data and control paths to form a digital computer system.
- 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.
- Examples are shift, count, clear, and load.
- Some of the digital components from before are registers that implement microoperations.
- The internal hardware organization of a digital computer is best defined by specifying
  - The set of registers it contains and their functions.
  - The sequence of microoperations performed on the binary information stored.
  - The control that initiates the sequence of microoperations.
- Use symbols, rather than words, to specify the sequence of microoperations.
- The symbolic notation used is called a *register transfer language*.
- A programming language is a procedure for writing symbols to specify a given computational process.
- Define symbols for various types of microoperations and describe associated hardware that can implement the microoperations.

### Register Transfer

- Designate computer registers by capital letters to denote its function.
- The register that holds an address for the memory unit is called MAR.
- The program counter register is called PC.
- IR is the instruction register and R1 is a processor register.
- The individual flip-flops in an  $n$ -bit register are numbered in sequence from 0 to  $n-1$
- Refer to Figure4.1 for the different representations of a register.

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



- Designate information transfer from one register to another by  $R2 \leftarrow R1$ .
- This statement implies that the hardware is available
  - The outputs of the source must have a path to the inputs of the destination.
  - The destination register has a parallel load capability.
- If the transfer is to occur only under a predetermined control condition, designate it by
 

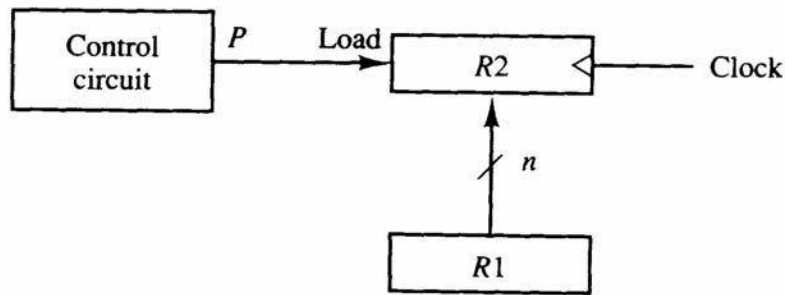
$If(P = 1) \text{ then } (R2 \leftarrow R1)$

or,

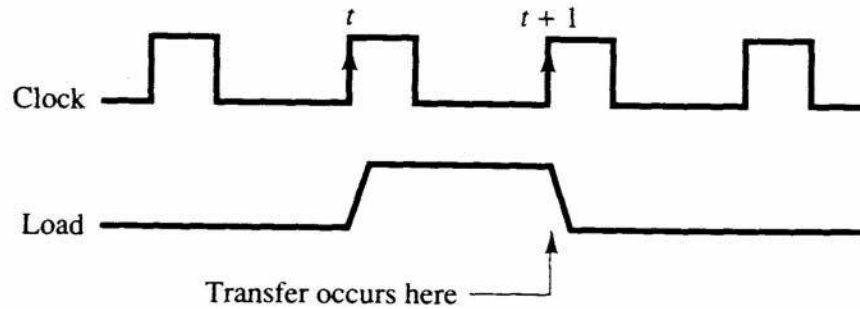
$P: R2 \leftarrow R1,$

Where  $P$  is a control function that can be either 0 or 1
- Every statement written in register transfer notation implies the presence of the required hardware construction.

**Figure 4-2** Transfer from R1 to R2 when  $P = 1$ .



(a) Block diagram



(b) Timing diagram

- It is assumed that all transfers occur during a clock edge transition.
- All microoperations written on a single line are to be executed at the same time  
 $T: R2 \leftarrow R1, R1 \leftarrow R2$

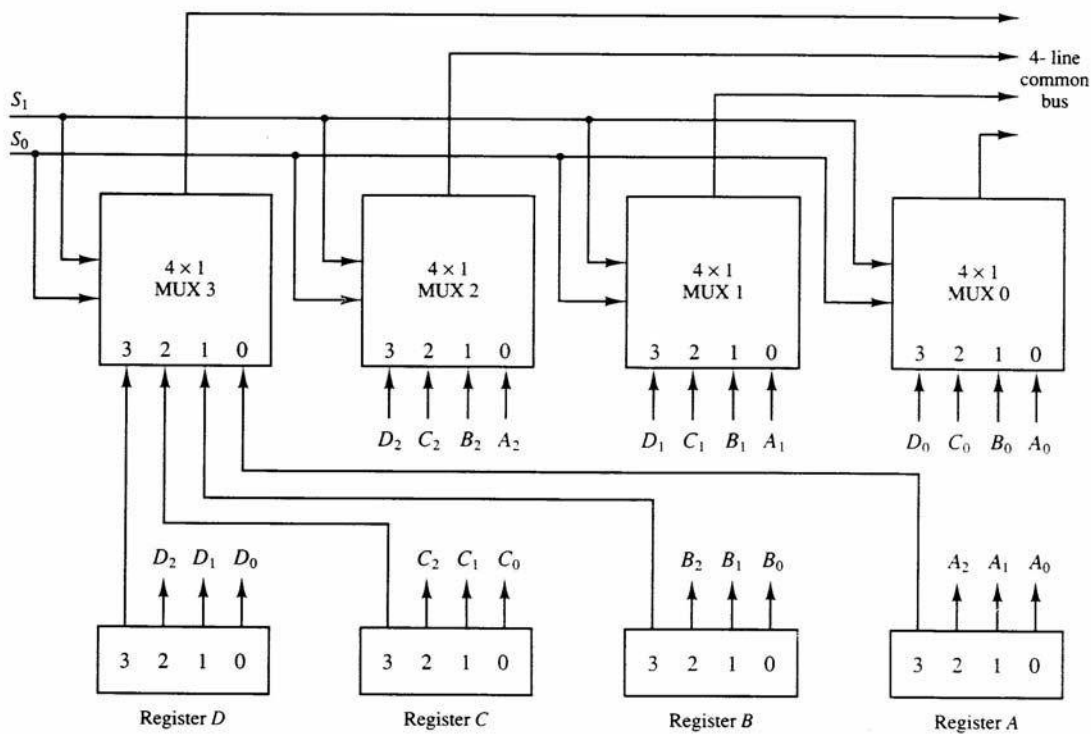
**TABLE 4-1** Basic Symbols for Register Transfers

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	$R2 \leftarrow R1, R1 \leftarrow R2$

## Bus and Memory Transfers

- Rather than connecting wires between all registers, a common bus is used.
- A bus structure 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 transfer.
- Multiplexers can be used to construct a common bus.
- Multiplexers select the source register whose binary information is then placed on the bus.
- The select lines are connected to the selection inputs of the multiplexers and choose the bits of one register.

Figure 4-3 Bus system for four registers.

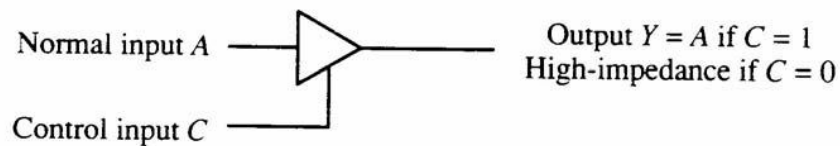


- In general, a bus system will multiplex  $k$  registers of  $n$  bits each to produce an  $n$ -line common bus.
- This requires  $n$  multiplexers— one for each bit.
- The size of each multiplexer must be  $k \times 1$ .
- The number of select lines required is  $\log k$ .
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated.
- Rather than listing each step as

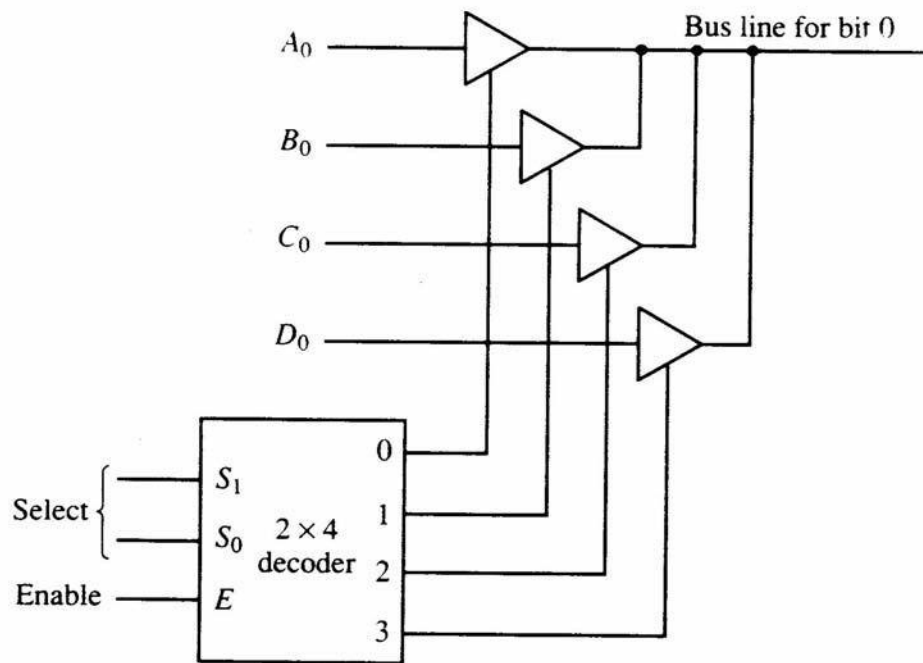
BUS  $\leftarrow$  C, R1  $\leftarrow$  BUS,  
use R1  $\leftarrow$  C, since the bus is implied.

- Instead of using multiplexers, *three-state gates* can be used to construct the bus system.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0.
- The third state is a *high-impedance* state this behaves like an open circuit, which means the output is disconnected and does not have logic significance.

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



- The three-state buffer gate has a normal input and a control input which determines the output state.
- With control 1, the output equals the normal input.
- With control 0, the gate goes to a high-impedance state.
- This enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loading effects.

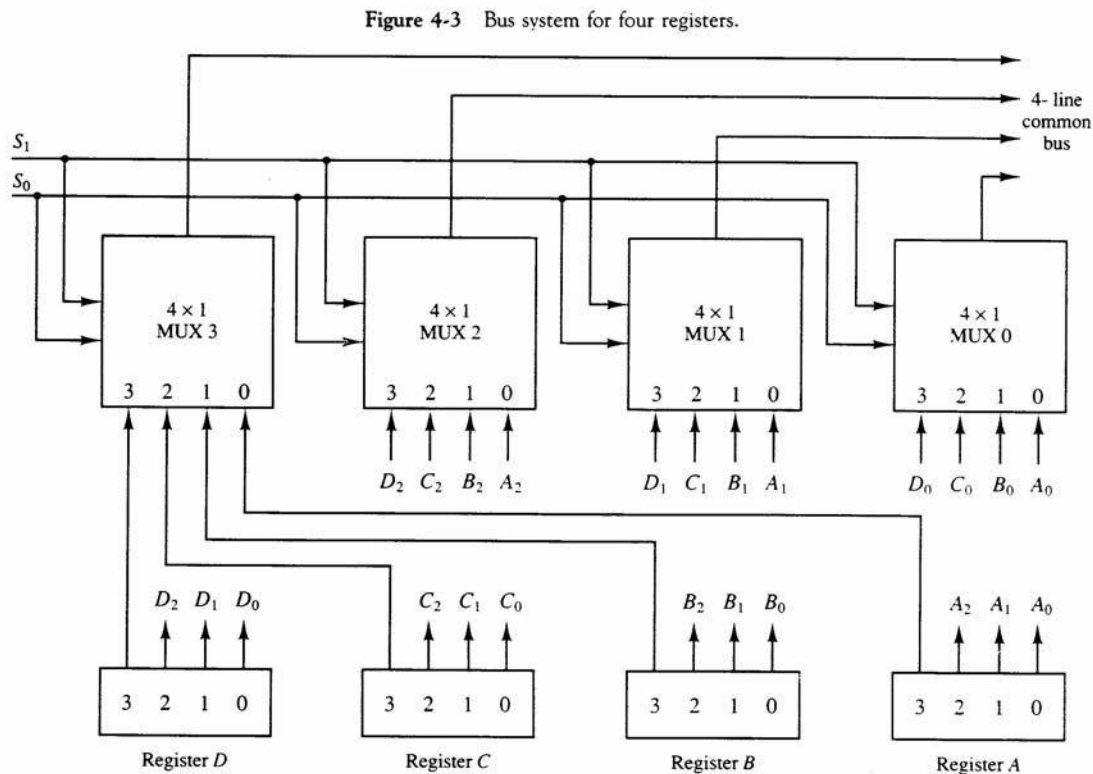


**Figure 4-5** Bus line with three state-buffers.

- Decoders are used to ensure that no more than one control input is active at any given time.
- This circuit can replace the multiplexer in Figure 4.3.
- To construct a common bus for four registers of  $n$  bits each using three-state buffers, we need  $n$  circuits with four buffers in each.
- Only one decoder is necessary to select between the four registers.
- Designate a memory word by the letter  $M$ .
- It is necessary to specify the address of  $M$  when writing memory transfer operations.
- Designate the address register by  $AR$  and the data register by  $DR$ .
- The read operation can be stated as:  
Read:  $DR \leftarrow M[AR]$
- The write operation can be stated as:  
Write:  $M[AR] \leftarrow R1$

## Arithmetic Microoperations

- There are four categories of the most common microoperations:
  - Register transfer: transfer binary information from one register to another.
  - Arithmetic: perform arithmetic operations on numeric data stored in registers.
  - Logic: perform bit manipulation operations on non-numeric data stored in registers.
  - Shift: perform shift operations on data stored in registers.
- The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift.
- Example of addition:  $R3 \leftarrow R1 + R2$ .
- Subtraction is most often implemented through complementation and addition
- Example of subtraction:  $R3 \leftarrow R1 + \bar{R2} + 1$  (strike through denotes bar on top -1's complement of R2)
- Adding 1 to the 1's complement produces the 2's complement.
- Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting.



- Multiply and divide are not included as microoperations.
- A microoperation is one that can be executed by one clock pulse.

- Multiply (divide) is implemented by a sequence of add and shift microoperations (subtract and shift).
- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the addition.
- A full – adder adds two bits and a previous carry.



- A *binary adder* is a digital circuit that generates the arithmetic sum of two binary numbers of any length.
- A binary adder is constructed with full-adder circuits connected in cascade.
- An  $n$ -bit binary adder requires  $n$  full-adders.

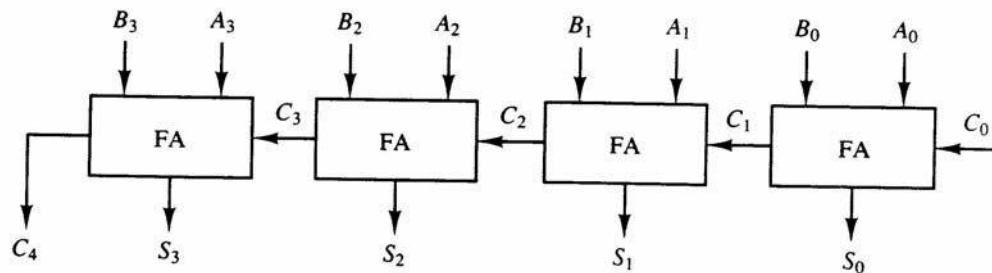


Figure 4-6 4-bit binary adder.

- The subtraction  $A-B$  can be carried out by the following steps
  - Take the 1's complement of B (invert each bit).
  - Get the 2's complement by adding 1.
  - Add the result to A.
- The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full-adder.

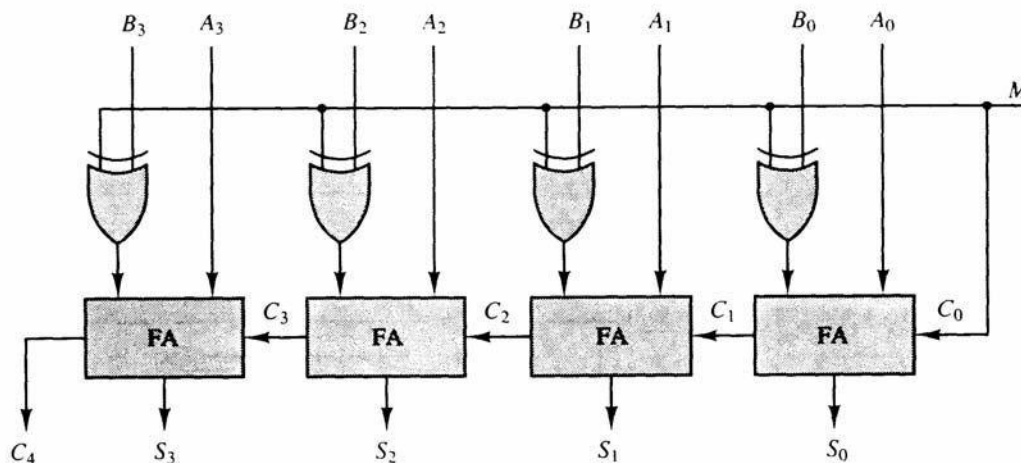
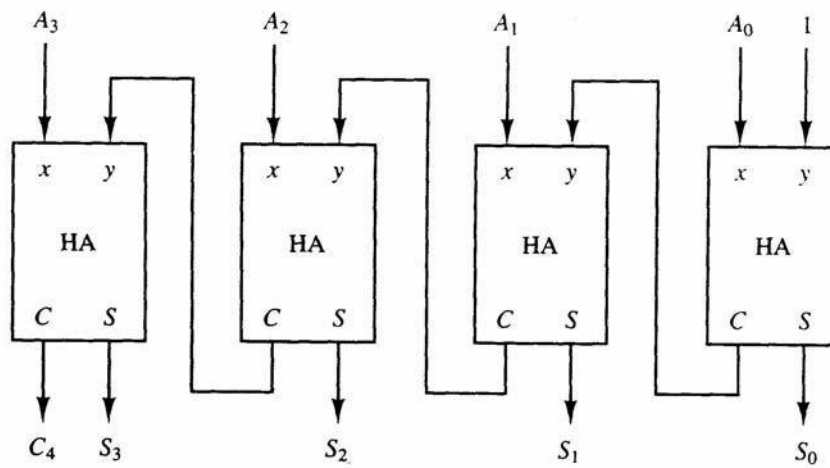


Figure 4-7 4-bit adder-subtractor.

- The increment microoperation adds one to a number in a register.
- This can be implemented by using a binary counter—every time the count enable is active, the count is incremented by one.
- If the increment is to be performed independent of a particular register, then use half-adders connected in cascade.

- An  $n$ -bit binary incrementer requires  $n$  half-adders.



**Figure 4-8** 4-bit binary incrementer.

- Each of the arithmetic microoperations can be implemented in one composite arithmetic circuit.
- The basic component is the parallel adder
- Multiplexers are used to choose between the different operations.
- The output of the binary adder is calculated from the following sum:  

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

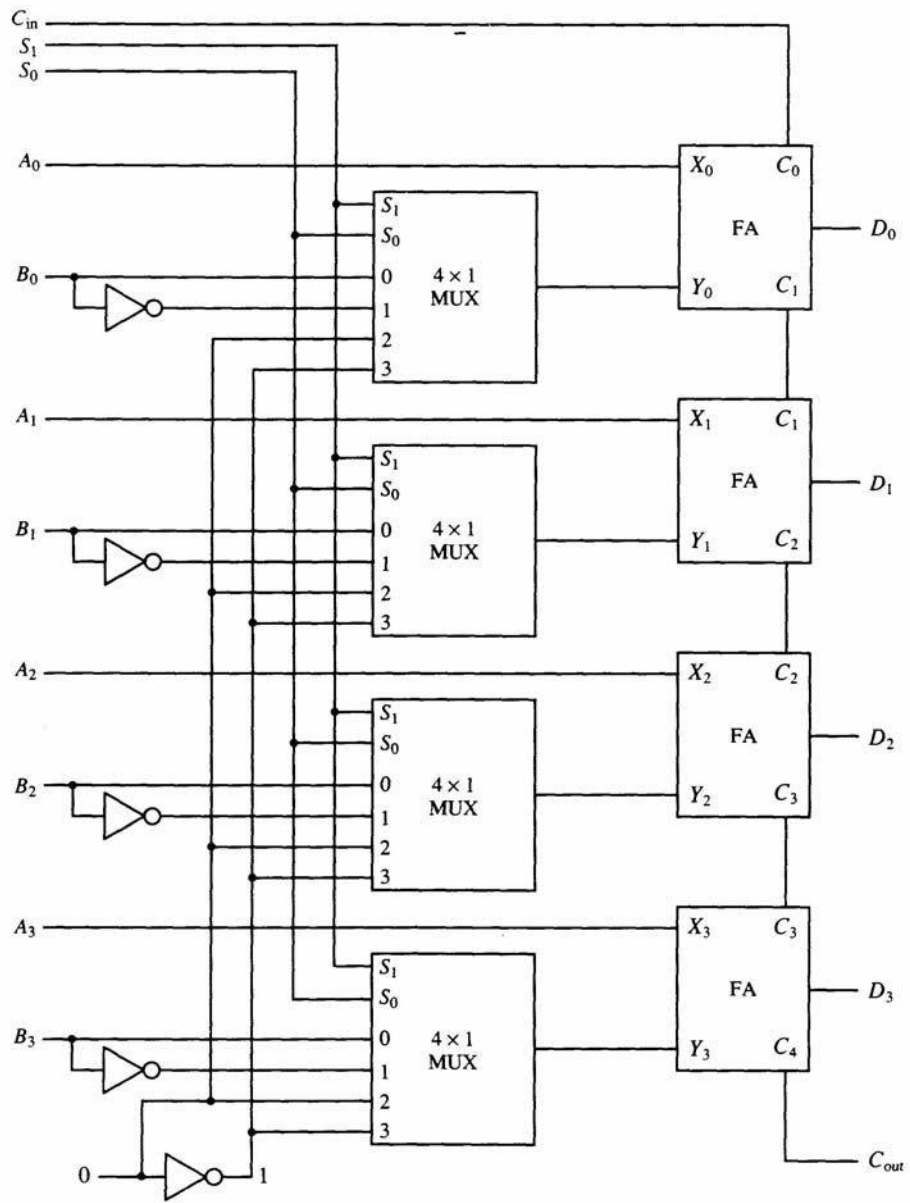


Figure 4-9 4-bit arithmetic circuit.

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	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	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 Microoperations

- Logic operations specify binary operations for strings of bits stored in registers and treat each bit separately.
- Example: the XOR of R1 and R2 is symbolized by  

$$P: R1 \leftarrow R1 \oplus R2$$
- Example: R1=1010 and R2=1100  

$$\begin{array}{ll} 1010 & \text{Content of R1} \\ 1100 & \text{Content of R2} \\ \hline 0110 & \text{Content of R1 after } P=1 \end{array}$$
- Symbols used for logical microoperations:
  - OR:  $\vee$
  - AND:  $\wedge$
  - XOR:  $\oplus$
- The + sign has two different meanings: logical OR and summation.
- When + is in a microoperation, then summation
- When + is in a control function, then OR
- Example:  

$$P+Q: R1 \leftarrow R2+R3, R4 \leftarrow R5 \vee R6$$
- There are 16 different logic operations that can be performed with two binary variables.

TABLE 4-5 Truth Tables for 16 Functions of Two Variables

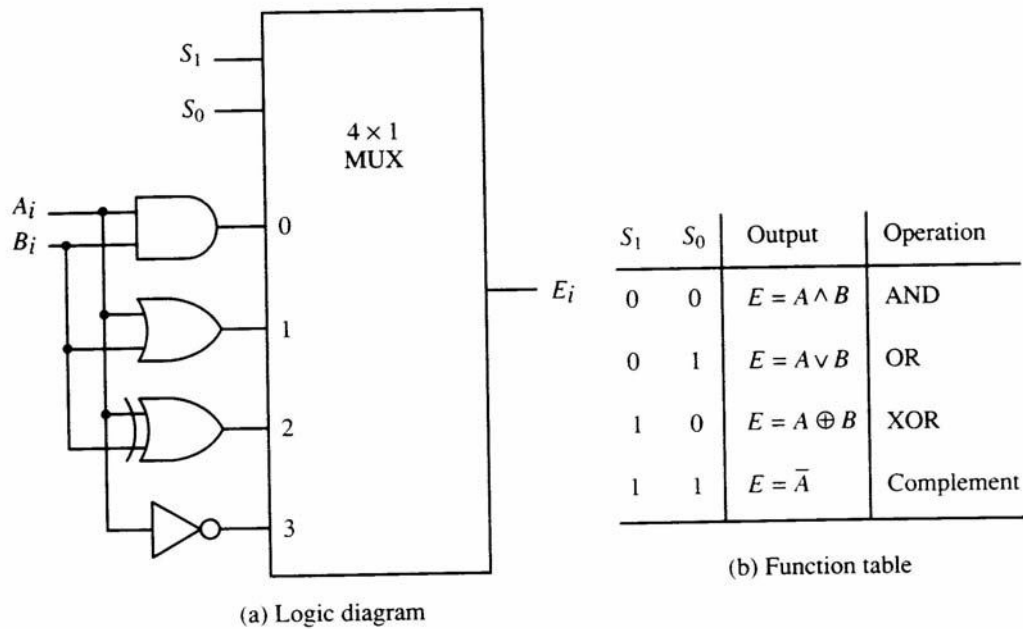
$x$	$y$	$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$	$F_{11}$	$F_{12}$	$F_{13}$	$F_{14}$	$F_{15}$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer $A$
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	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
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement $B$
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement $A$
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow \text{all 1's}$	Set to all 1's

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers.
- All 16 microoperations can be derived from using four logic gates.

Figure 4-10 One stage of logic circuit.



- Logic microoperations can be used to change bit values, delete a group of bits, or insert new bit values into a register.
- The *selective-set* operation sets to 1 the bits in A where there are corresponding 1's in B

1010 A before  
1100 B (logic operand)  
 1110 A after

$$A \leftarrow A \vee B$$

- The *selective-complement* operation complements bits in A where there are corresponding 1's in B

1010 A before  
1100 B (logic operand)  
 0110 A after

$$A \leftarrow A \oplus B$$

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

1010 A before  
1100 B (logic operand)  
 0010 A after

$$A \leftarrow A \wedge \bar{B}$$

- 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

1010	A before
<u>1100</u> B(logicoperand)100	
0	A after

$A \leftarrow A \wedge B$

- The *insert* operation inserts a new value into a group of bits
- This is done by first masking the bits to be replaced and then Oring them with the bits to be inserted

01101010	A before
<u>0000 1111</u>	B(mask)
0000 1010	A after masking

00001010	A before
<u>1001 0000</u>	B (insert)
1001 1010	A after insertion

- The *clear* operation compares the bits in A and B and produces an all 0's result if the two numbers are equal

1010	A
<u>1010</u>	B
0000	$A \leftarrow A \oplus B$

## Section 4.6–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
- There are three types of shifts: logical, circular, and arithmetic
- A *logical shift* is one that transfers 0 through the serial input
- The symbols *shl* and *shr* are for logical shift-left and shift-right by one position

$R1 \leftarrow \text{shl} R1$

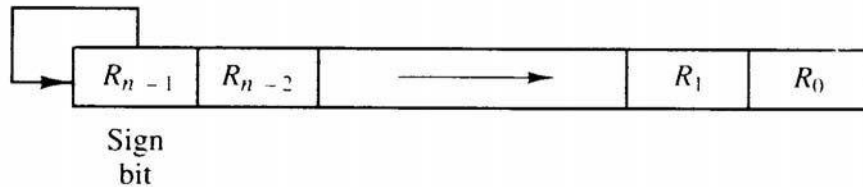
- The *circular shift* circulates the bits of the register around the two ends without loss of information
- The symbols *cil* and *cir* are for circular shift left and right

**TABLE 4-7 Shift Microoperations**

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register $R$
$R \leftarrow \text{shr } R$	Shift-right register $R$
$R \leftarrow \text{cil } R$	Circular shift-left register $R$
$R \leftarrow \text{cir } R$	Circular shift-right register $R$
$R \leftarrow \text{ashl } R$	Arithmetic shift-left $R$
$R \leftarrow \text{ashr } R$	Arithmetic shift-right $R$

- The *arithmetic shift* shifts assigned binary number to the left or right
- To the left is multiplying by 2, to the right is dividing by 2
- Arithmetic shifts must leave the sign bit unchanged
- A sign reversal occurs if the bit in  $R_{n-1}$  changes in value after the shift
- This happens if the multiplication causes an overflow
- An over flow flip-flop  $V_s$  can be used to detect the overflow

$$V_s = R_{n-1} \oplus R_{n-2}$$



**Figure 4-11** Arithmetic shift right.

- A bi-directional shift unit with parallel load could be used to implement this
- Two clock pulses are necessary with this configuration: one to load the value and another to shift
- In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit
- The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into the register
- This can be constructed with multiplexers



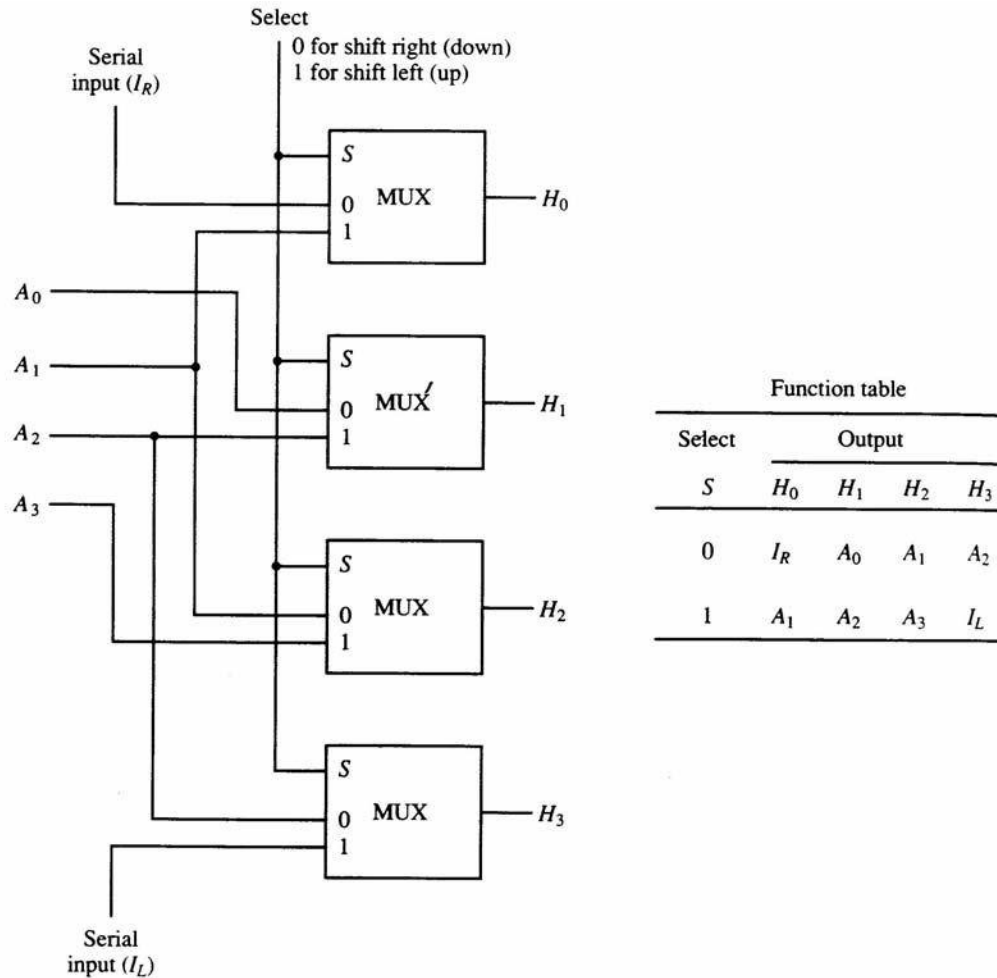


Figure 4-12 4-bit combinational circuit shifter.

### Arithmetic Logic Shift Unit

- The *arithmetic logic unit (ALU)* is a common operational unit connected to a number of storage registers.
- To perform a microoperation, the contents of specified registers are placed in the inputs of the ALU.
- The ALU performs an operation and the result is then transferred to a destination register.
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.

Figure 4-13 One stage of arithmetic logic shift unit.

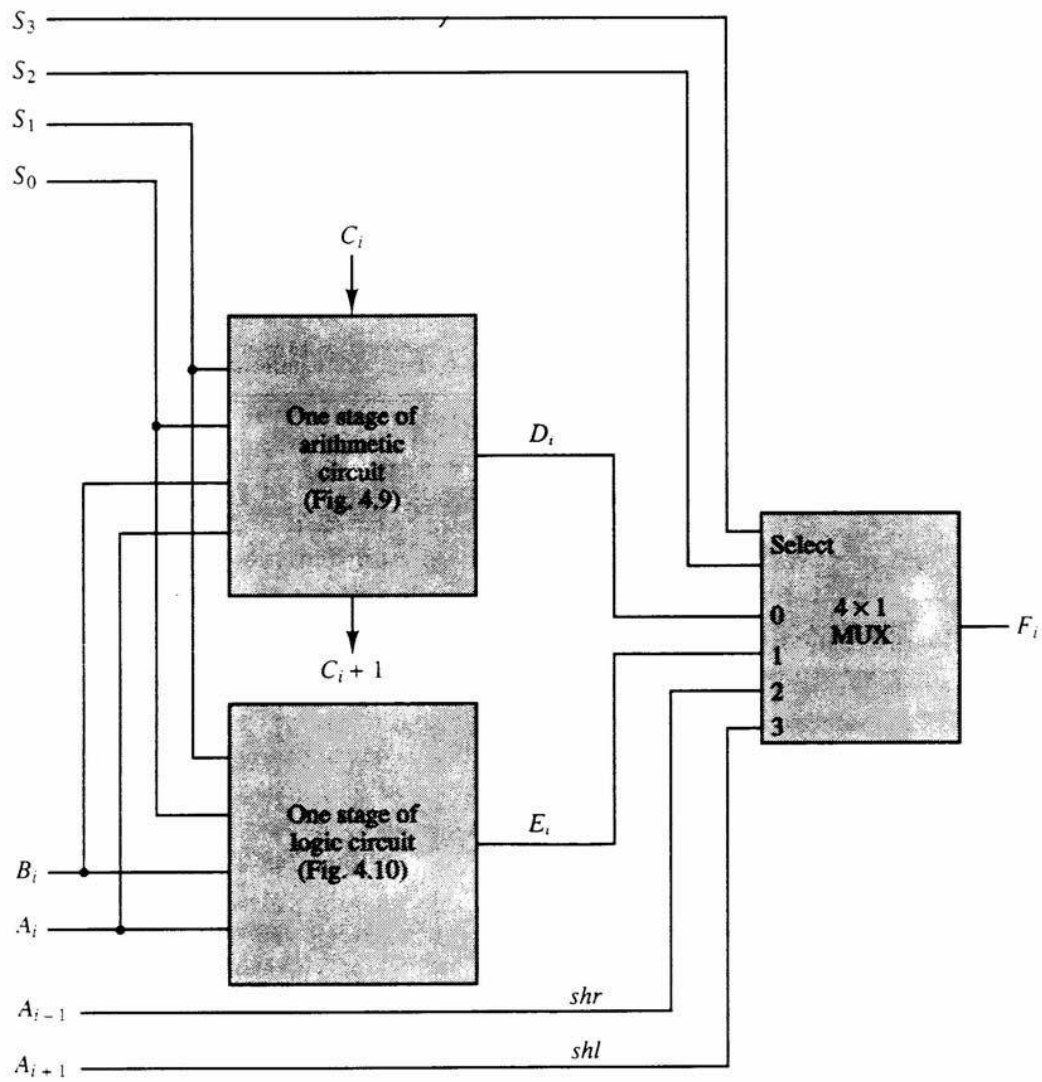


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select					Operation	Function
$S_3$	$S_2$	$S_1$	$S_0$	$C_{in}$		
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	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement $A$
0	0	1	1	1	$F = A$	Transfer $A$
0	1	0	0	$\times$	$F = A \wedge B$	AND
0	1	0	1	$\times$	$F = A \vee B$	OR
0	1	1	0	$\times$	$F = A \oplus B$	XOR
0	1	1	1	$\times$	$F = \overline{A}$	Complement $A$
1	0	$\times$	$\times$	$\times$	$F = \text{shr } A$	Shift right $A$ into $F$
1	1	$\times$	$\times$	$\times$	$F = \text{shl } A$	Shift left $A$ into $F$

