UNIT-III

REGISTER TRANSFER LANGUAGE AND DESIGN OF CONTROL UNIT

Register Transfer:

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Micro operations
- Logic Micro operations
- Shift Micro operations

Control Unit:

- Control Memory
- Address Sequencing
- Micro program Example
- Design of Control Unit.

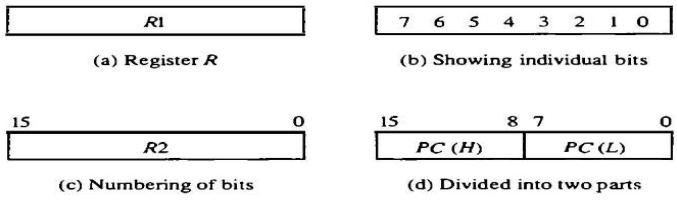
REGISTER TRANSFER LANGUAGE

- ✓ 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 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
- ✓ 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

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

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) *then* (R2 ← R1)

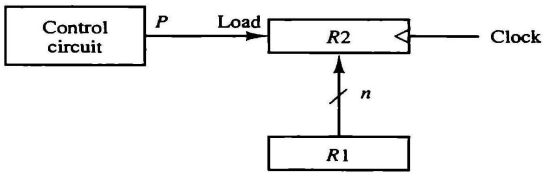
or,

P: R2 ← R1,

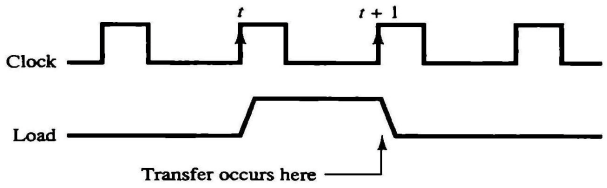
where P is a control function that can be either 0 or 1

- ✓ 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 ← R1, R1 ← R2

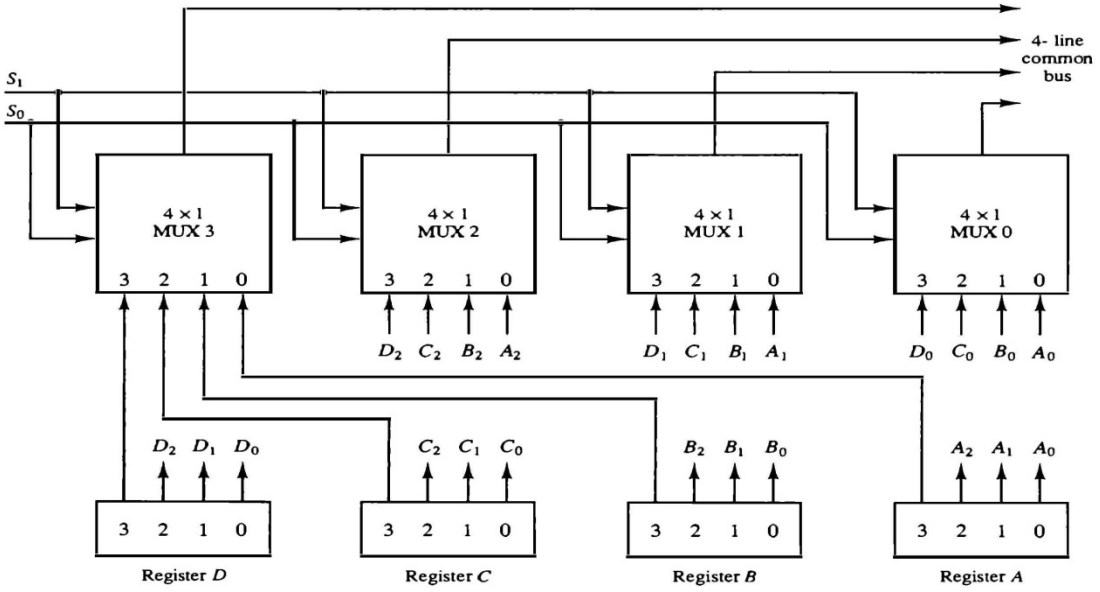
- ✓ The basic symbols of the register transfer notation are

Symbol	Description	Examples
Letters (and 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*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *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.



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

- ✓ 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 x 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

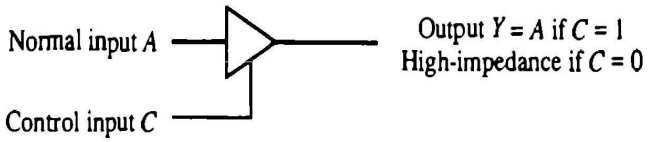
  BUS ← C, R1 ← BUS,

  use R1 ← C, since the bus is implied

Three-State Bus Buffers

- ✓ 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 a 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
- ✓ Decoders are used to ensure that no more than one control input is active at any given time
- ✓ 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

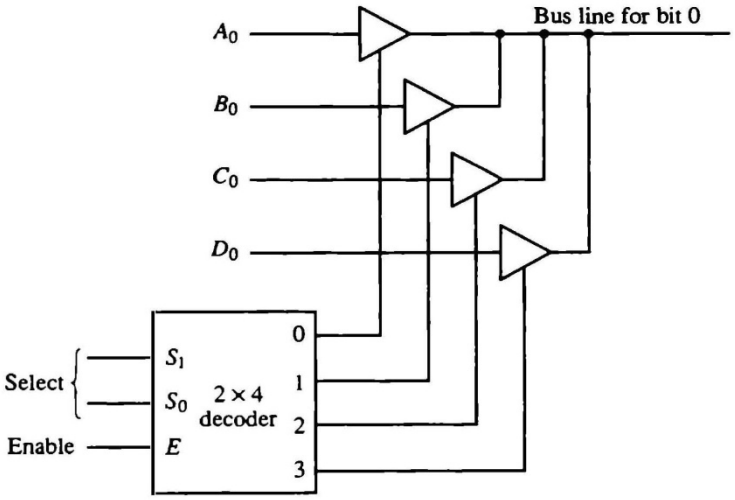


Figure 4-5 Bus line with three state-buffers.

Memory Transfer

- ✓ 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 ← M [AR]

- ✓ The write operation can be stated as:

  Write: M [AR] ← R1

ARITHMETIC MICROOPERATIONS

- ✓ A micro operation is an elementary operation performed with the data stored in registers. 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 register transfer microoperation does not change the information content when the binary information moves from the source register to the destination register. The other three types of microoperation change the information content during the transfer.
- ✓ The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift
- ✓ Example of addition: R3 ← R1 +R2
- ✓ Subtraction is most often implemented through complementation and addition
- ✓ Example of subtraction: R3 ← R1 +$\overline{R2}$+ 1 ($\overline{R2}$ is the symbol for the 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
- ✓ The increment and decrement microoperations are symbolized by plus-one and minus-one operations, respectively. These microoperations are implemented with a combinational circuit or with a binary up-down counter.

TABLE 4-3 Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one

- ✓ 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)

Binary Adder

- ✓ 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 added is constructed with full-adder circuits connected in cascade
- ✓ An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.

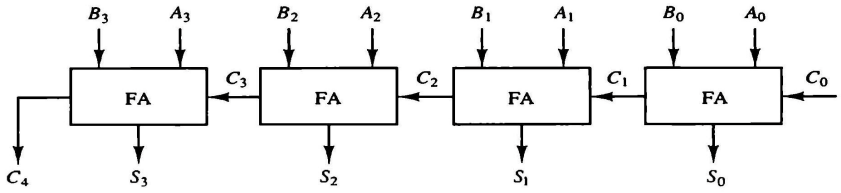


Figure 4-6 4-bit binary adder.

Binary Adder-Subtractor

- ✓ The subtraction of binary numbers can be done most conveniently by means of complements.
- ✓ The subtraction A –B can be done by taking the 2's complement of B and adding it to A.
- ✓ The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.
- ✓ The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full-adder

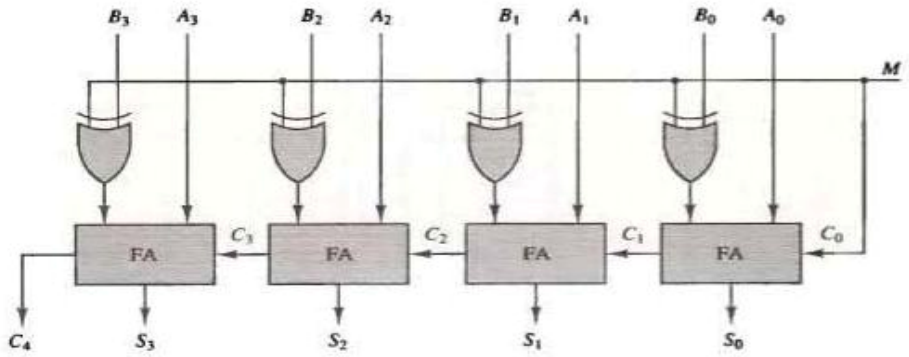


Figure 4-7 4-bit adder-subtractor.

- ✓ A 4-bit adder-subtract or circuit is shown in Fig. The mode input M controls the operation. When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.

Binary Incrementer

- ✓ 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

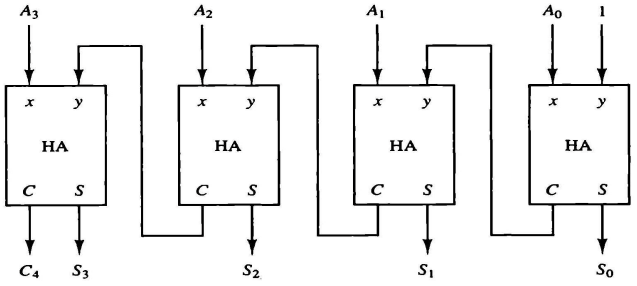


Figure 4-8 4-bit binary incrementer.

Arithmetic Circuit

- ✓ Each of the arithmetic microoperations can be implemented in one composite arithmetic circuit
- ✓ The basic component is the parallel adder
- ✓ The diagram of a 4-bit arithmetic circuit is shown in Fig.

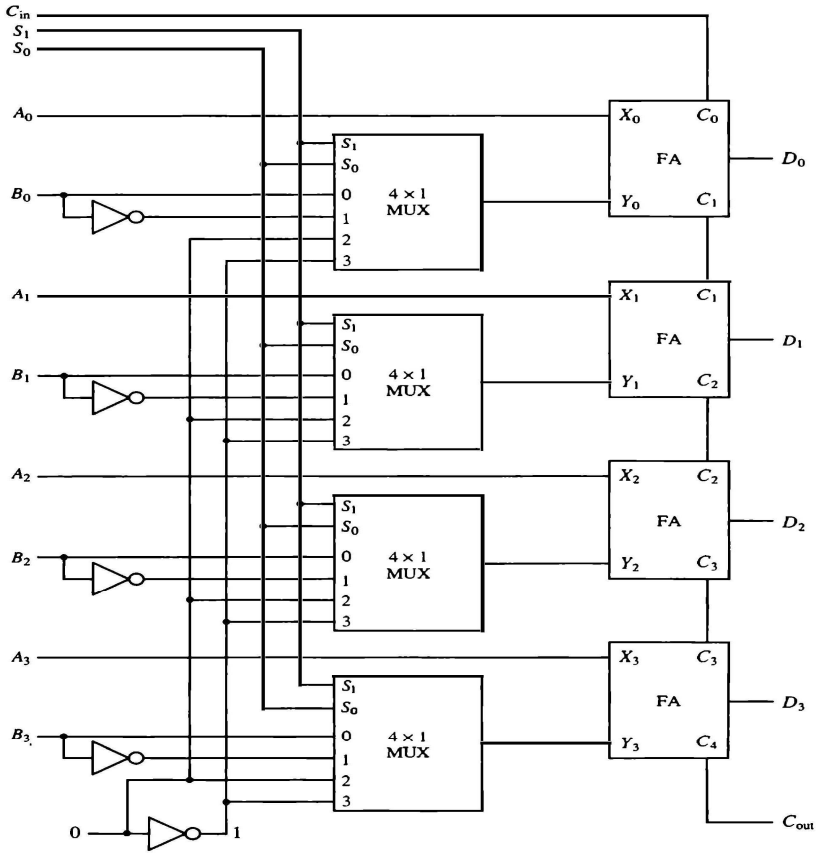


Figure 4-9 4-bit arithmetic circuit.

- ✓ It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. 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 multiplexer's data inputs also receive the complement of B. The other two data inputs are connected to logic-0 ad logic -1
- ✓ The four multiplexers are controlled by two selection inputs, S1 and S0. The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- ✓ The output of the binary adder is calculated from the following sum: $D = A + Y + C_{in}$
- ✓ 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 microoperations listed in Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
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

- ✓ When S1 S0 = 00, the value of B is applied to the Y inputs of the adder. If Cin = 0, the output D = A + B. If Cin = 1, output D = A + B + 1. Both cases perform the add microoperation with or without adding the input carry.
- ✓ When S1 S0 = 01, the complement of B is applied to the Y inputs of the adder. If Cin = 1, then D =A +B+ 1. This produces A plus the 2's complement of B, which is equivalent to a subtract with borrow, that is, A – B – 1.
- ✓ When S1S0 = 10, the input from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 +Cin. This gives D = A when Cin = 0 and D = A +1 when Cin = 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 S1 S0 = 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 Cin = 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 microoperations in the arithmetic circuit.

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 ← R1 ⊕ R2
- ✓ Example: R1 = 1010 and R2 = 1100

  1010 Content of R1

  1100 Content of R2

  0110 Content of R1 after P = 1
- ✓ Symbols used for logical microoperations:
  - OR: ∨
  - AND: ∧
  - XOR: ⊕
- ✓ 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 ← R2 + R3, R4 ← R5 ∨ R6

List of Logic Microoperations

- ✓ There are 16 different logic operations that can be performed with two binary variables

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

- ✓ The 16 logic microoperations 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

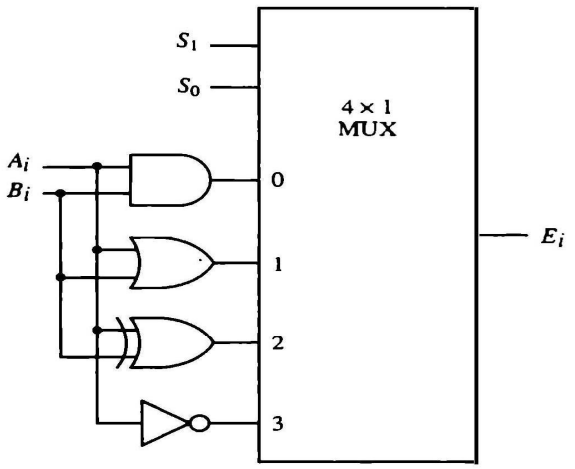
Sixteen Logic Microoperations

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
$F_{15} = 1$	$F \leftarrow$ 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
- ✓ All 16 microoperations can be derived from using four logic gates

Figure One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

Some applications

- ✓ 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

  $A \leftarrow A \wedge \overline{B}$
- ✓ 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
  1100 B (logic operand)
  1000 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

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

  0000 1010 A before
  1001 0000 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 number are equal

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

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

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

- ✓ 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

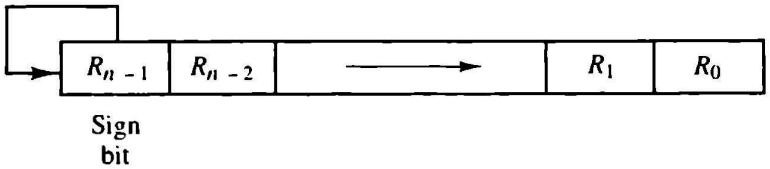
  R1 ← shl R1

  R2 ← shr R2

- ✓ The *circular shift* (also known as rotate operation) 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

- ✓ The *arithmetic shift* shifts a signed binary number to the left or right
- ✓ To the left is multiplying by 2, to the right is dividing by 2
- ✓ Bit R_{n-1} in the leftmost position holds the sign bit. R_{n-2} is the most significant bit of the number and R0 is the least significant bit. Arithmetic shifts must leave the sign bit unchanged and shifts the number (including the sign bit) to the right.

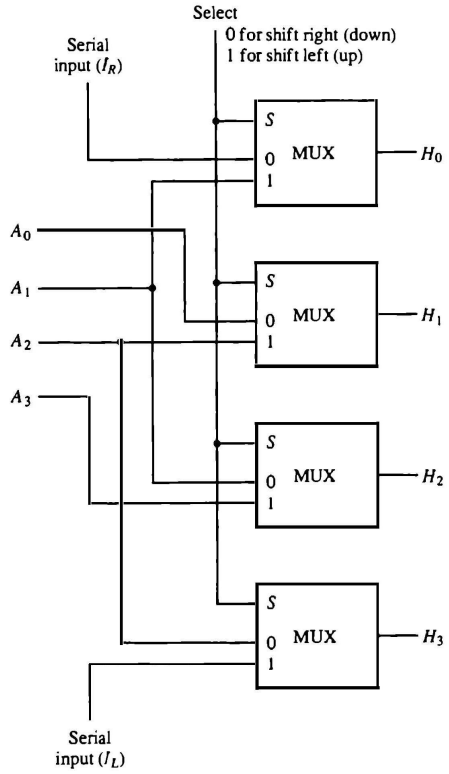


- ✓ The arithmetic shift-left inserts a 0 into R0, and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in Rn-1 changes in value after the shift. This happens if the multiplication causes an overflow
- ✓ An overflow flip-flop V_s can be used to detect the overflow $V_s = R_{n-1} \oplus R_{n-2}$

- ✓ If Vs = 0, there is no overflow, bit if Vs = 1, there is an overflow and a sign reversal after the shift. Vs must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Hardware implementation

- ✓ 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



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

4-bit combinational circuit shifter.

- ✓ The 4-bit shifter has four data inputs, A0 through A3, and four data outputs, H0 through H3. There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).
- ✓ 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).
- ✓ A shifter with n data inputs and outputs requires n multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.