1. Explain the Register Transfer Language.

Definition: The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.

- 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.
- 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 describeing the internal organization of digital computers in concise and precise manner.
- It can also be used to facilitate the design process of digital systems.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.
- The statement below denotes a transfer of the content of register R1 into register R2.

$$R2 \leftarrow R1$$

- A statement that specifies a register transfer implies that circuits are available from the outputs of the destination register has a parallel load capability.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.

2. Explain the Register Transfer in detail.

Definition: Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.

$$R2 \leftarrow R1$$

Denotes a transfer of the content of register R1 into register R2.

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.
  For example:

MAR	Holds address of memory unit
PC	Program Counter
IR	Instruction Register
R_1	Processor Register

- Below figure1.1 shows the 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 shown in figure1.1.

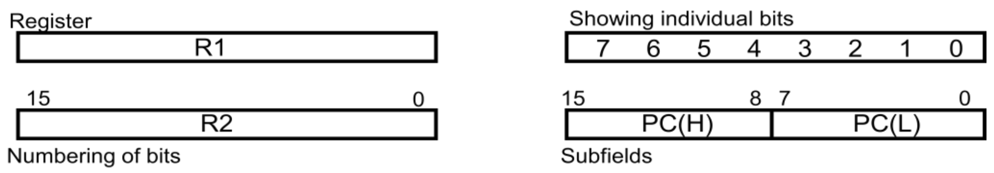


Figure 1.1: Block diagram of register

- 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.
- The statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability.
- The basic symbols of the register transfer notation are listed in Table below:

Symbol	**Description**	**Examples**
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R 2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2←R1
Comma ,	Separates two micro operations	R2←R1, R1←R2

Table 1.1: **Basic Symbols for Register Transfers**

- Registers are denoted by capital letters, and numerals may follow the letters.
- Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register.
- The arrow denotes a transfer of information and the direction of transfer.
- A comma is used to separate two or more operations that are executed at the same time.

3. Draw and explain the block diagram for register transfer of data from R1 to R2 when control p = 1.

Register Transfer with control function:

- If we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.
  `If (P = 1) then (R2← R1)`
  where P is a control signal.
- It is sometimes convenient to separate the control variables from the register transfer operation control function by specifying a control function.
- A **control function** is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:
  `P: R2 ←R1`
- The control condition is terminated with a colon. It symbolizes the requirement that the 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. Below figure shows the block diagram that depicts the transfer from R1 to R2.

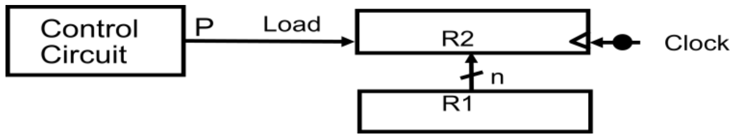


Figure 1.2: Transfer from R1 to R2 when P = 1

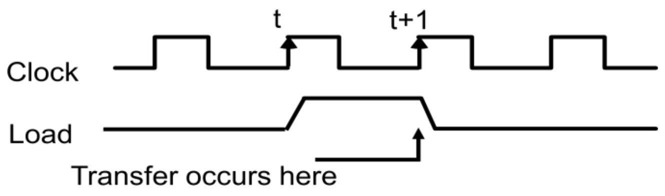


Figure 1.3: Timing diagram

- 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.
- 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.
- The statement below, denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.
  `T: R2← R1, R1← R2`

4. Design and explain a common bus system for four register.

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.
- 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 bus during each particular register transfer.
- One way of constructing a 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 figure1.4.
- Each register has four bits, numbered 0 through 3.
- 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.
- 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.

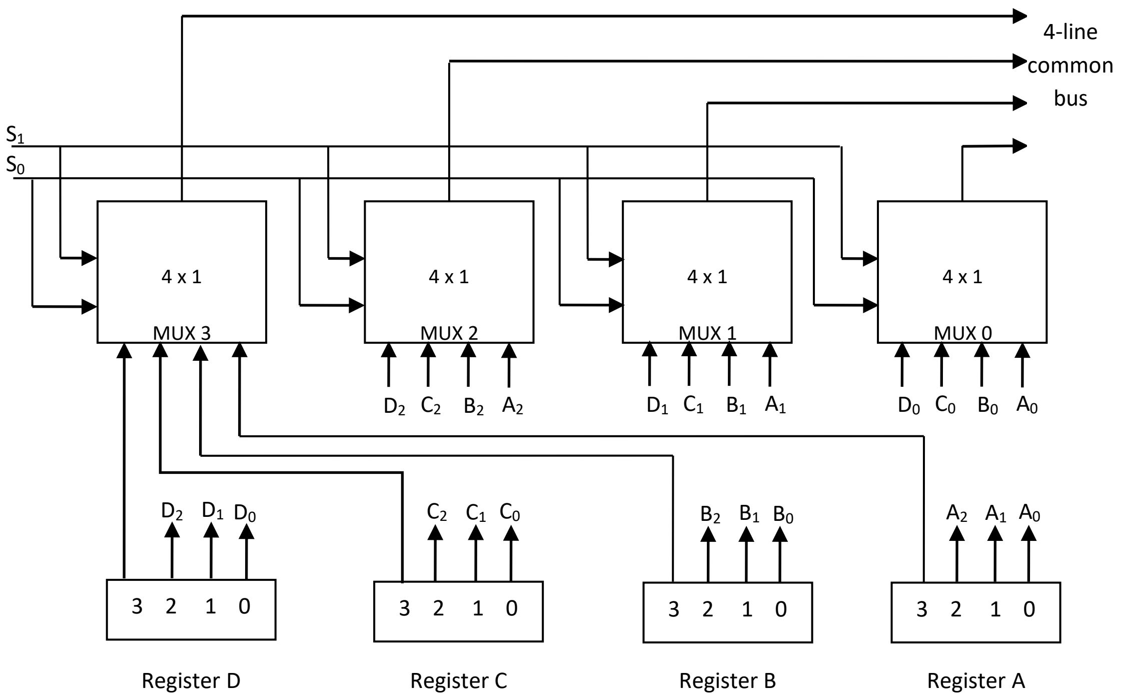


Figure 1.4: Bus system for four registers

- 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.

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

Table 1.2: Function Table for 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 shows the register that is selected by the bus for each of the four possible binary values of the selection lines.
- In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus.
- The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.
- The size of each multiplexer must be K x 1 since it multiplexes K data lines.
  For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

5. A digital computer has a common bus system for 16 registers of 32 bits each. (i) How many selection input are there in each multiplexer? (ii) What size of multiplexers is needed? (iii) How many multiplexers are there in a bus?

(i) How many selection input are there in each multiplexer?
2^n=No. of Registers; n=selection input of multiplexer
2^n=16; here n=4
Therefore 4 selection input lines should be there in each multiplexer.

(ii) What size of multiplexers is needed?
size of multiplexers= Total number of register X 1
= 16 X 1
Multiplexer of 16 x 1 size is needed to design the above defined common bus.

(iii) How many multiplexers are there in a bus?
No. of multiplexers = bits of register

= 32

32 multiplexers are needed in a bus.

6. Explain the operation of three state bus buffers and show its use in design of common bus.

- 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.
  State 1: Signal equivalent to Logic 1
  State 2: Signal equivalent to Logic 0
  State 3: High Impedance State (behaves as open circuit)
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- The most commonly used design of a bus system is the buffer gate.
- The graphic symbol of a three-state buffer gate is shown in figure 1.5.

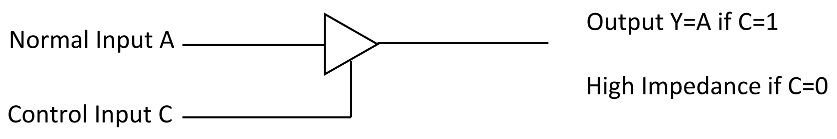


Figure 1.5: Graphic symbols for three-state buffer

- 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 C 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 C is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.
- The high-impedance state of a three-state gate provides a special feature not available in other gates.
- 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 construction of a bus system with three-state buffers is demonstrated in figure 1.6.
- 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 figure: Bus line with three state-buffers.

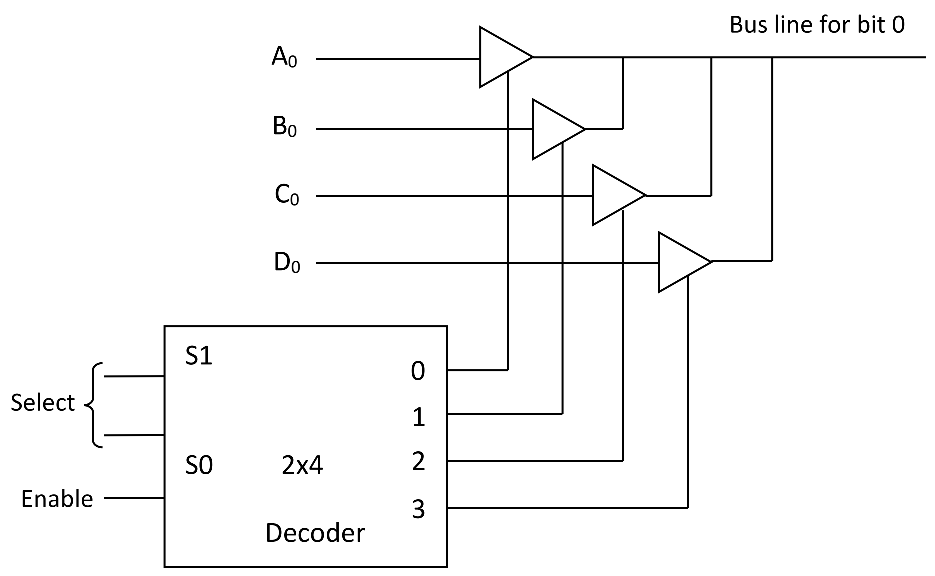


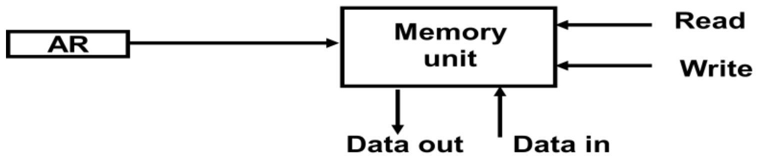
Figure 1.6: Bus line with three state-buffers

- 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.
- 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 as shown in figure1.6.
- Each group of four buffers receives one significant bit from the four registers.
- Each common output produces one of the lines for the common bus for a total of n lines.
- Only one decoder is necessary to select between the four registers.

7. Explain Memory Transfer

- **Read Operation:** The transfer of information from a memory word to the outside environment is called a read operation.
- **Write 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.
- 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.
- Write operation can be stated symbolically as follows:
  Write: M[AR] ←R1
- This causes a transfer of information from R1 into memory word M selected by address AR.



8. Explain Arithmetic Micro-operation.

- The basic arithmetic micro-operations are:
  1. Addition
  2. Subtraction
  3. Increment
  4. Decrement
  5. Shift
- The additional arithmetic micro operations are:
  1. Add with carry
  2. Subtract with borrow
  3. Transfer/Load , etc.
- Summary of Typical Arithmetic Micro-Operations:

R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 – 1	Decrement

9. Explain Binary Adder in detail

- To implement the add micro operation with hardware, we need :
    1. Registers : that hold the data
    2. Digital component: that performs the arithmetic addition.
- **Full-adder**
  The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder.
- **Binary adder**
  The digital circuit that generates the arithmetic sum of two binary numbers of any lengths is called a binary adder.
- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

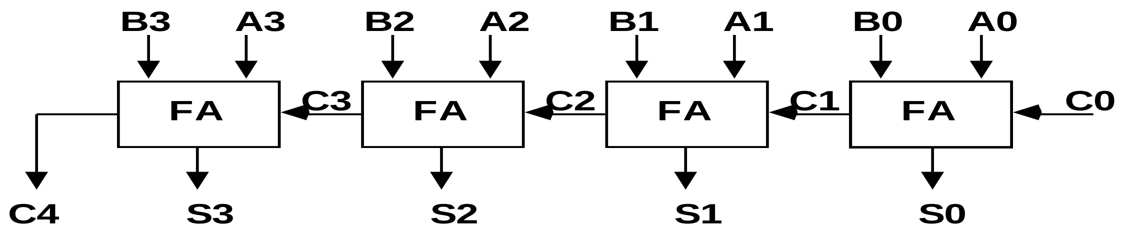


Figure 1.7: 4-bit binary adder

- Above figure 1.7 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.
- The augends bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders.
- The input carry to the binary adder is C0 and the output carry is C4.
- The S outputs of the full-adders generate the required sum bits.
- 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.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2). The sum can be transferred to a third register or to one of the source registers (R1 or R2), replacing its previous content.

10. Explain Binary Adder-Subtractor in detail.

- The subtraction of binary numbers can be done most conveniently by means of complements.
- Remember that 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 l's complement and adding one to the least significant pair of bits. The l'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 exclusive-OR gate with each full-adder.
- The mode input M controls the operation.
  When **M = 0** the circuit is an **Adder**
  When **M = 1** the circuit becomes a **Subtractor**
- Each exclusive-OR gate receives input M and one of the inputs of B.
  When M = 0,
  We have $C_0=0$
  $B \oplus 0 = B$
  The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
  When M = 1,
  We have $C_0=1$
  $B \oplus 1 = B`$; B complement
  The B inputs are all complemented and 1is added through the input carry. The circuit performs the operation A plus the 2's complement of B.

  A + 2's compliment of B
- A 4-bit adder-subtractor circuit is shown as follows:

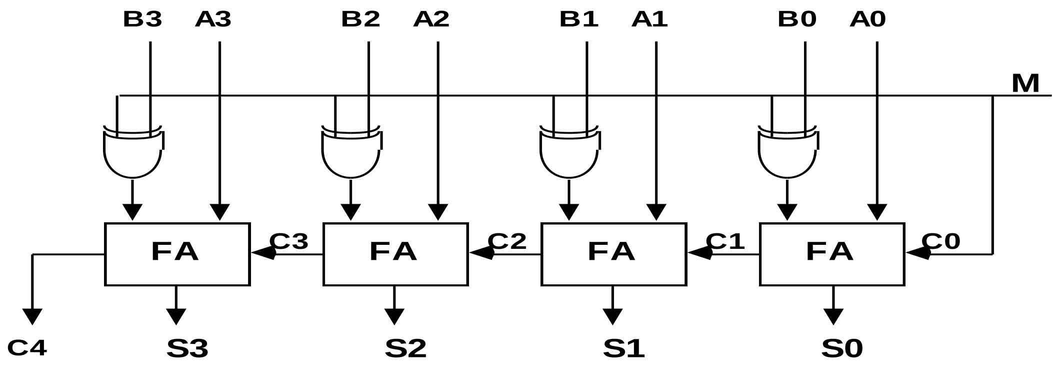


Figure 1.8: 4-bit Adder-Subtractor

- For unsigned numbers,
  If A>=B, then A-B
  If A<B, then B-A
  For signed numbers,
  Result is A-B, provided that there is no overflow

11. Explain Binary Incrementer

- The increment micro operation adds one to a number in a register.
- For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.

```
0 1 1 0
+     1
--------
0 1 1 1
```

- The diagram of a 4-bit combinational circuit incrementer is shown above. One of the inputs to the least significant half-adder (HA) is connected to logic-1 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 circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3.
- The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

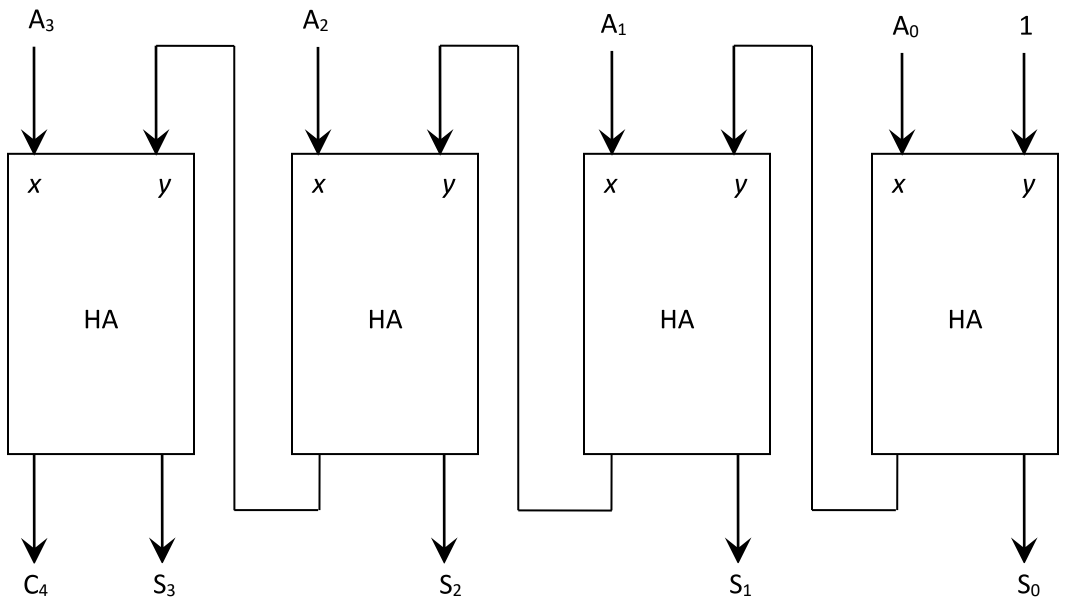


Figure 1.9: 4-bit binary incrementer

12. Draw block diagram of 4-bit arithmetic circuit and explain it in detail.

- The arithmetic micro operations 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.
- Hardware implementation consists of:
  1. 4 full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
  2. There are two 4-bit inputs A and B
     The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B is connected to the data inputs of the multiplexers. The multiplexer's data inputs also receive the complement of B.
  3. The other two data inputs are connected to logic-0 and logic-1. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0.
  4. The four multiplexers are controlled by two selection inputs, S_1 and S_0.
  5. 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.
  6. 4-bit output $D_0 \ldots D_3$
- The diagram of a 4-bit arithmetic circuit is shown below figure 1.10:

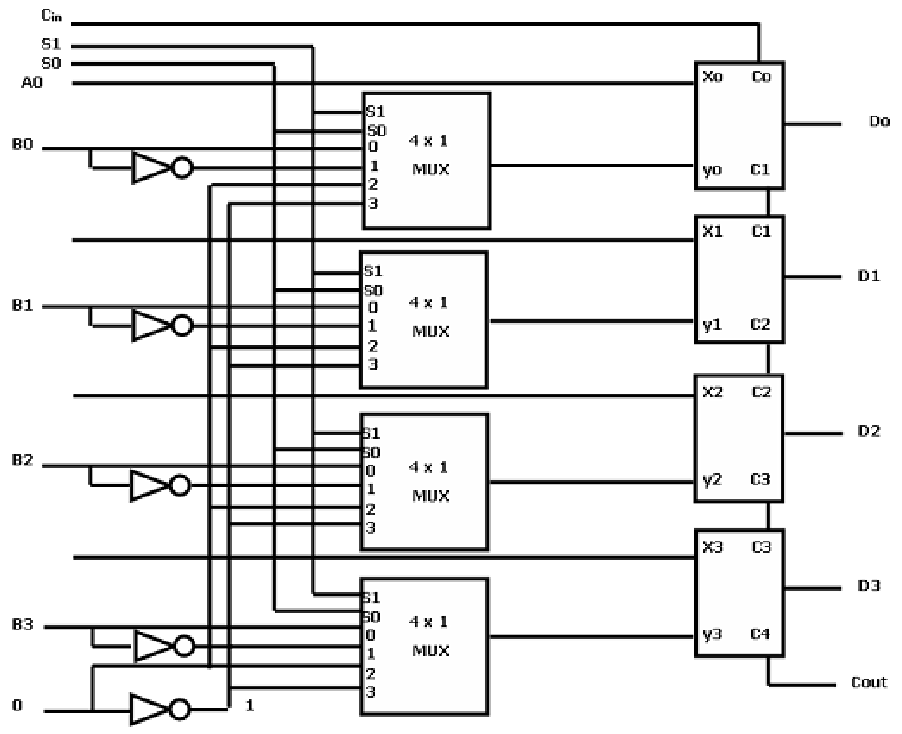


Figure 1.10: 4-bit arithmetic circuit

- The output of binary adder is calculated from arithmetic sum.

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

Select S_1 S_0 C_{in}			**Input** Y	**Output** D = A + Y + Cin	**Microoperation**
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with Carry
0	1	0	B’	D = A + B’	Subtract with Borrow
0	1	1	B’	D = A + 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

TABLE 1.3: 4-4 Arithmetic Circuit Function Table

- When S_1 S_0= 0 0
  If Cin=0, D=A+B; Add
  If Cin=1, D=A+B+1;Add with carry
- When S_1 S_0= 0 1
  If Cin=0, D=A+$\overline{B}$; Subtract with borrow
  If Cin=1, D=A+$\overline{B}$+1;A+2’s compliment of B i.e. A-B
- When S_1 S_0= 1 0
  Input B is neglected and Y=> logic ‘0’
  D=A+0+ Cin
  If Cin=0, D=A; Transfer A
  If Cin=1, D=A+1;Increment A
- When S_1 S_0= 1 1
  Input B is neglected and Y=> logic ‘1’
  D=A-1+ Cin
  If Cin=0, D=A-1; 2’s compliment
  If Cin=1, D=A; Transfer A

- Note that the micro-operation D = A is generated twice, so there are only seven distinct micro-operations in the arithmetic circuit.

13. Draw and explain Logic Micro-operations in detail.

- Logic micro operations 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 micro-operation with the contents of two registers R1 and R2 is symbolized by the statement:

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

```
          1 0 1 0   Content of R1
     ⊕    1 1 0 0   Content of R2
-----------------------------------
          0 1 1 0   Content of R1 after P = 1
```

- 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.
- **Notation:**
  The symbol **V** will be used to denote an **OR** microoperation and the symbol **Λ** 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.
- Although the + symbol has two meanings, it will be possible to distinguish between them by noting where the symbol occurs. When the symbol + occurs in a microoperation, it will denote an arithmetic plus. When it occurs in a control (or Boolean) function, it will denote an OR operation.

$$P + Q: R1 \leftarrow R2 + R3,\ R4 \leftarrow R5 \vee R6$$

- 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 R5and R6.

List of Logic Micro operations

- 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 below.

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 1.4: Truth Tables for 16 Functions of Two Variables

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
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all l's

TABLE 1.5: Sixteen Logic Microoperation

Hardware Implementation

- The hardware implementation of logic microoperation 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 microoperation, most computers use only four—AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.
- Below figure shows one stage of a circuit that generates the four basic logic micro operations.

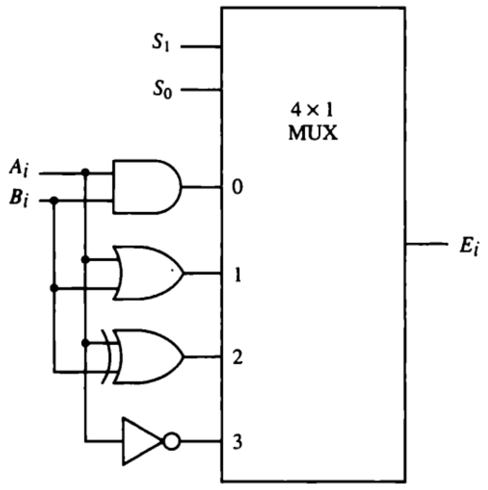


Figure 1.11 : One stage of logic circuit

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	$E = \overline{A}$	Compliment

Table 1.6: Function table

- Hardware implementation 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.
- The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for i = 0, 1, 2, . . . n - 1. The selection variables are applied to all stages.

14. Explain selective set, selective complement and selective clear.

Selective-Set operation:

- 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:

```
1010   A before
1100   B (logical operand)
1110   A after
```

- The two leftmost bits of B are 1's, so the corresponding bits of A are set to 1.
- One of these two bits was already set and the other has been changed from 0 to 1.
- The two bits of A with corresponding 0's in B remain unchanged. The example above serves as a truth table since it has all four possible combinations of two binary variables.
- The OR microoperation can be used to selectively set bits of a register.

Selective-Complement operation:

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

```
1010   A before
1100   B (logical operand)
0110   A after
```

- Again the two leftmost bits of B are 1's, so the corresponding bits of A are complemented.
- The exclusive-OR microoperation can be used to selectively complement bits of a register.

Selective-Clear operation:

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

```
1010   A before
1100   B (logical operand)
0010   A after
```

- Again the two leftmost bits of B are 1's, so the corresponding bits of A are cleared to 0.
- One can deduce that the Boolean operation performed on the individual bits is AB'.
- The corresponding logic microoperation is $A \leftarrow A \wedge B'$.

15. Explain shift micro operations and draw 4-bit combinational circuit shifter.

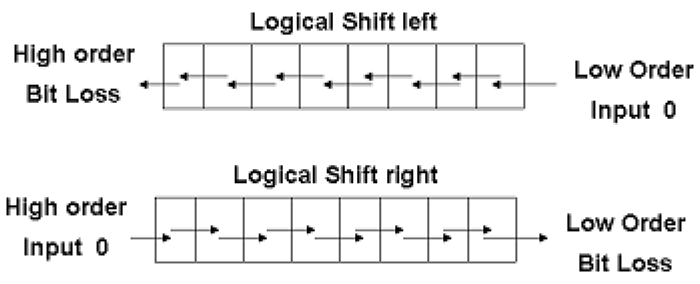
There are 3 types of shift micro-operations:

Logical Shift:

- A logical shift is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right micro-operations.
- For example:

$$R1 \leftarrow shl\ R1$$
$$R2 \leftarrow shr\ R2$$

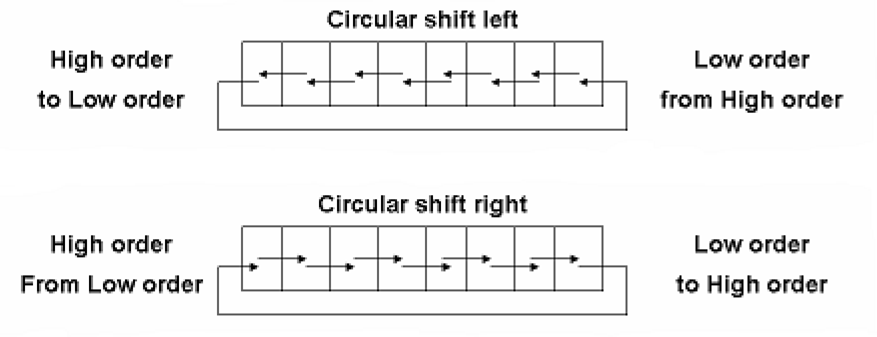
  are two micro-operations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2.
- The register symbol must be the same on both sides of the arrow.
- The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.



Circular Shift:

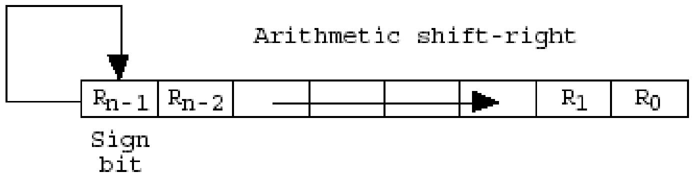
- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input. We will use the symbols cil and cir for the circular shift left and right, respectively.

$$R1 \leftarrow Cil\ R1$$
$$R2 \leftarrow Cir\ R2$$

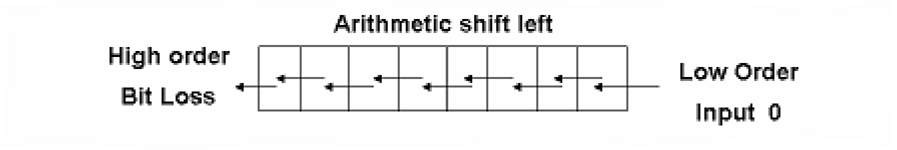


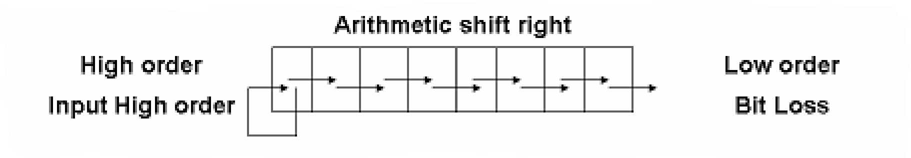
Arithmetic Shift:

- An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.
- The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative.
- Negative numbers are in 2's complement form.



- Figure shows a typical register of n bits. Bit R_{n-1} in the leftmost position holds the sign bit.
- R_{n-2} is the most significant bit of the number and R_0 is the least significant bit.
- The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right.
- Thus R_{n-1} remains the same; R_{n-2} receives the bit from R_{n-1}, and so on for the other bits in the register.
- The bit in R_0 is lost.
- The arithmetic shift-left inserts a 0 into R_0, 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 by 2 causes an overflow.





4-bit combinational circuit shifter

- A combinational circuit shifter can be constructed with multiplexers as shown in Figure below.

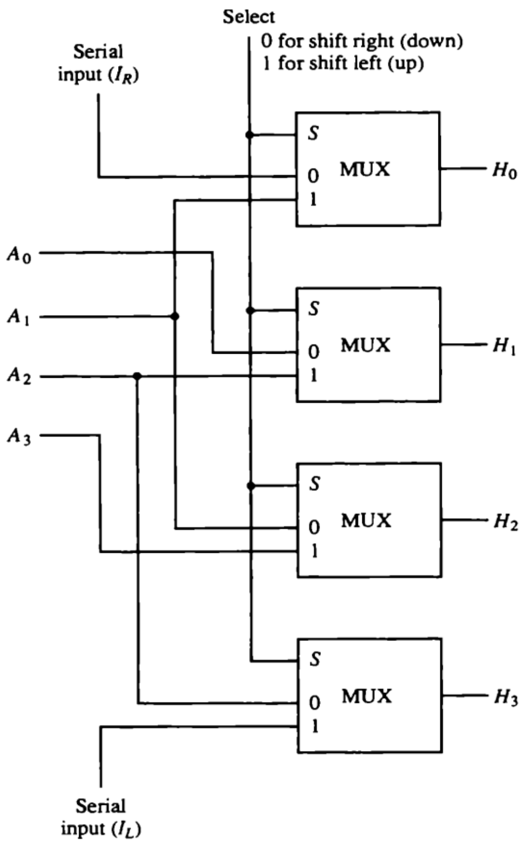


Figure 1.12: 4-bit combinational circuit shifter

- 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 (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 Figure shows which input goes to each output after the shift.

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

- 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.

16. Draw and explain one stage of arithmetic logic shift unit.

- Instead of having individual registers performing the micro operations 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.
- 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 dock pulse period.
- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables.
- One stage of an arithmetic logic shift unit is shown in figure below:

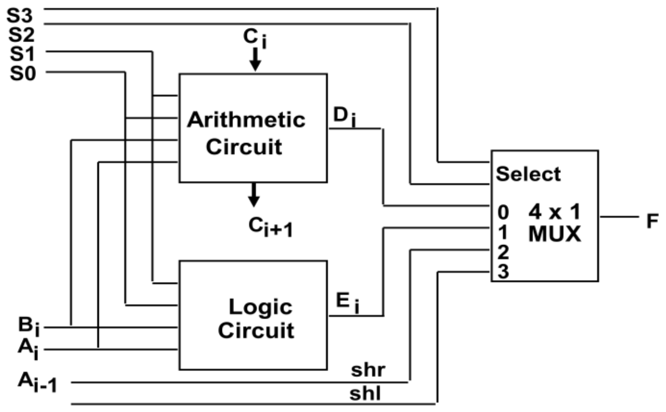


Figure 1.13: One stage of arithmetic logic shift unit

- The subscript i designates a typical stage. Inputs A_i and B_i are applied to both the arithmetic and logic units.
- A particular microoperation is selected with inputs S_1 and S_0.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in D_i and a logic output in E_i.
- The data in the multiplexer are selected with inputs S_3 and S_2.
- The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.
- Note that the diagram shows just one typical stage. The circuit shown in figure must be repeated n times for an n-bit ALU.
- The outputs carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_{in} of the next stage in sequence.
- The input carry to the first stage is the input carry on, which provides a selection variable for the arithmetic operations.

- The circuit whose one stage is specified in figure 1.13 provides
  - 8 arithmetic operation
  - 4 logic operations
  - 2 shift operations
- Each operation is selected with the five variables S_3, S_2, S_1, S_0, and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.
- Table below lists the 14 operations of the ALU.

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	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = $\overline{A}$	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

Table 1.7: Function Table for Arithmetic Logic Shift Unit

- The first eight are arithmetic operations and are selected with S_3S_2 = 00.
- The next four are logic operations are selected with S_3S_2 = 01. The input carry has no effect during the logic operations and is marked with don't-care X's.
- The last two operations are shift operations and are selected with S_3S_2 = 10 and 11.
- The other three selection inputs have no effect on the shift.