UNIT – 1 :
Overview of Register Transfer and Micro Operations

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 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.

- 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.

$$\mathbf{R2 \leftarrow R1}$$

- 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.

Register Transfer in detail with block diagram and timing diagram :

- **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

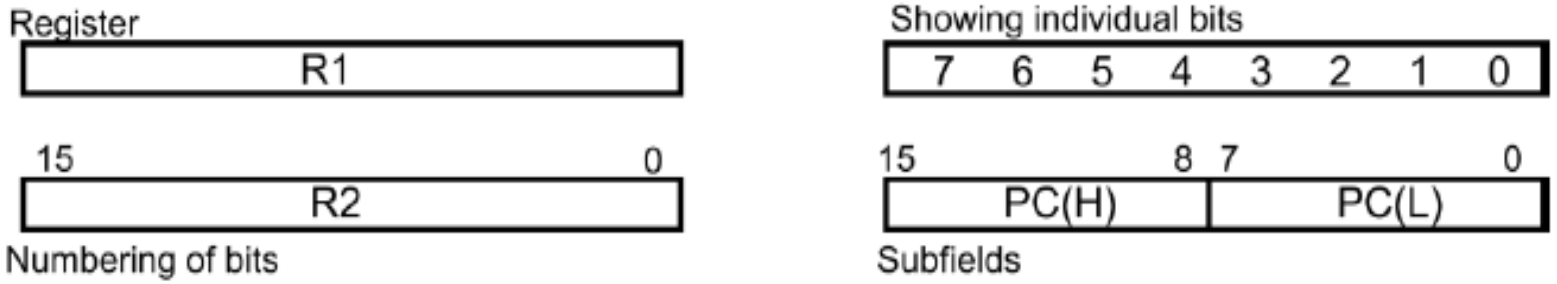


Figure 1.1: Block diagram of register

• The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in figure.

• 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.

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`

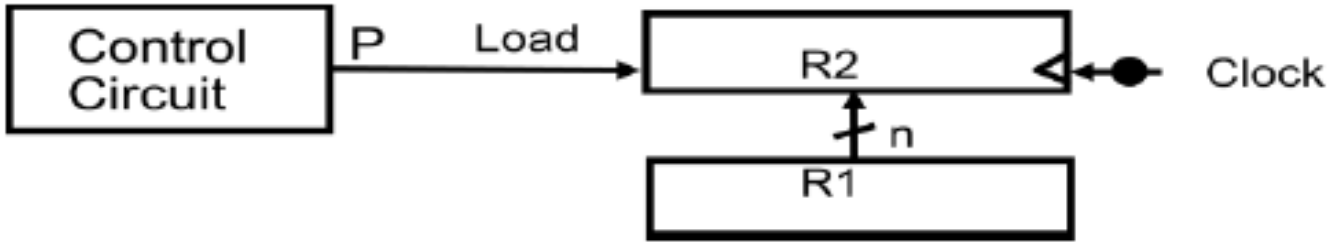


Figure 1.2: Transfer from R1 to R2 when P = 1

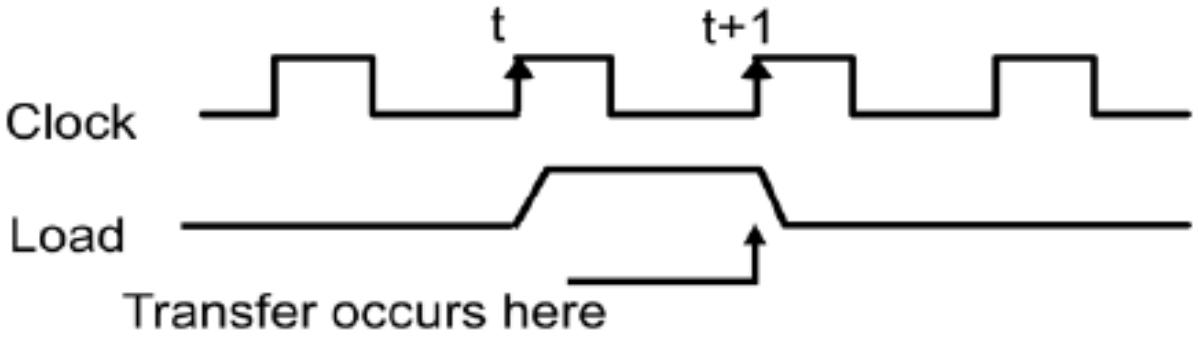


Figure 1.3: Timing diagram

- 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.

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

- A comma is used to separate two or more operations that are executed at the same time.
- 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, PC ← PC+1`

- This simultaneous operation is possible with registers that have edge-triggered flipflops.

Common Bus System :

- 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 figure below.

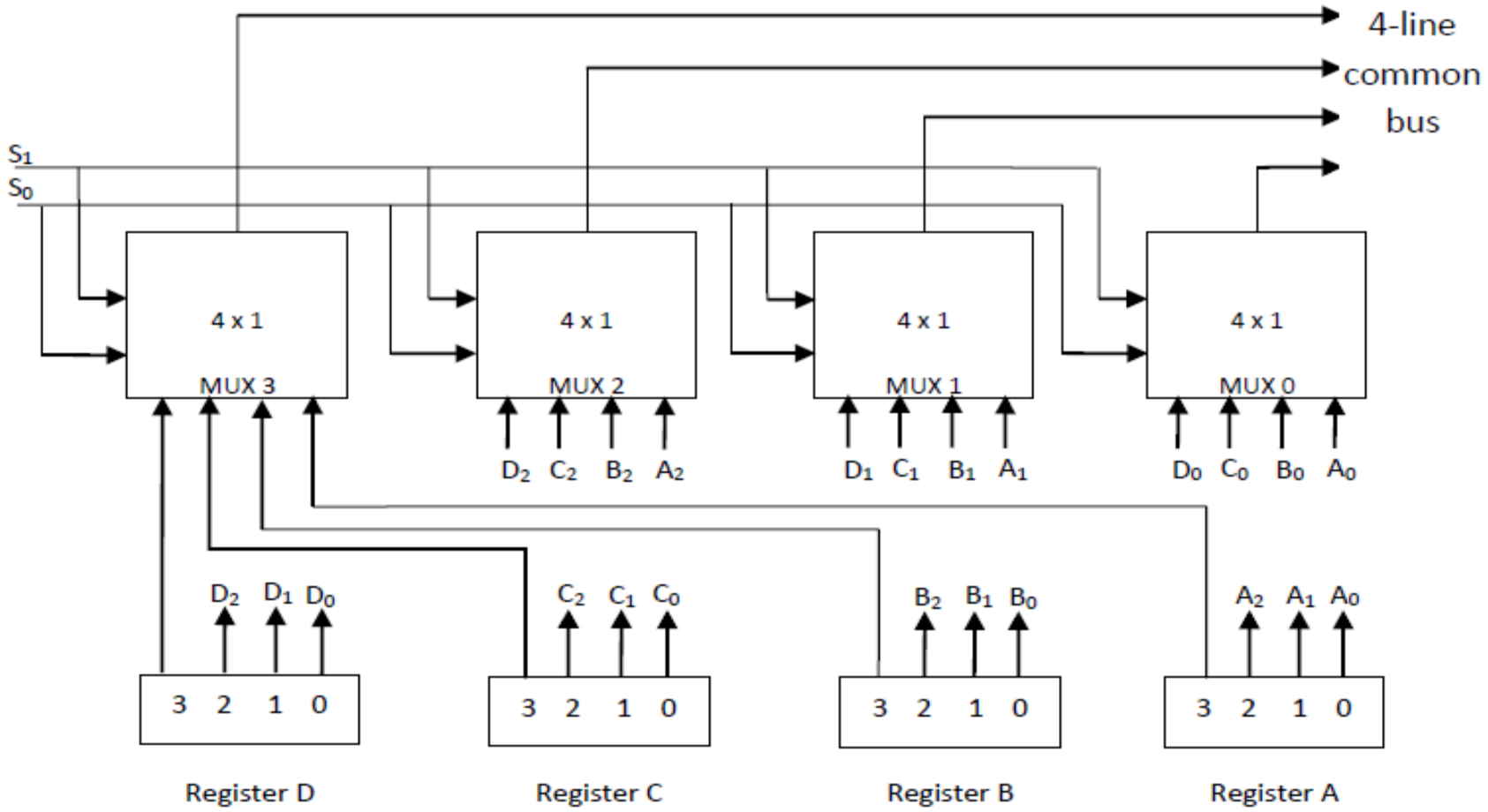


Figure 1.4: Bus system for four registers

- 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, S1 and S0.
- 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.
- The two selection lines S1 and S0 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

- When S1S0 = 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 S1S0 = 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.

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.

Three-state bus buffer : Tri state buffer

- 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 below:

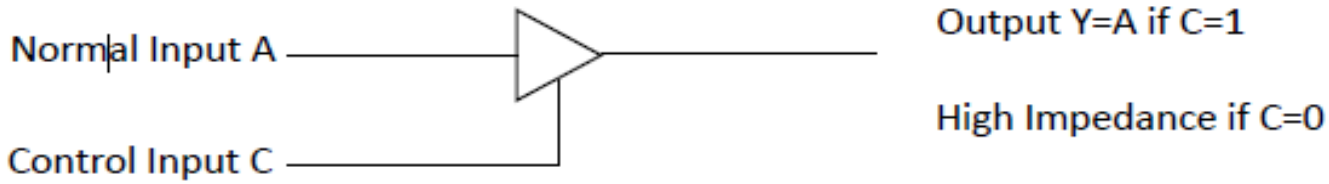


Figure 1.5: Graphic symbols for three-state buffer

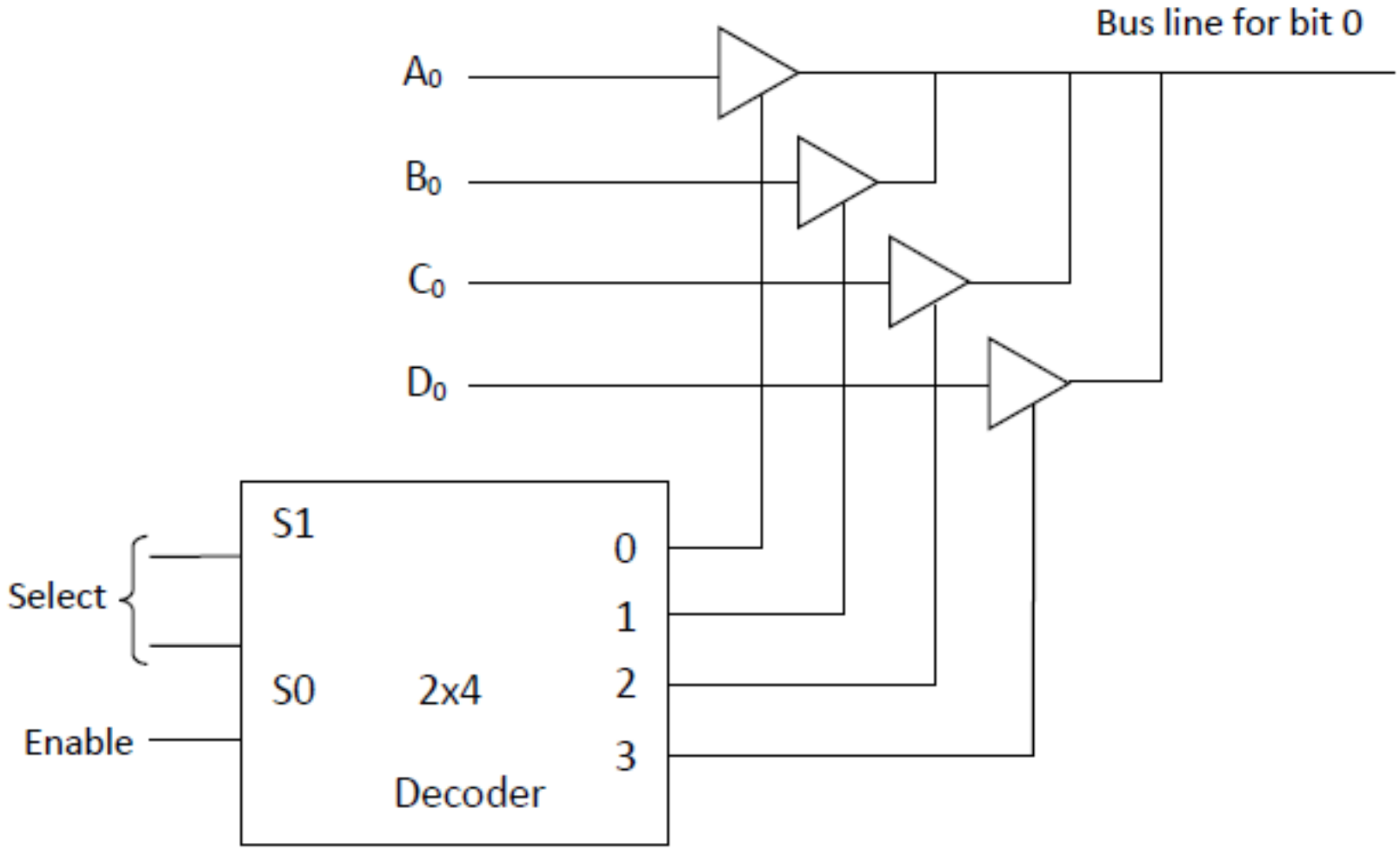


Figure 1.6: Bus line with three state-buffers

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.
- 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:

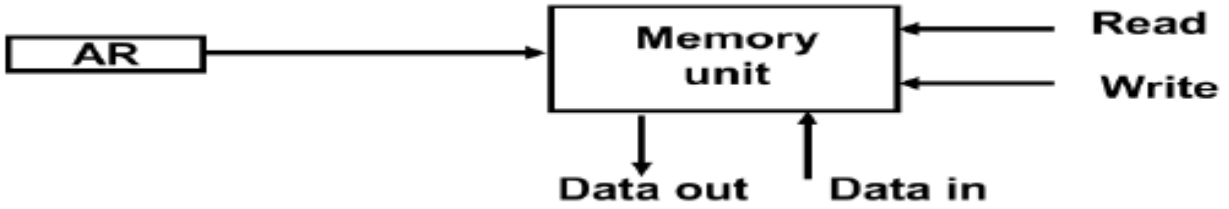
Read: DR $\leftarrow$M[AR]

- 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.



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.

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

Binary Adder:

- 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.

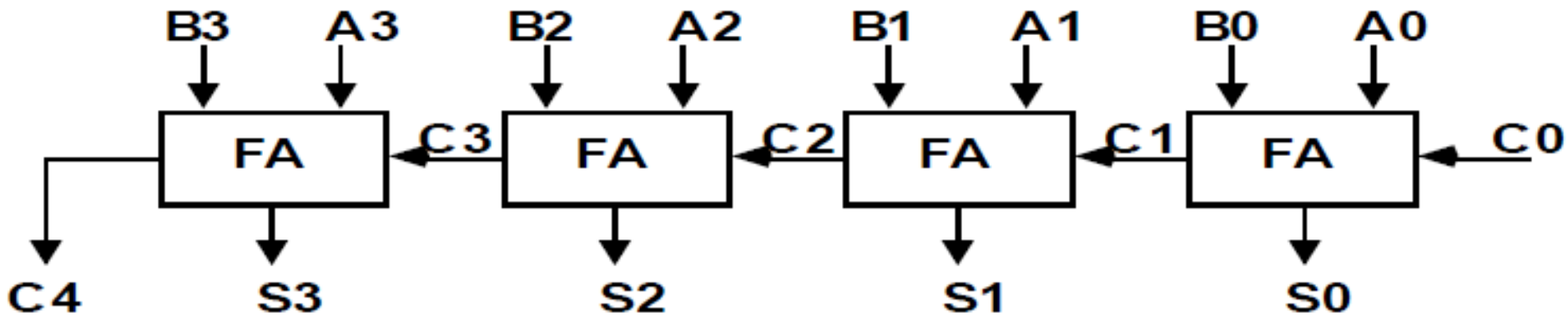


Figure 1.7: 4-bit binary adder

Binary Adder-Subtractor :

- 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 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 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 C0=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 C0=1

$$B \oplus 1 = B`$$; B complément

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:

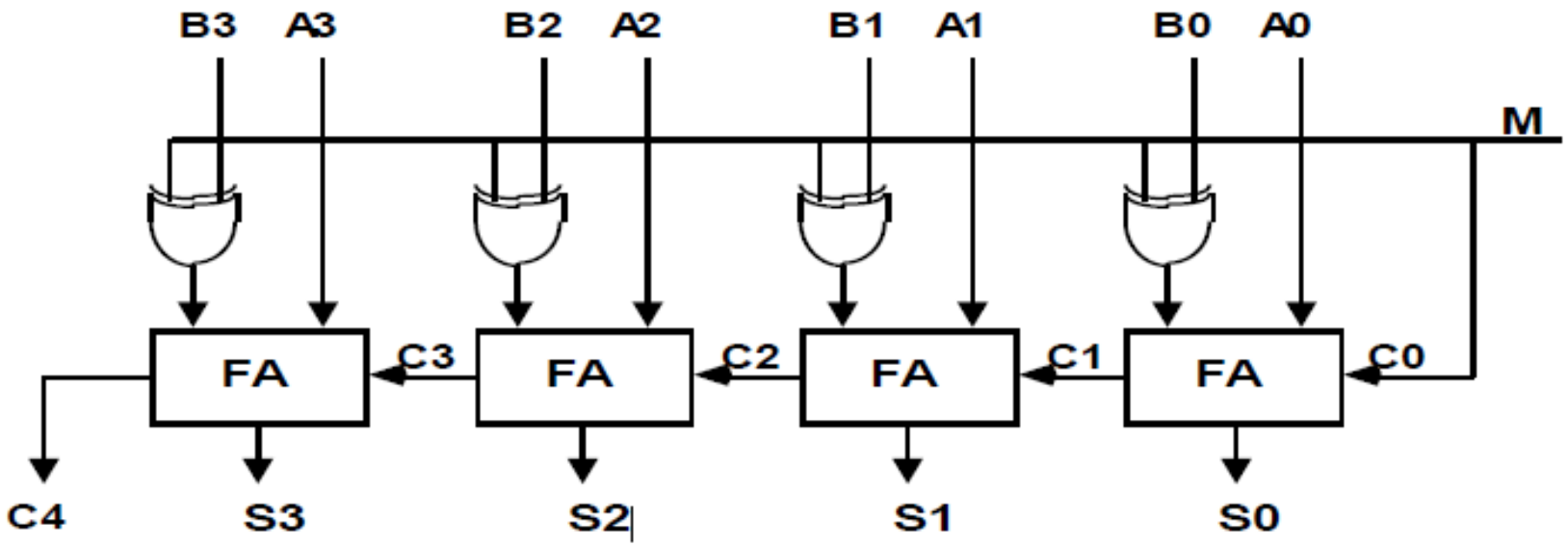


Figure 1.8: 4-bit Adder-Subtractor

- 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.

Binary Incrementer :

- The increment micro operation add 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 A0 through A3, adds one to it, and generates the incremented output in S0 through S3.
- The output carry C4 will be 1 only after incrementing binary 1111. This also causes outputs S0 through S3 to go to 0.

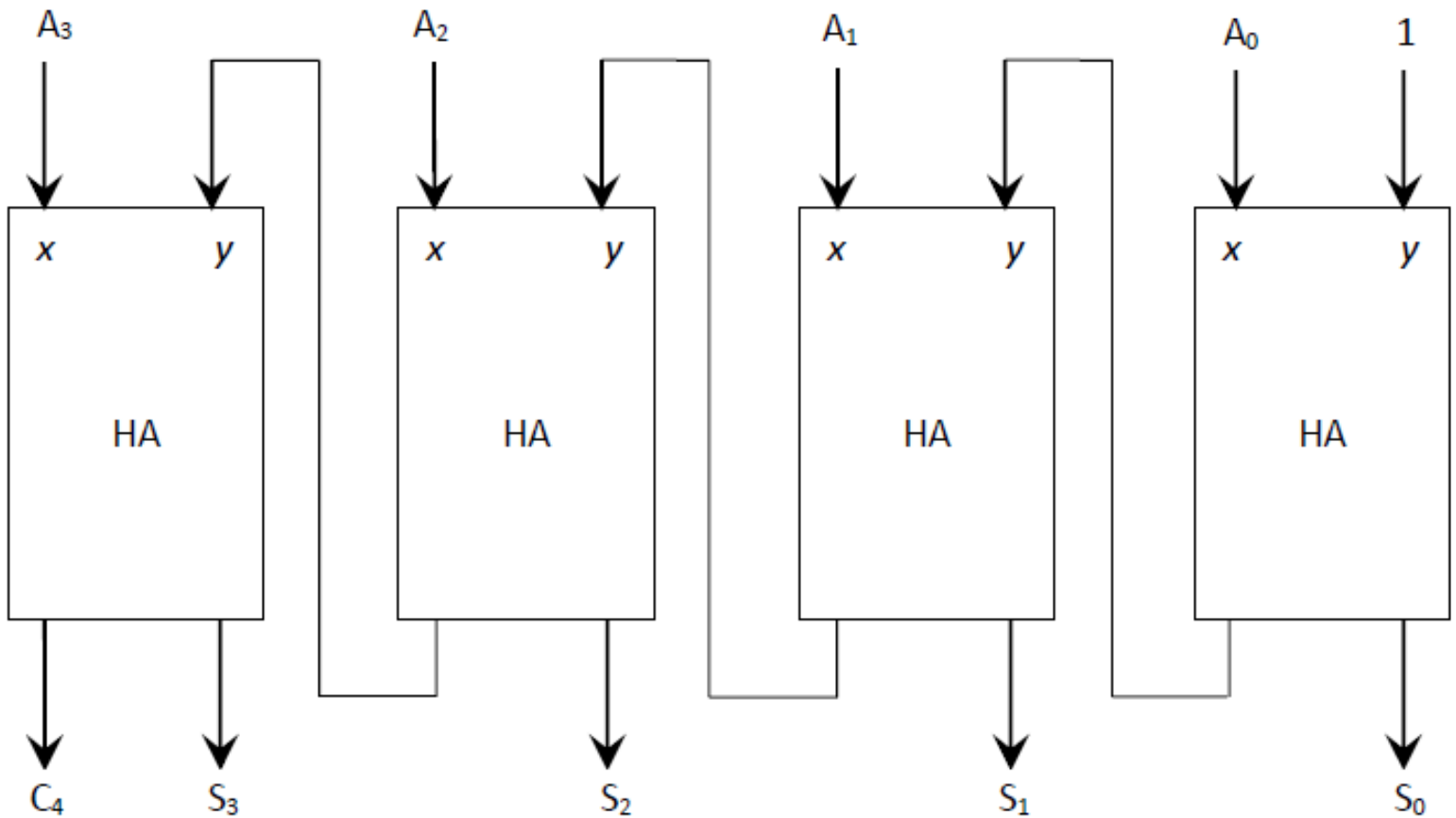
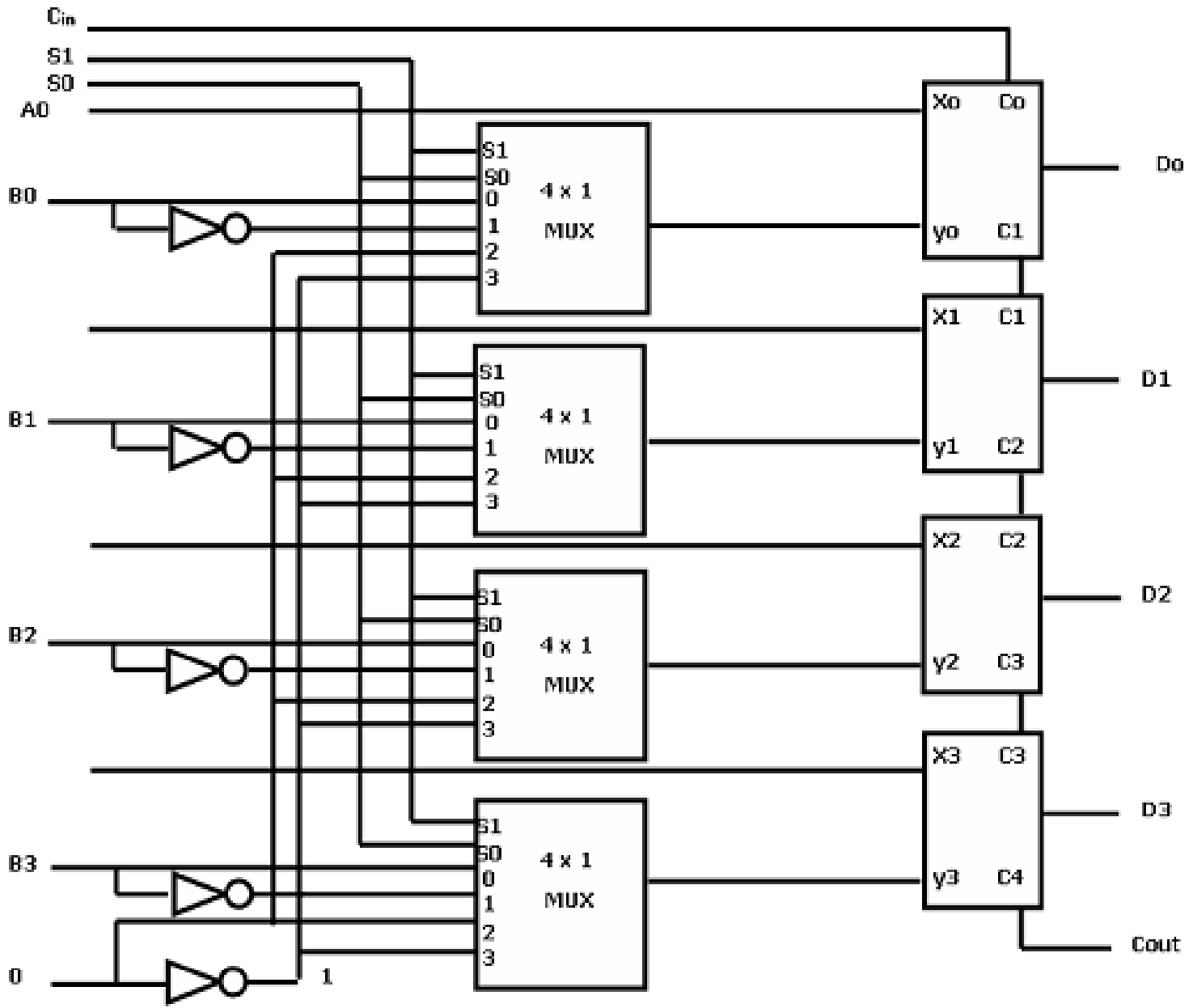


Figure 1.9: 4-bit binary incrementer

4-bit arithmetic circuit :



- 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, S1 and S0.

5. 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.
6. 4-bit output D0…D3

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

D=A+Y+Cin

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

Logic Micro-operations :

- 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 ←R1 ⊕ 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 ←R2 + R3, R4 ←R5 V 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

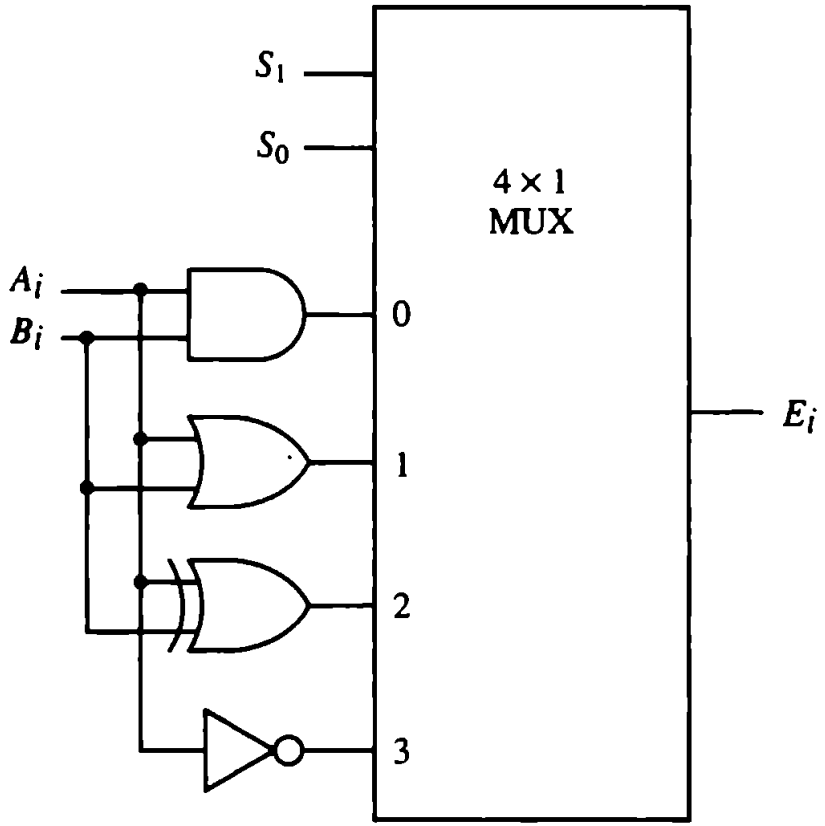
TABLE 1.4: Truth Tables for 16 Functions of Two Variables

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

Hardware Implementation :

• 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.



One stage of logic circuit

S_1	S_0	Output	Operation
0	0	E = A ∧ B	AND
0	1	E = A ∨ B	OR
1	0	E = A ⊕ B	XOR
1	1	E = $\bar{A}$	Compliment

Table 1.6: Function table

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 **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

• 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

- The corresponding logic microoperation is **A ← A ∧ B'.**

Shift micro operations :

- There are 3 types of shift micro-operations:

1. Logical Shift:

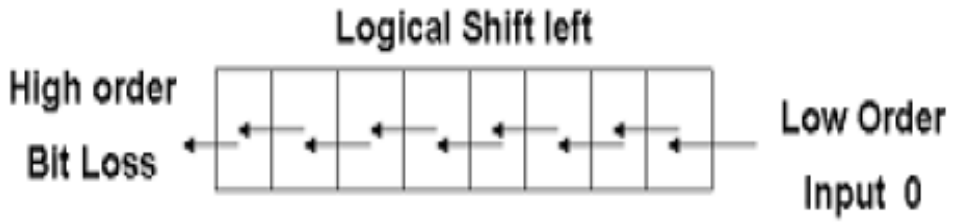
- A logical shift is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right micro-operations.
- For example:

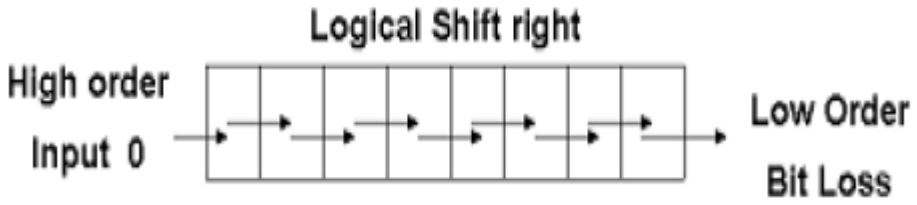
R1 ← shl R1

R2 ← 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.

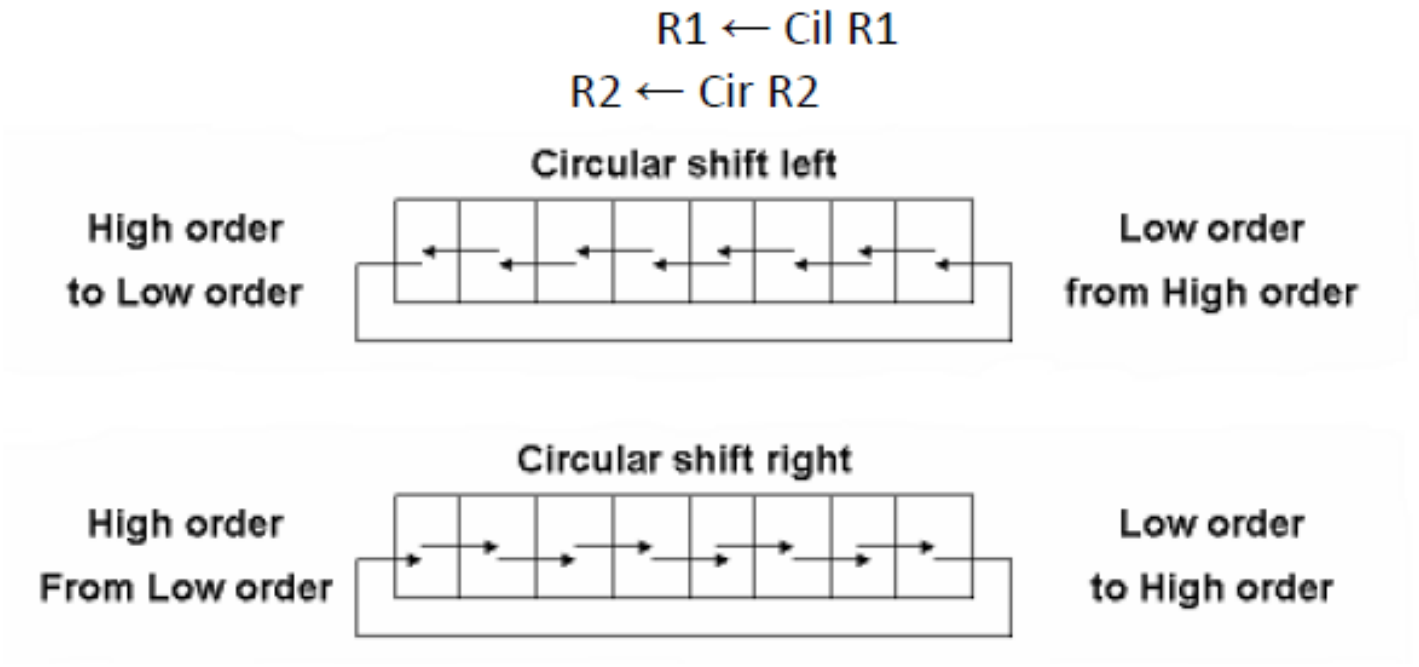




2. Circular Shift:

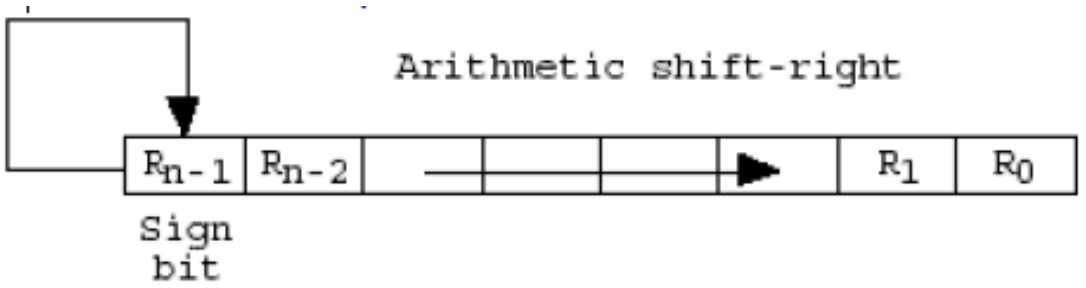
- The circular shift (also known as a **rotate operation**) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.

We will use the symbols cil and cir for the circular shift left and right, respectively.



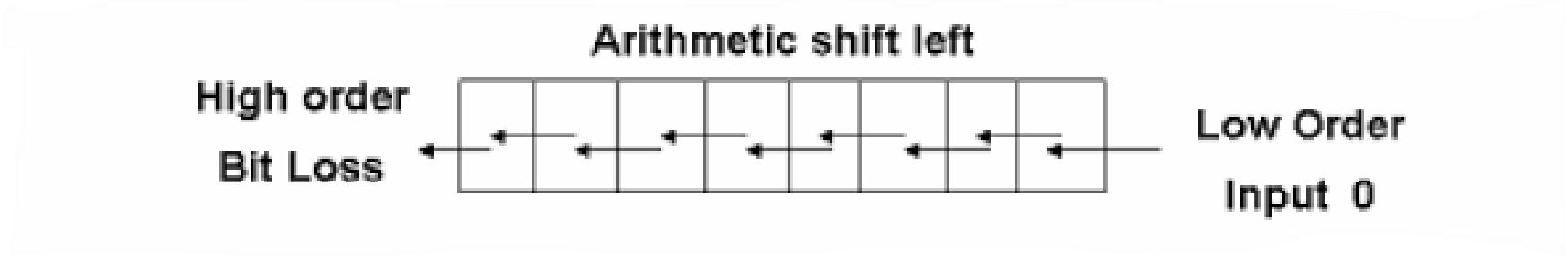
3. Arithmetic Shift:

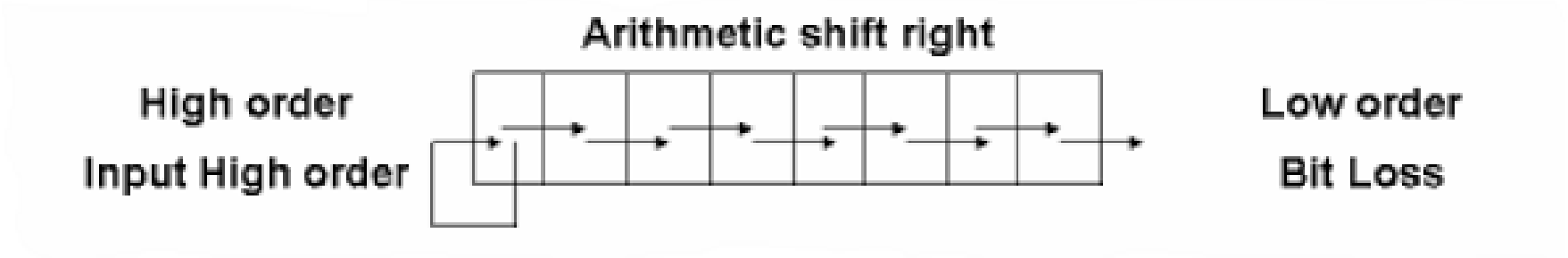
- An arithmetic shift is a micro-operation that **shifts a signed binary number to the left or right**.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.
- Arithmetic shifts **must leave the sign bit unchanged** because the sign of 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 Rn-1 in the leftmost position holds the sign bit.
- Rn-2 is the most significant bit of the number and R0 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 Rn-1 remains the same; Rn-2 receives the bit from Rn-1, and so on for the other bits in the register.
- **The bit in R0 is lost.**
- The arithmetic **shift-left inserts a 0 into R0**, and shifts all other bits to the left.
- The **initial bit of Rn-1 is lost and replaced by the bit from Rn-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.





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 clock 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

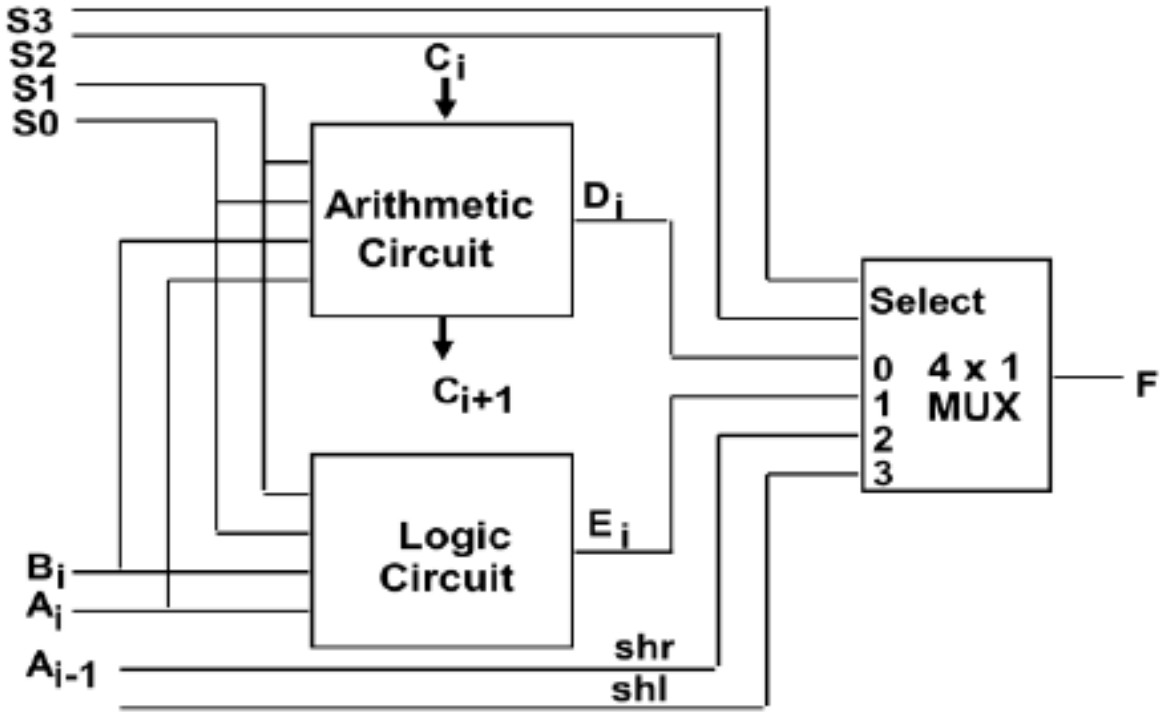


Figure 1.13: One stage of arithmetic logic shift unit

- The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units.
- A particular microoperation is selected with inputs S1 and S0.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in Di and a logic output in Ei .

- The data in the multiplexer are selected with inputs S3 and S2.
- The other two data inputs to the multiplexer receive inputs **Ai-1 for the shift-right operation** and **Ai+1 for the shift-left operation**.
- Note that the diagram shows just one typical stage. The circuit shown in figure must be repeated n times for an n-bit ALU.
- The outputs carry Ci+1 of a given arithmetic stage must be connected to the input carry Cin 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 provides
  - 8 arithmetic operation
  - 4 logic operations
  - 2 shift operations
- Each operation is selected with the **five variables S3, S2, Si, S0, and Cin**. The input **carry Cin is used for selecting an arithmetic operation only.**
- Table below lists the 14 operations of the ALU.

TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

- The **first eight are arithmetic operations** and are selected with **S3S2 = 00.**
- The **next four are logic operations** are selected with **S3S2 = 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 **S3S2 = 10 and 11.**
- The other **three selection inputs have no effect on the shift**.