



IT helloprogrammers

Note by Roshan Bist

"IT Helloprogrammers"

Chapter-2

Register transfer and Micro-operations:-

* Micro operation:

The operation performed on the data stored in register is called microoperation. It is elementary operation performed on data stored in registers. e.g add, subtract, load, store, clear, shift etc.

* Register transfer language & Register transfer:

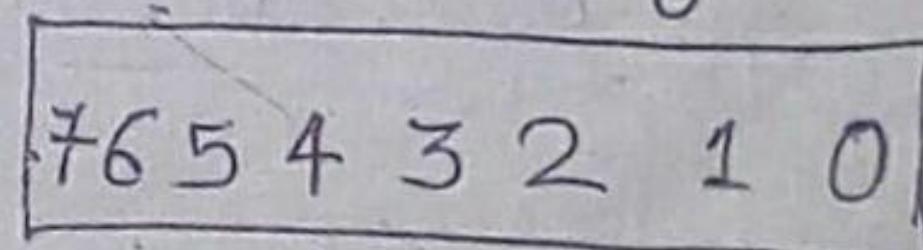
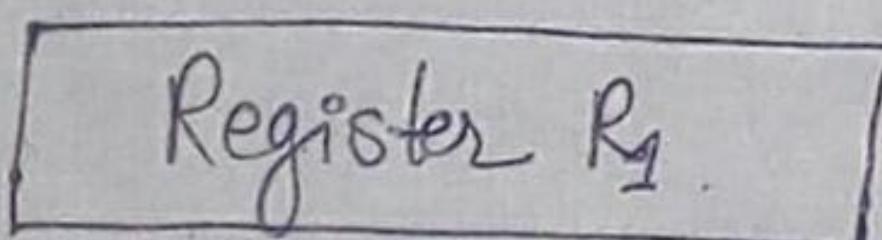
The symbolic notation used to describe the micro operation transfer among register is called a register transfer language.

e.g.

$R_1 = R_2 + R_3 \Rightarrow R_1 \leftarrow R_1 + R_3$
 microoperation . Register transfer language (RTL).

We designate (दर्शयां) computer registers by capital letters to denote its function. For example program counter register is designated by PC and instruction register by IR.

The individual flip-flops in a register are numbered in sequence from 0 to $n-1$ as shown in figure below:-



Showing Individual bits

- Showing Individual bits

 - We designate information transfer from one register to another by $R_2 \leftarrow R_1$.
 - This implies that the outputs of source must have a path to the inputs of the destination.
 - The destination register has a parallel load capability.
 - It is assumed that all transfers occur during a clock edge transmission.
 - All microoperations written on a single line are to be executed at the same time.

Basic symbols for register transfer:

The register that holds address for memory unit is called MAR.

Symbol	Description	Examples
Letters and numerals	Denotes a register	MAR, R2
Parenthesis ()	Denotes a part of 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

Control Function:

In digital system, this is done through a control signal. It is similar to "if" statement.

For example. If ($P=1$) then ($R_2 \leftarrow R_1$)

i.e, $P : R_2 \leftarrow R_1$

If $P=1$, the action takes place, otherwise no, where P is a control function that can be either 0 or 1.

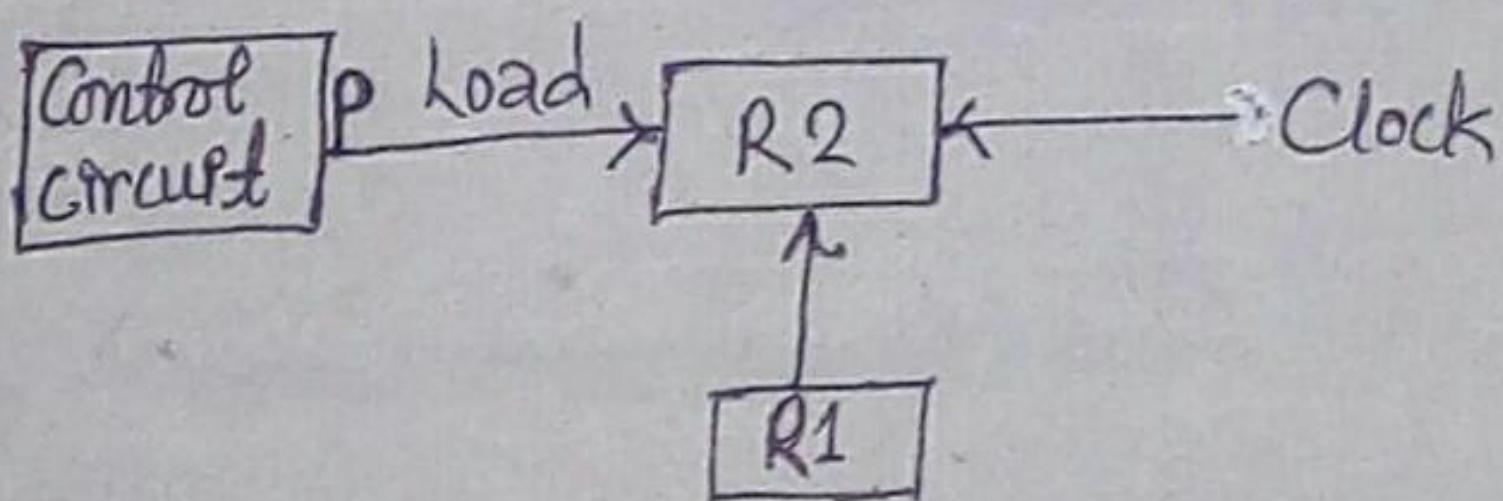


fig. Block diagram

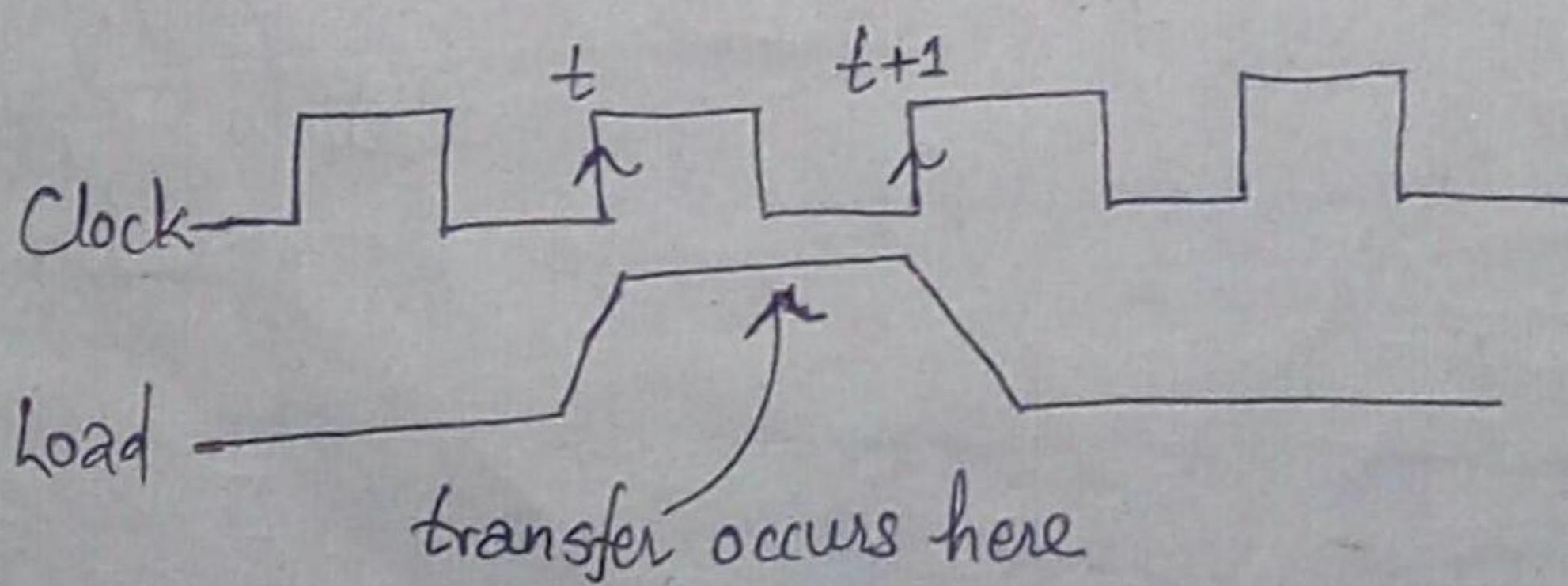
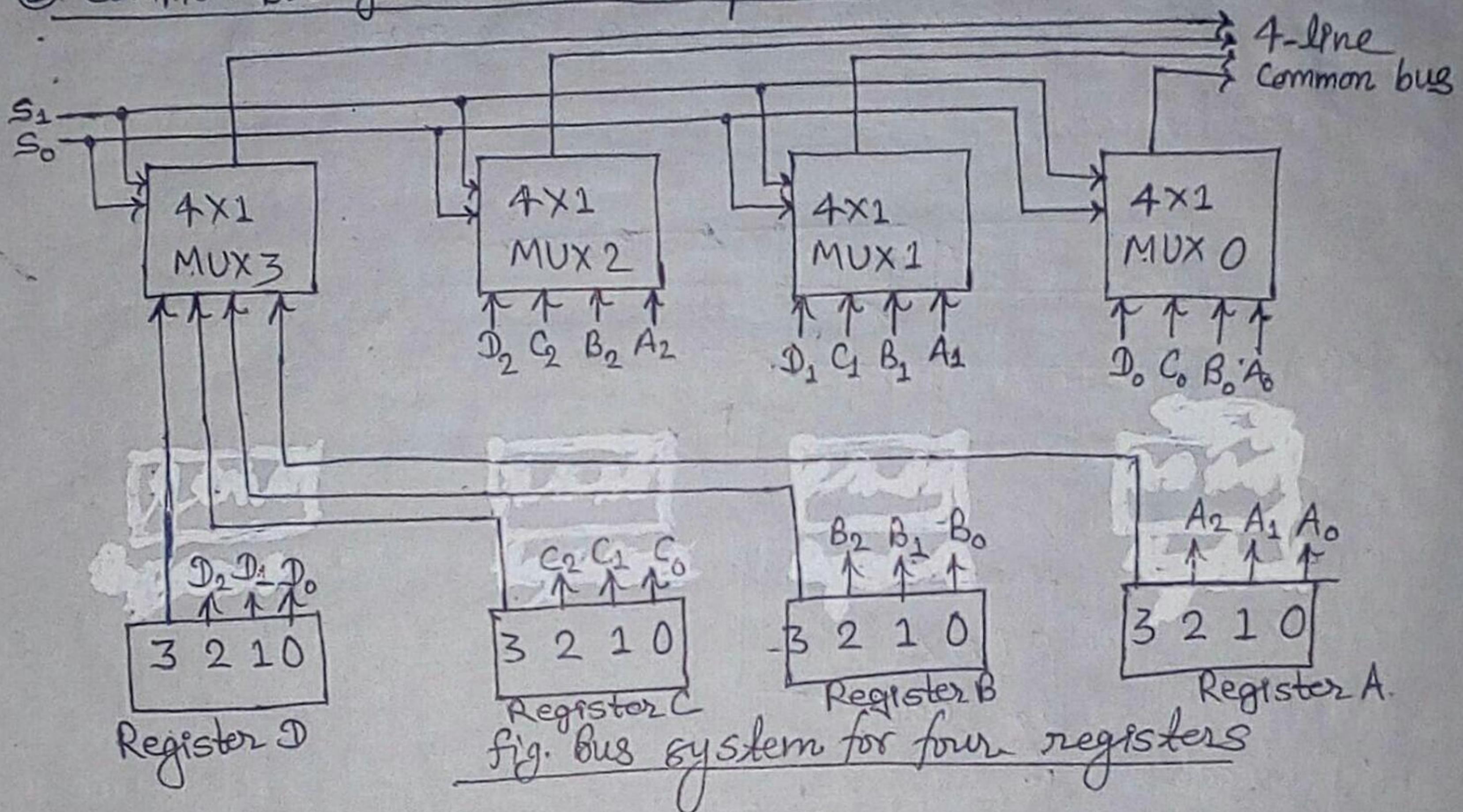


fig. timing diagram

Bus and Memory transfers:

A common bus is used rather than connecting wires between all registers. Common bus can be constructed either by using MUX or three-state buffers.

Common bus system with multiplexer



S ₁	S ₀	Register select
0	0	A
0	1	B
1	0	C
1	1	D

Function table

Multiplexers can be used to construct a common bus and it selects 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.

⑥. Common bus system with three-state buffers:-

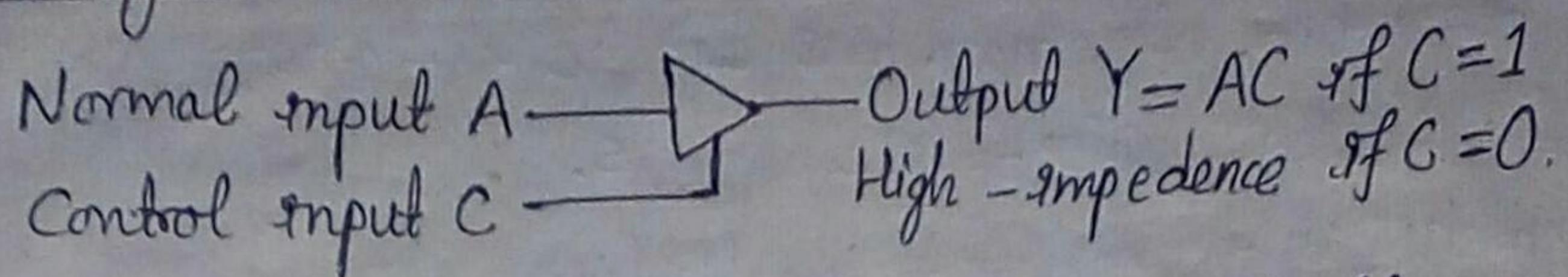


fig. Graphical symbols for three-state buffer.

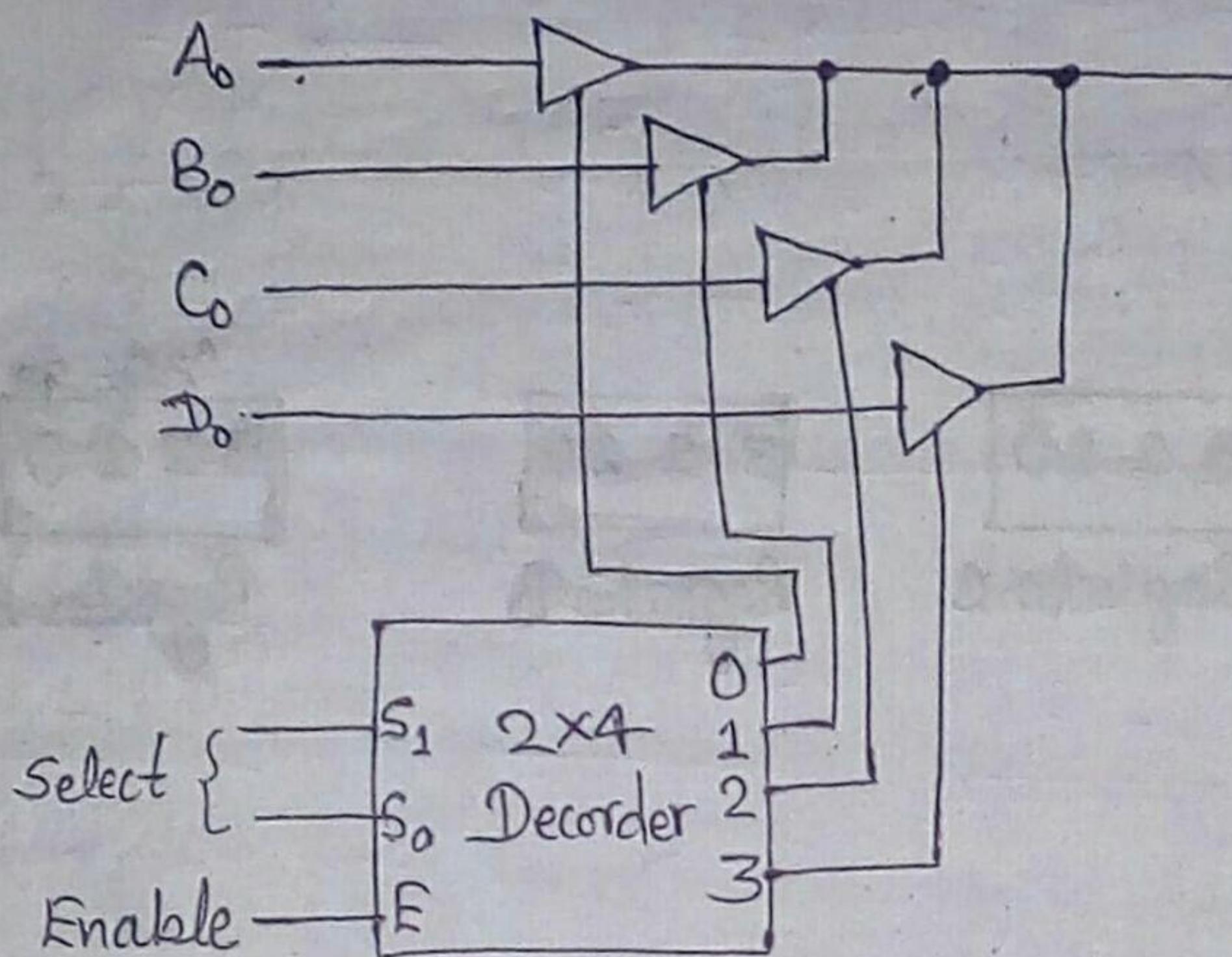


fig. Bus line with three state-buffers

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, which enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loading effects.

④ Arithmetic Microoperations:

④ List of Arithmetic Microoperations:

v) $R \leftarrow A + B$

vi) $R \leftarrow A + B + 1$

vii) $R \leftarrow A + \bar{B}$

viii) $R \leftarrow A + \bar{B} + 1$

vix) $R \leftarrow A - B$

vi) $R \leftarrow A + 1$

vi) $R \leftarrow A - 1$

Arithmetic microoperation addition

" " " addition with carry.

" " " subtraction with borrow

" " " subtraction using 2's complement.

" " " subtraction

" " " increment

" " " decrement.

④ Binary Adder:

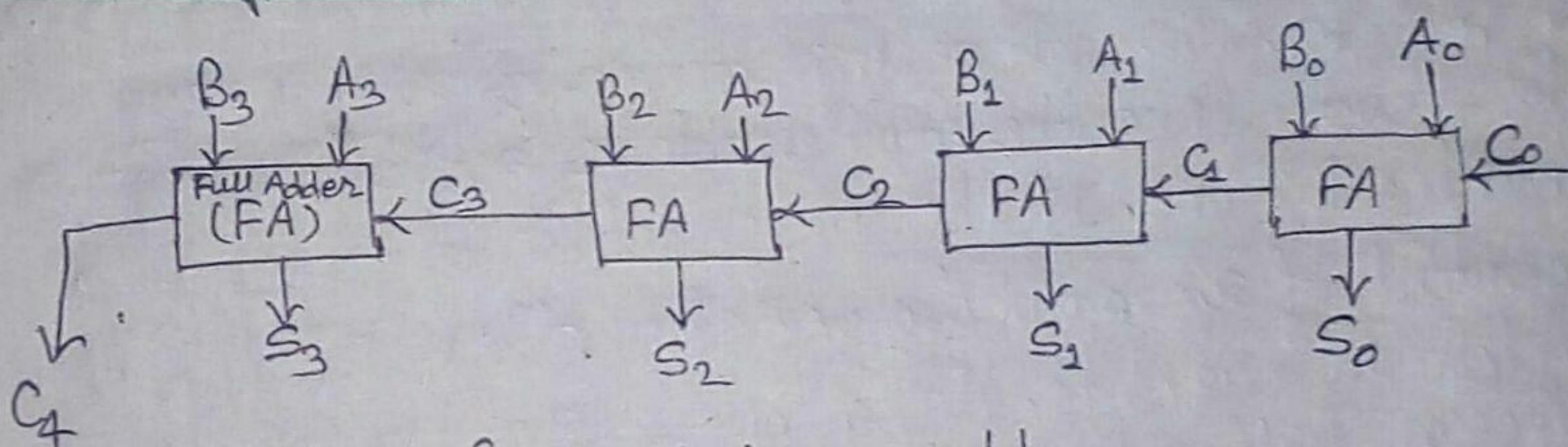


fig. 4-bit binary adder

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.

④ Binary adder-subtractor:

The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full-adder.

→ The subtraction $A - B$ can be carried out as follows:-

v) Take 1's complement of B (invert each bit)

vi) Get 2's complement by adding 1.

vii) Add result to A.

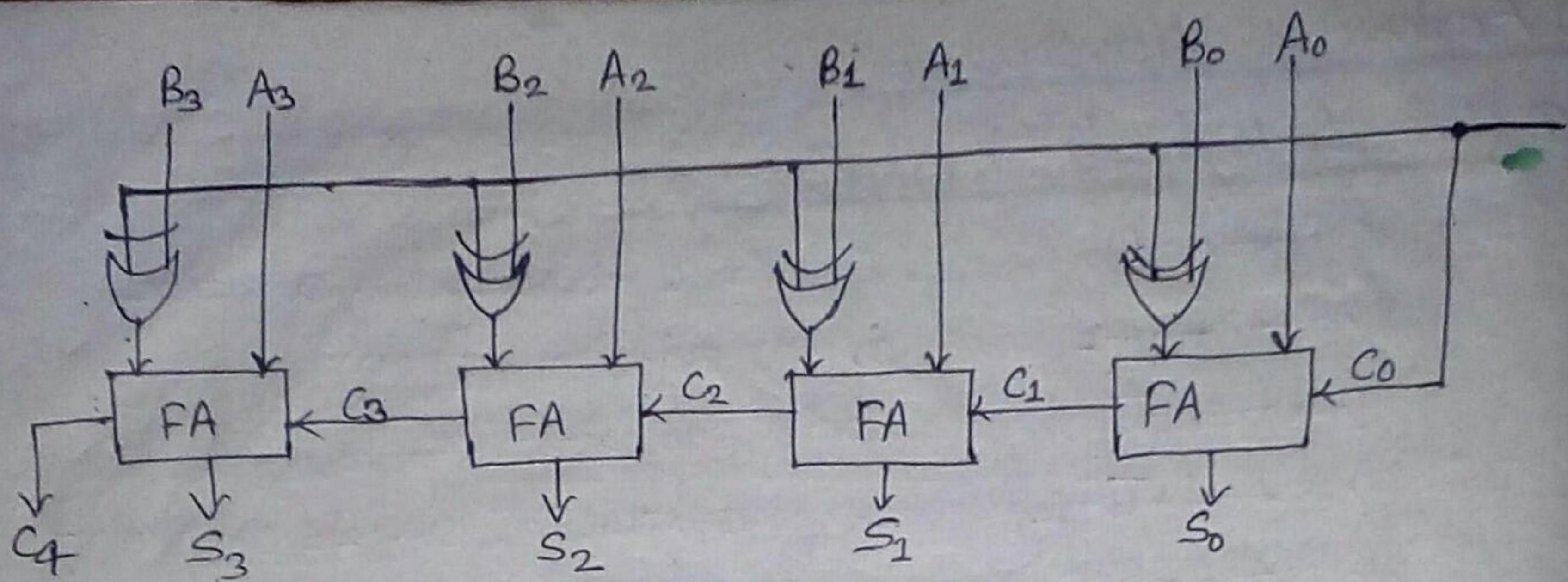
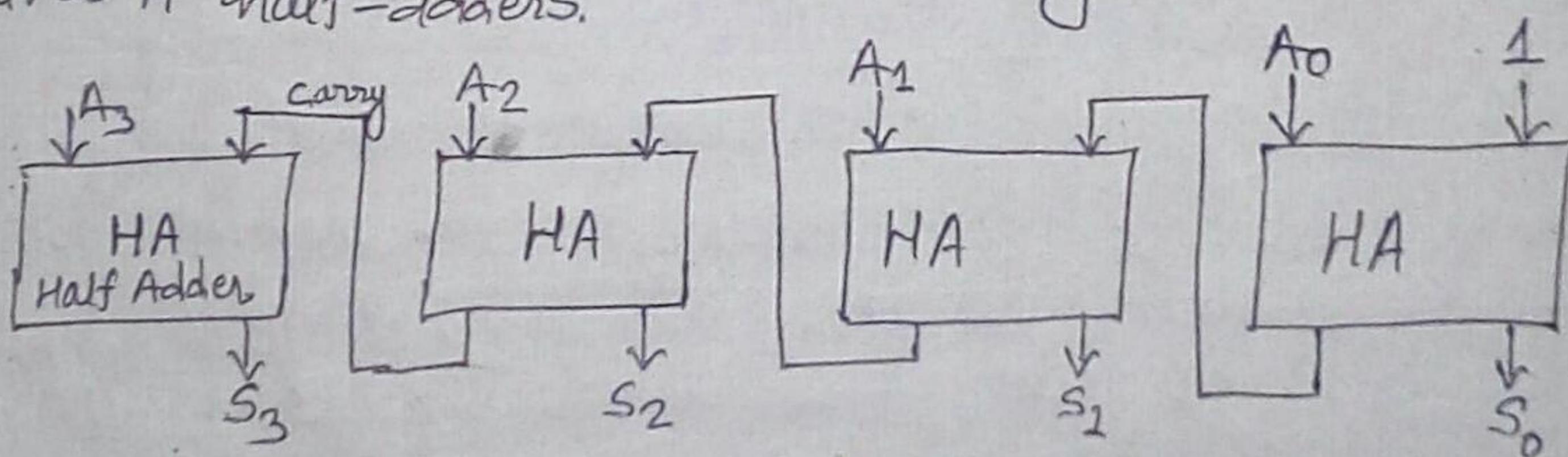


fig. 4-bit adder-subtractor

* Binary Incrementer:-

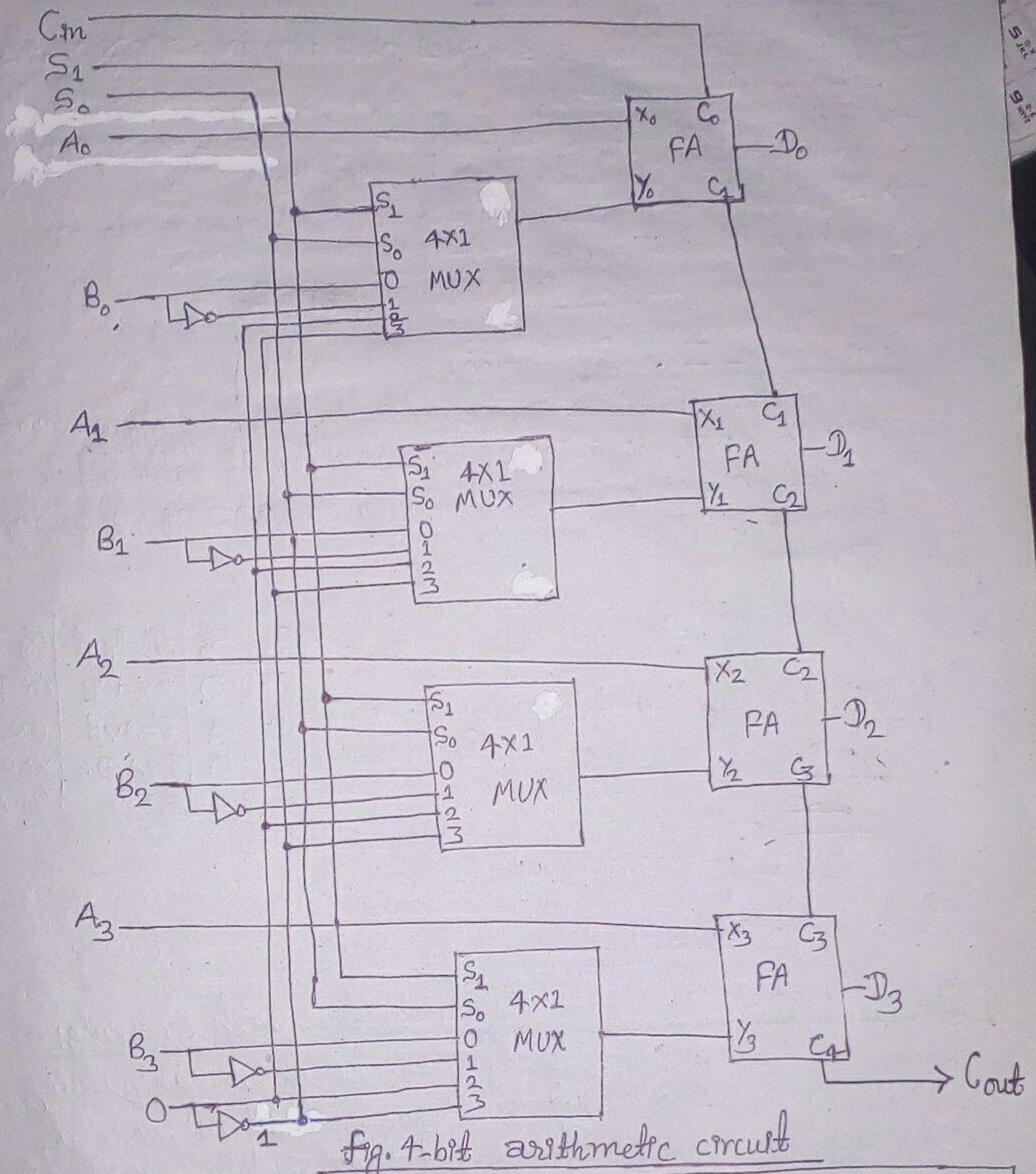
The increment microoperation adds one to a number in a register. This can be implemented by using a binary counter - every time the count enable is active, the count is incremented by one. If the increment is to be performed independent of a particular register, then use half-adders connected in cascade.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



4-bit binary incrementer

* Arithmetic Circuits:-

Each of the microoperations can be implemented in one composite arithmetic circuit. It can be constructed using full adders and multiplexers. The output of arithmetic circuit is calculated by $D = A + Y + C_{in}$. The arithmetic circuit and its function table are as follows:-



Select			Input Y	Output $D = A + Y + C_{in}$	Micro operation
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
1	0	0	B	$D = A + \bar{B}$	Subtract with borrow
1	0	1	0	$D = A + \bar{B} + 1$	Subtract
1	1	0	0	$D = A$	Transfer A
1	1	1	1	$D = A + 1$	Increment A
				$D = A - 1$	Decrement A
				$D = A$	Transfer A

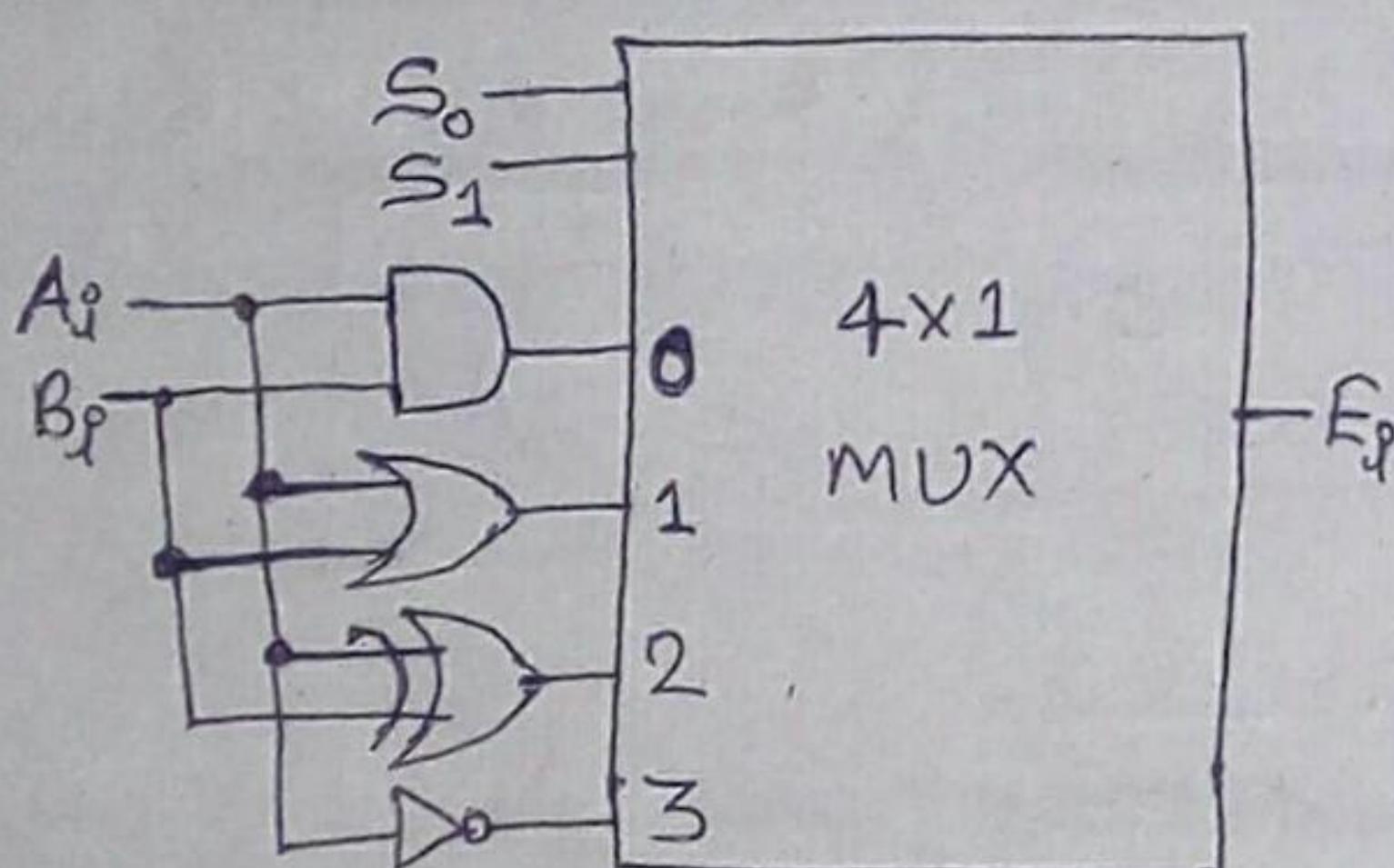
fig. Arithmetic Circuit function table.

④ Logic Microoperations:

- Logic microoperations specifies operation for strings of bits stored in registers.
- These operations consider each bit of register separately and treats them as binary variables.
- It is very useful for bit manipulation of data and for making logical decisions.
e.g. AND, OR, XOR, NOT etc.

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.



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	$E = \bar{A}$	Complement

V.V.I Applications of Logical Microoperations with example:

Logical microoperations are useful for manipulating individual bits or a portion of word stored in register. They can be used to change bit value, delete a group of bit, or insert new bit values into a register.

Applications are as follows:-

Assume: processor register (A) = 1010

operand register (B) = 1100

- a) Selective set operation → It sets to 1 bits in register A when there are corresponding 1's in register B.
- It does not affect bit position that have 0's in B.

e.g.

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

b) Selective Complement operation

→ The selective complement operation complements bits in A when there are corresponding 1's in B.

→ It also does not affect bit position that have 0's in B.

e.g.

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

c) Selective clear operation:

→ It clears to 0 the bits in A when there are corresponding 1's in B.

→ It is similar to $A \leftarrow A \wedge \bar{B}$

e.g.

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

d) Mask operations

→ It is similar to the selective clear operation except that the bits of A are cleared only when there are corresponding 0's in B.

→ It is AND operation.

e.g.

1010	A (before)
1100	B (logic operand)
1000	A (after)

e) Insert operation → It inserts new value into a group of bits.

→ It is done first by masking and then ORing with required value.

→ It is OR operation.

e.g. Consider an example of inserting 1010 in place of 0110

Moving bits to be replaced
Original bits to be inserted

Mask

0110	1010 - A (before)
0000	1111 - B (Mask)
0000	1010 A (after masking)

ORing

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

f) Clear operation

→ It compares the values of A and B and produces all 0's if A and B are equal or same.

→ It is similar to XOR microoperation.

e.g.
$$\begin{array}{r} 1010 \text{ A (before)} \\ 1010 \text{ B (logic operand)} \\ \hline 0000 \text{ A} \leftarrow \text{A} \oplus \text{B} \text{ (A cleared).} \end{array}$$

④ Shift Microoperations:

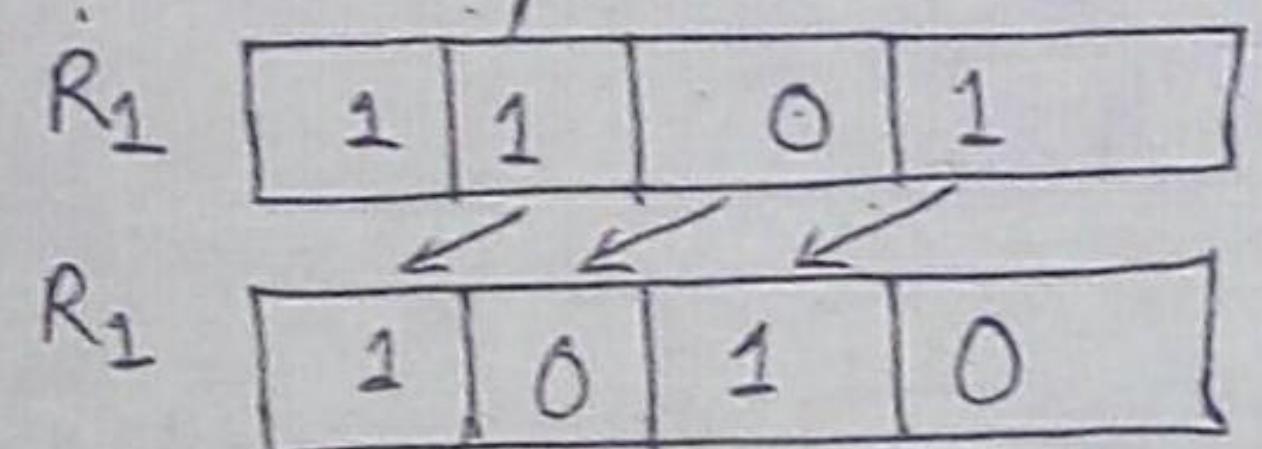
→ Used for transfer of data.

→ Used in conjunction with arithmetic, logic and other data processing operations.

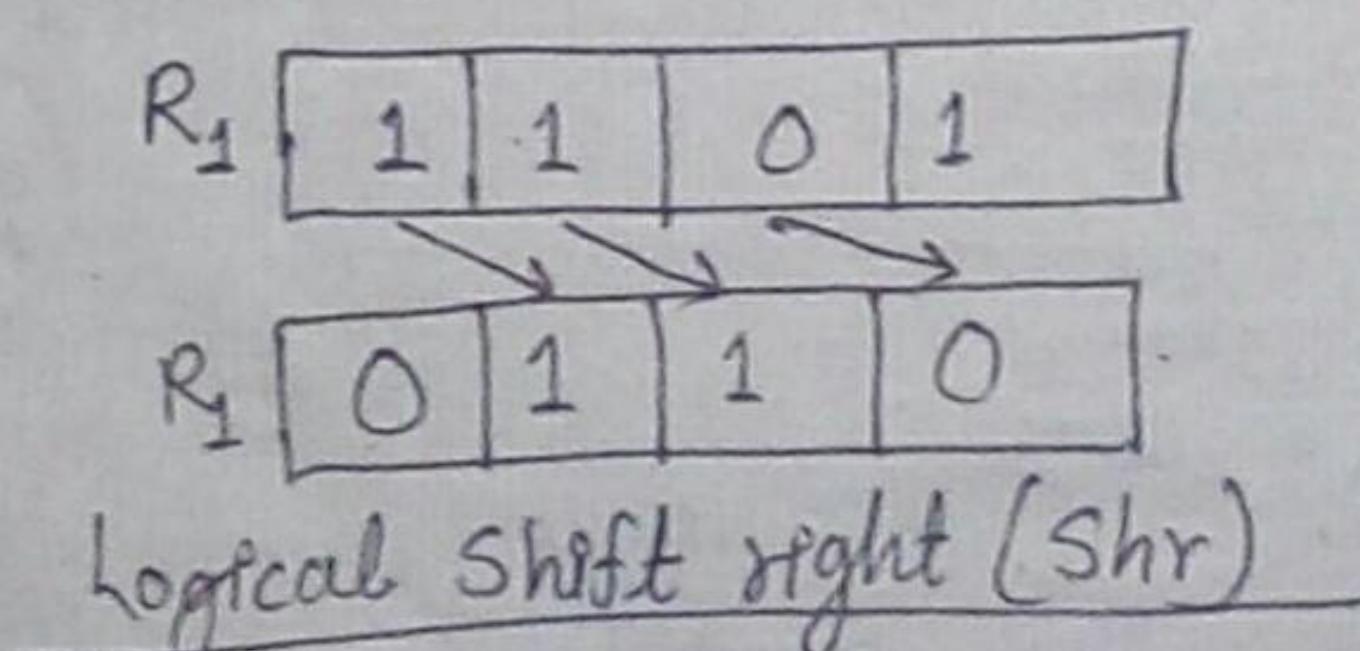
→ The content of a register can be shifted to the left or right.

Types:

a) Logical shift → A logical shift is one that transfers 0 through the serial input.

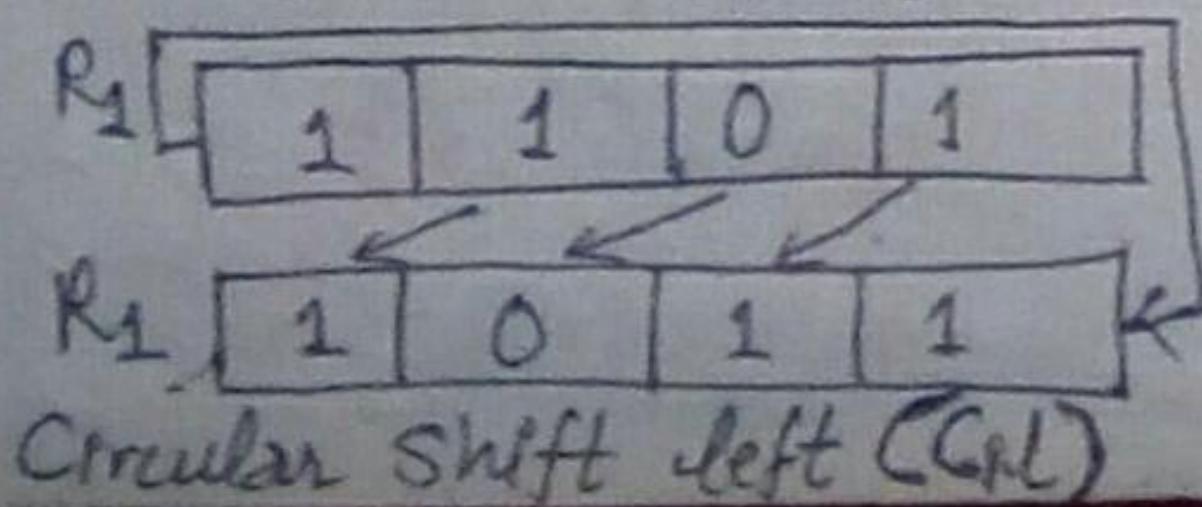


Logical Shift Left (Lsh)

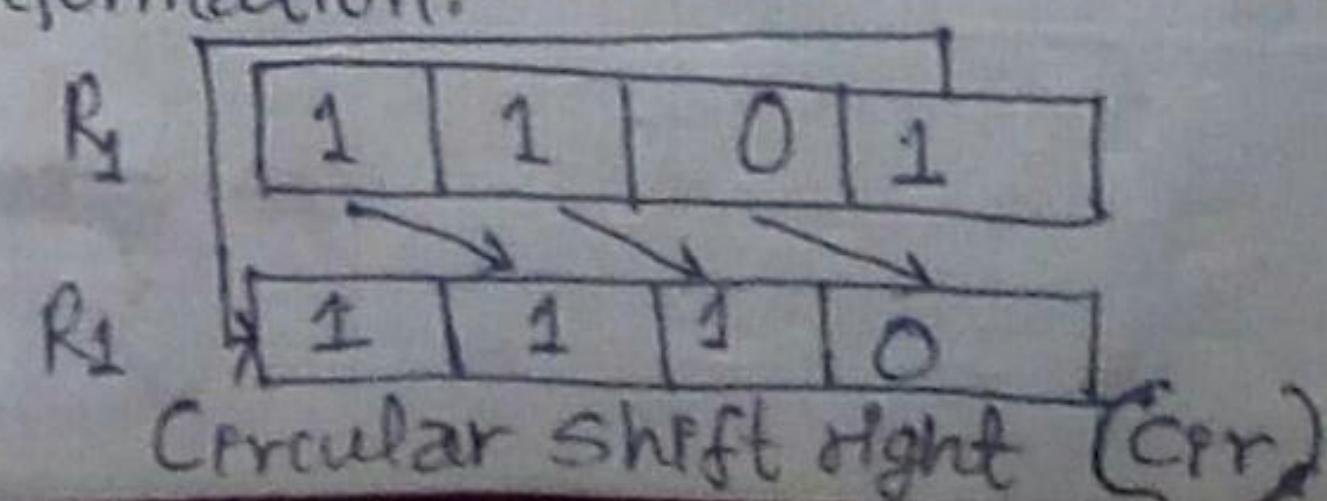


Logical Shift Right (Lsh)

b) Circular shift → It is also known as rotate operation which circulates the bits of the registers around the two ends without loss of information.



Circular Shift Left (Crl)



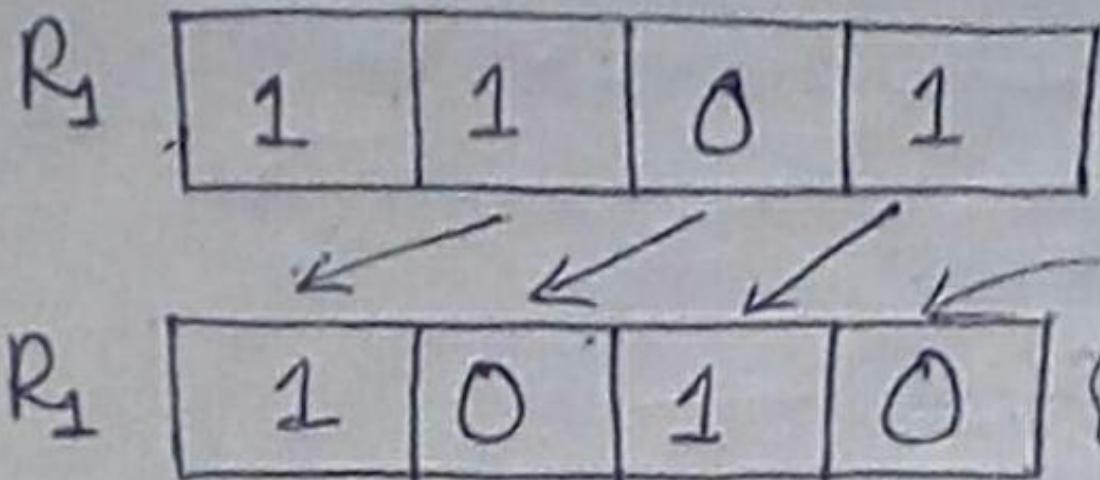
Circular Shift Right (Crr)

c) Arithmetic Shift

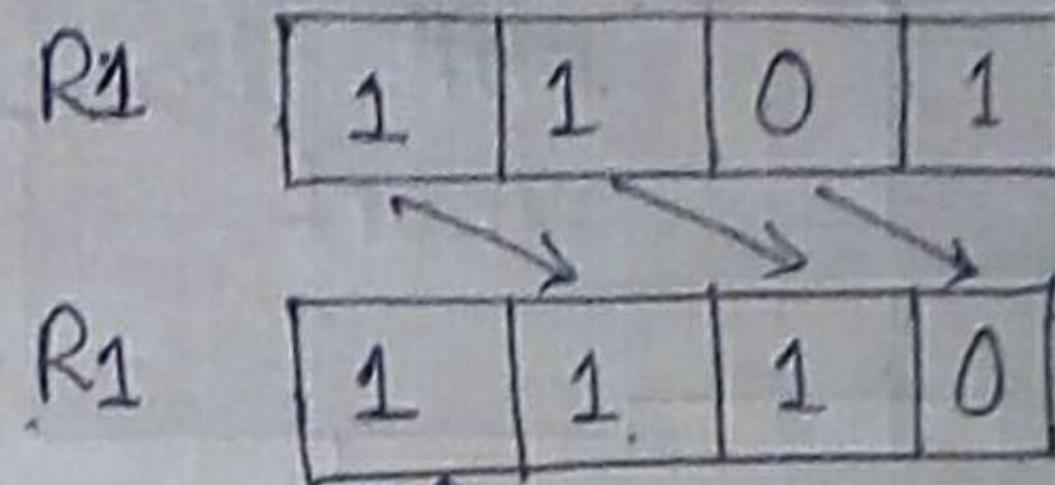
→ It is microoperation that shifts a signed binary number to the left or right.

→ Arithmetic shift left multiplies a signed binary no. by 2.

→ Arithmetic shift right divides a signed binary no. by 2.

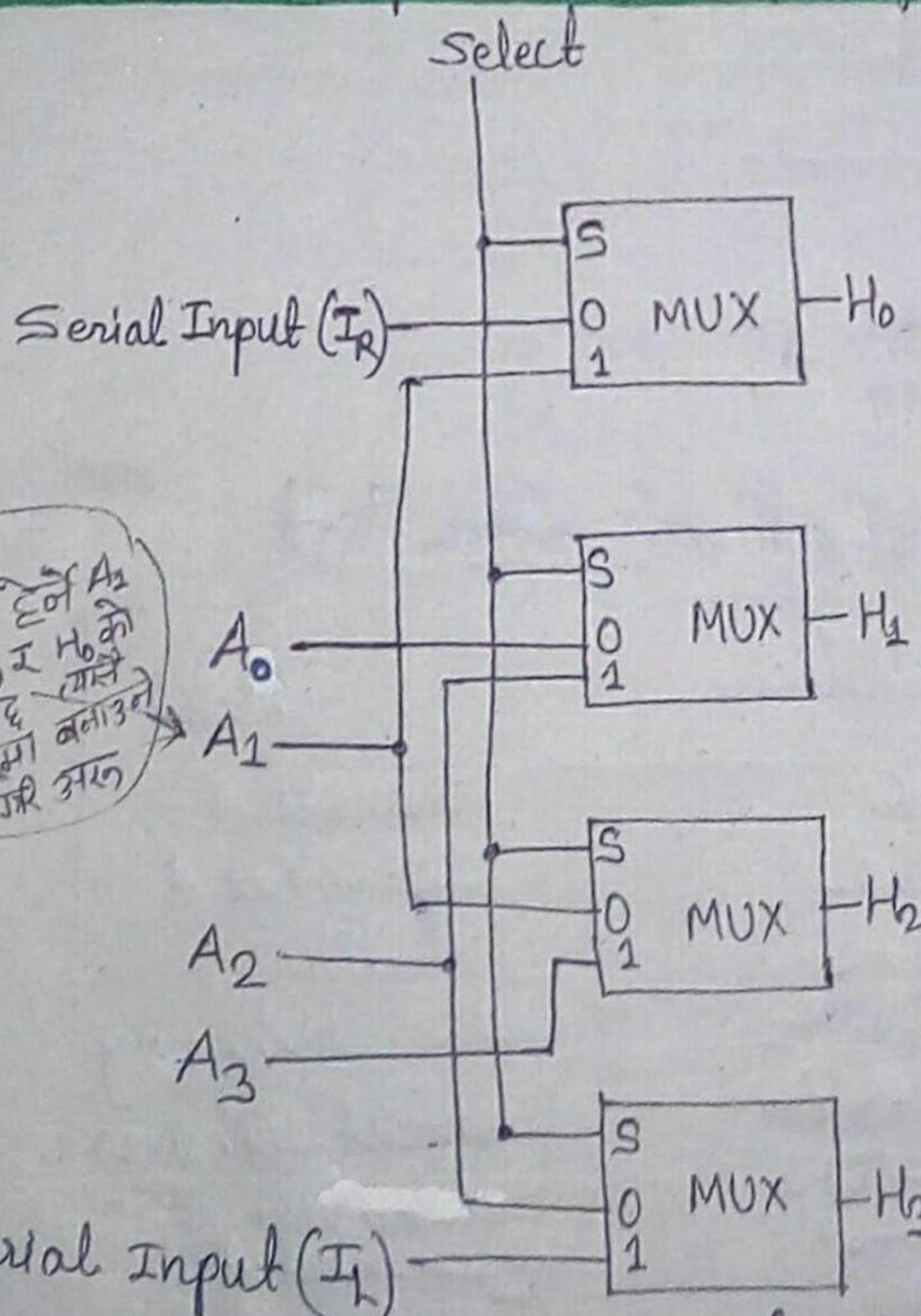


arithmetic shift left (ashl)



arithmetic shift right (ashr)

④. Hardware implementation of shift microoperation:



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

पहला टेबल लिया जाता है अब को अपरिवर्तित I_R, A₀...
A₃, I_L लिया जाता है इसके लिए 50%

fig. 4-bit combinational circuit shifter

* Arithmetic Logic Shift Unit:

Hardware Implementation

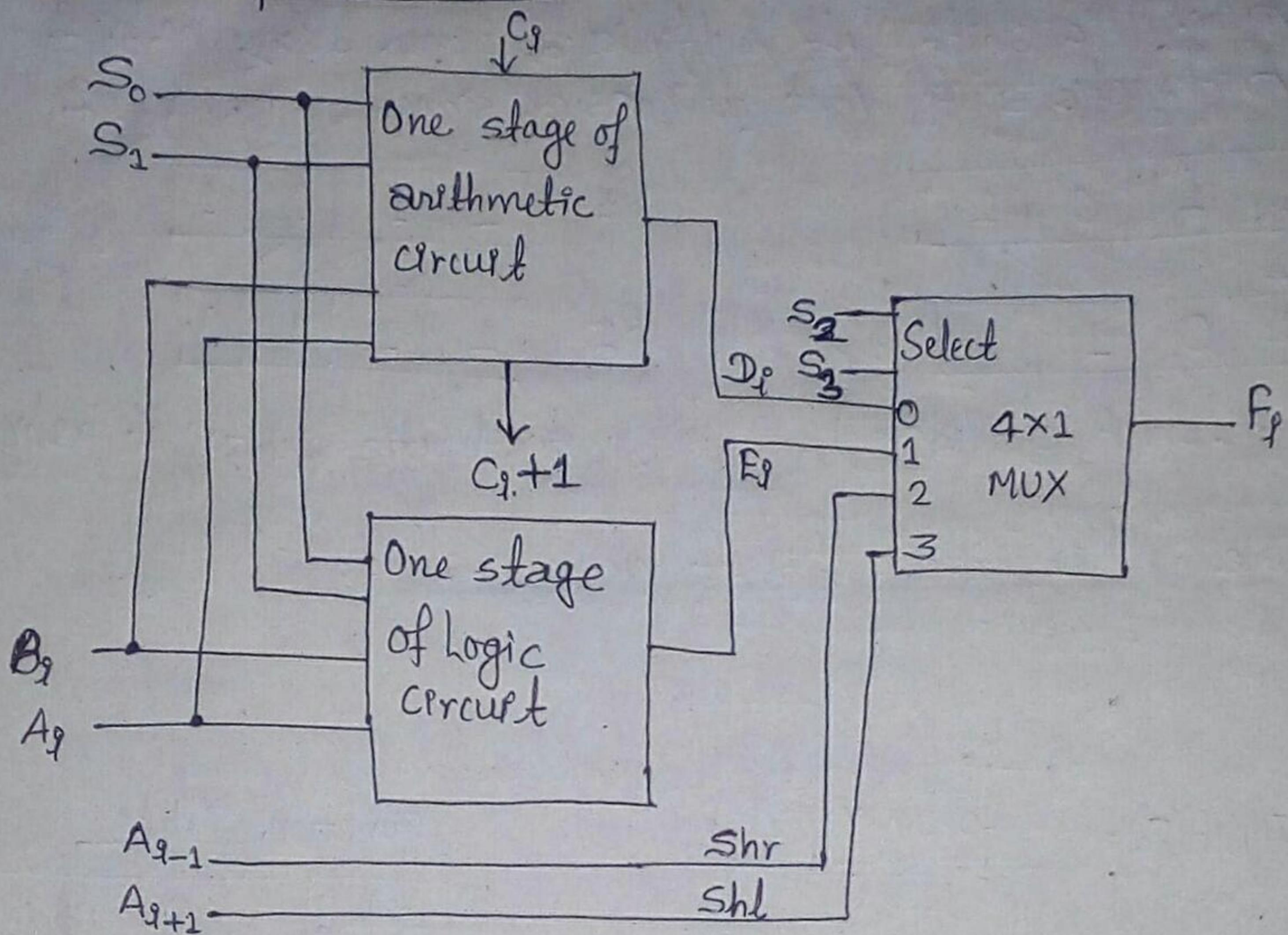


fig. One stage of arithmetic-logic shift

Operation Select						Operation	Function
S ₃	S ₂	S ₁	S ₀	C _m			
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 + B̄	Subtract with borrow
0	0	1	0	1		F = A + B̄ + 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 = Ā	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: Function table for arithmetic logic shift unit.

Q. The 8-bit registers AR, BR, CR and DR initially have the following values.

$$AR = 11110010$$

$$BR = 11111111$$

$$CR = 10111001$$

$$DR = 11101010$$

Determine the 8-bit values in each register after execution of the following sequence of micro-operation.

$$AR \leftarrow AR + BR$$

$$CR \leftarrow CR \wedge DR$$

$$BR \leftarrow BR + 1$$

$$AR \leftarrow AR - CR$$

Solution

For $AR \leftarrow AR + BR$

$$AR : 11110010$$

$$BR : +11111111$$

$$\begin{array}{r} \text{carry} \\ \hline AR \Rightarrow 11110001 \end{array}$$

$$AR \Rightarrow 11110001$$

For $CR \leftarrow CR \wedge DR$

$$CR : 10111001$$

$$DR : 11101010$$

$$\hline 10101000$$

$$CR \Rightarrow 10101000$$

For $BR \leftarrow BR + 1$

$$BR : 11111111$$

$$+ 1$$

$$\begin{array}{r} \text{carry} \\ \hline 10000000 \end{array}$$

$$BR \Rightarrow 00000000$$

For $AR \leftarrow AR - CR$

$$AR : 11110010$$

$$CR + 1 : 01011000$$

$$\begin{array}{r} \text{carry} \\ \hline 01001001 \end{array}$$

$$AR \Rightarrow 01001001$$

$$AR : 11110010$$

$$CR - 01011000$$

$$\hline AR \Rightarrow 00111001$$

