Scanned by TapScanner

Chapter-2

Register transfer and Micro-operations:-

⊛. Microoperation:

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 microoperation transfer among register is called a register transfer language.

e.g.

$R_1 = R_2 + R_3$ ⇒ $R_1 \leftarrow R_1 + R_3$

microoperation. Register transfer language (RTL).

We designate (हुट्याउनु) 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 an register are numbered in sequence from 0 to n-1 as shown in figure below:-

Register R_1

7	6	5	4	3	2	1	0

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.

Scanned by TapScanner

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 ←	Denotes transfer of information	R2 ← R1
Comma ,	Seperates two microoperations	R2←R1, R1←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.

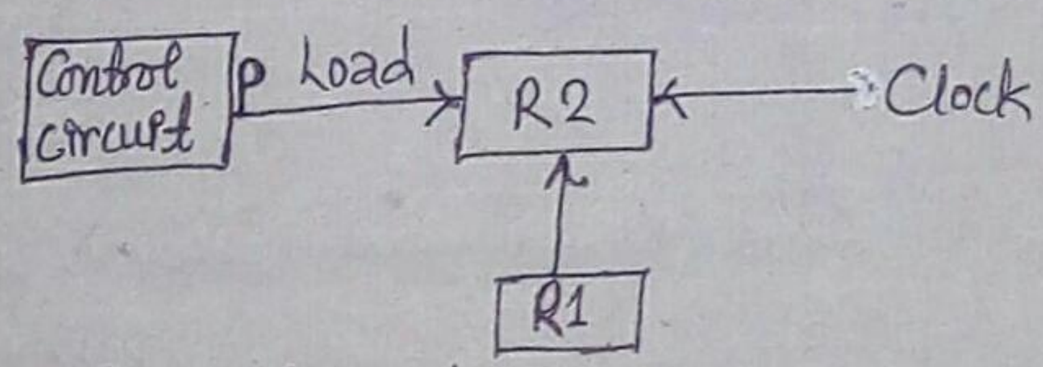


fig. Block diagram

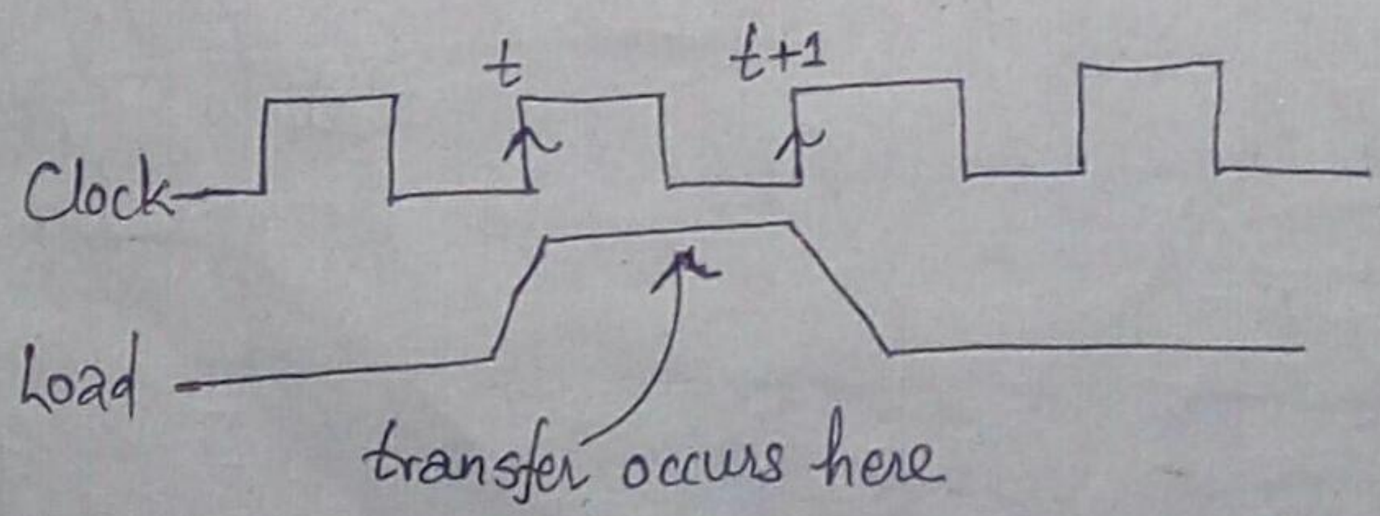


fig. timing diagram

Bus and Memory transfers:

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

① Common bus system with multiplexer

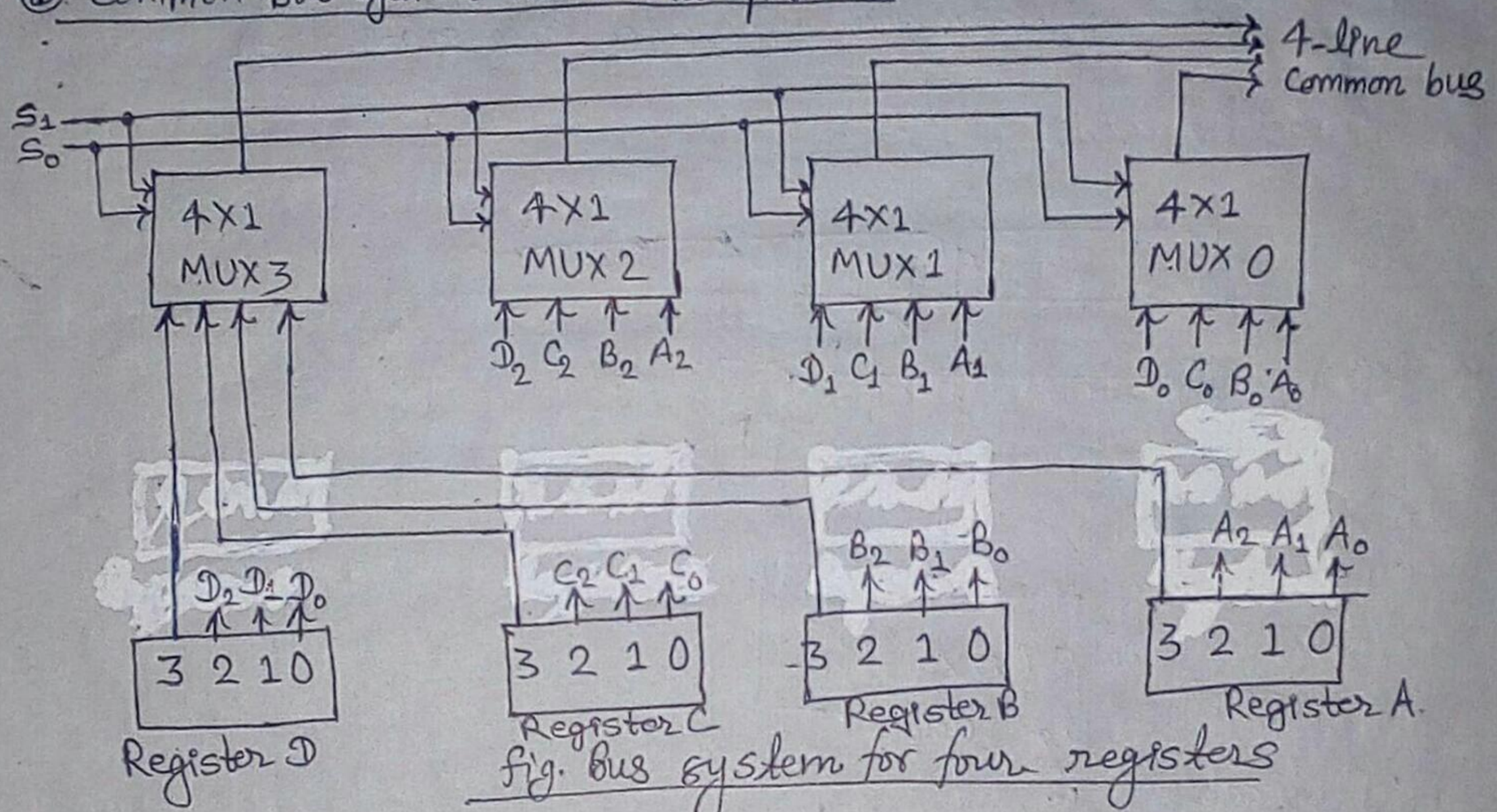


fig. Bus system for four registers

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.

(b). Common bus system with three-state buffers:-

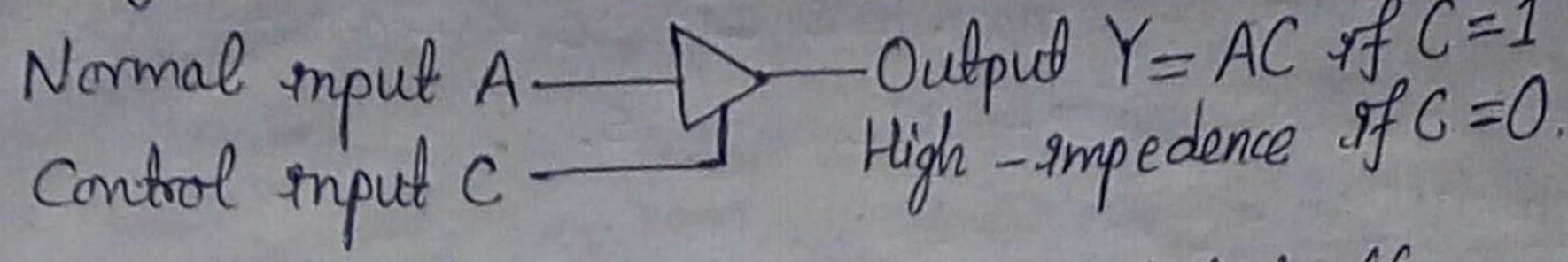


fig. Graphical symbols for three-state buffer.

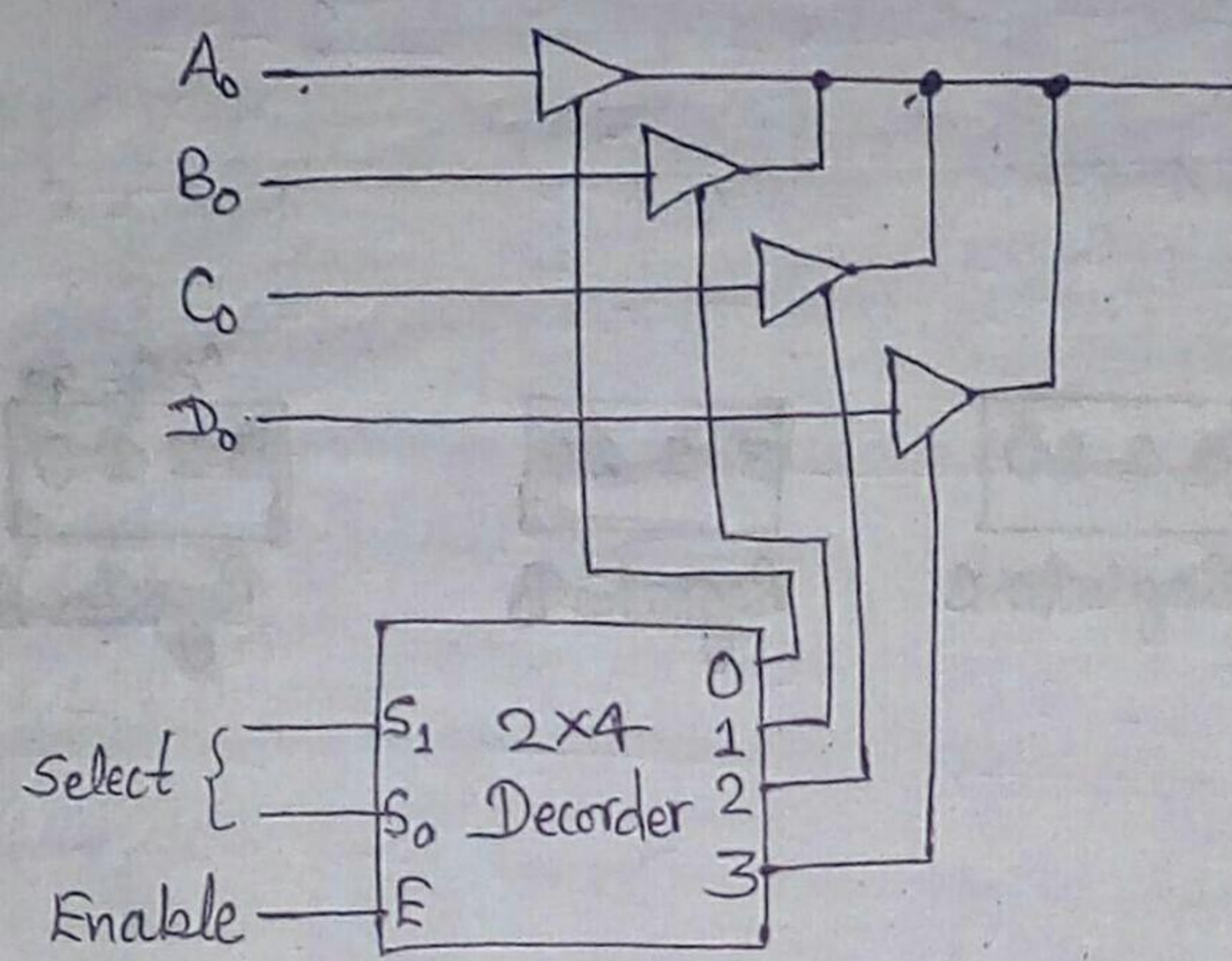


fig. Bus line with three state-buffers

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.

Scanned by TapScanner

⊛ Arithmetic Microoperations:

⊛ List of Arithmetic Microoperations:

i) $R \leftarrow A+B$ — Arithmetic microoperation addition

ii) $R \leftarrow A+B+1$ — " " addition with carry

iii) $R \leftarrow A+\overline{B}$ — " " subtraction with borrow

iv) $R \leftarrow A+\overline{B}+1$ — " " subtraction using 2's complement.

v) $R \leftarrow A-B$ — " " subtraction

vi) $R \leftarrow A+1$ — " " increment

vii) $R \leftarrow A-1$ — " " decrement.

⊛ Binary Adder:

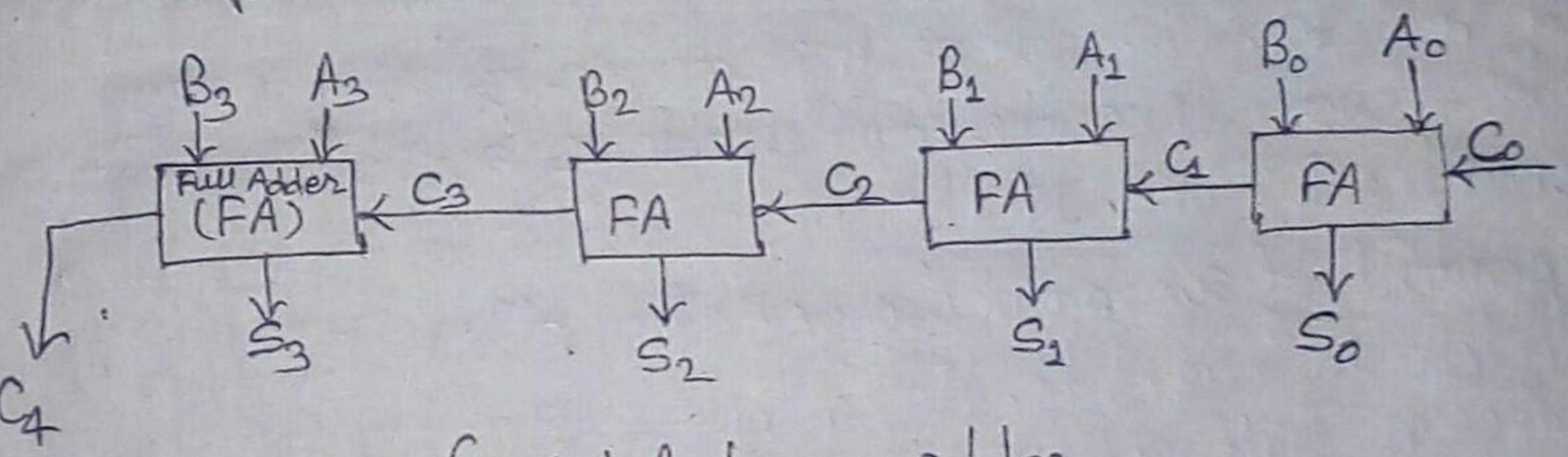


fig. 4-bit binary adder

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.

⊛ 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:-

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

ii) Get 2's complement by adding 1.

iii) Add result to A.

Scanned by TapScanner

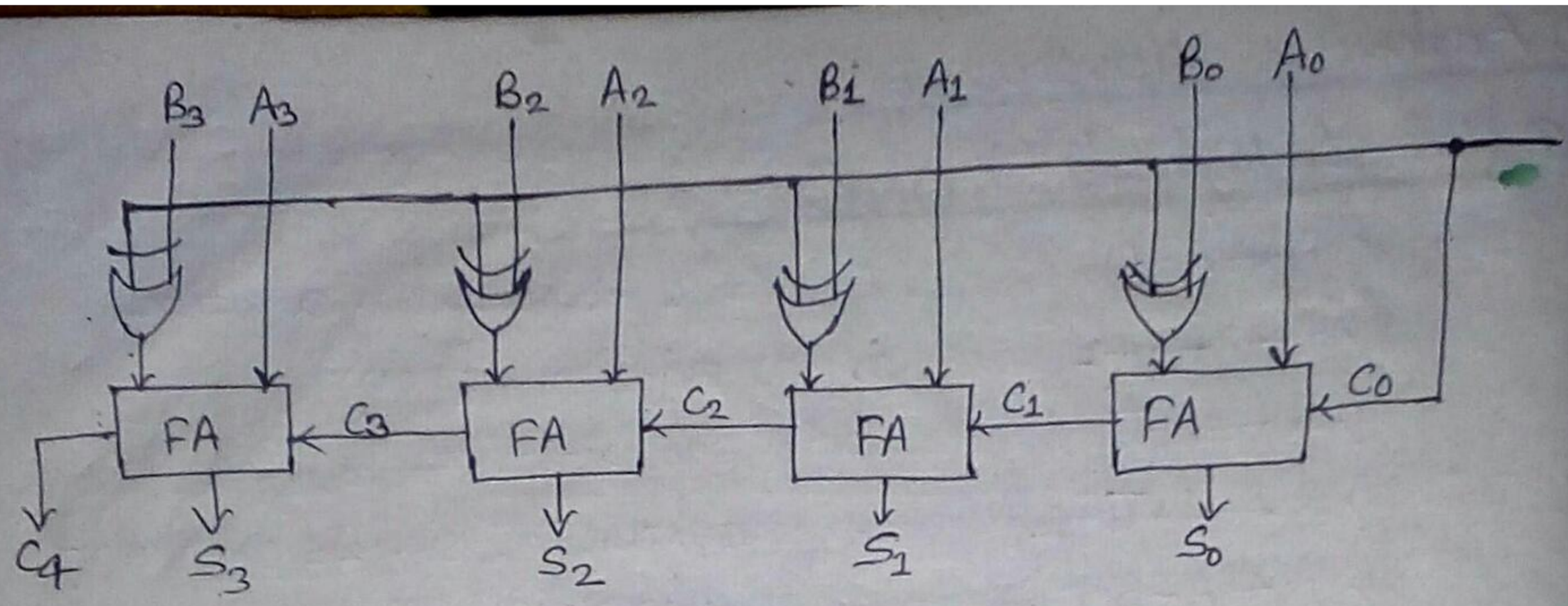
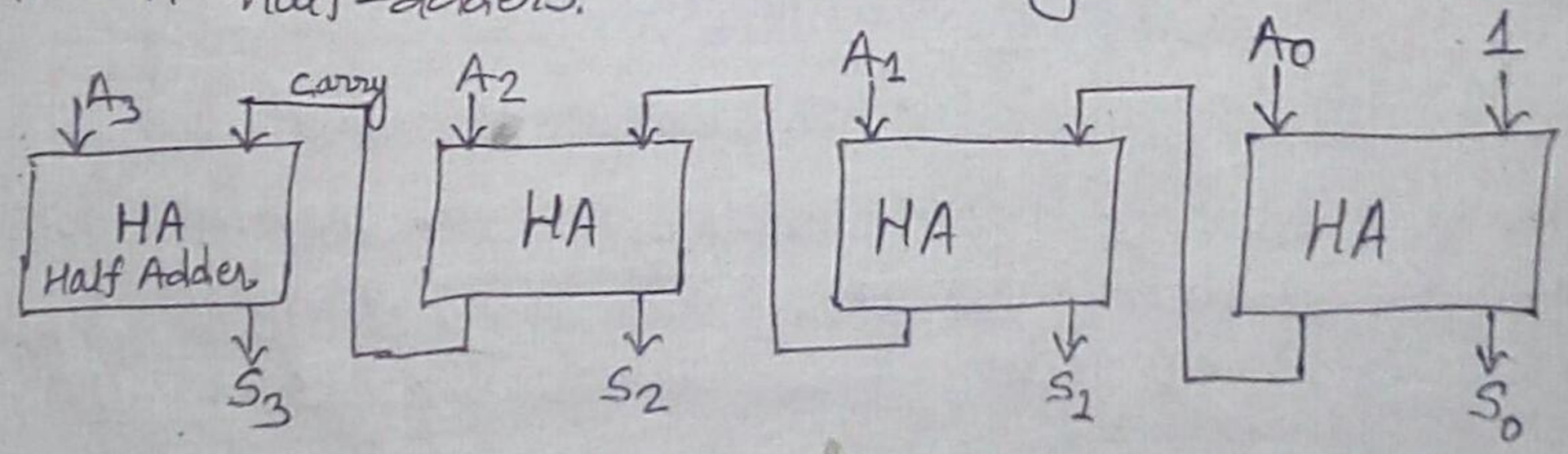


fig. 4-bit adder-subtractor

⊛ 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 Circuit:-

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:-

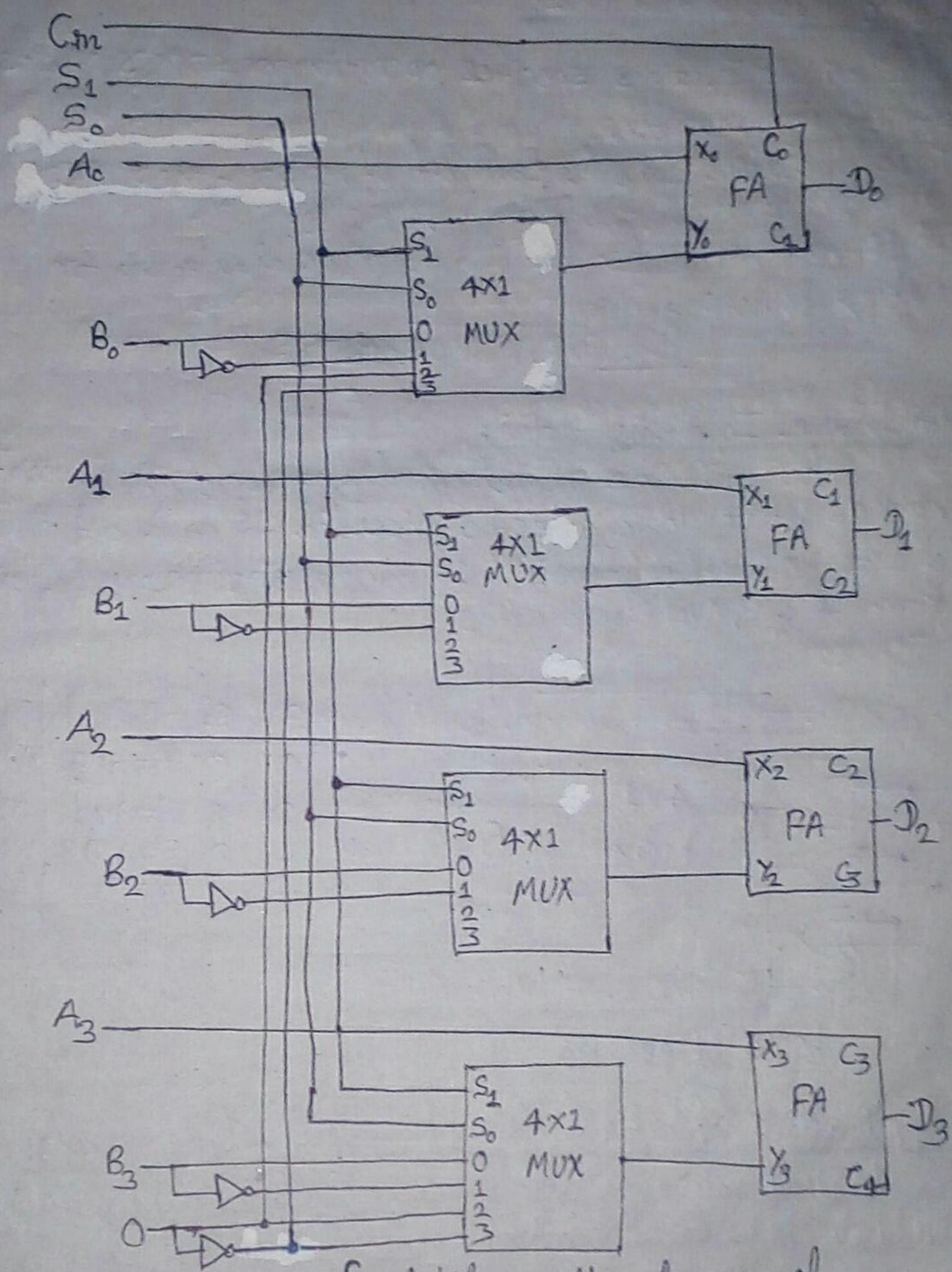


fig. 4-bit arithmetic circuit

Select			Input Y	Output $D = A + Y + C_{in}$	Microoperation
S_1	S_0	C_{in}			
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	B	$D = A + \overline{B}$	Subtract with borrow
0	1	1	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

fig. Arithmetic Circuit function table.

⊛. Logic Microoperations:

→ Logic microoperations specifies operation for strings of bits stored in registers.

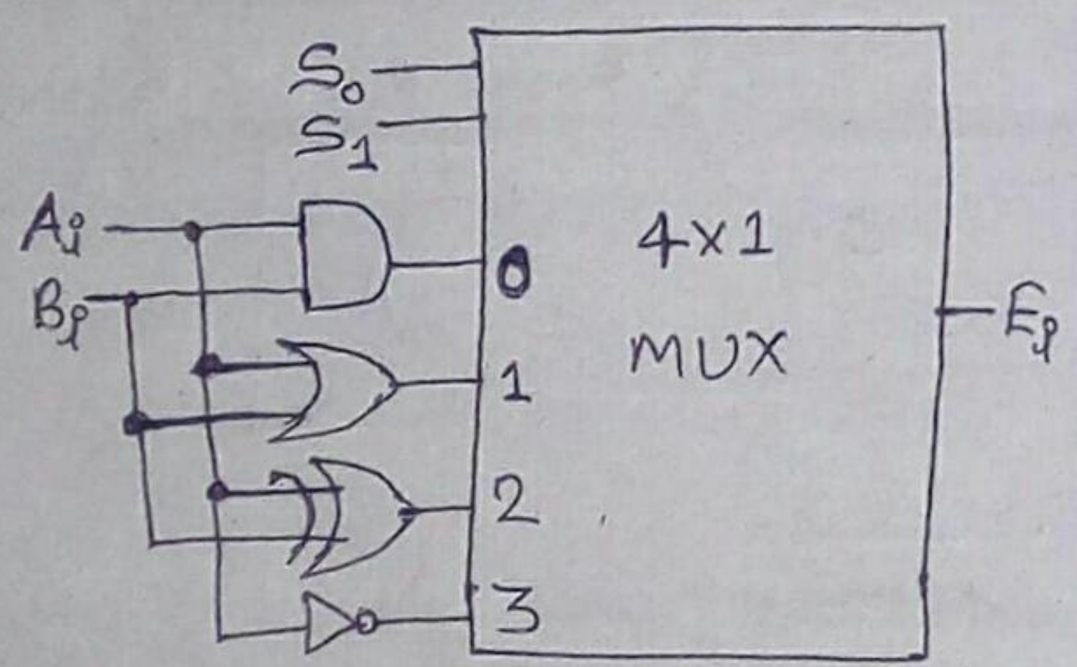
→ These operation consides each bit of register seperately and treats them as binary variables.

→ It is very useful for bit manipulation of data and for making logical decision.

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.



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	$E = \bar{A}$	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 0s 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 $A \leftarrow A \wedge \overline{B}$

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 1001 in place of 0110

Mask			ORing		
0110	1001	A (before)	0000	1010	A (before)
0000	1111	B (Mask)	1001	0000	B (insert)
0000	1010	A (after masking)	1001	1010	A (after insertion)

Scanned by TapScanner

f) Clear operation

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

→ It is similar to XOR microoperation.

e.g.
```
1010  A (before)
1010  B (logic operand)
----------
0000  A ← A⊕B (A cleared).
```

⊛ Shift Microoperations:

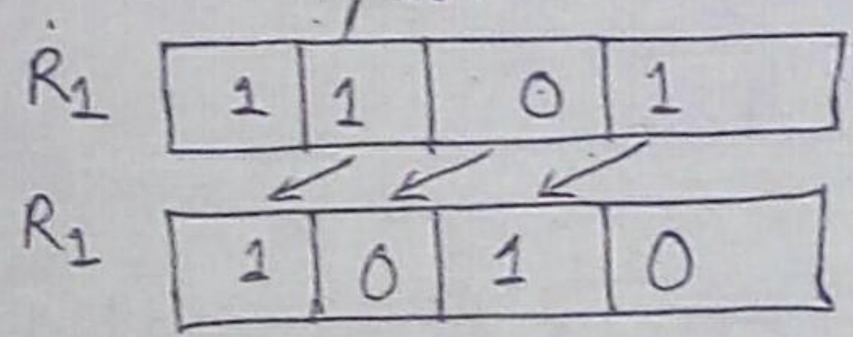
→ Used for transfer of data.

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

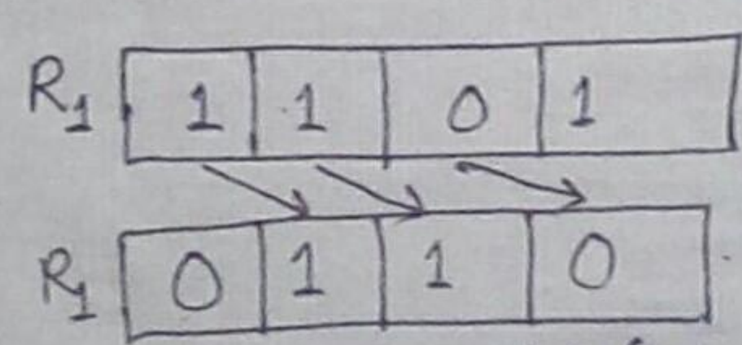
→ The content of a register can be shifted to the left or right.

Types:

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

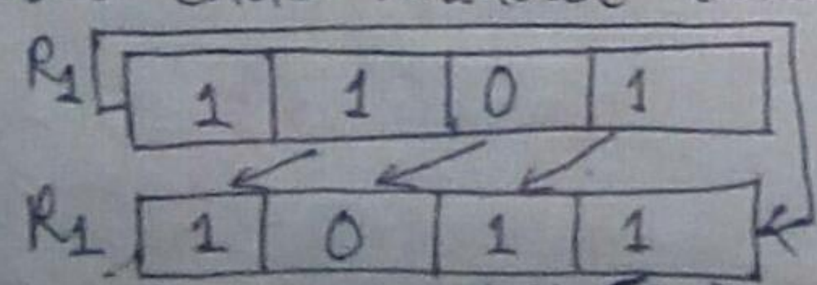


Logical shift left (Shl)

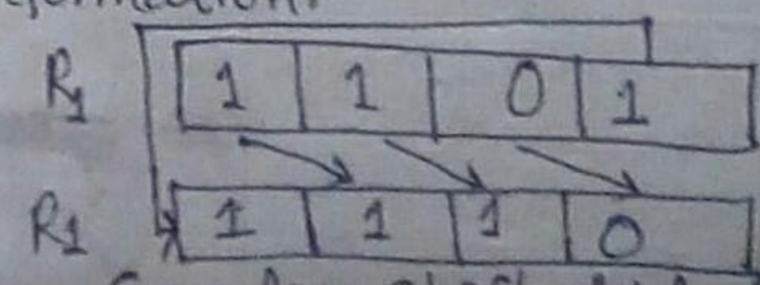


Logical shift right (Shr)

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 (Cil)



Circular shift right (Cir)

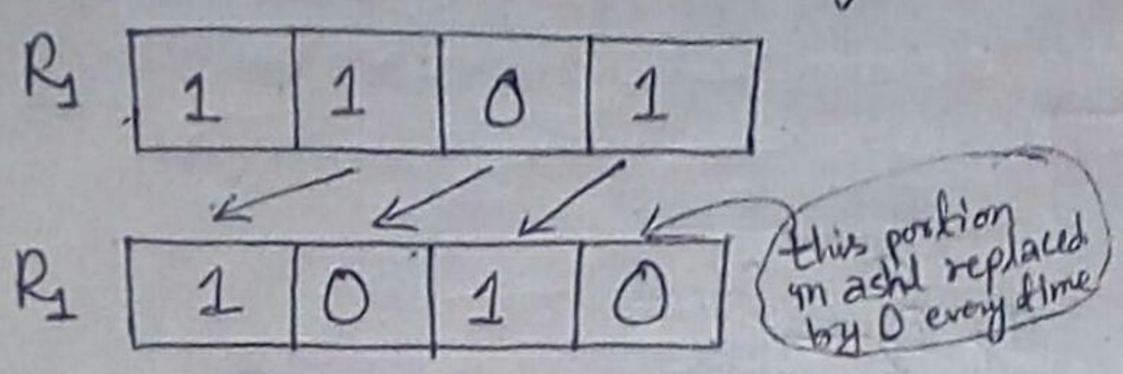
Scanned by TapScanner

c) Arithmetic Shift

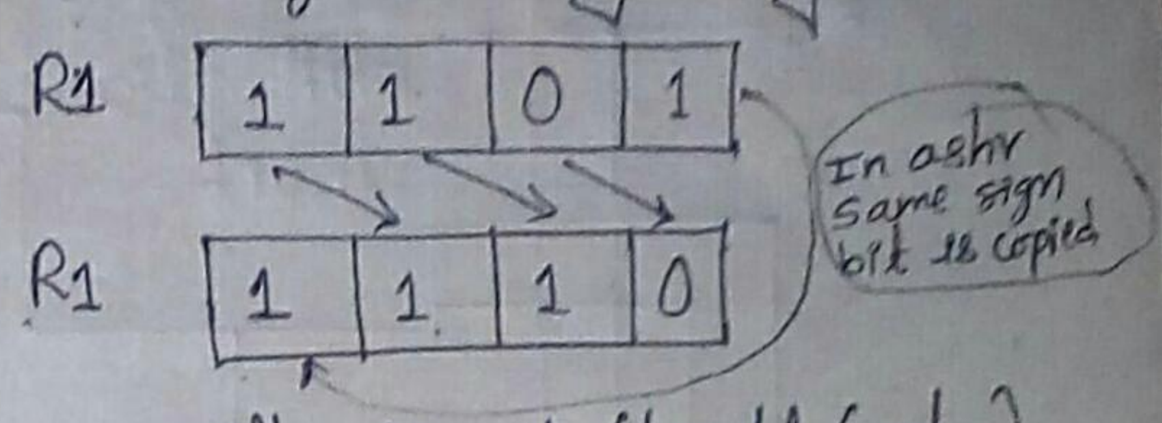
→ It is microoperation that shifts a signed binary number to the left or right.

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

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

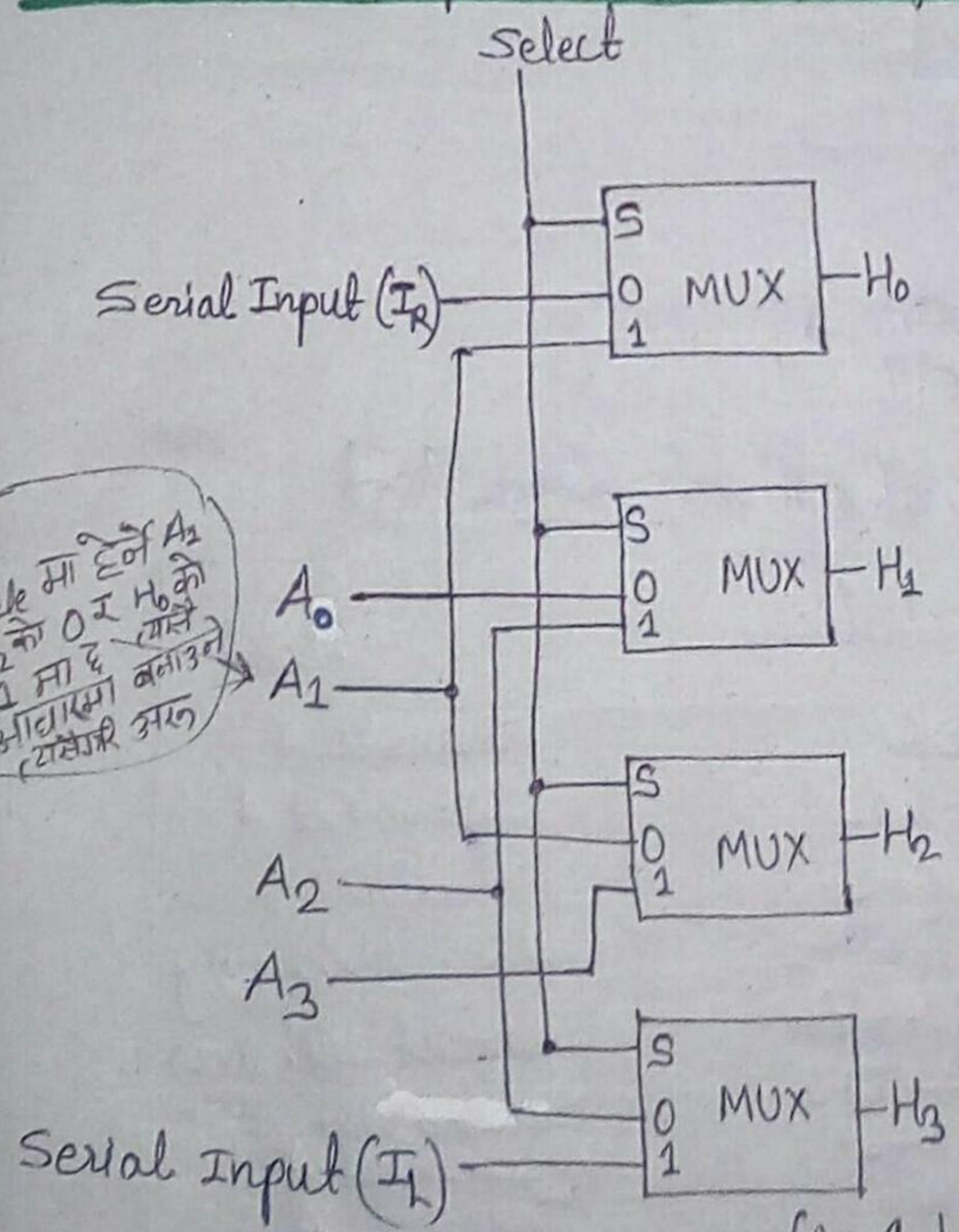


arithmetic shift left (ashl)



arithmetic shift right (ashr)

⊛. 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

पहिला table बनाउने अस्को आधारमा $I_R, A_0 \ldots A_3\ I_L$ बनाउने यस जिलो हुन्छ

fig. 4-bit combinational circuit shifter

⊛. Arithmetic Logic Shift Unit:

Hardware Implementation

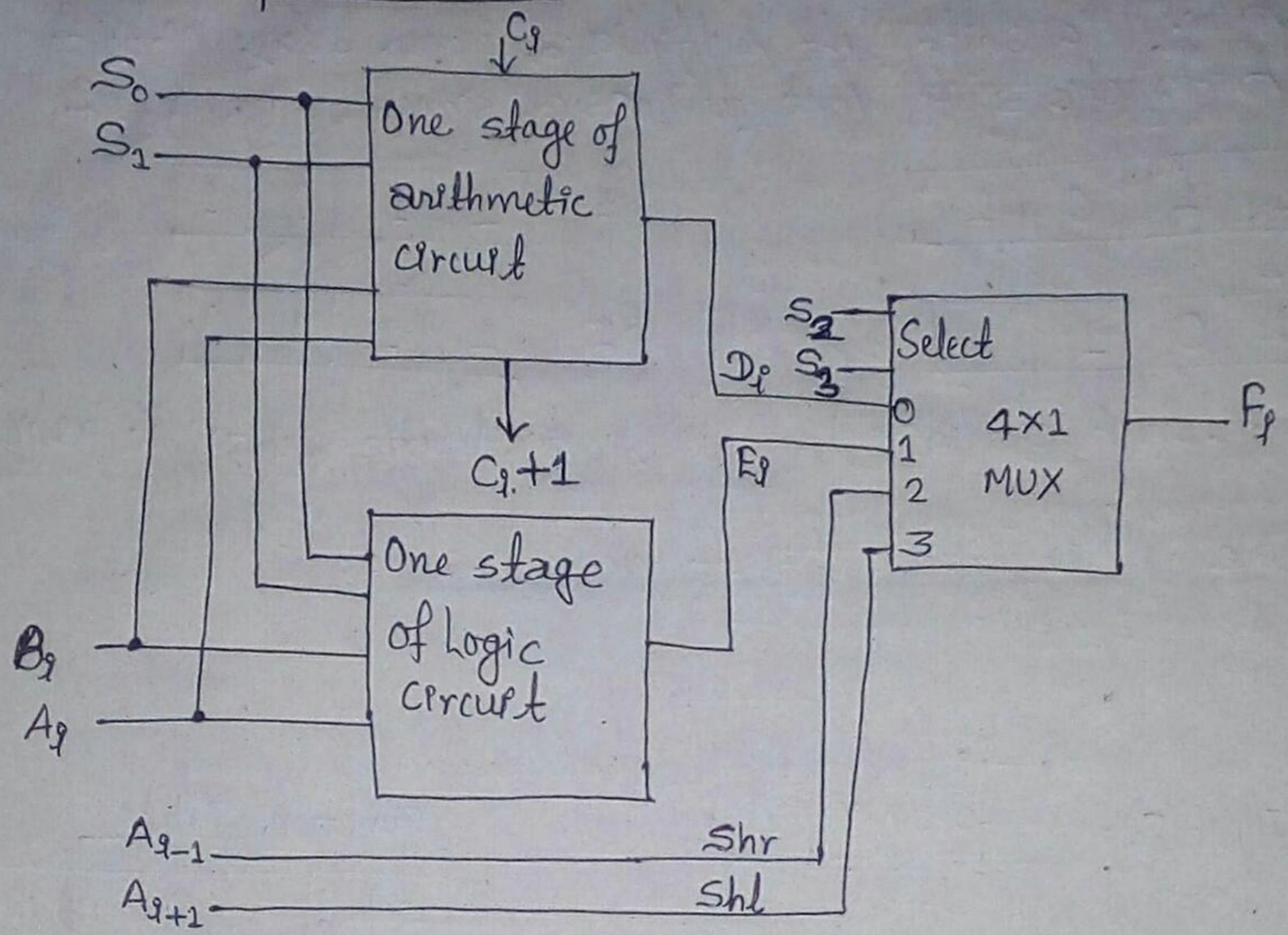


fig. One stage of arithmetic-logic shift

Operation select					Operation	Function
S_3	S_2	S_1	S_0	C_{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	$F=A \wedge B$	AND
0	1	0	1	X	$F=A \vee B$	OR
0	1	1	0	X	$F=A \oplus B$	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

Arithmetic operations

Logical operations

Shift operations

Logical operation में carry होता ही नहीं X

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 ← AR + BR
CR ← CR ^ DR
BR ← BR + 1
AR ← AR − CR.

Solution.

For AR ← AR+BR

```
      AR : 11110010
      BR : +11111111
carry
      AR ⇒ 1 11110001
      AR ⇒ 11110001
```

For CR ← CR ^ DR

```
      CR : 10111001
      DR : 11101010
      ----------
           10101000
      CR ⇒ 10101000
```

For BR ← BR+1

```
      BR : 11111111
           +1
      ----------
carry → 1 00000000
      BR ⇒ 00000000.
```

For AR ← AR − CR

```
      AR : 11110010
      CR+1 : +10111000
      ----------
carry → 1 01001001
      AR ⇒ 01001001
```

```
AR : 11110010
CR − 01011000
-------------
AR ⇒ 00111001
```

Rough

```
  10111001      128
                +64
  01000110      +32
                +16
  11110010      +2
− 10111001      242
  --------
  10000000
```

```
00111001
128  64 32 16 8 4 2 1
1    1  1  1        1
✓    ✓  ✓  ✓
```

128
+ 32
+ 16
+ 8
+ 4
185

242
− 185
57

28
+ 32
+ 1
51