Prof. Usman Sindhi
CSE Department GUNI-ICT

2. REGISTER TRANSFER AND MICRO-OPERATIONS

CONTENTS

REGISTER TRANSFER LANGUAGE

Modules of a digital system are best defined by a set of registers and the operations that are performed on the binary information stored in them.

Micro-operation: Operation executed on data stored in registers.

Result of operations: may replace data in registers, or transfer it to another register.

Examples of Micro-operations: shift, count, clear and load.

REGISTER TRANSFER LANGUAGE

Organization of a digital system is best defined by:

I . The set of registers in the system.

2. The operations that are performed on the data stored in the registers.

3. The control that supervises the sequence of operations in the system

register transfer operations: Information flow and processing performed on the data stored in the registers.

Register Transfer Language(RTL): The symbolic notation used to describe the micro-operation transfers among registers.

REGISTER TRANSFER LANGUAGE

Why *Register transfer language?*

symbolic language

convenient tool for describing the

internal organization of digital computers

Can also be used to facilitate the design

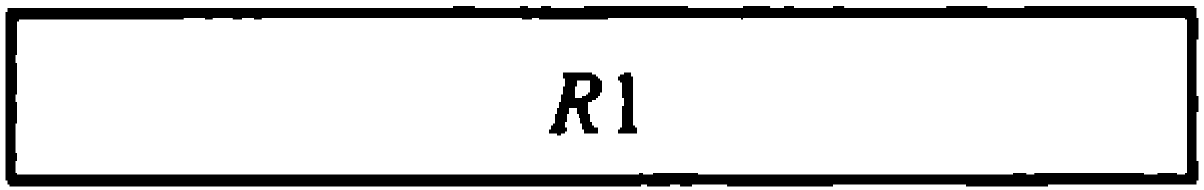
process of digital systems.

REGISTER TRANSFER

Registers are designated by capital letters with numbers to denote the function of the register.

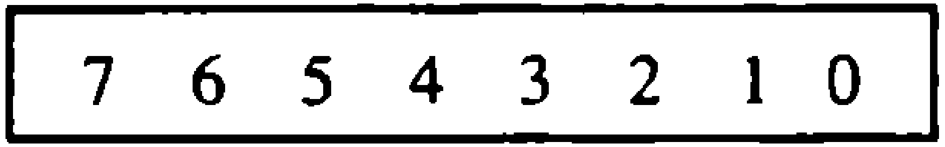
- MAR : register that holds an address for the memory unit (memory address register).
- R1: processor register.

The most common representation for a register :

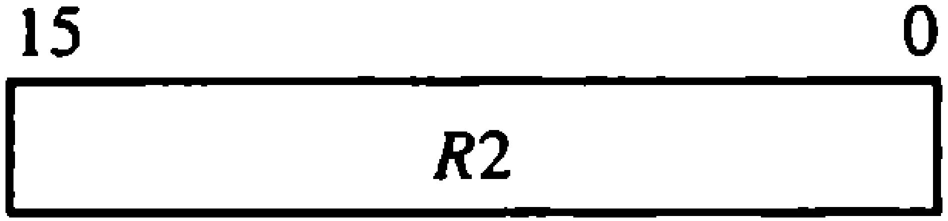


REGISTER TRANSFER

Individual flip-flops in n-bit register are numbered in sequence from 0 to n-1. and is represented as:

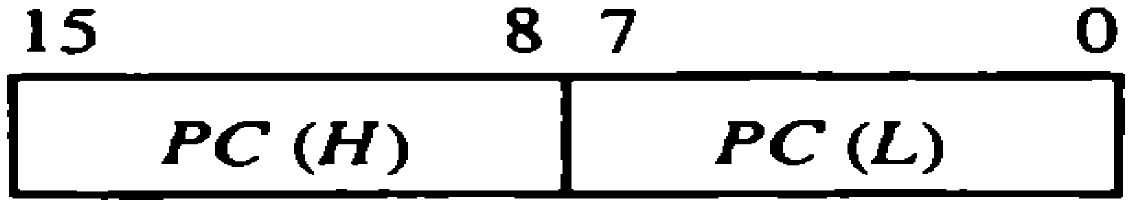


Numbering of bits in 16-bit register can be marked as:



REGISTER TRANSFER

16-bit register can be partitioned into two parts:



Bits 0 to 7 are assigned L(low byte)

Bits 8 to 15 are assigned H(high byte)

The name for the 16-bit register is PC.

PC(L) or PC(0-7) refers to the low byte

PC(H) or PC(8-16) refers to the high byte

REGISTER TRANSFER, EXAMPLE 1

Information transfer from one register to another is represented as:

$$R2 \leftarrow R1$$

Means: transfer the content of register R1 into register R2.

Note: R2 content will be replaced by R1 content.

R1 content will not change.

Normally transfer occur only under a control condition :

$$\textit{If } (P = 1) \textit{ then } (R2 \leftarrow R1)$$

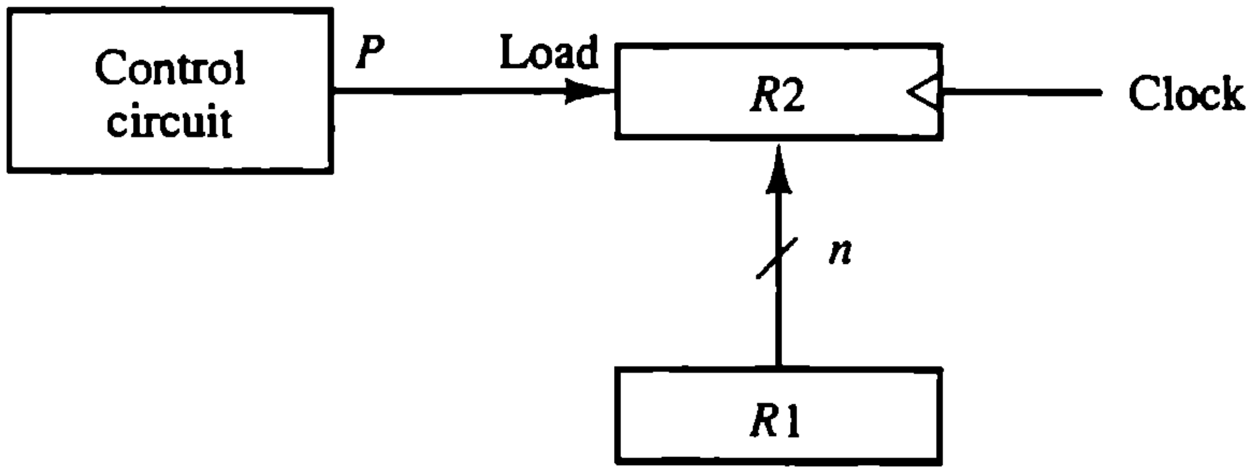
Where p, is a control signal .

Control function :

$$P: \quad R2 \leftarrow R1$$

REGISTER TRANSFER, EXAMPLE 1

The previous statement implies the following hardware construction:

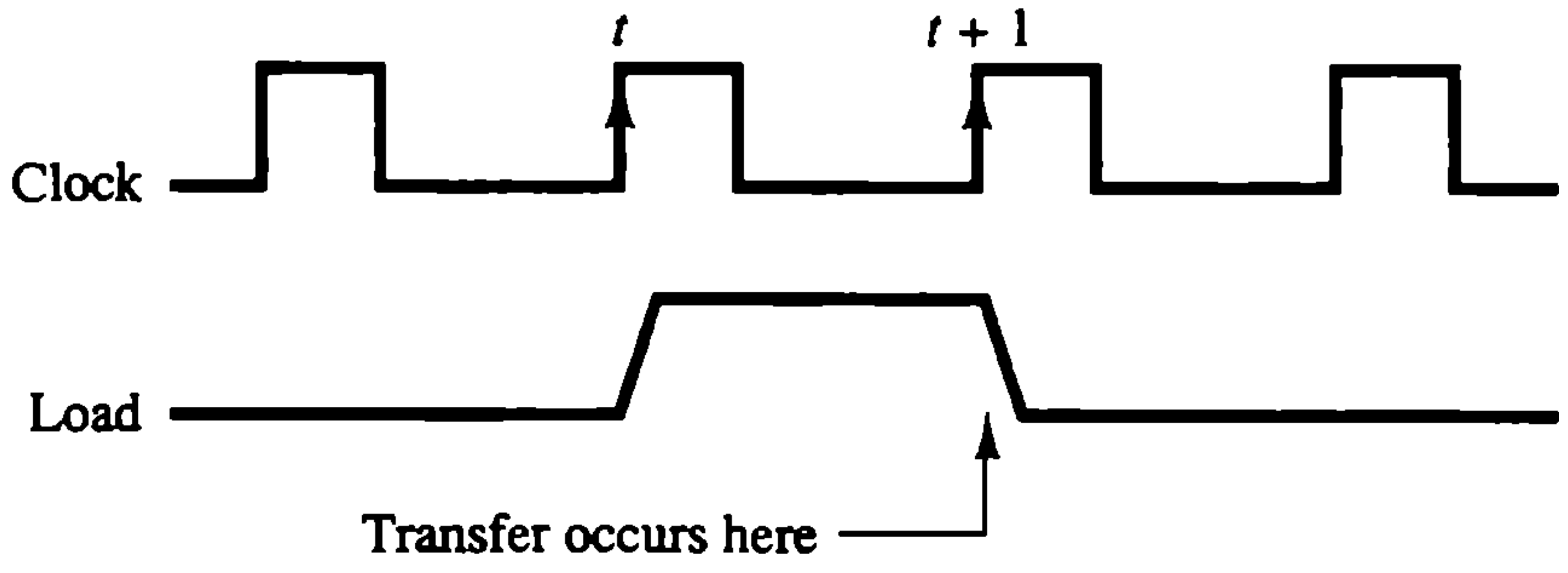


Where n indica

R2 has load input that is activated by the control input (p)

Clock is not included as a variable in the register transfer statement, why?

REGISTER TRANSFER, EXAMPLE 1



(b) Timing diagram

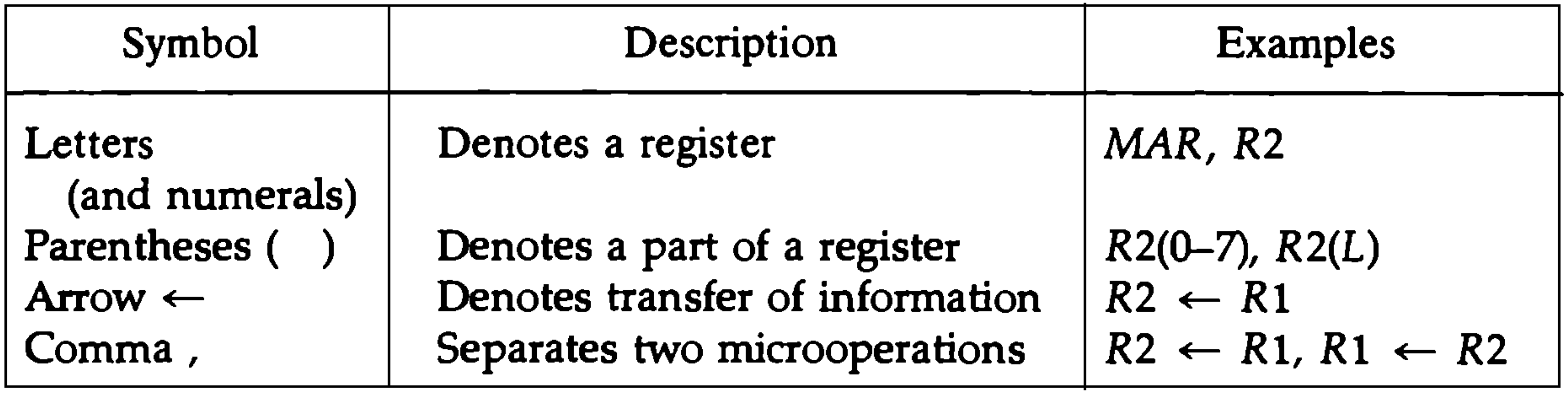
REGISTER TRANSFER, EXAMPLE 2

Comma separates two or more operations executed at the same time.

T: $R2 \leftarrow R1$, $R1 \leftarrow R2$

An operation that exchanges the contents of the two registers during one common clock pulse provided T=1.

BASIC SYMBOLS FOR REGISTER TRANSFER

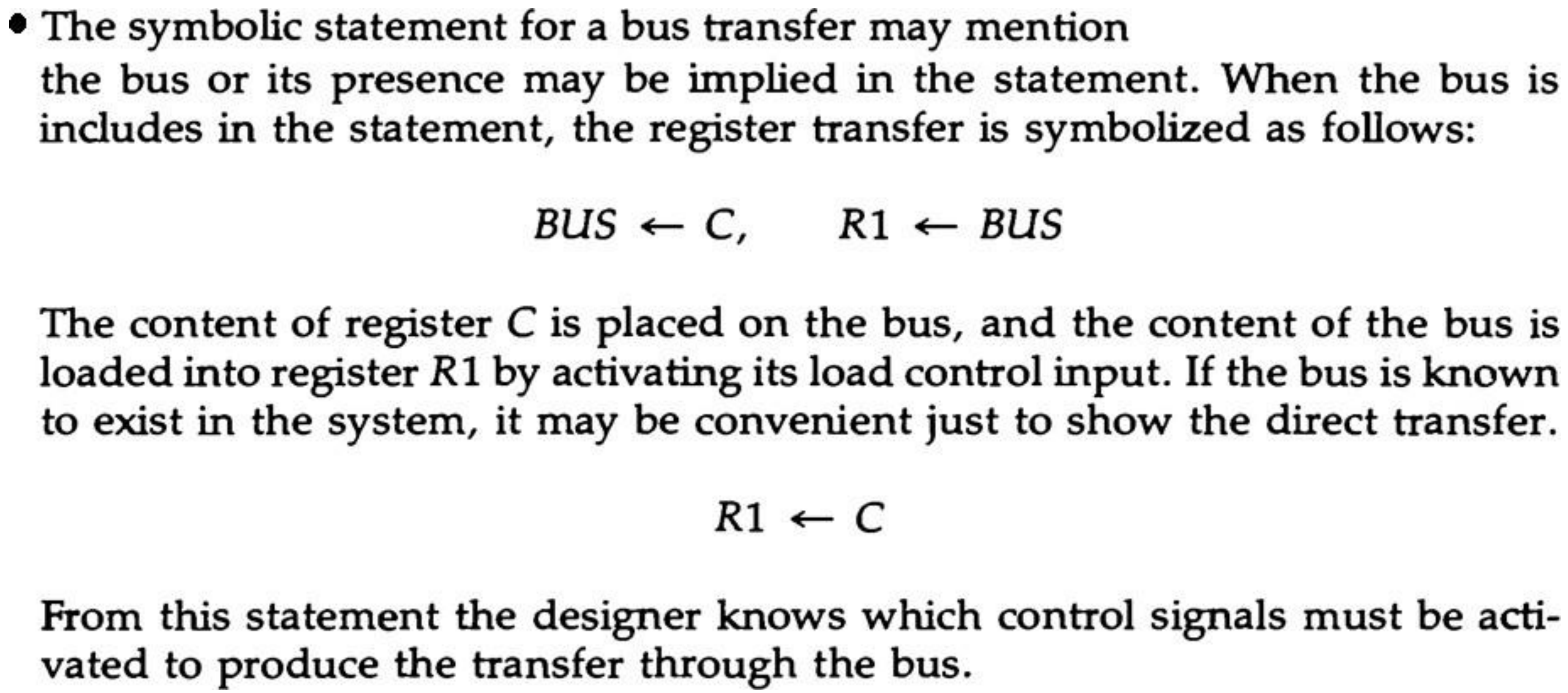


Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R*2
Parentheses ( )	Denotes a part of a register	*R*2(0–7), *R*2(*L*)
Arrow ←	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1, R1 \leftarrow R2$

BUS AND MEMORY TRANSFER

Bus is a path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

From a register to bus: BUS 🡐 R



- The symbolic statement for a bus transfer may mention
the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows:

$$BUS \leftarrow C, \qquad R1 \leftarrow BUS$$

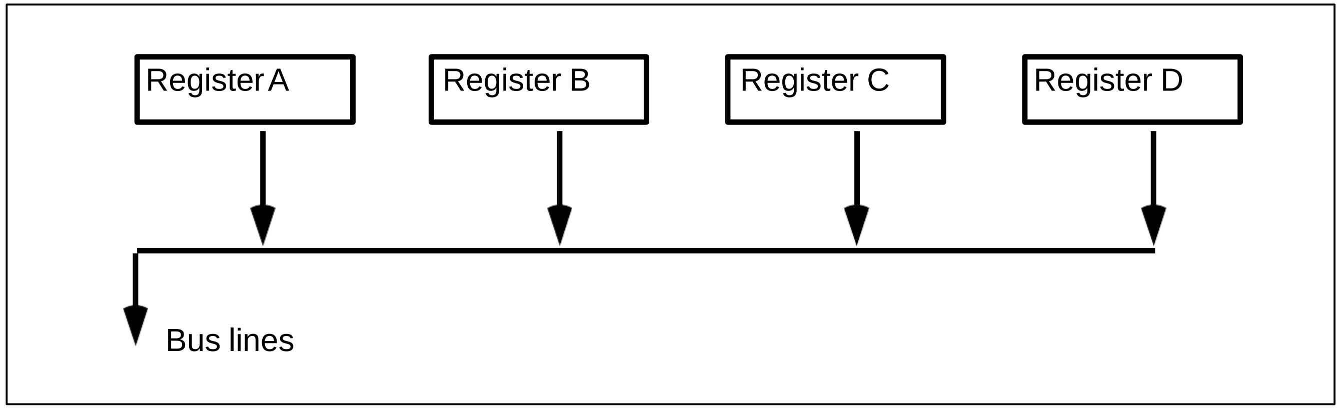
The content of register C is placed on the bus, and the content of the bus is loaded into register $R1$ by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

$$R1 \leftarrow C$$

From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

COMMON BUS SYSTEM (USING MULTIPLEXERS)

To reduce number of wires, we construct a Common bus system, one way is using **Multiplexers**



COMMON BUS SYSTEM (USING MULTIPLEXERS)

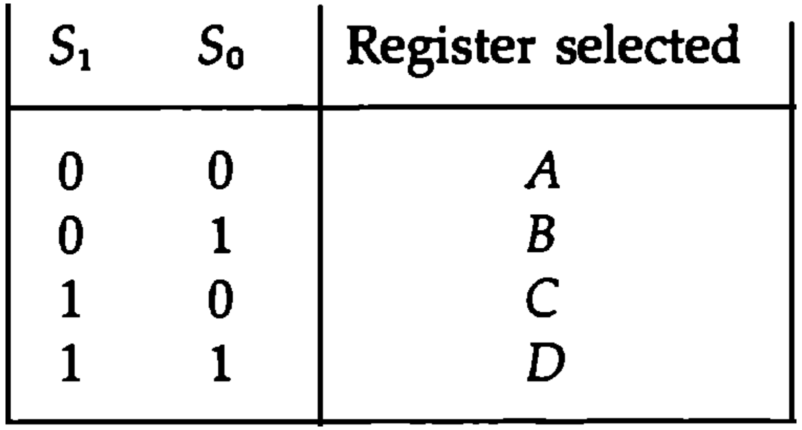
A bus system to multiplex **K registers** with **n bits** each, needs:

- **n multiplexers**.
- **Size** of each multiplexer is **Kx1**

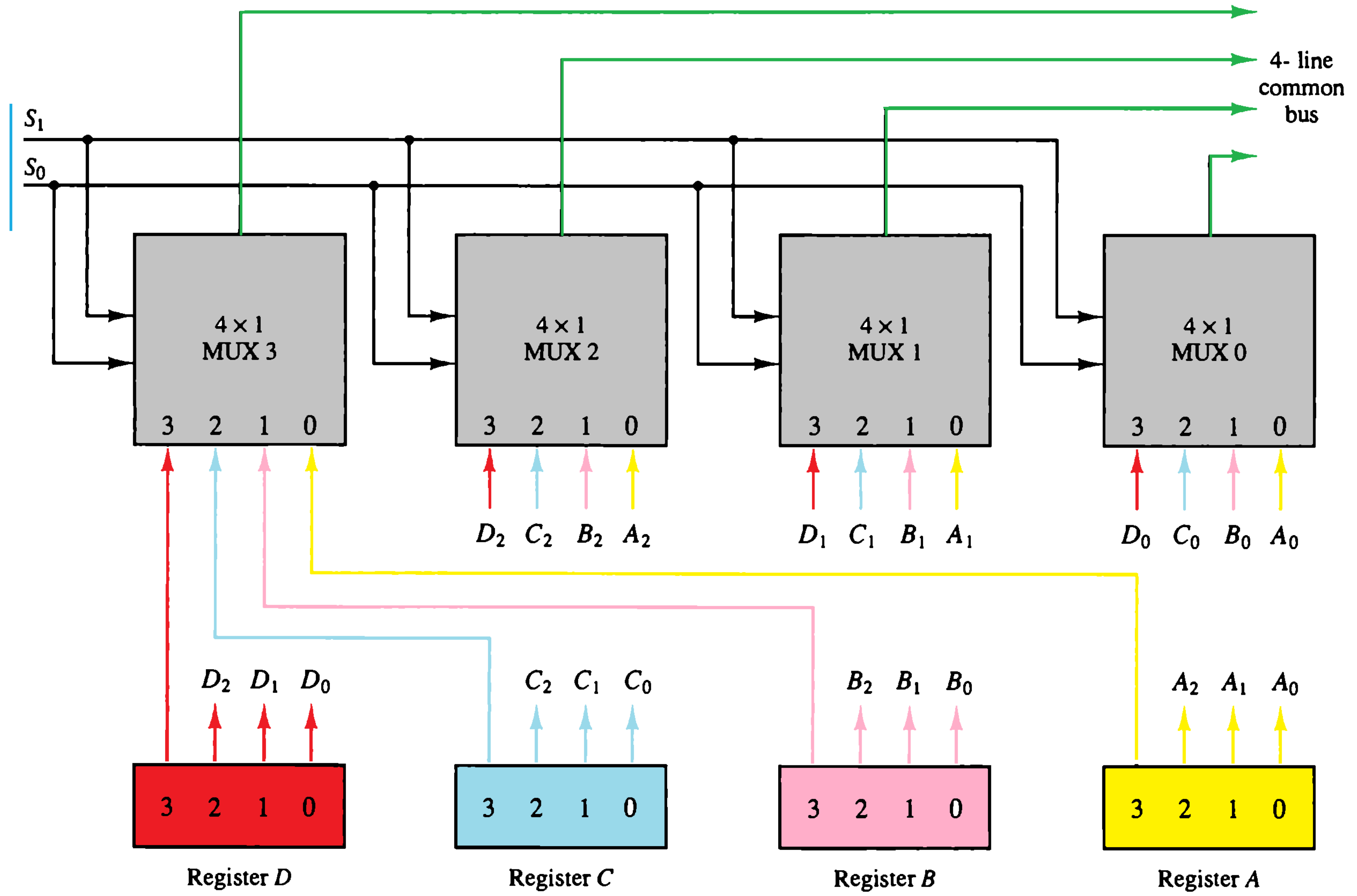
Example:

The common bus for 4 register of 4 bits each needs 4 (4x1) multiplexers

- Will need 2 selection lines:



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



4- line common bus
S_1
S_0
4 × 1 MUX 3
4 × 1 MUX 2
4 × 1 MUX 1
4 × 1 MUX 0
3 2 1 0
D_2 C_2 B_2 A_2
D_1 C_1 B_1 A_1
D_0 C_0 B_0 A_0
D_2 D_1 D_0
C_2 C_1 C_0
B_2 B_1 B_0
A_2 A_1 A_0
Register *D*
Register *C*
Register *B*
Register *A*

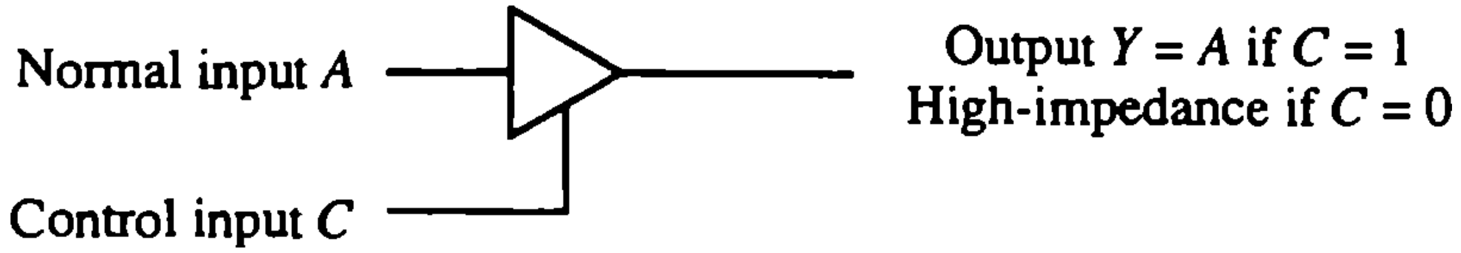
COMMON BUS SYSTEM (USING THREE-STATE BUFFER)

What is three-state buffer?

Or (tri-state buffer) is logic circuit element that has three states: 0, 1 and high impedance (means the output is disconnected).

Has two inputs: a data input A and a control input C.

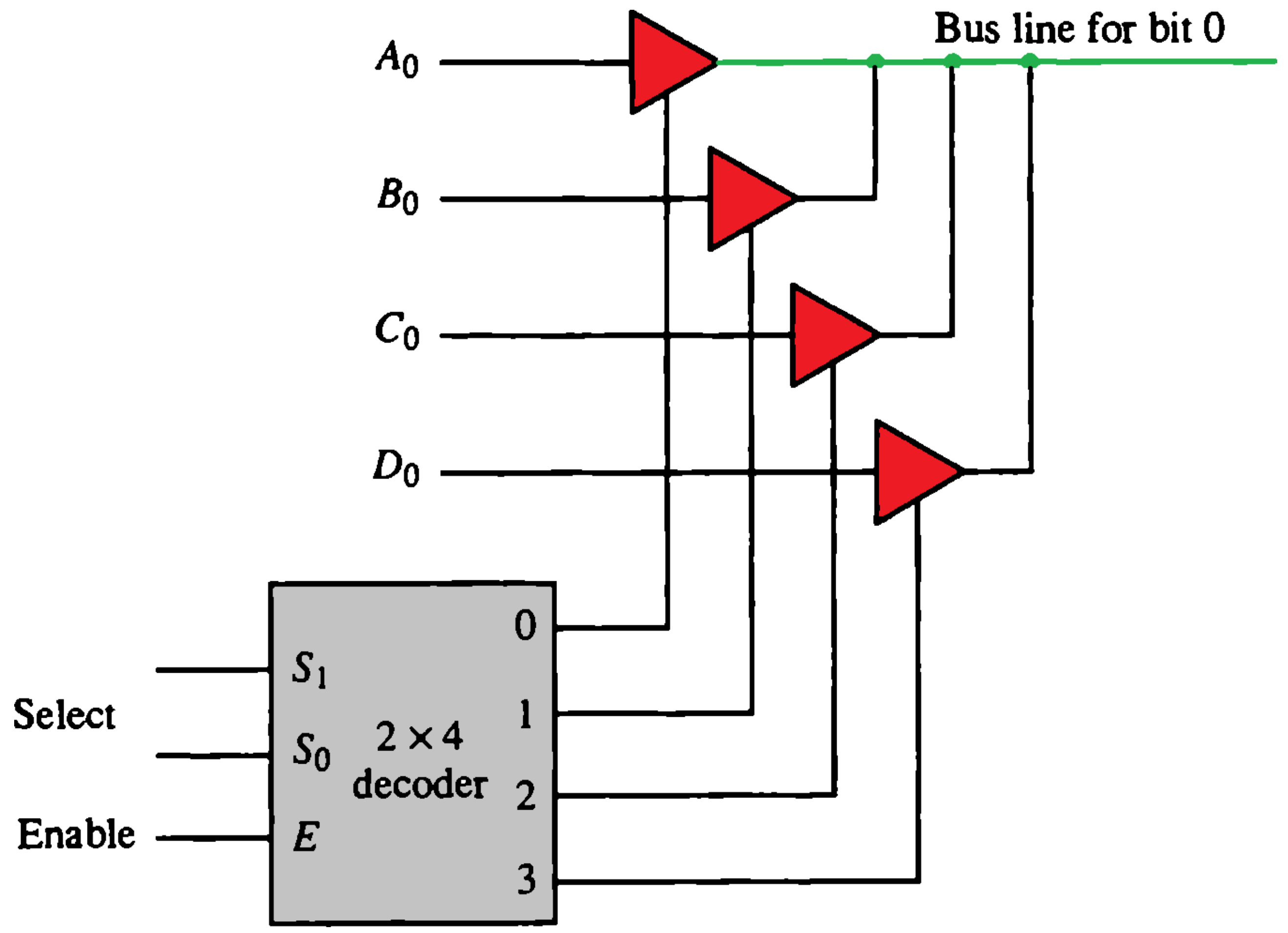
- When C=1, the output is the input.
- when C=0, the output disabled, gate goes to high impedance.



COMMON BUS SYSTEM (USING THREE-STATE BUFFER)

To construct common bus for K registers of n bits, we need:

- One Decoder (size depending on the number of registers).
- n circuits with K buffers.



COMMON BUS SYSTEM (USING THREE-STATE BUFFER)

Example:

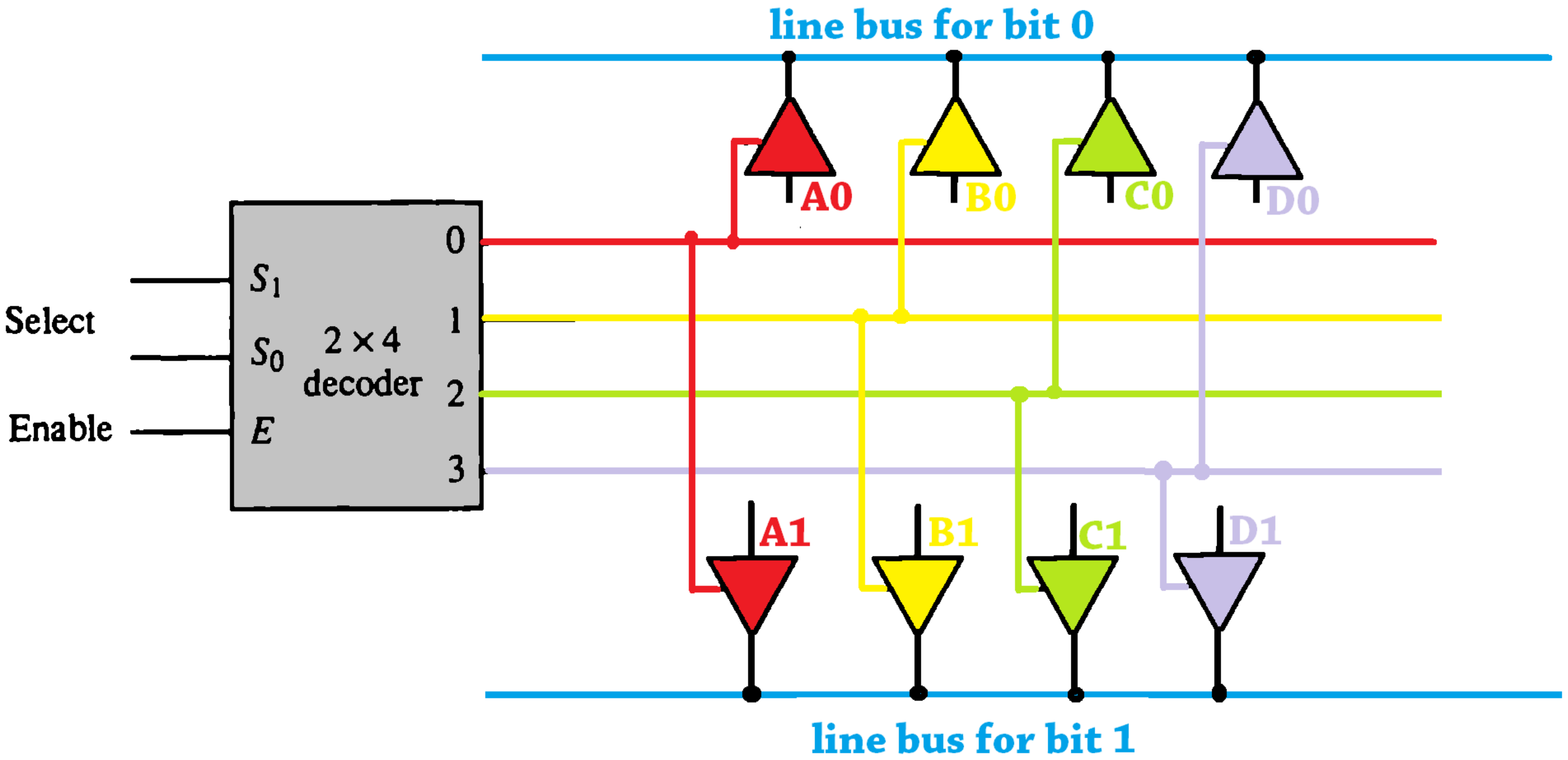
Draw the diagram of a bus system using tri-state buffers for 4 registers of 2 bits each.

Solution:

We will need:

One (2x4) decoder

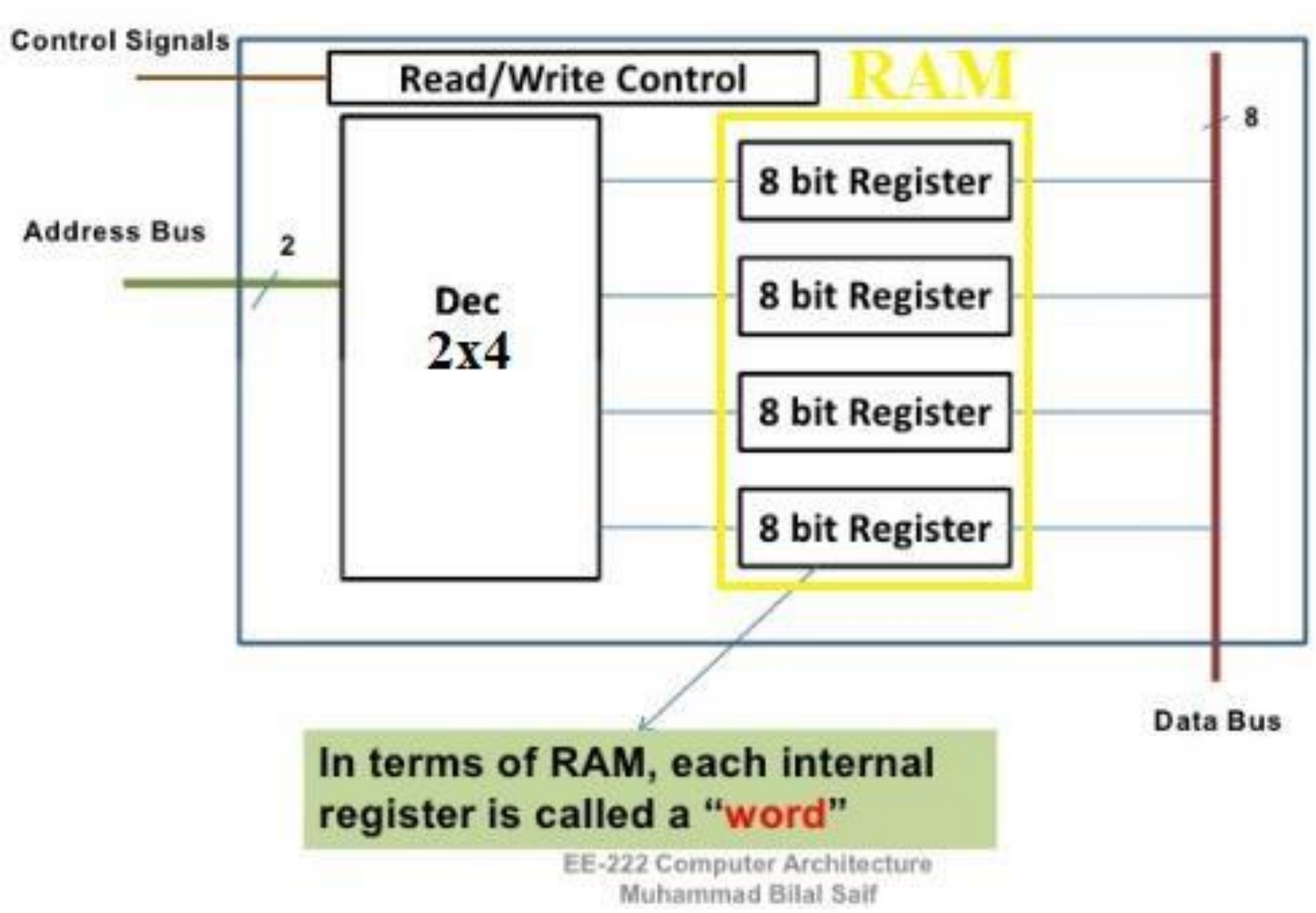
2 circuits (bus lines) with 4 buffers.


line bus for bit 0
A0
B0
C0
D0
Select
Enable
S_1
S_0
E
2 × 4 decoder
0
1
2
3
A1
B1
C1
D1
line bus for bit 1

PRACTICE

Draw the diagram of a bus system using tri-state buffers for 3 registers of 4 bits each

SIMPLE RAM DESIGN



MEMORY TRANSFER

Read operation: The transfer of information from a memory word to the outside environment.

Write operation: The transfer of new information to be stored into the memory. A

memory word will be symbolized by the letter **M**.

- It is necessary to specify the address of M when writing memory transfer operations, example **M [address]**

Read: DR← M [AR] means:

- data from the memory unit M which has the address AR, will be transferred to the data register DR.

MEMORY TRANSFER

Write: $M[AR] \leftarrow R1$ means:

- transfers the content of the register R1 to a memory word M selected by the address AR.

ARITHMETIC MICRO OPERATIONS

Elementary operations performed with the data stored in registers.

Classified into four categories:

1. **Register transfer** micro-operations transfer binary information from one register to another.
2. **Arithmetic** micro-operations perform arithmetic operation on numeric data stored in registers.
3. **Logic** micro-operations perform bit manipulation operations on non-numeric data stored in registers.
4. **Shift** micro-operations perform shift operations on data stored in registers.

ARITHMETIC MICRO OPERATIONS

Basic arithmetic micro-operations:

- Addition
- Subtraction
- Increment
- Decrement
- Shift

$R3 \leftarrow R1 + R2$

- specifies an **add** micro-operation
- It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

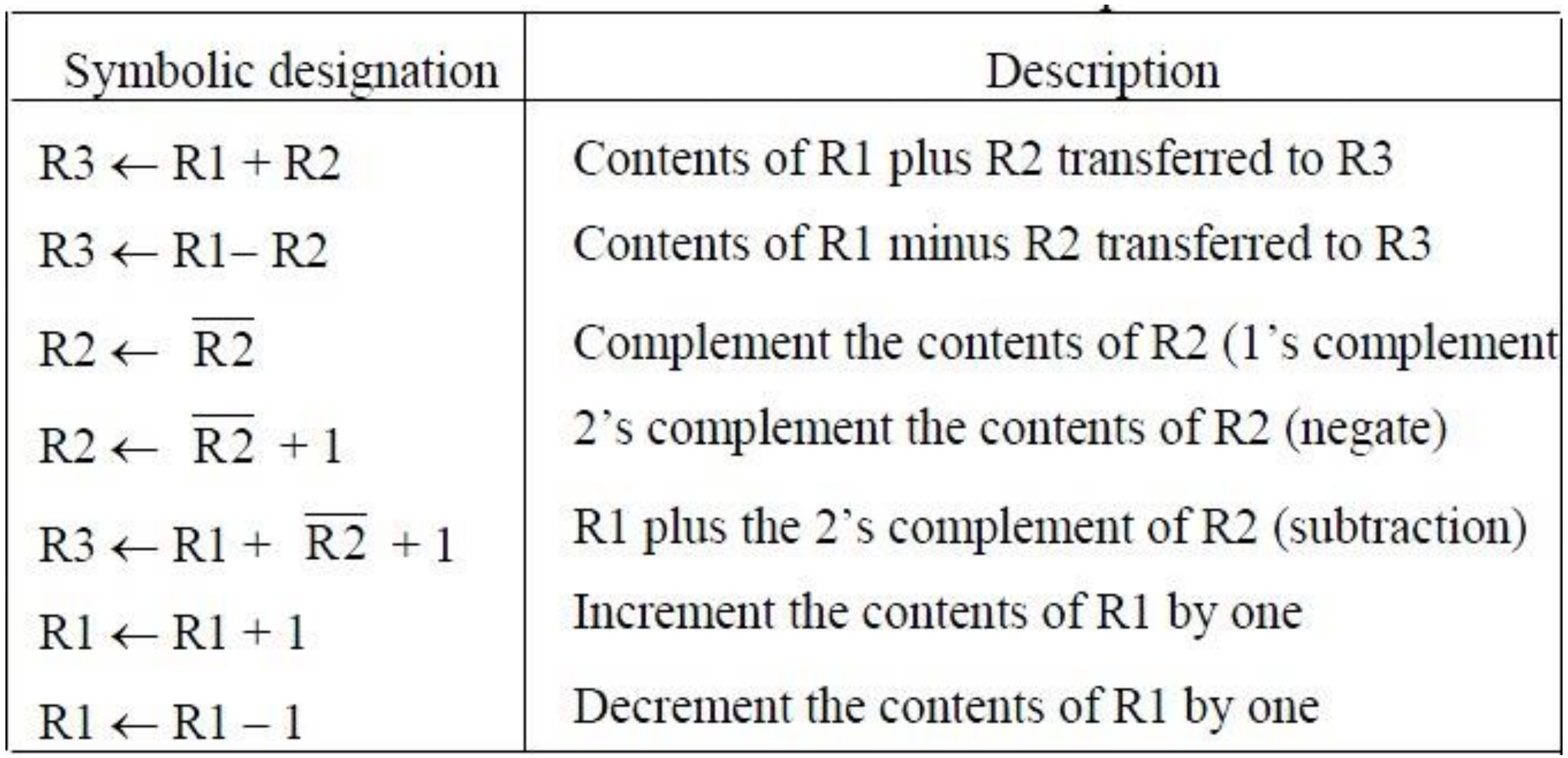
ARITHMETIC MICRO OPERATIONS

- $R3 \leftarrow R1 + \overline{R2} + 1$

Subtraction:

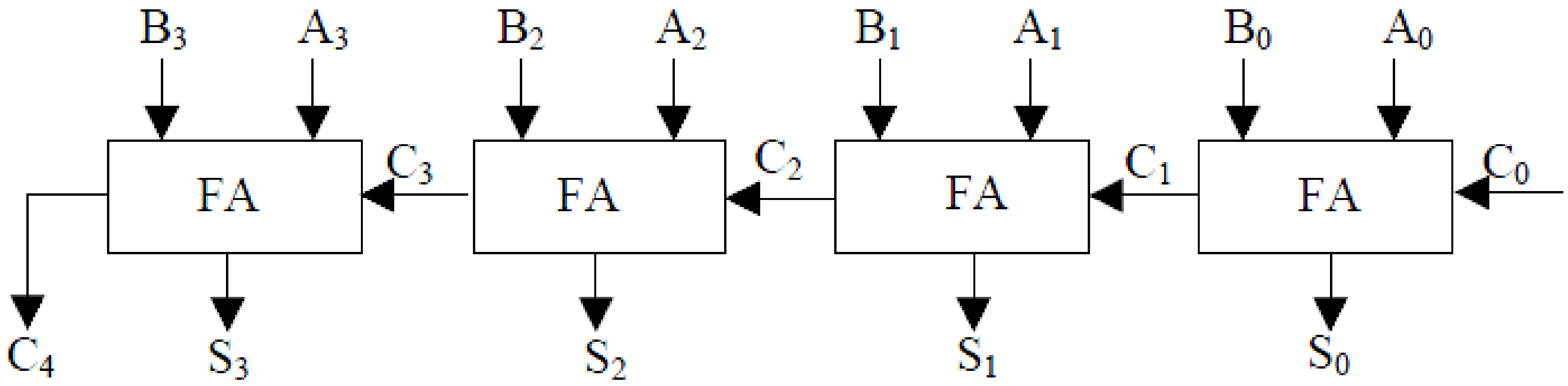
$\overline{R2}$ is the symbol for the 1’s complement of R2. Adding 1 to the 1’s complement produces the 2’s complement. Adding the contents of R1 to the 2’s complement of R2 is equivalent to R1 – R2.

BASIC SET OF MICRO-OPERATIONS

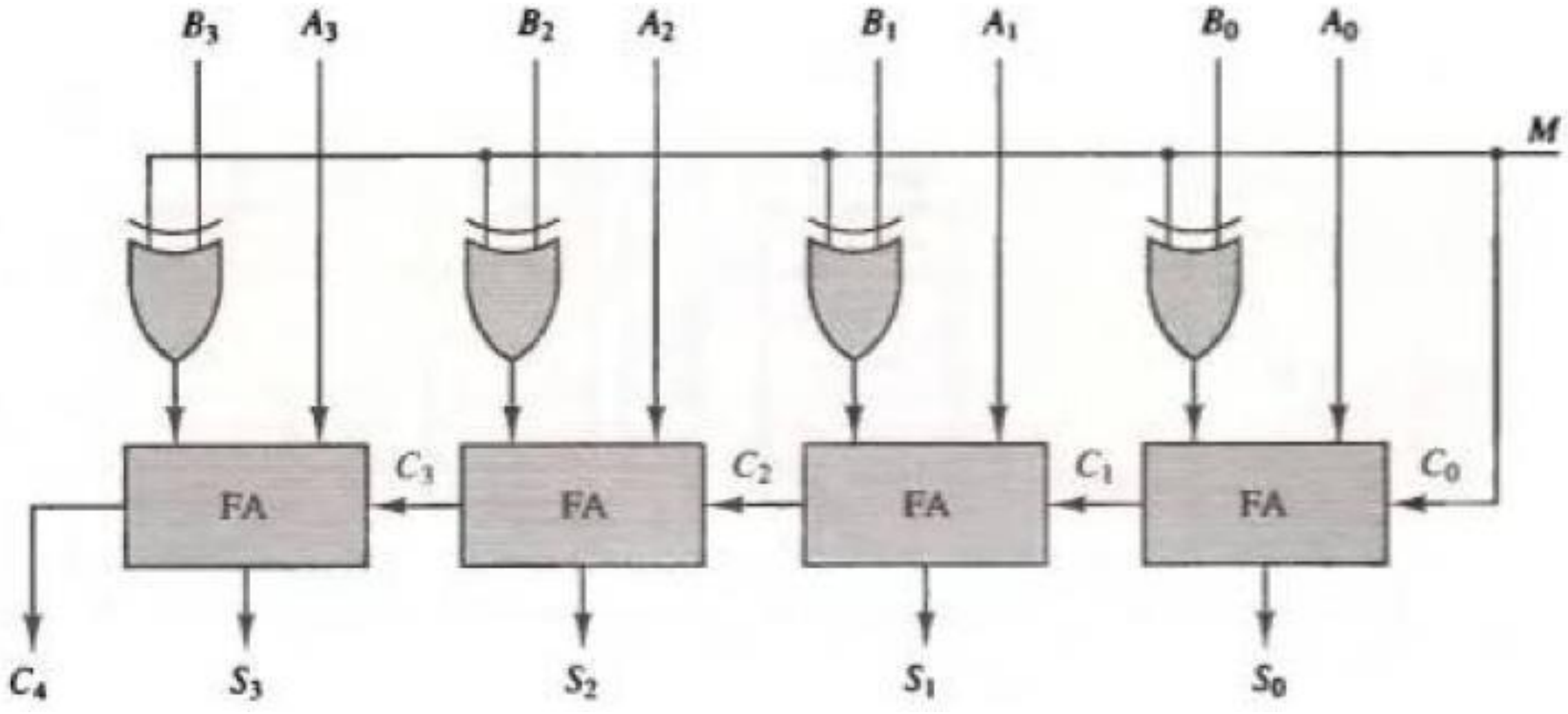


Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents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

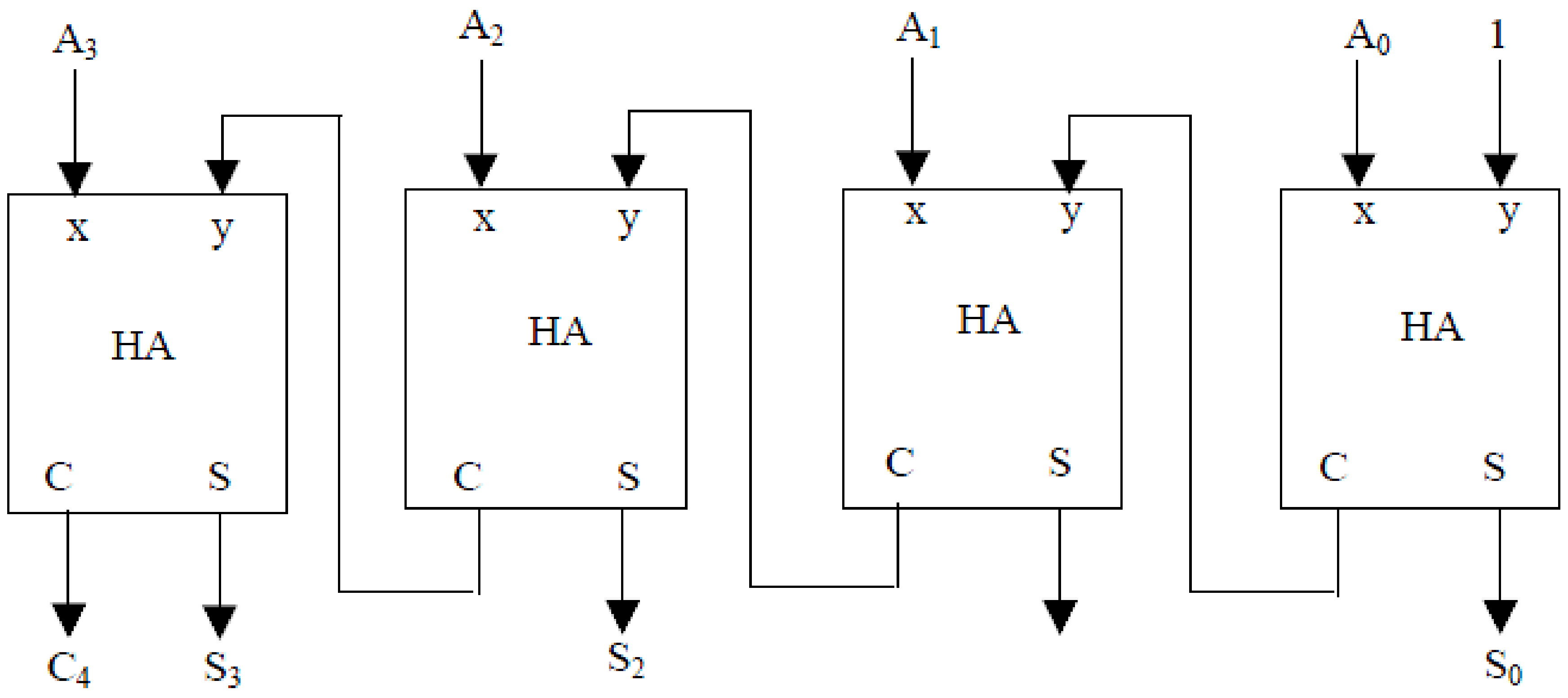


BINARY ADDER-SUBTRACTOR



BINARY INCREMENTER

The increment micro-operation adds one to a number in a register



ARITHMETIC CIRCUIT

The arithmetic micro-operations can be implemented in one composite arithmetic circuit.

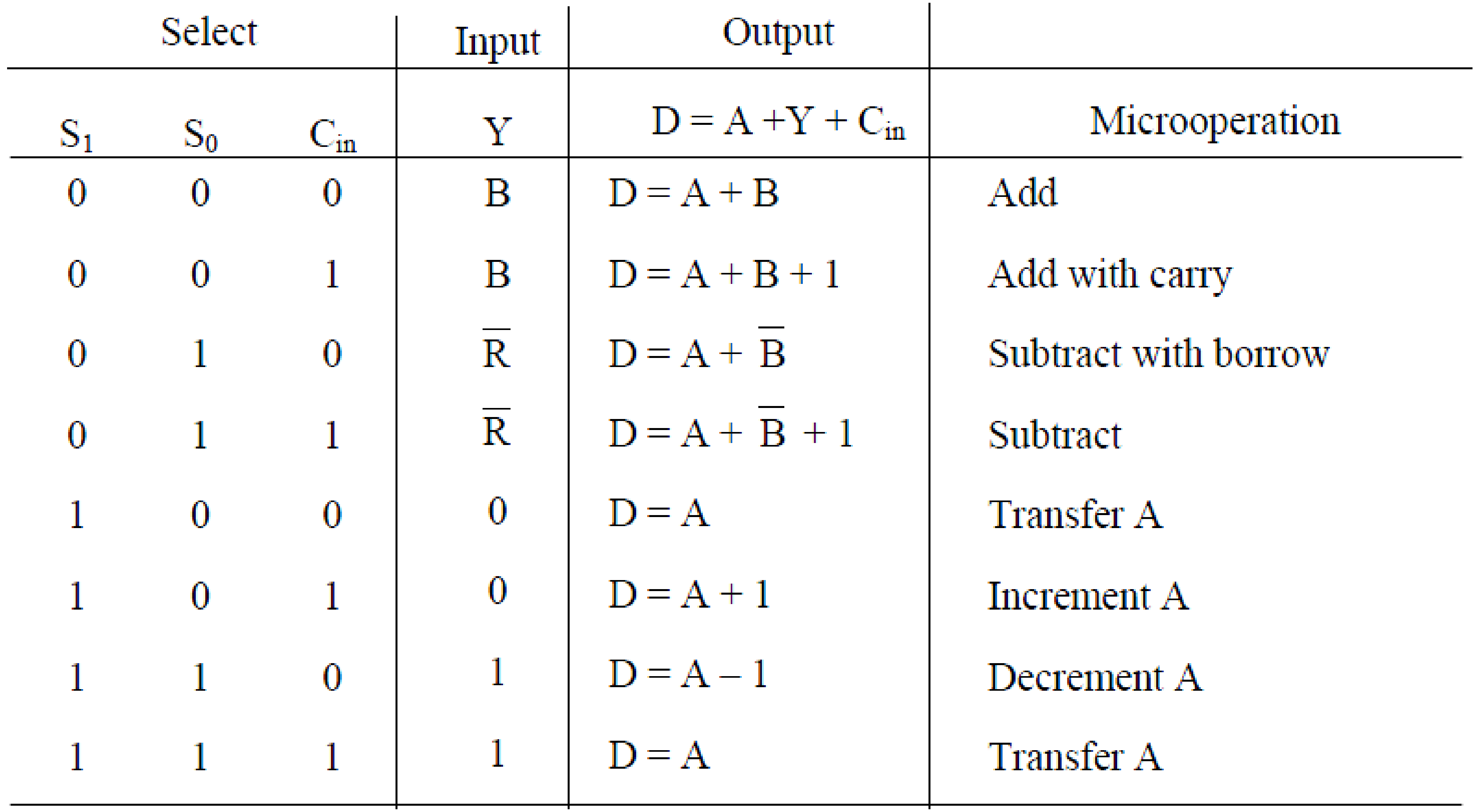
The diagram of a 4-bit arithmetic circuit contains:

- 4 full-adder circuits.
- 4 multiplexers for choosing different operations.
- 2 (4-bit) inputs A and B.
- One (4-bit) output D.

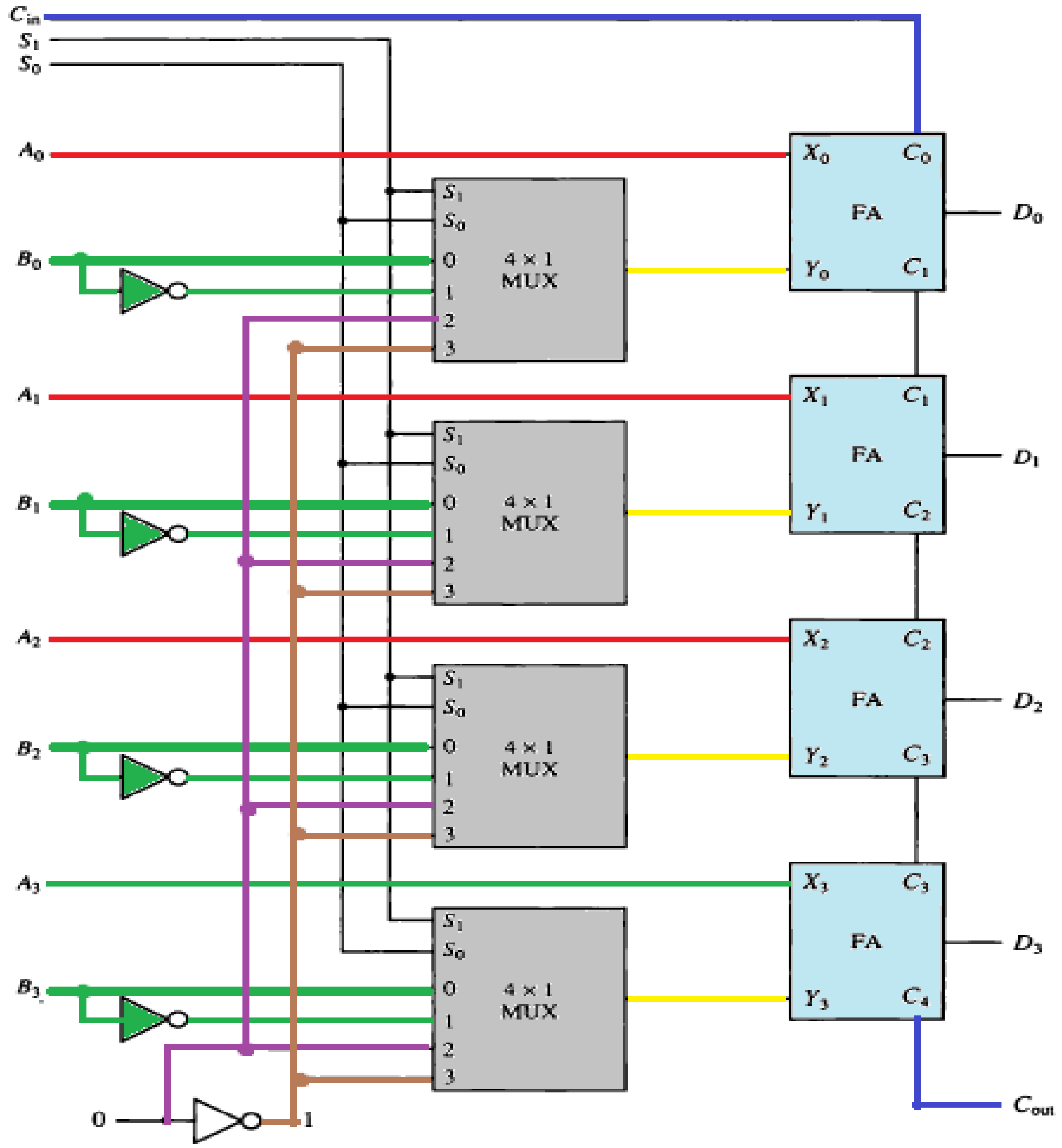
The output of the binary adder is calculated from the following arithmetic sum:

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

ARITHMETIC CIRCUIT FUNCTION TABLE



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


C_{in}
S_1
S_0
A_0
B_0
A_1
B_1
A_2
B_2
A_3
B_3
0
1
S_1
S_0
0
1
2
3
4 × 1 MUX
X_0
C_0
FA
Y_0
C_1
D_0
X_1
C_1
Y_1
C_2
D_1
X_2
C_2
Y_2
C_3
D_2
X_3
C_3
Y_3
C_4
D_3
C_{out}

LOGIC MICRO-OPERATIONS

Logic micro operations specify binary operations for strings of bits stored in registers.

Each bit of the register is considered separately.

Example:

$P: R1 \leftarrow R1 \oplus R2$

- Means exclusive-OR microoperation with the contents of two registers R1 and R2, when p=1.
- if content of R1=1010, the content of R2=1100, then:

1010	Content of R1
1100	Content of R2
$\overline{0110}$	Content of R1 after P = 1

LOGIC MICRO-OPERATIONS

Symbols:

Special symbols will be adopted for the logic micro-operations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions.

Operation	Boolean Function/ variables/control	Register	Micro-operation
OR	+		∨
Add		+	
AND	.		∧

LOGIC MICRO-OPERATIONS

Example:

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

- The + between P and Q : OR operation (two binary variables)
- The + between R2 and R3: Add micro-operation.
- the symbol ∨: OR micro-operation (between register R5 and R6).

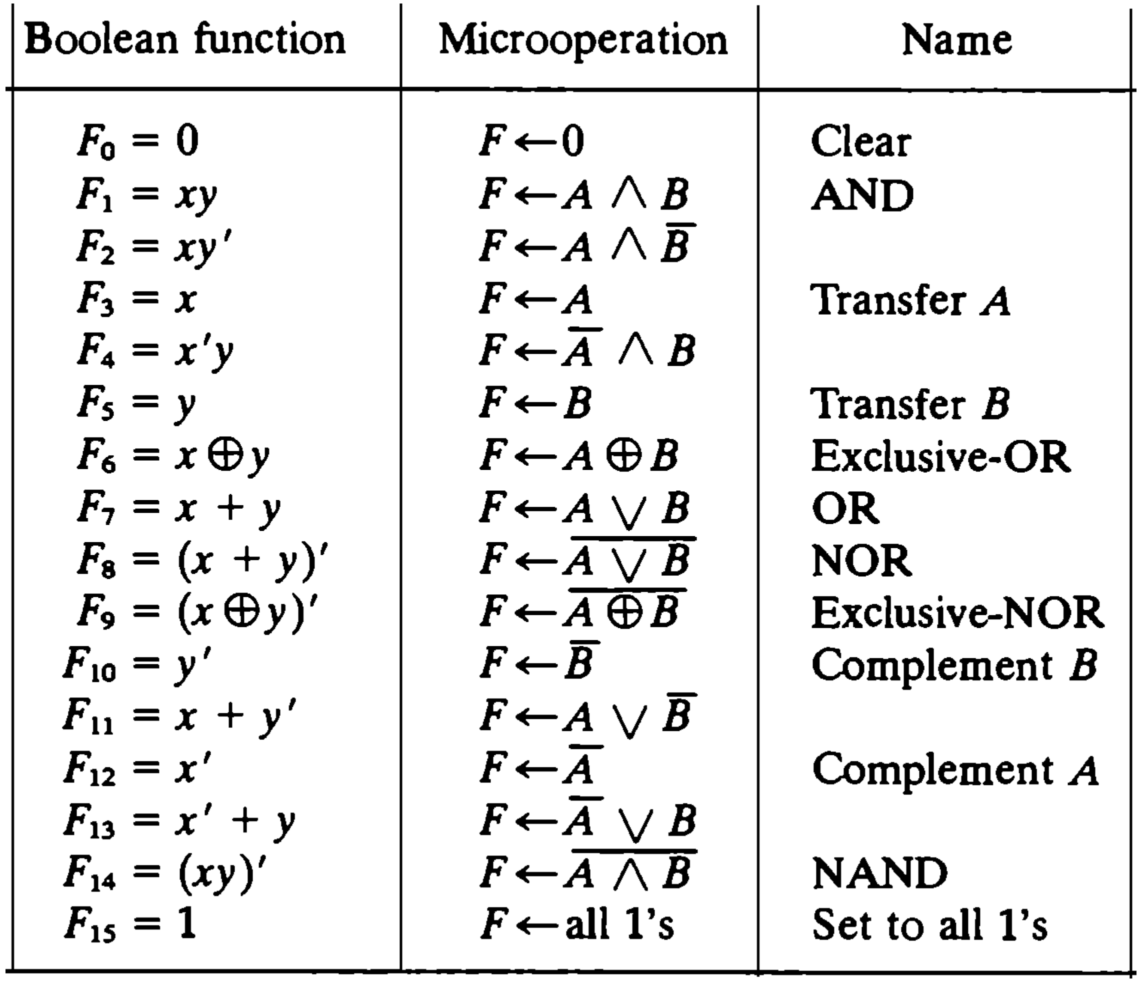
LOGIC MICRO-OPERATIONS

The table shows Sixteen Logic Micro operations.

the Boolean functions listed in the first column represent a relationship between **two binary variables x and y**.

The logic micro-operations listed in the second column represent a relationship between the **binary content of two registers** A and B.

(**Each bit of the register is treated as a binary variable**)

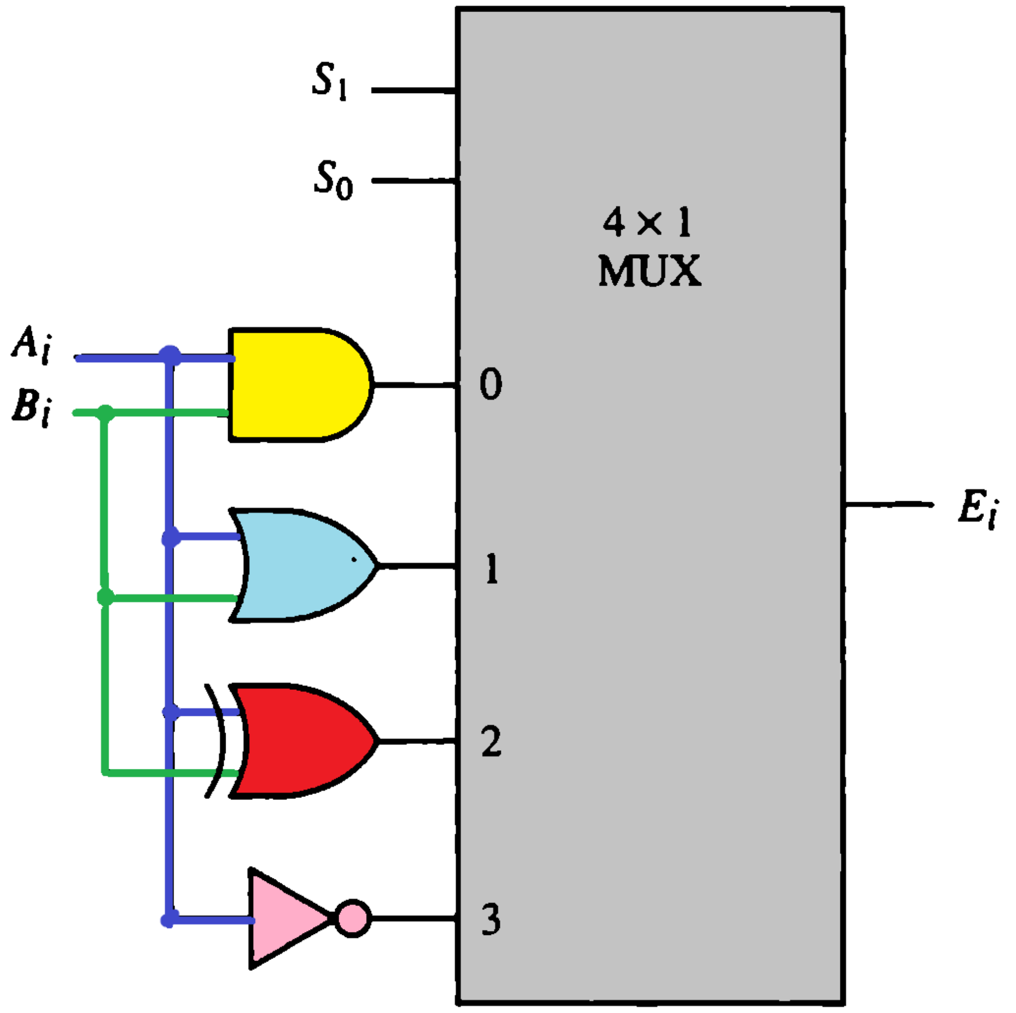


Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	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

Function table

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



SHIFT MICRO-OPERATIONS

Shift micro-operations are used for serial transfer of data.

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

There are three type of shifts :

- 1. Logical Shift
- 2. Circular Shift
- 3. Arithmetic Shift

LOGICAL SHIFT

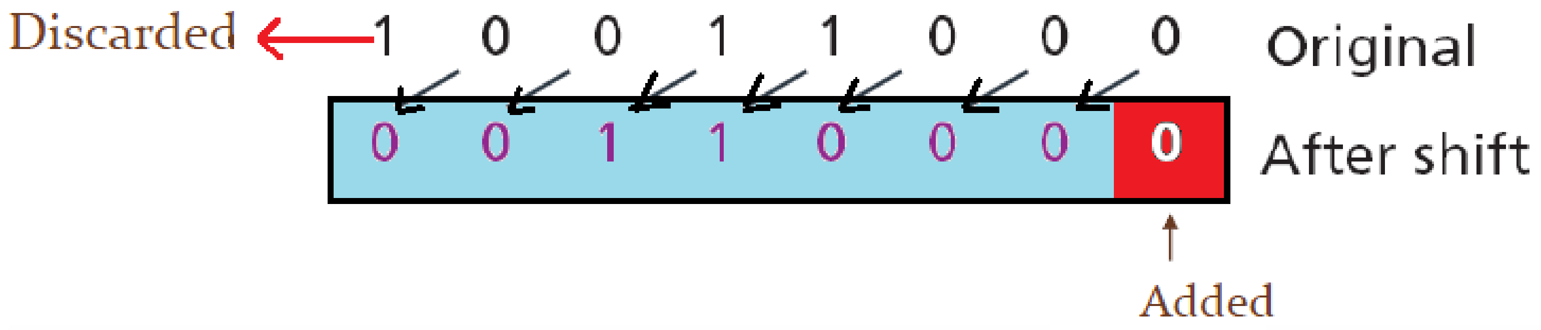
A logical shift is one that transfers 0 through the serial input.

Symbols:

- R1← shl R1 (means: R1 shifts left)
- R2 ← shr R2 (means: R1 shifts right)

Example:

Use a **Logical Left Shift Operation** on the bit pattern 10011000.



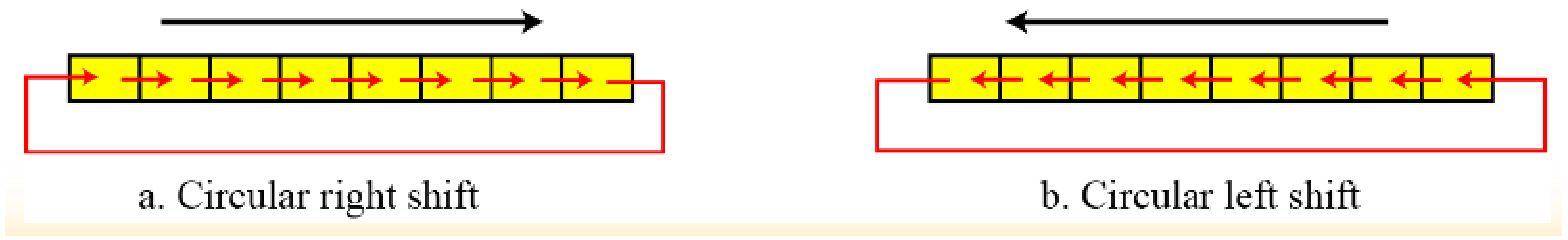
CIRCULAR SHIFT

The circular shift circulates the bits of the register around the two ends without loss of information.

This is accomplished by connecting serial output of the shift register to its serial input.

Symbols:

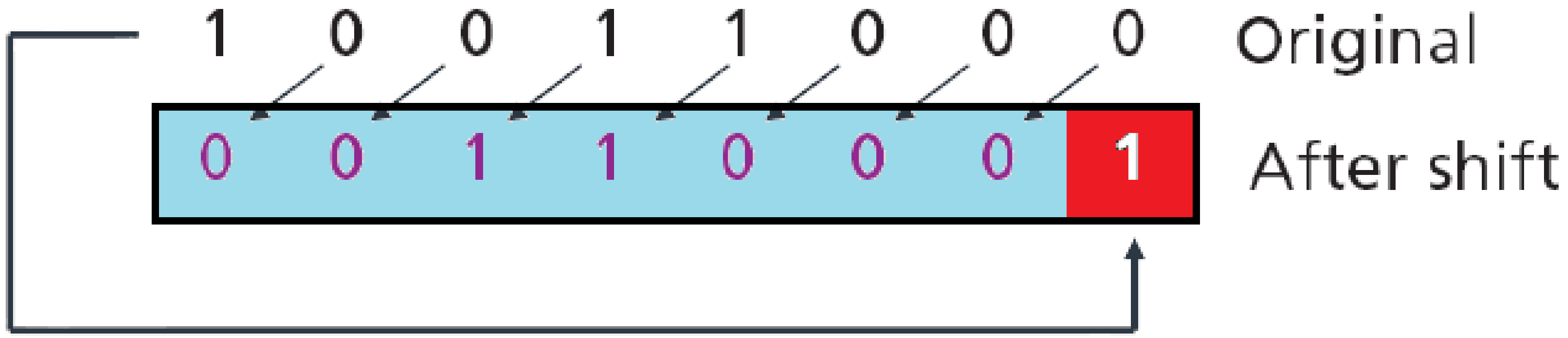
- R1 ←cil R1 (means: circular shift left)
- R2 ←cir R2 (means: circular shift right)



CIRCULAR SHIFT

Example:

Use a **Circular Left Shift Operation** on the bit pattern 10011000.



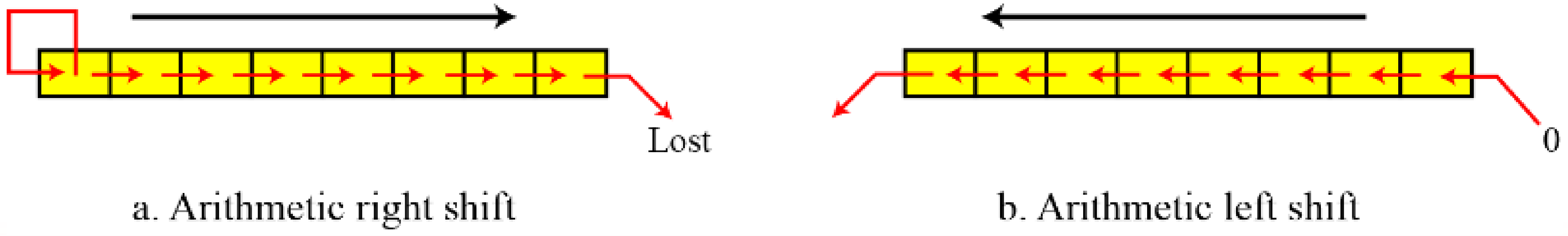
ARITHMETIC SHIFT

An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right.

Arithmetic **right** shift **divides** the number by two.

Arithmetic **left** shift **multiplies** a signed binary number by two.

Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by two.

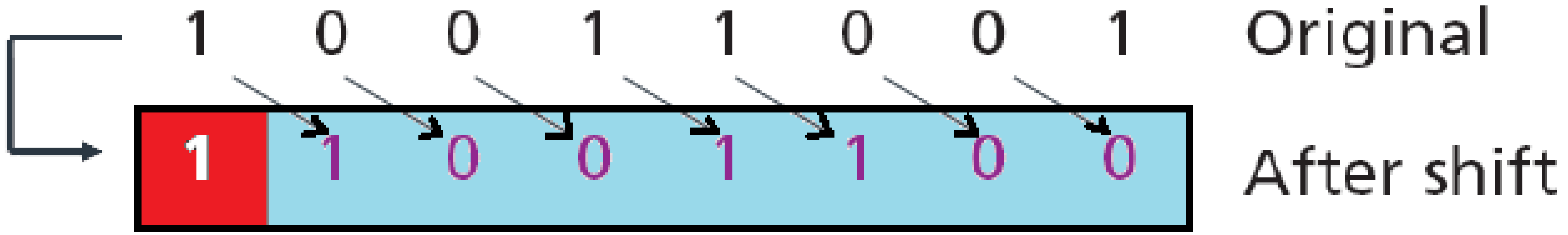


ARITHMETIC SHIFT

Example 1:

Use an **Arithmetic right shift operation** on the bit pattern 10011001.

(The pattern is an integer in two's complement format)



The leftmost bit is retained and also copied to its right neighbor bit.

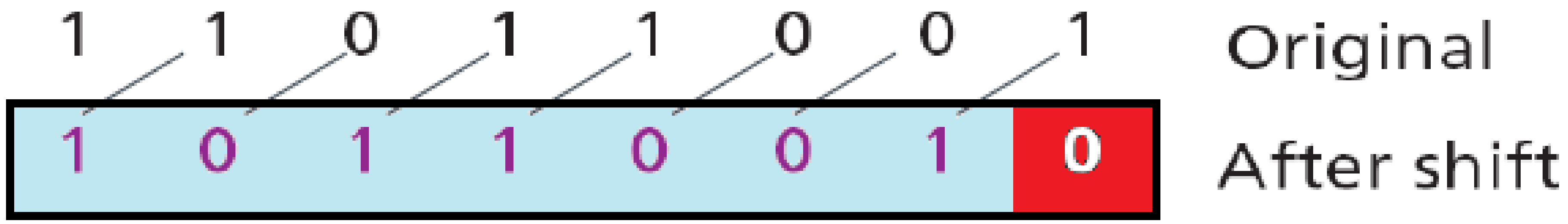
The original number was −103 and the new number is −52, which is the result of dividing −103 by 2 truncated to the smaller integer.

ARITHMETIC SHIFT

Example 2:

Use an **Arithmetic left shift operation** on the bit pattern 11011001.

(The pattern is an integer in two's complement format)



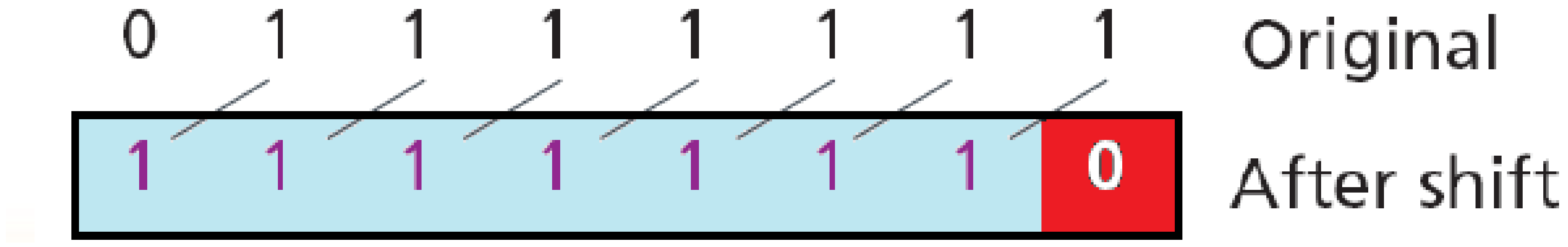
The leftmost bit is lost and a 0 is inserted as the rightmost bit.

The original number was −39 and the new number is −78. The original number is multiplied by two. The operation is valid because no underflow occurred.

ARITHMETIC SHIFT

Example 3:

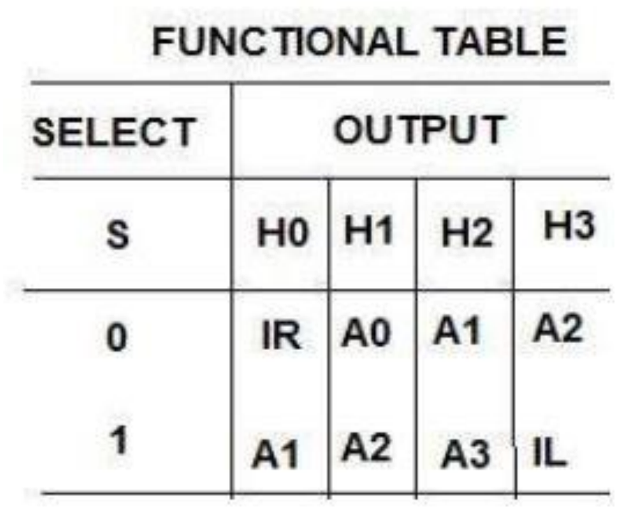
Use an **Arithmetic left shift operation** on the bit pattern **01111111**. The pattern is an integer in twc



The leftmost bit is lost and a 0 is inserted as the rightmost bit.

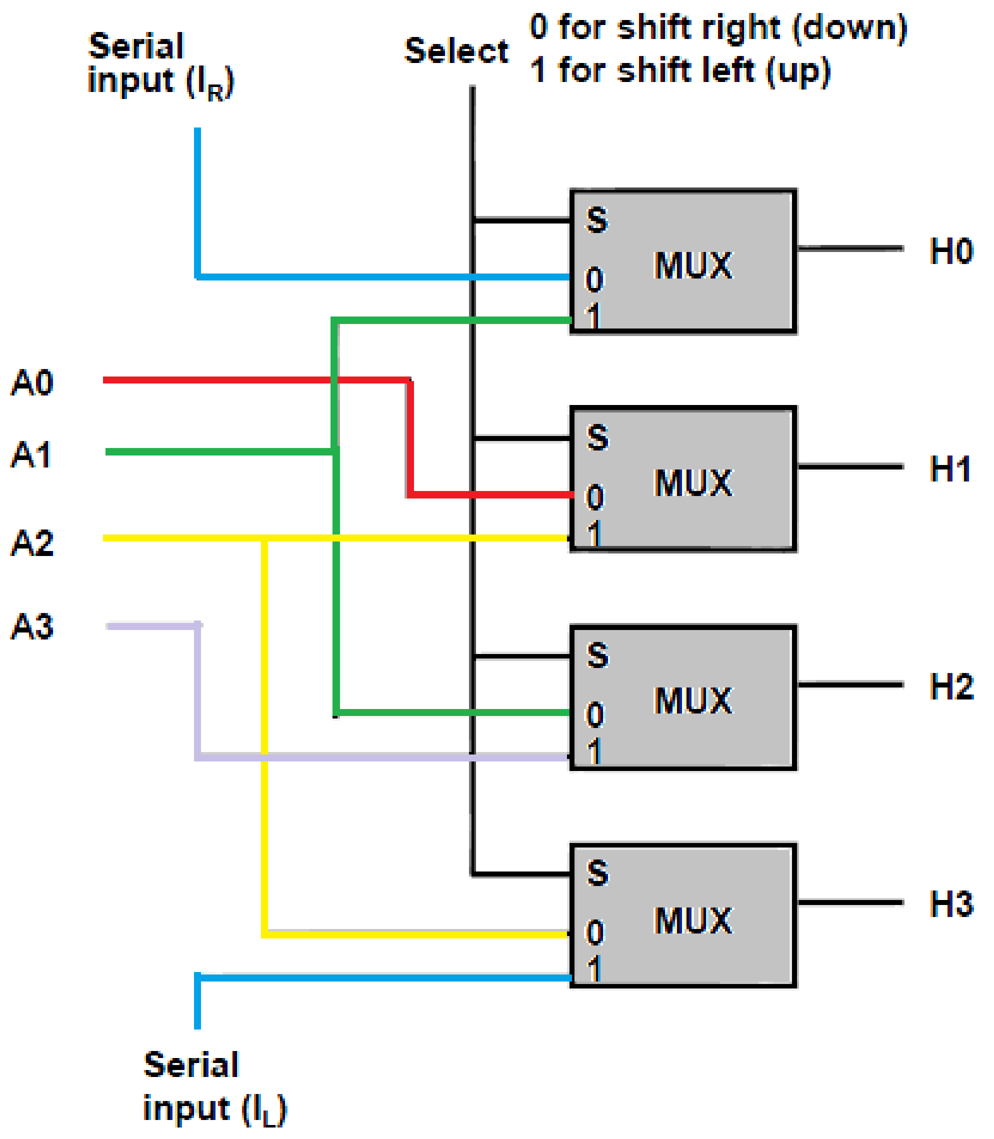
The original number was 127 and the new number is 0. Here the result is not valid because an overflow has occurred. The expected answer 127x2 = 254, so it cannot be represented by an 8-bit sign pattern.

HARDWARE IMPLEMENTATION 4-BIT COMBINATIONAL CIRCUIT SHIFTER



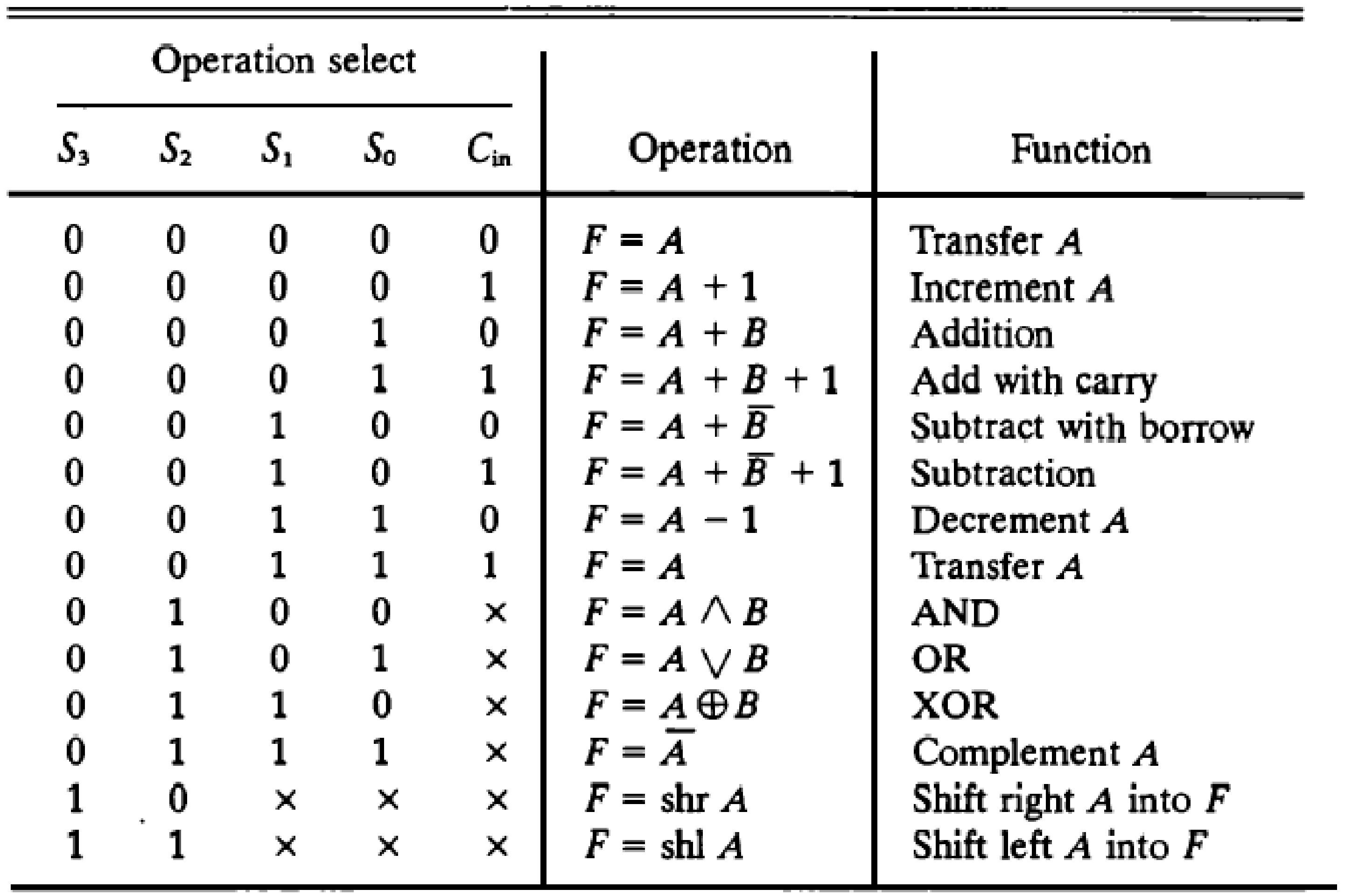
FUNCTIONAL TABLE

SELECT	OUTPUT			
S	H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

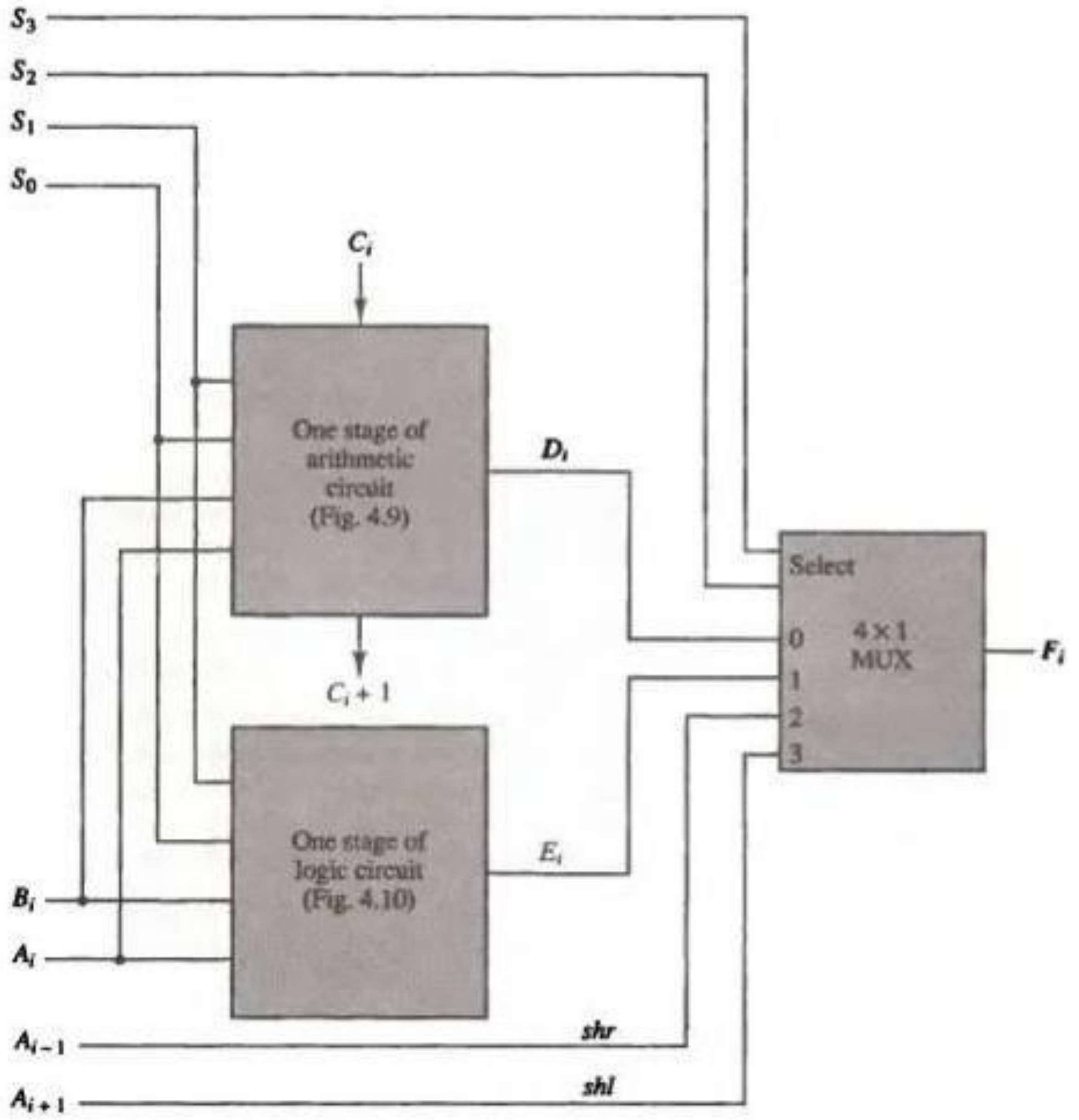


ARITHMETIC LOGIC SHIFT UNIT

Function Table for Arithmetic Logic Shift Unit



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	×	$F = A \wedge B$	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	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F


S_3
S_2
S_1
S_0
C_i
One stage of arithmetic circuit (Fig. 4.9)
D_i
Select
4×1 MUX
0
1
2
3
F_i
$C_i + 1$
One stage of logic circuit (Fig. 4.10)
E_i
B_i
A_i
A_{i-1}
shr
A_{i+1}
shl

Most things in life that are worth a big pay off are worth the wait and the investment of hard work. We know you are willing to sacrifice and be patient.