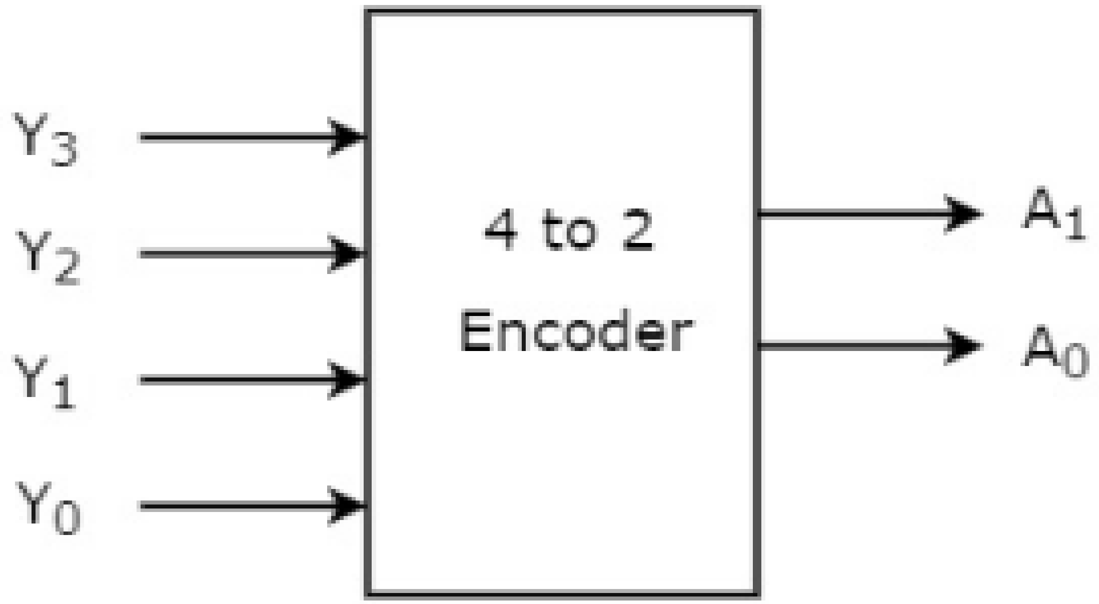
Encoders

An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

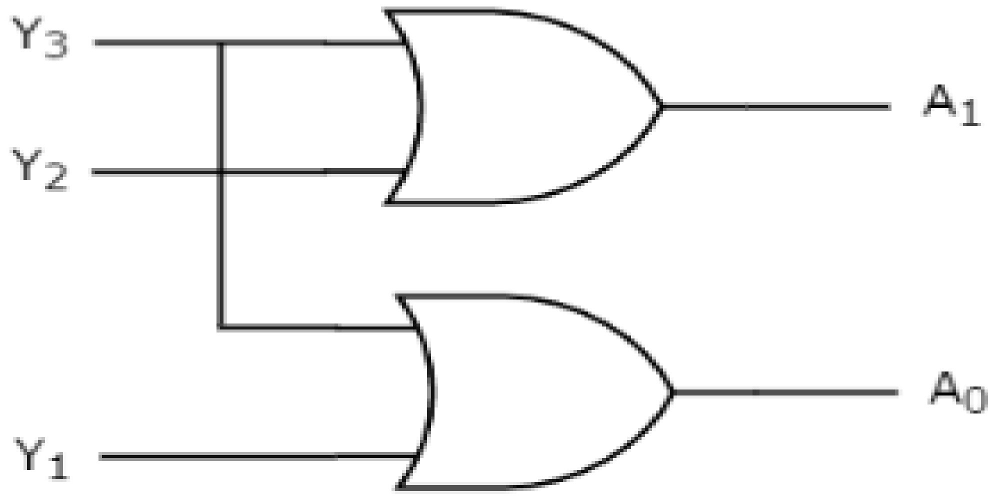
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

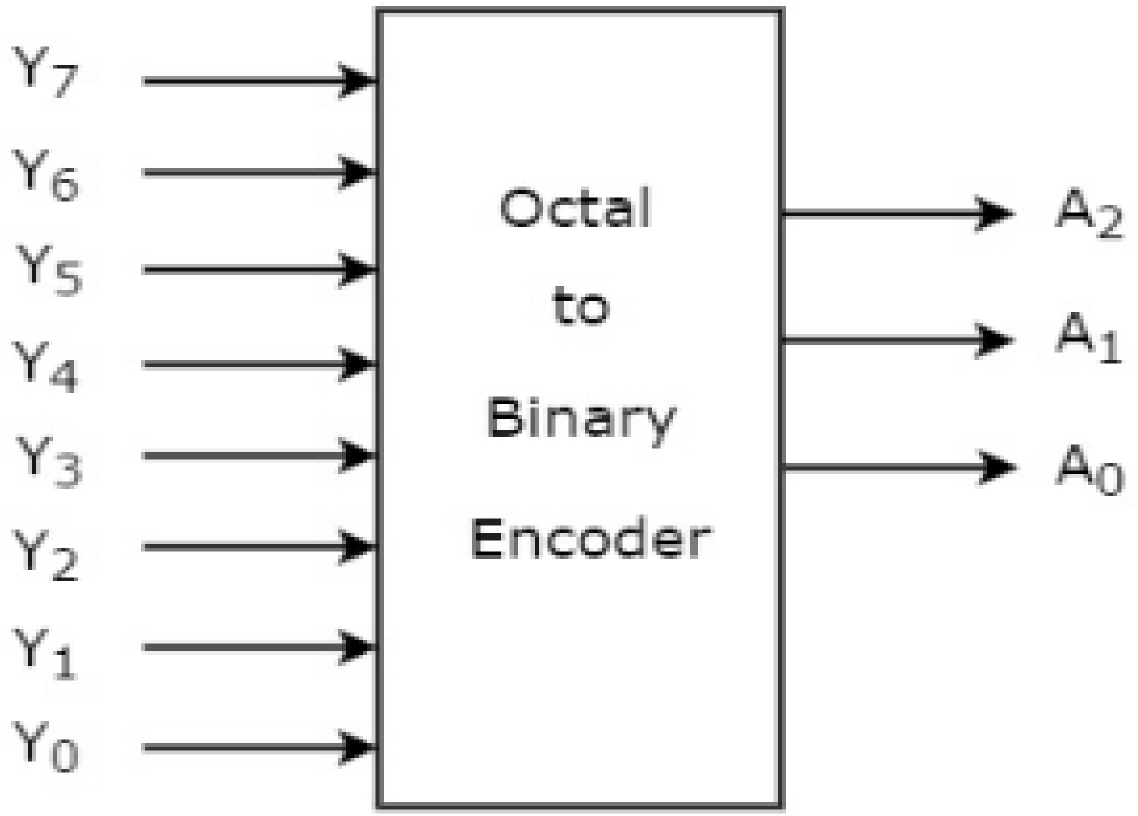
We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.



At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs								Outputs		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

From Truth table, we can write the **Boolean functions** for each output as

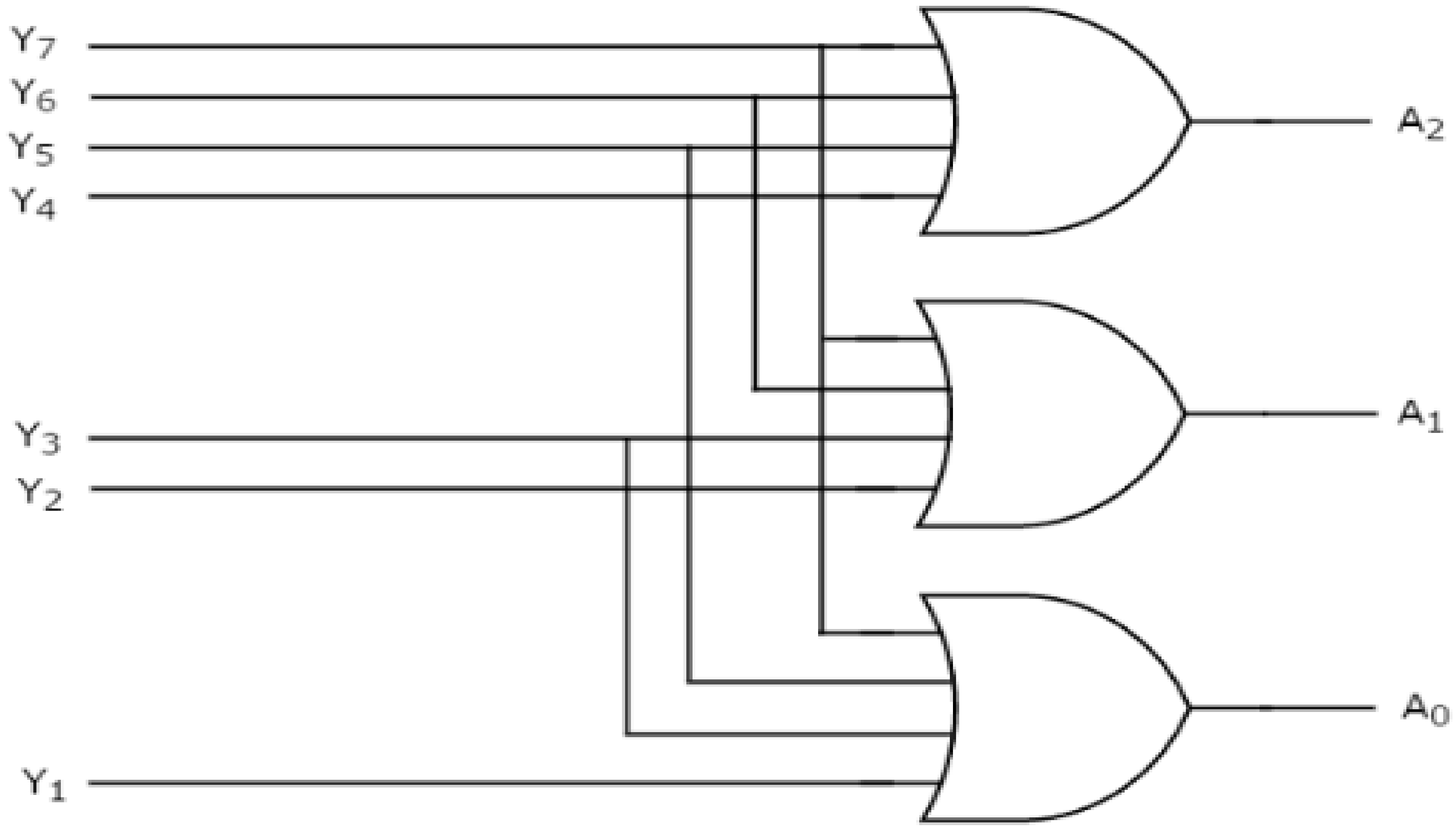
$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.

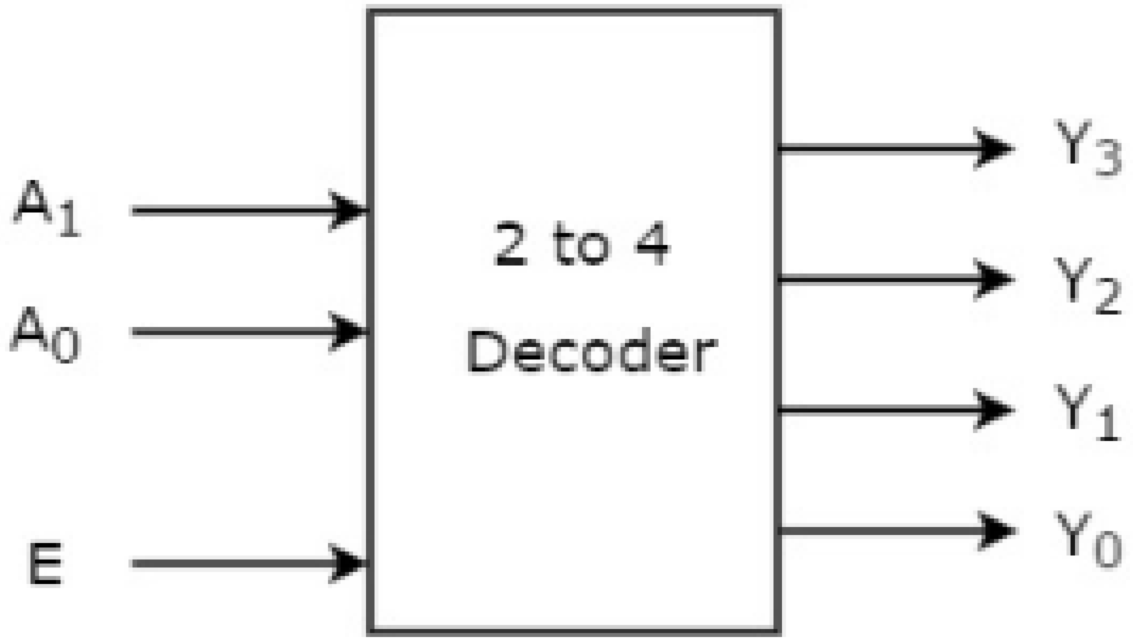


The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of 'n' input variables $lines$, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’. The **Truth table** of 2 to 4 decoder is shown below.

One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the Boolean functions for each output as

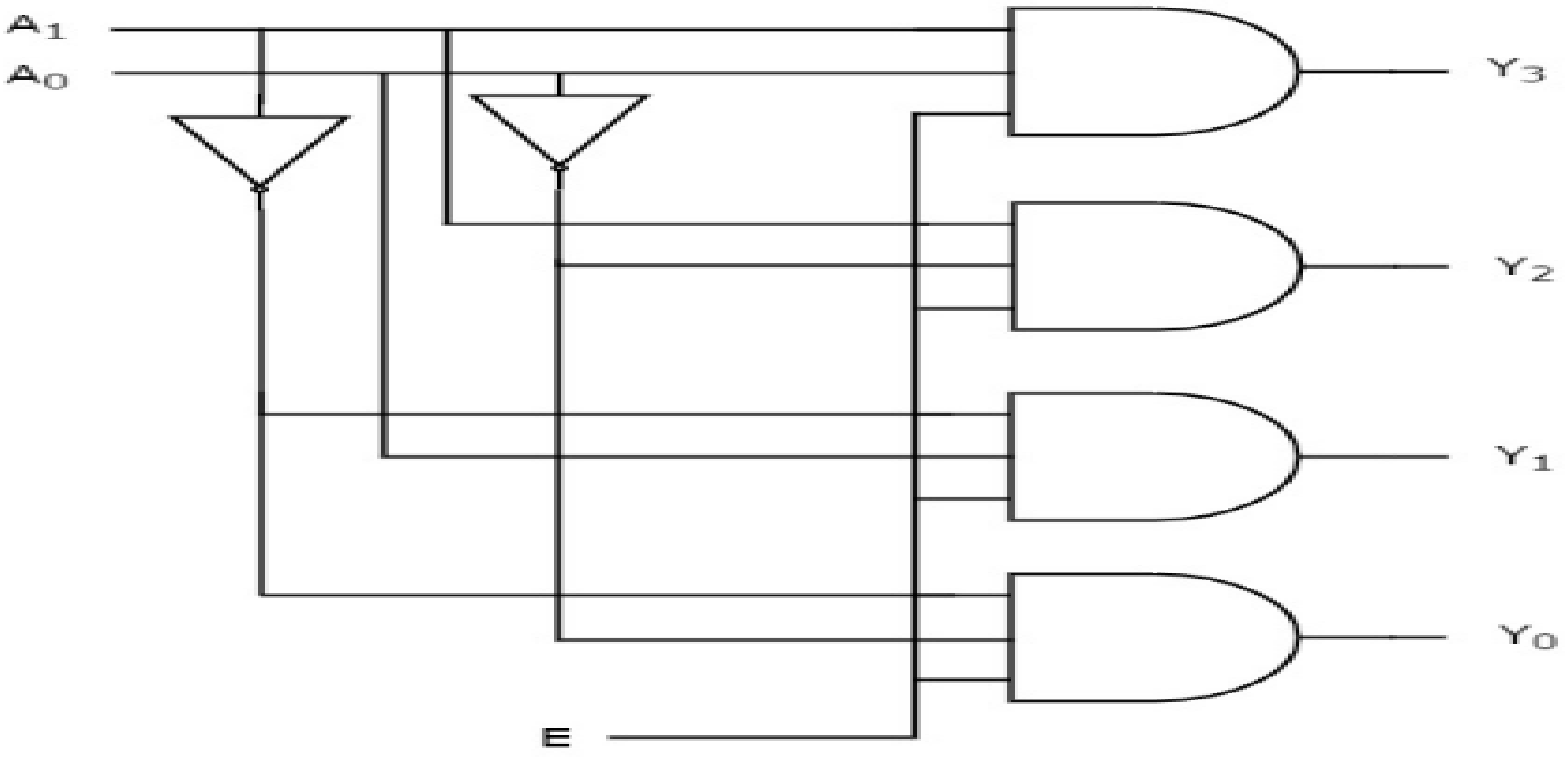
$$Y_3 = E.\, A_1 .\, A_0$$

$$Y_2 = E.\, A_1 .\, A_0'$$

$$Y_1 = E.\, A_1' .\, A_0$$

$$Y_0 = E.\, A_1' .\, A_0'$$

Each output is having one product term. So, there are four product terms in total. We can implemen these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



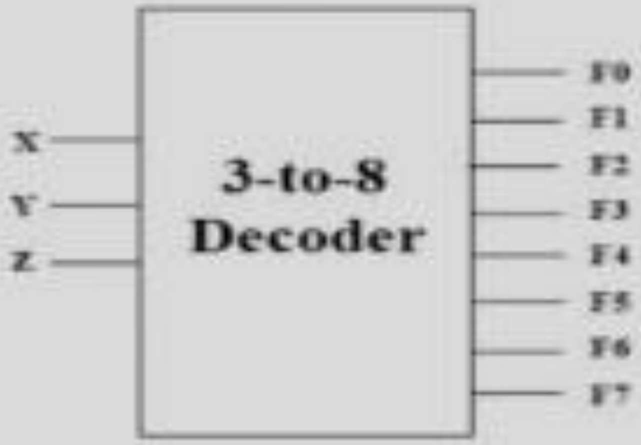
Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

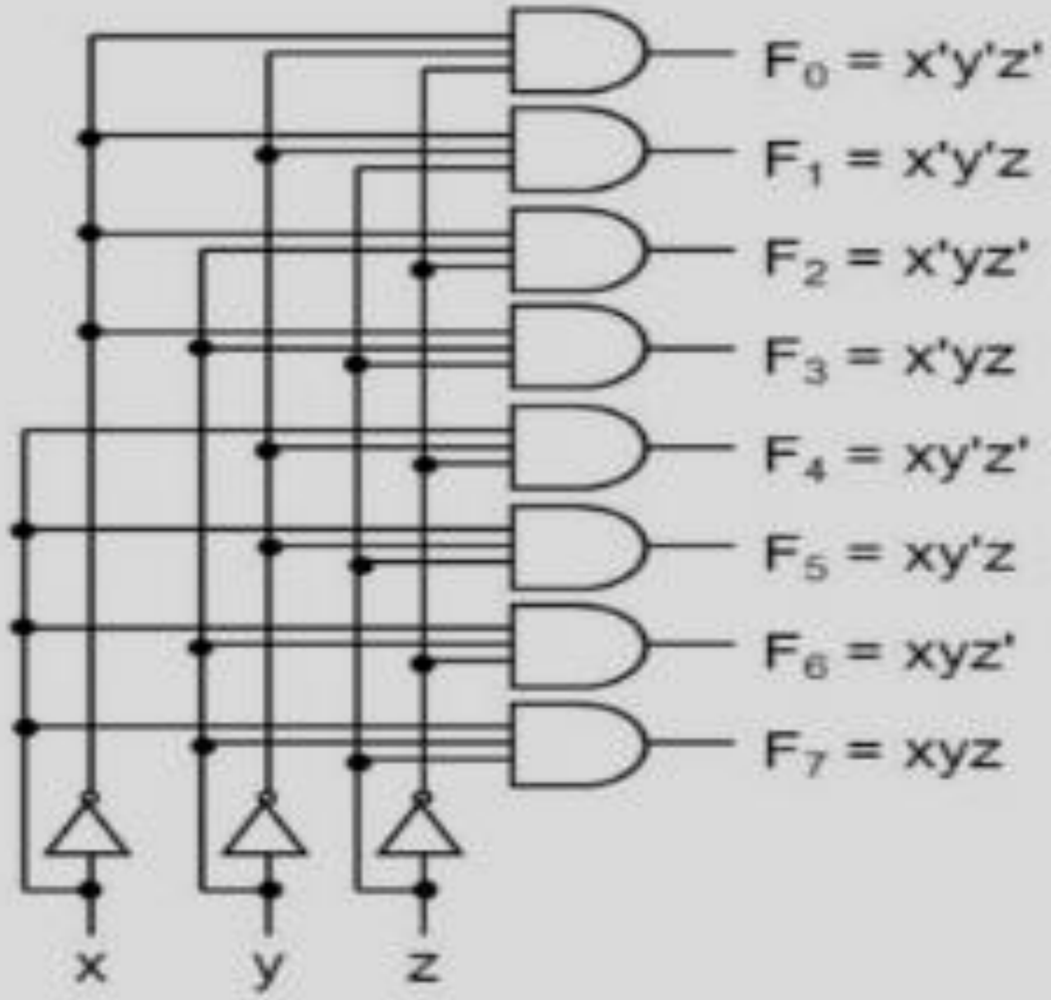
Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

3-to-8 Binary Decoder

Truth Table:

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



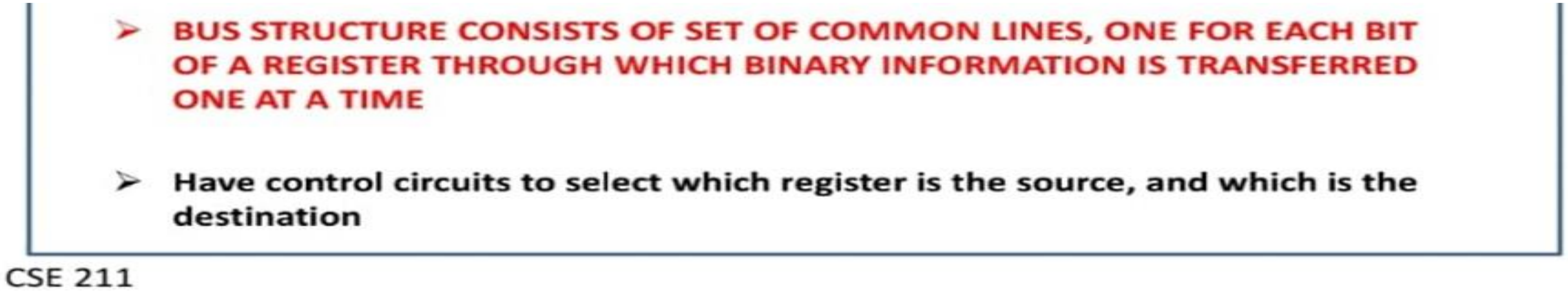


Bus and Memory transfer

- ❖ 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.

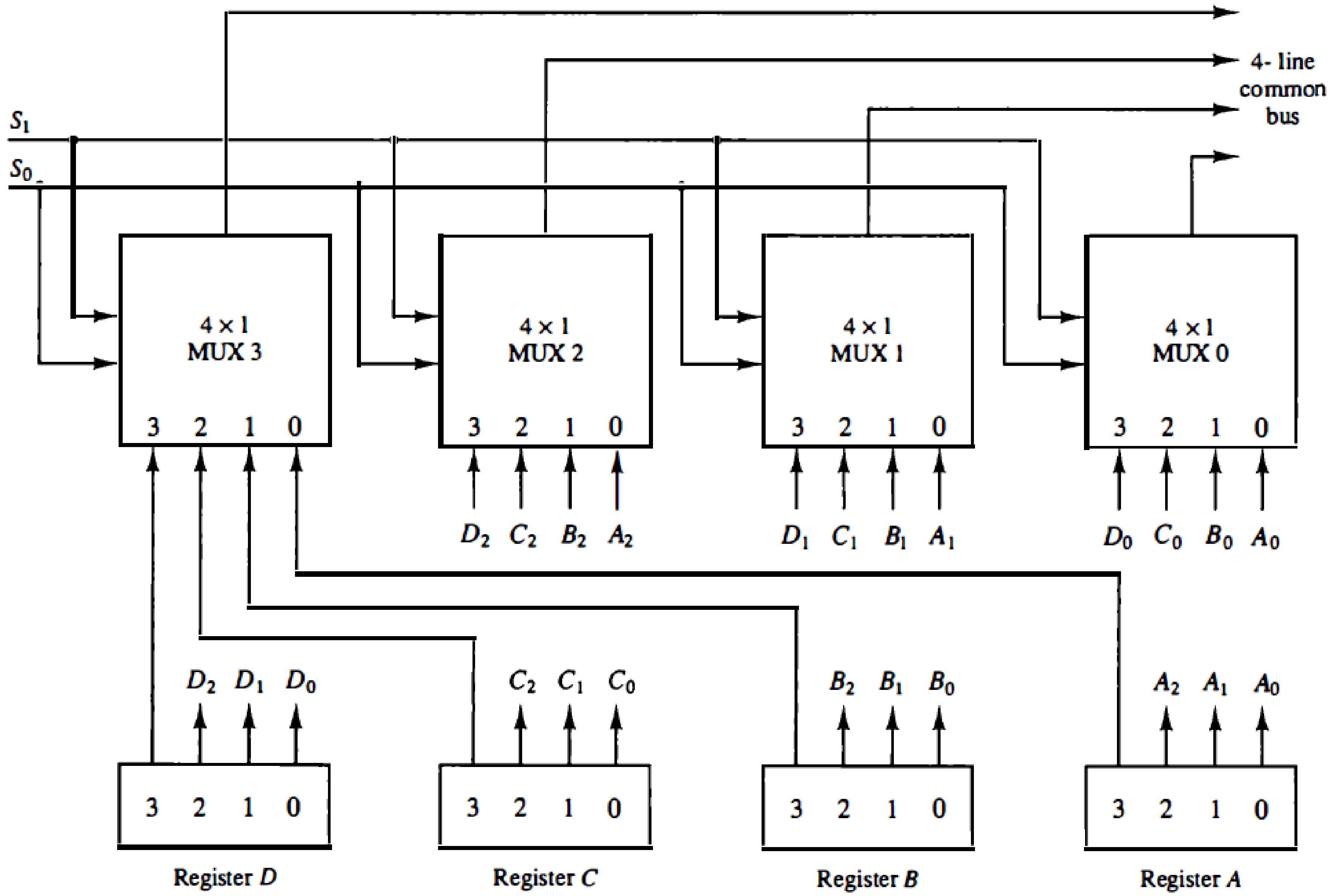
- ❖ A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system.

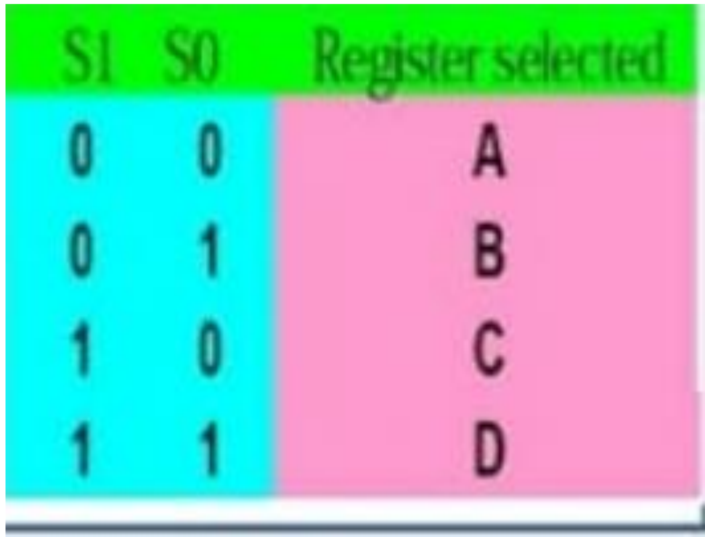


- ➢ BUS STRUCTURE CONSISTS OF SET OF COMMON LINES, ONE FOR EACH BIT OF A REGISTER THROUGH WHICH BINARY INFORMATION IS TRANSFERRED ONE AT A TIME

- ➢ **Have control circuits to select which register is the source, and which is the destination**

Figure 4-3 Bus system for four registers.





S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Discussions

- 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.
- 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.
- In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers.

Discussions

- For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A1.

- 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.

- Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

Discussions

- 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. 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 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

Discussions

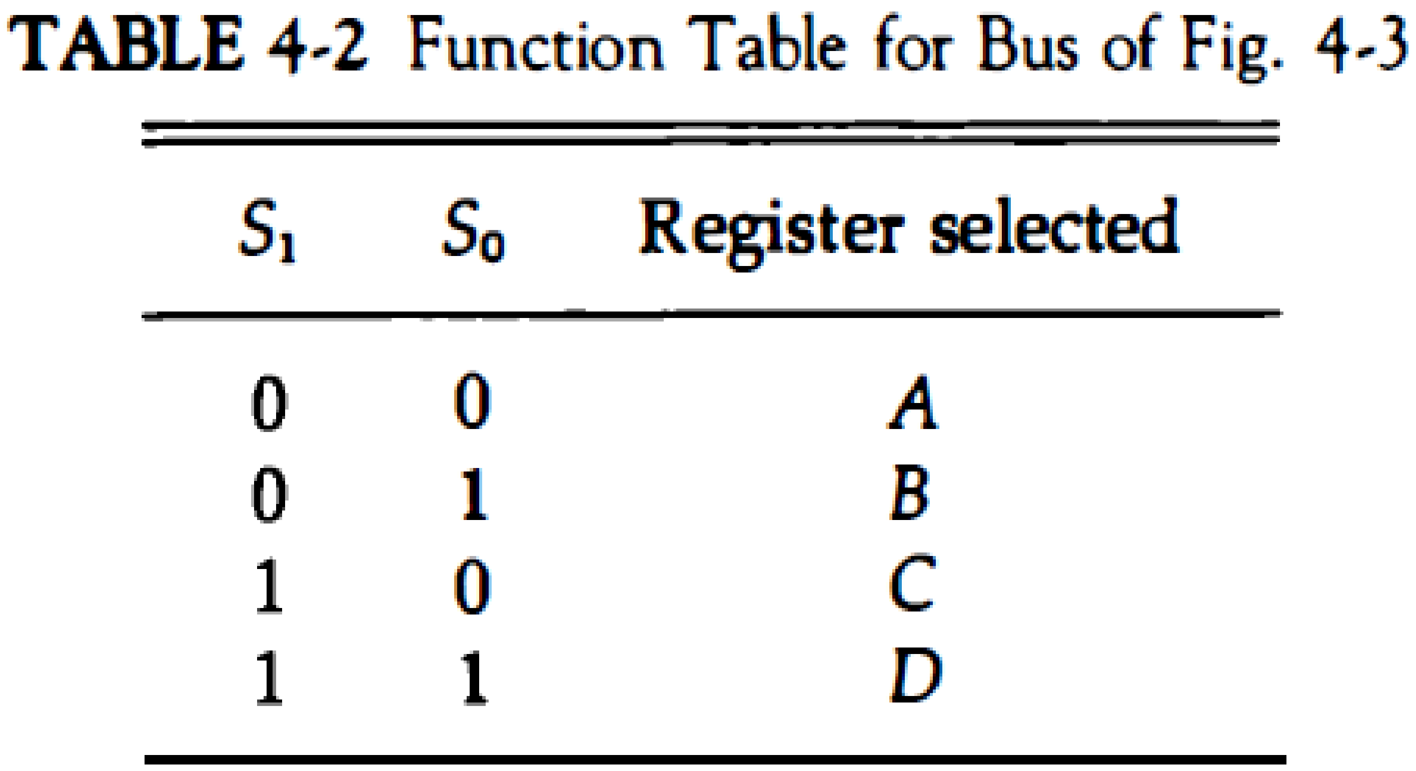
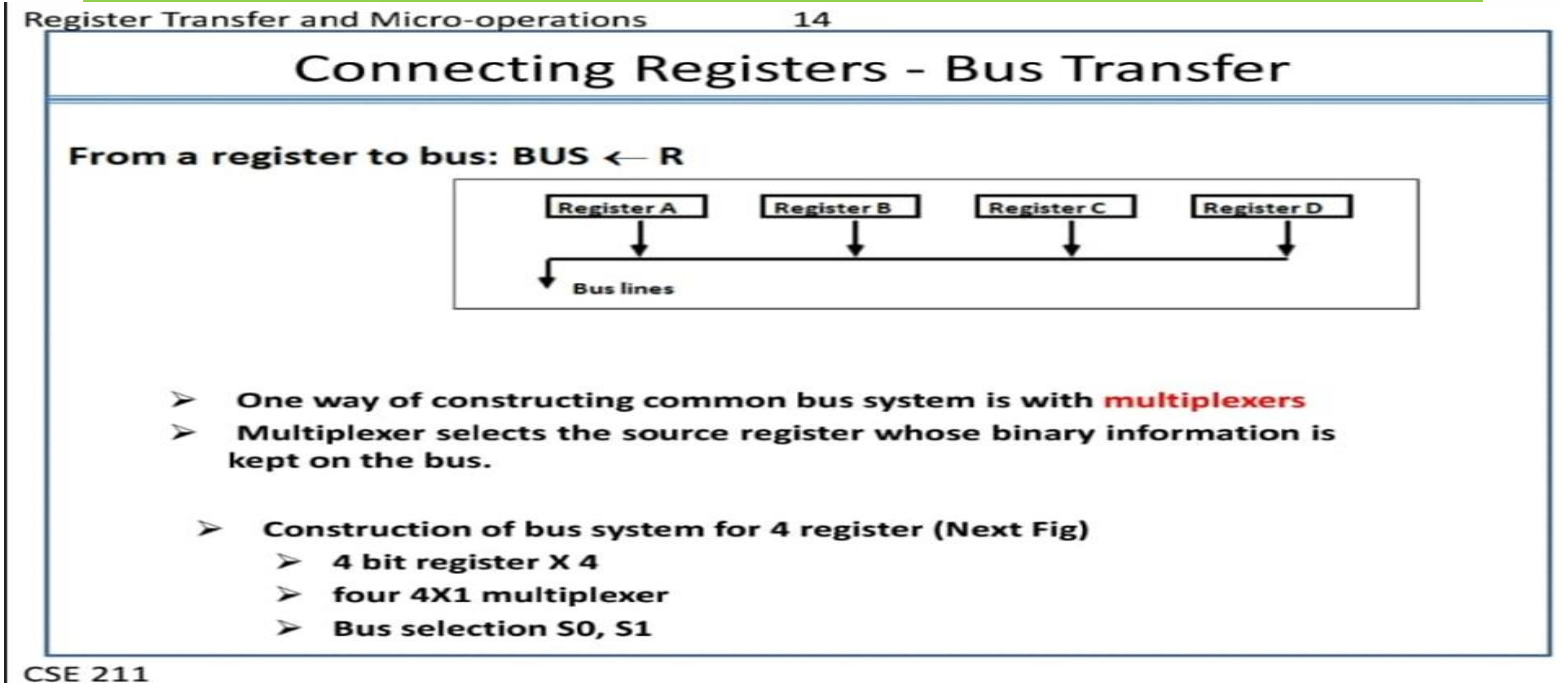


TABLE 4-2 Function Table for Bus of Fig. 4-3

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

Bus and Memory transfer



Connecting Registers - Bus Transfer

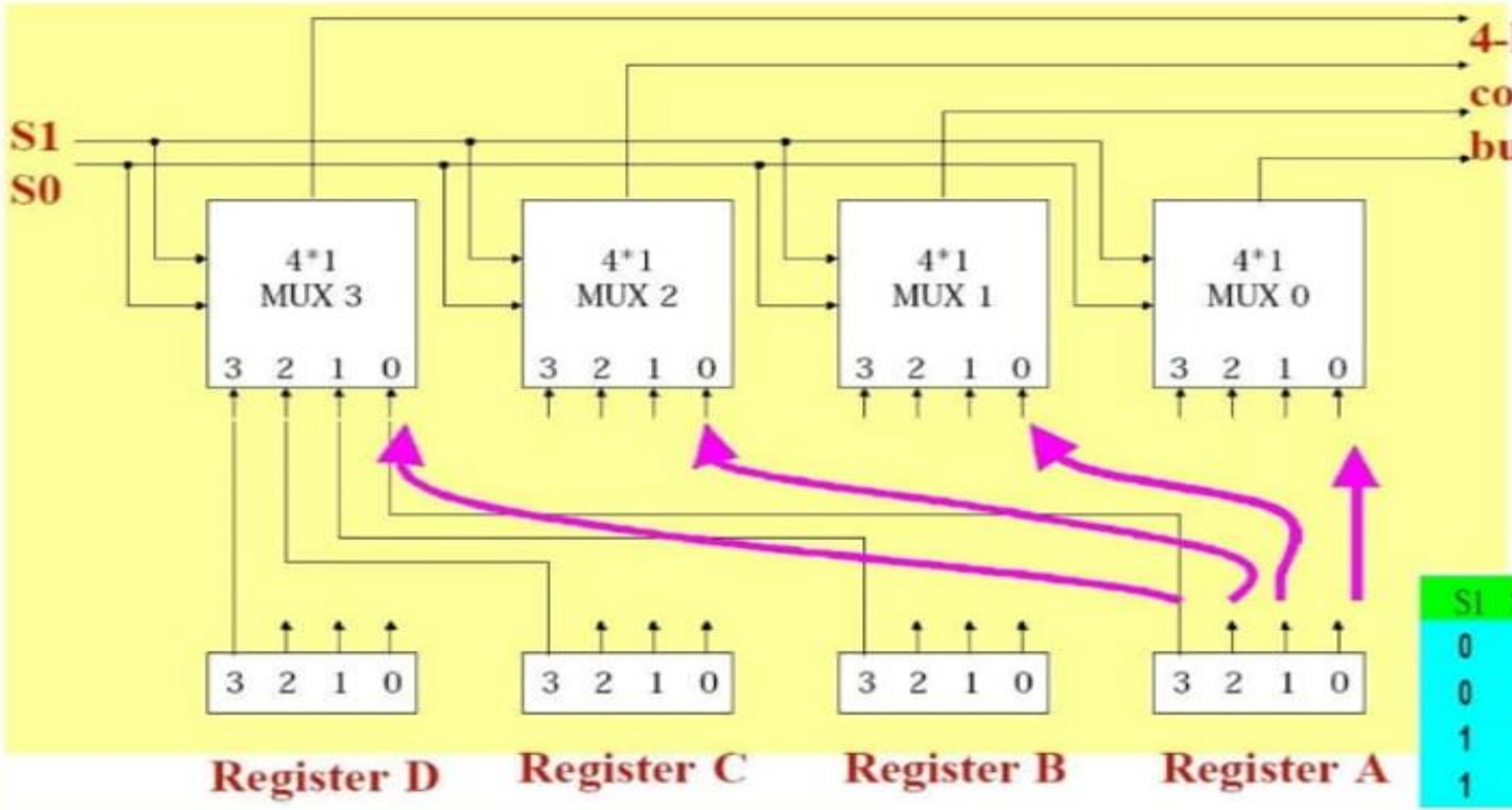
From a register to bus: BUS ← R

Register A
Register B
Register C
Register D
Bus lines

- **One way of constructing common bus system is with multiplexers**
- **Multiplexer selects the source register whose binary information is kept on the bus.**

- **Construction of bus system for 4 register (Next Fig)**
  - **4 bit register X 4**
  - **four 4X1 multiplexer**
  - **Bus selection S0, S1**

Connecting Registers - Bus Transfer



S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Connecting Registers - Bus Transfer

- For a bus system to multiplex k registers of n bits each
  - No. of multiplexer = n
  - Size of each multiplexer = k x 1

Discussions

- The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected.

- 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.

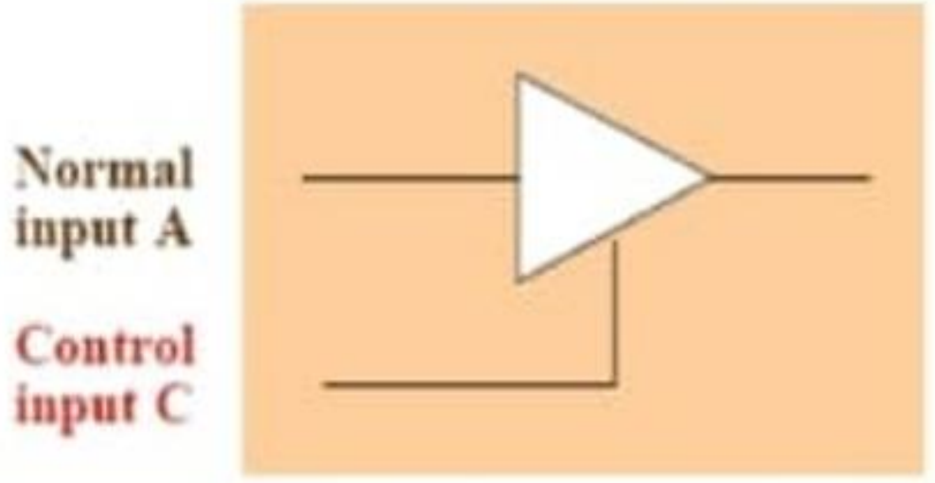
Three-State Bus Buffers

- A bus system can be constructed with three-state gates.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.

Three-State Bus Buffers

- However, the one most commonly used in the design of a bus system is the buffer gate.

- Tri-state buffer gate : Fig. 4-4
  - » When control input =1 : The output is enabled(output Y = input A)
  - » When control input =0 : The output is disabled(output Y = high-impedance)



If C=1, Output Y = A

If C=0, Output = High-impedance

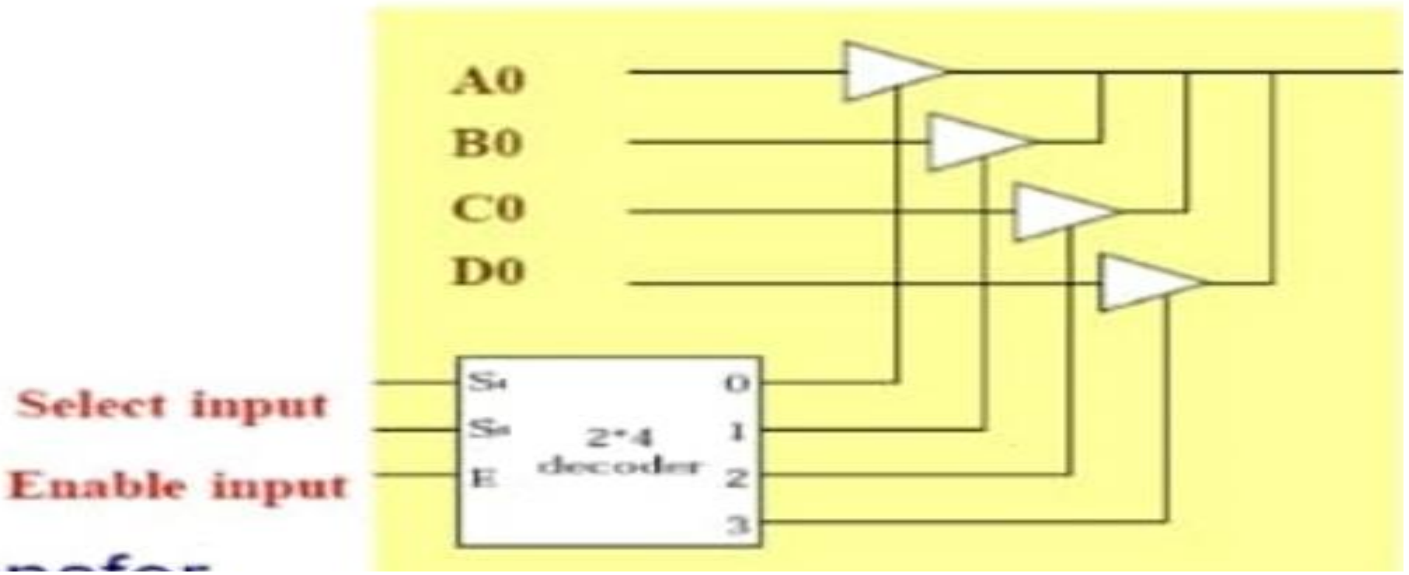
Three-State Bus Buffers

- When the control input 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 is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

Three-State Bus Buffers

- The construction of a bus system with tri-state buffer : *Fig.*
  - The outputs of four buffer are connected together to form a single bus line(Tri-state buffer
  - No more than one buffer may be in the active state at any given time(2 X 4 Decoder
  - To construct a common bus for 4 register with 4 bit : Fig.



Bus line with three state-buffers.

*When E=1 and S0 and S1=00
Then
0 output of decoder will be=1
*1st buffer will be active- due to that A0 will be output.
*Similarly- S0 and S1=11 then
*4th buffer will be active-due to that D0 will be output.

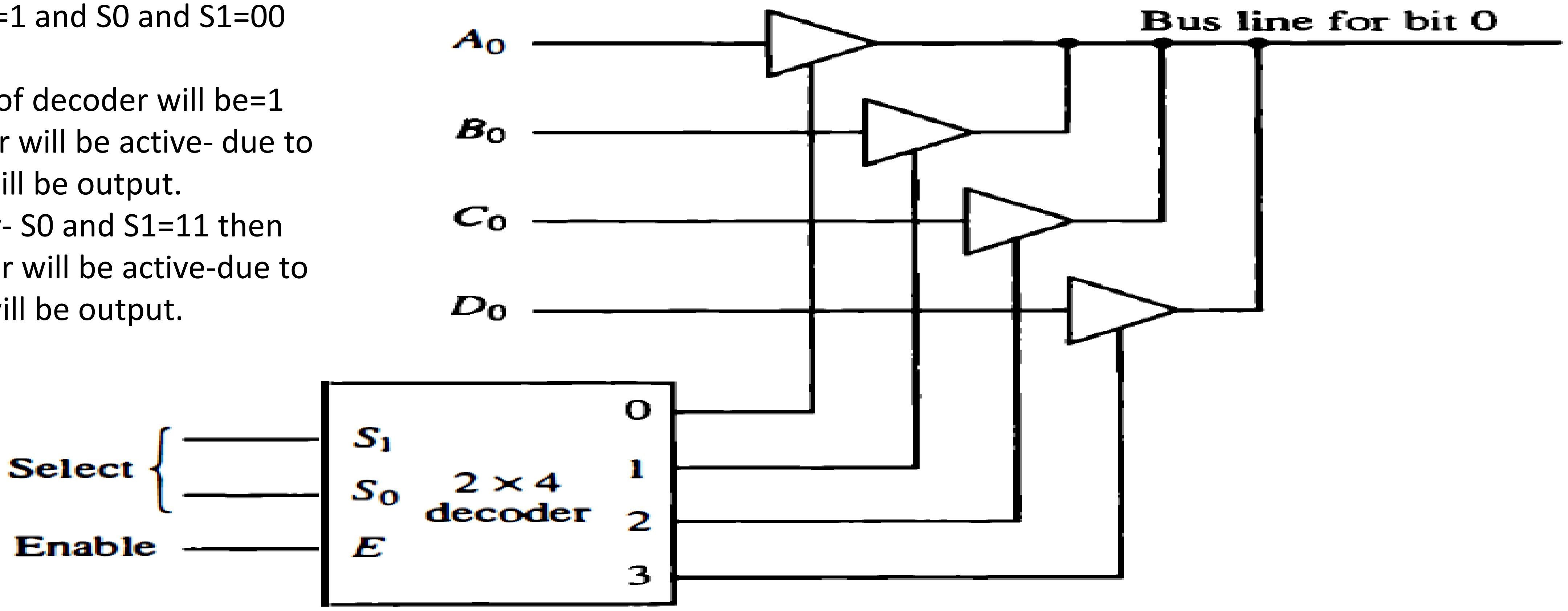


Figure 4-5 Bus line with three state-buffers.

Three-State Bus Buffers

- One way to ensure that no more than one control input is active at any given time is to use a decoder.

- 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.

Three-State Bus Buffers

- One way to ensure that no more than one control input is active at any given time is to use a decoder.

- 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.

Memory Transfer

- The transfer of information from a memory word to the outside environment is called a read 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 . The particular memory word among the many available is selected by the memory address during the transfer.

- 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 .
- Memory stores binary information in groups of bits called as words.

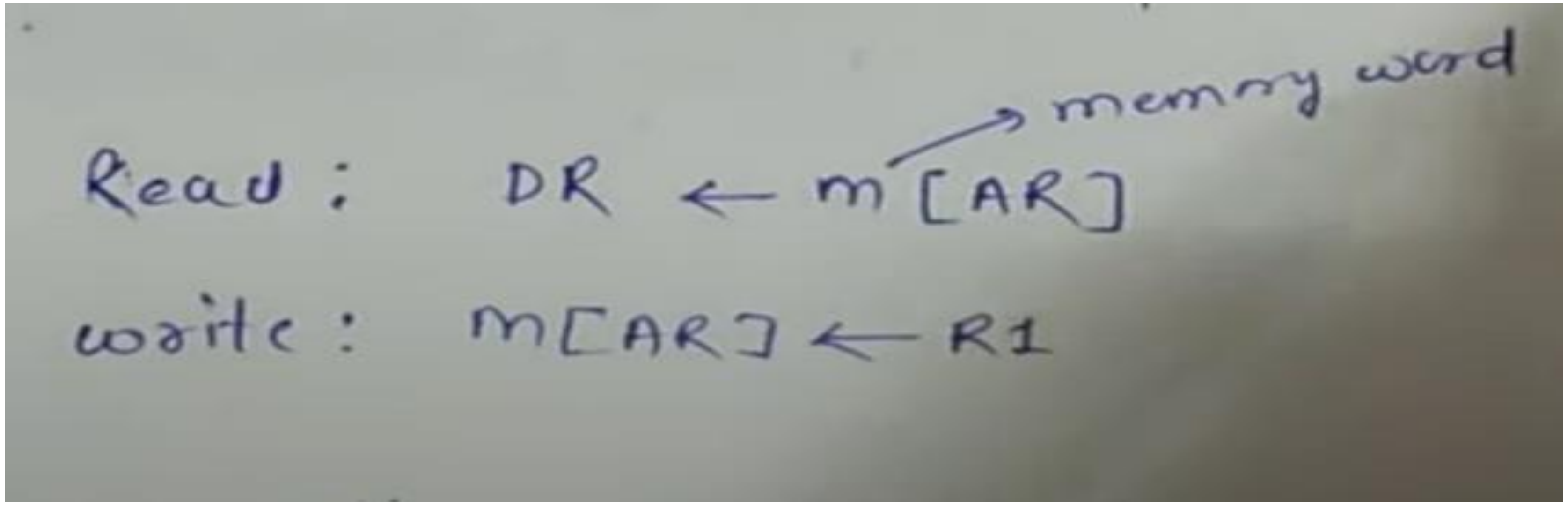
Memory Transfer

memory read

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:

$$\text{Read:} \quad DR \leftarrow M[AR]$$

This causes a transfer of information into *DR* from the memory word *M* selected by the address in *AR*.



Read : $DR \leftarrow m[AR]$ memory word

write : $m[AR] \leftarrow R1$

MICROOPERATIONS

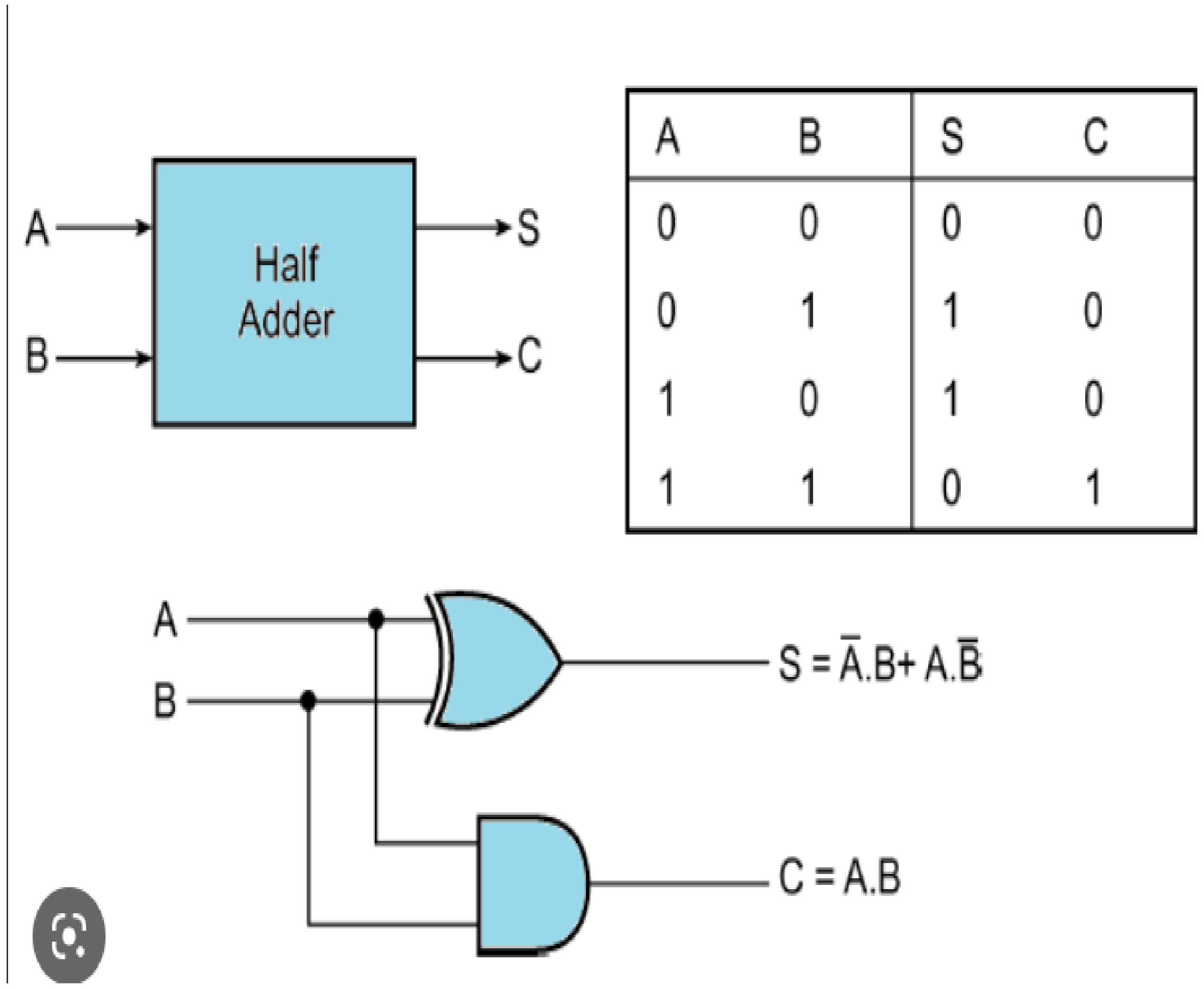
Computer system microoperations are of four types:

- ➢ **Register transfer microoperations**
- ➢ **Arithmetic microoperations**
- ➢ **Logic microoperations**
- ➢ **Shift microoperations**

Summary of Typical Arithmetic Micro-Operations

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

Adder



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

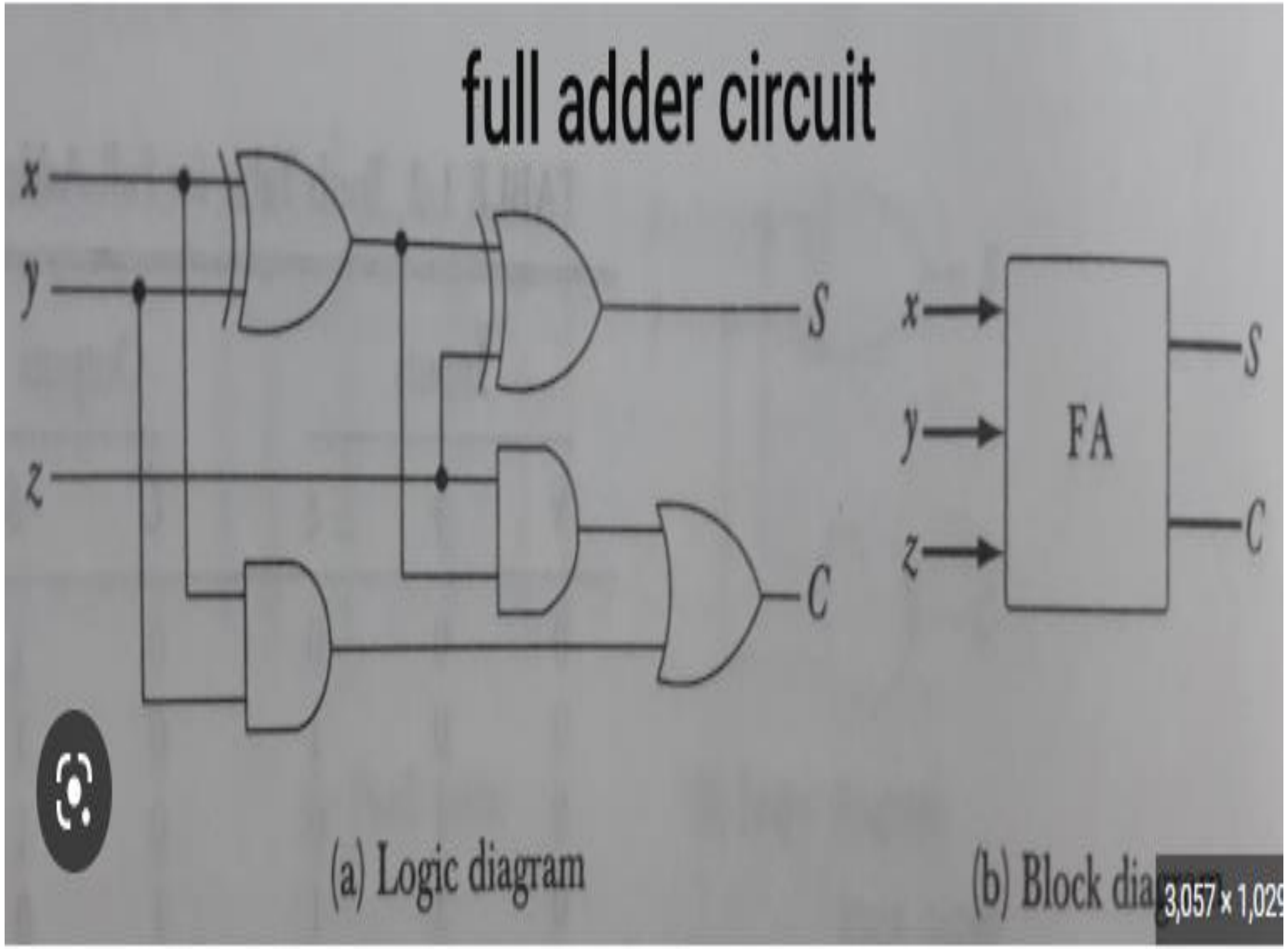
A half adder is a type of adder, an electronic circuit that performs the addition of numbers.

The half adder is able to add two single binary digits and provide the output plus a carry value.

It has two inputs, called A and B, and two outputs S (sum) and C (carry).

The common representation uses a XOR logic gate and an AND logic gate.

Adder

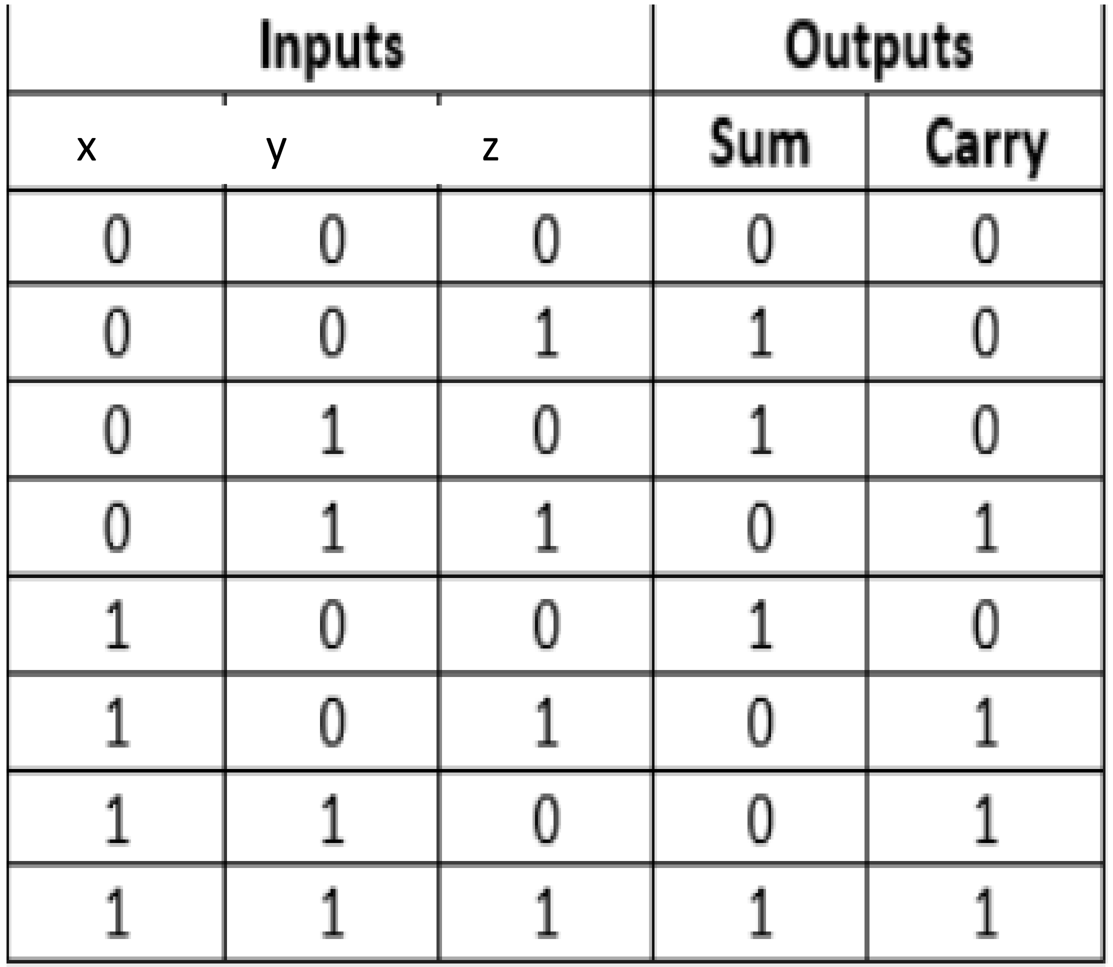


(a) Logic diagram (b) Block diagram

Sum:- x(xor)y(xor)z

Carry = xy+xz+yz

Adder

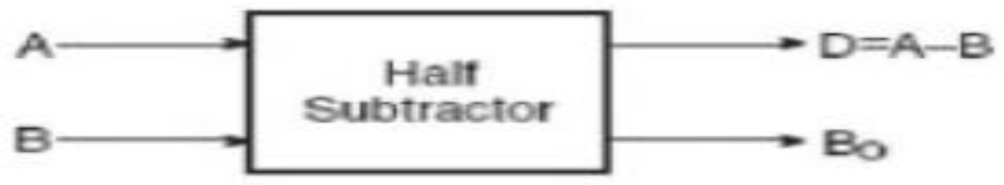


Inputs			Outputs	
x	y	z	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Subtractor

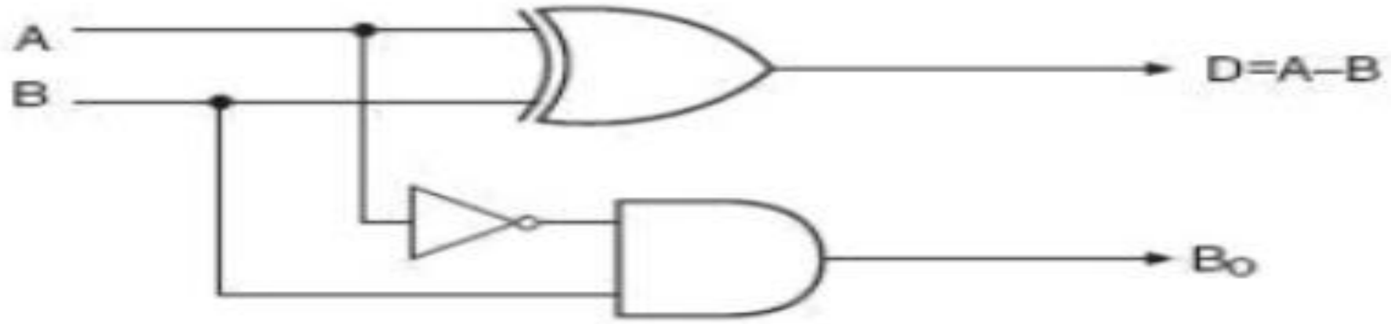
$$D = \overline{A}.B + A.\overline{B}$$

$$B_o = \overline{A}.B$$

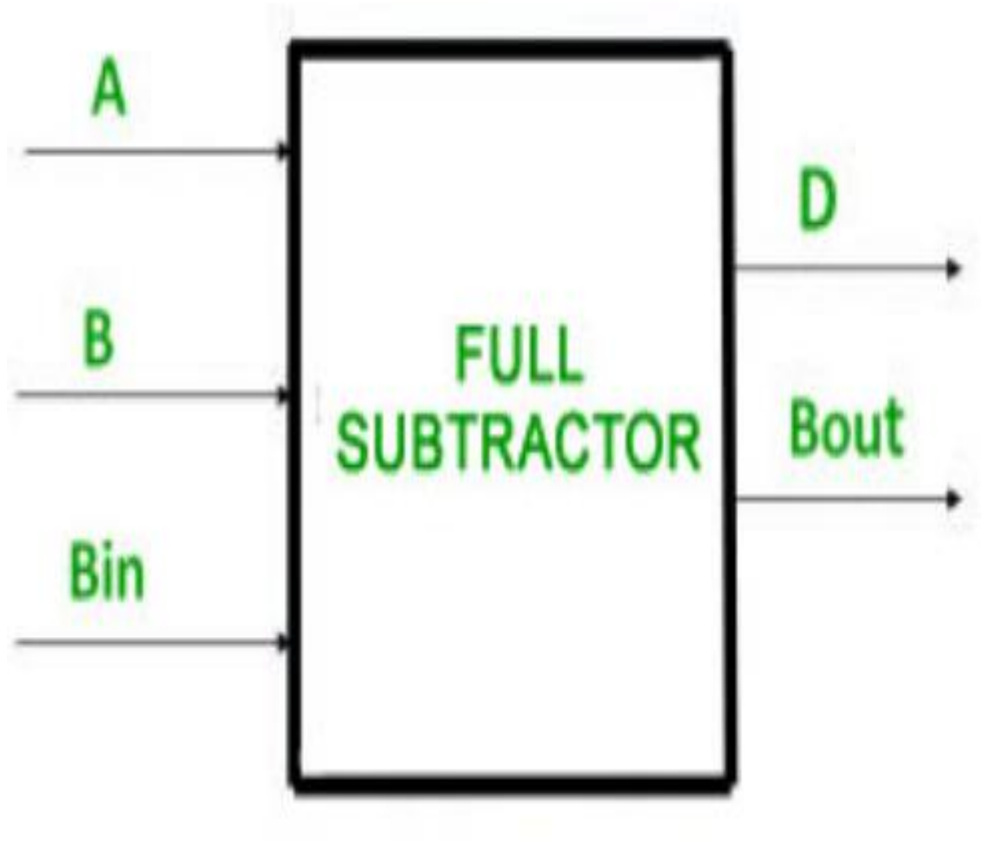


A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor



Subtractor



Logical expression for difference –

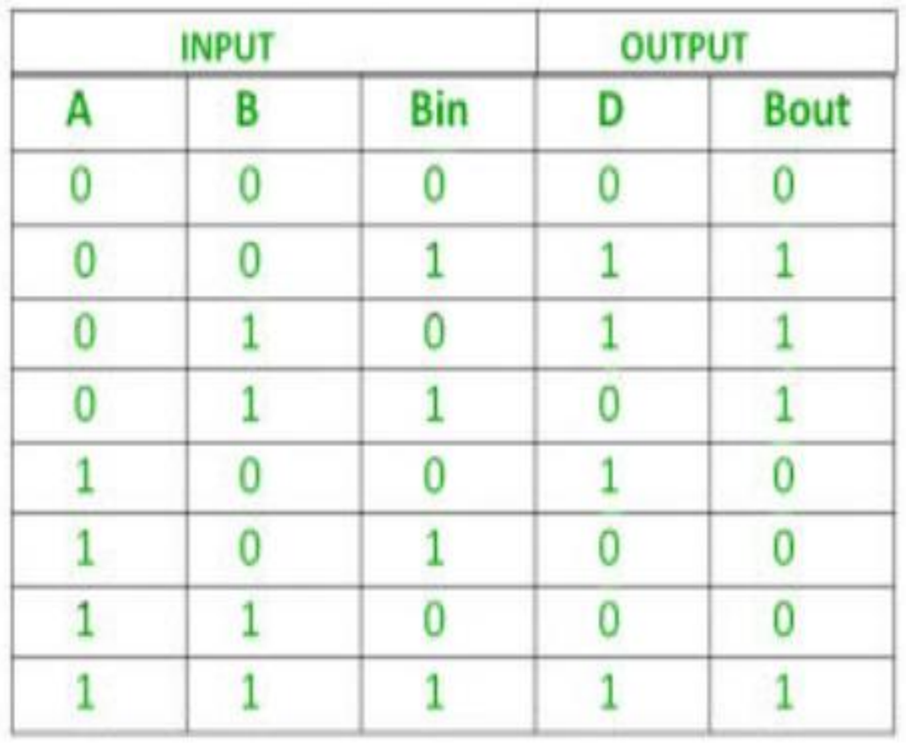
```
= (A XOR B) XOR Bin
```

Logical expression for borrow –

```
= A’Bin + A’B + BBin
```

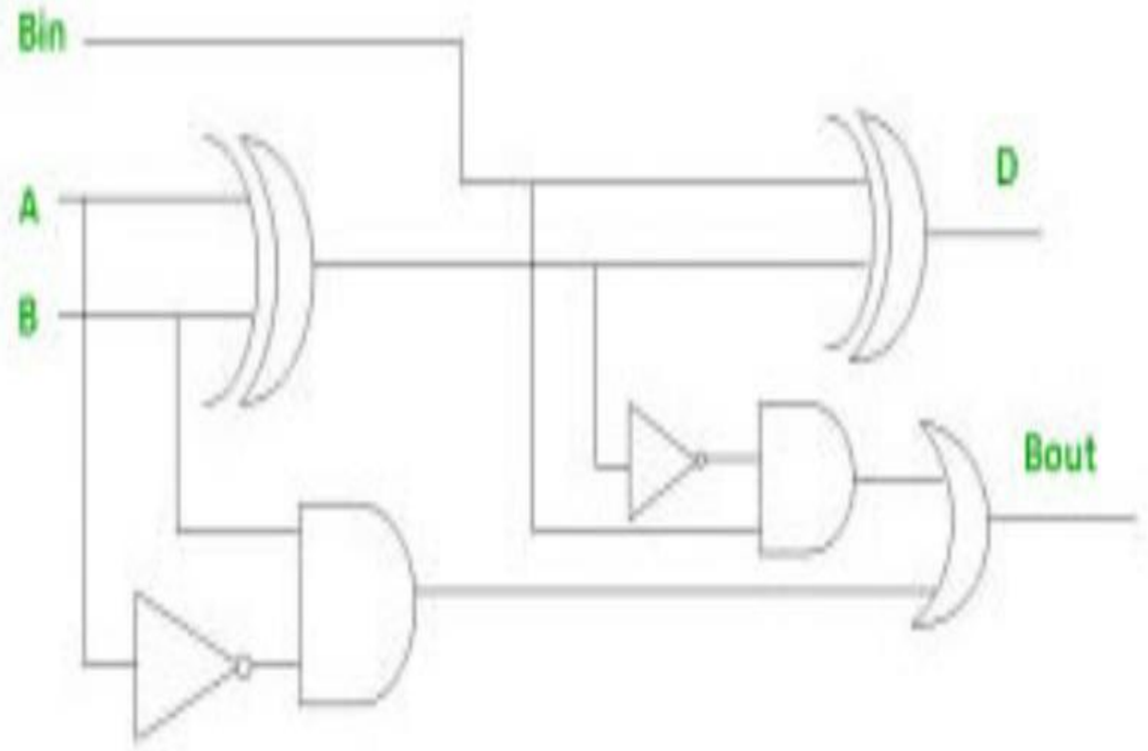
Subtractor

Truth Table –



INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Logic Circuit for Full Subtractor –



4-5 Logic Microoperations

Logic microoperations 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 microoperation with the contents of two registers $R1$ and $R2$ is symbolized by the statement

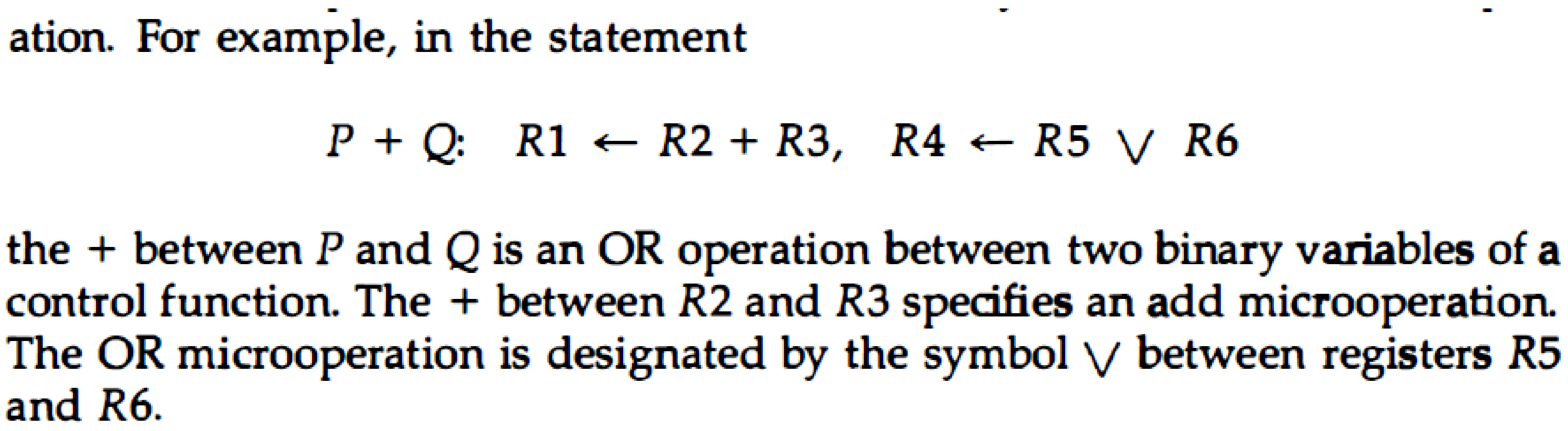
$$P: \quad R1 \leftarrow R1 \oplus R2$$

It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable $P = 1$. As a numerical example, assume that each register has four bits. Let the content of $R1$ be 1010 and the content of $R2$ be 1100. The exclusive-OR microoperation stated above symbolizes the following logic computation:

1010	Content of $R1$
1100	Content of $R2$
0110	Content of $R1$ after $P = 1$

The content of R1 , after the execution of the microoperation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of Rl .

Special symbols will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions. The symbol V will be used to denote an OR microoperation and the symbol $\wedge$ to denote an AND microoperation.



ation. For example, in the statement

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

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 $\vee$ between registers $R5$ and $R6$.

List of Logic Microoperations

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 4-5. In this table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the

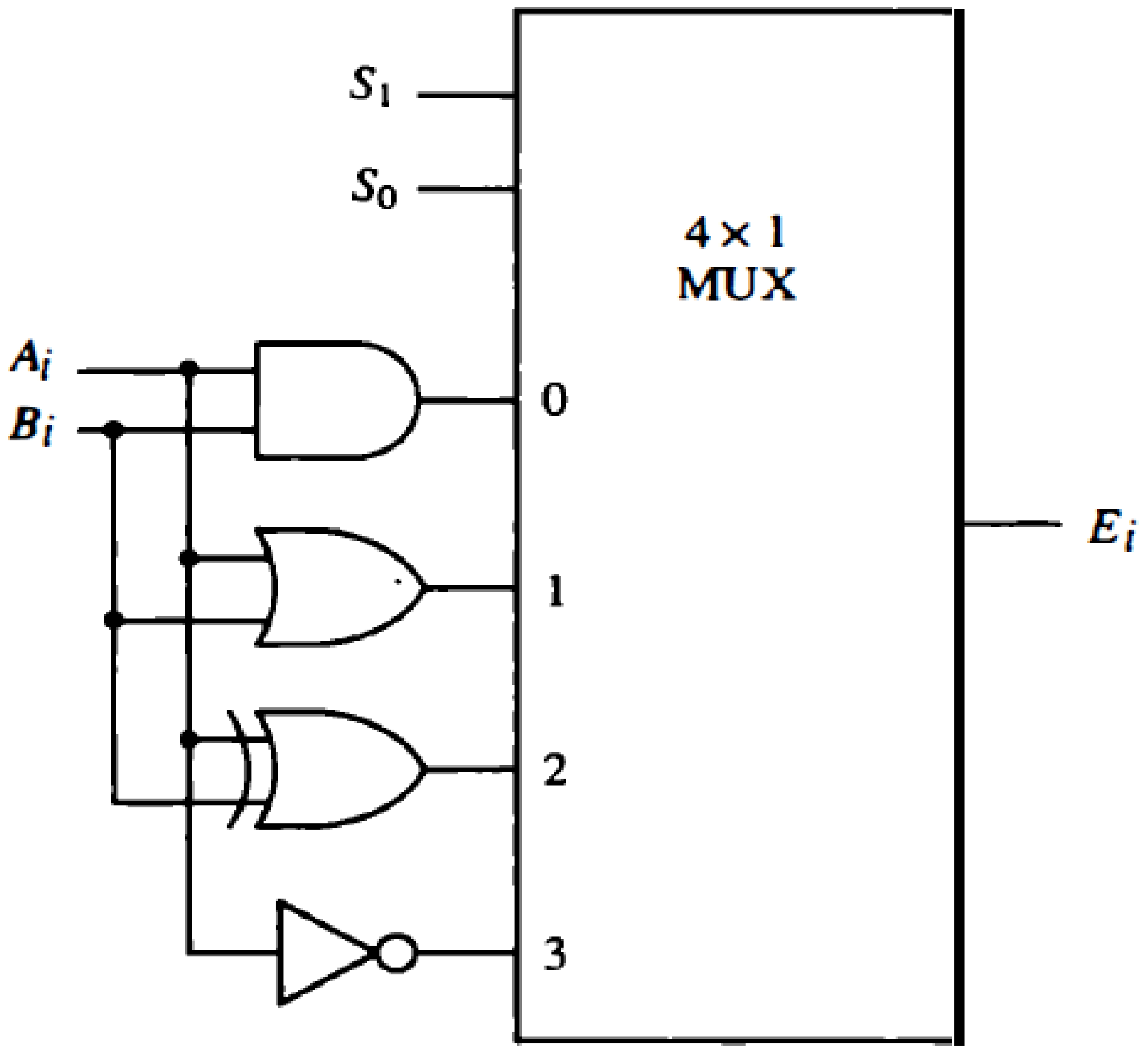
two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

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

The hardware implementation of logic microoperations 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 microoperations, most computers use only four—AND, OR, XOR (exclusive-OR), and complement—from which all others can be derived.



(a) 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	$E = \overline{A}$	Complement

(b) Function table