Computer System Architecture

Ms. Yuzana Hlaing
M.E(IT), Ph.D. Candidate(thesis)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Periods	Lectures
Week-1	Introduction to Computer System Architecture
Week-2	Data Presentations
Week-3	Register Transfer and Microoperations
Week-4	Basic Computer Organization and Design
Week-5	Programming the Basic Computer
Week-6	Microprogrammed Control
Week-7	Central Processing Unit
Week-8	Pipeline and Vector Processing
Week-9	Computer Arithmetic
Week-10	Input-Output Organization
Week-11	Memory Organization
Week-12	Multiprocessors

Lecture-3

Register Transfer and Microoperations

Lecture-3
Register Transfer and Microoperations

- Register Transfer Language
- Register transfer
- Bus and Memory Transfers
- Arithmetic Microoperations
- Logic Microoperations
- Shift Microoperations
- Arithmetic Logic Shift Unit

Register Transfer Language

Register Transfer Language

- A **digital system** is a interconnection of digital hardware modules that accomplish a specific information-processing task.

- The operations executed on data stored in registers are called **microoperations**.

- The symbolic notation used to describe the microoperation transfers among registers is called **register transfer language**.

- The internal hardware organization of a digital computer is defined by specifying
    1. The set of registers it contains and their function.
    2. The sequence of microoperations performed on the binary information stored in the registers.
    3. The control that initiates the sequence of microoperations.

Register Transfer

Register Transfer

- **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.
- Different types of register are defined as follows;
    - Memory Address Register (MAR)
    - Instruction Register (IR)
    - Program Counter (PC)
    - Processor register (R1, R2, etc)

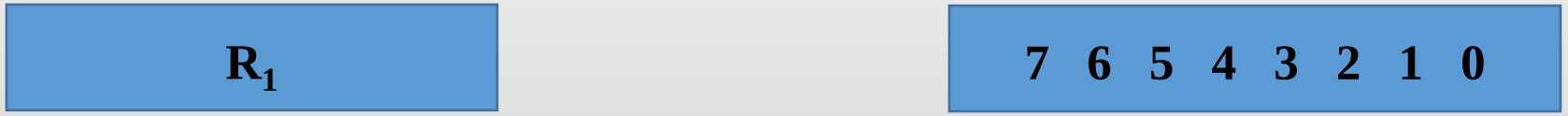
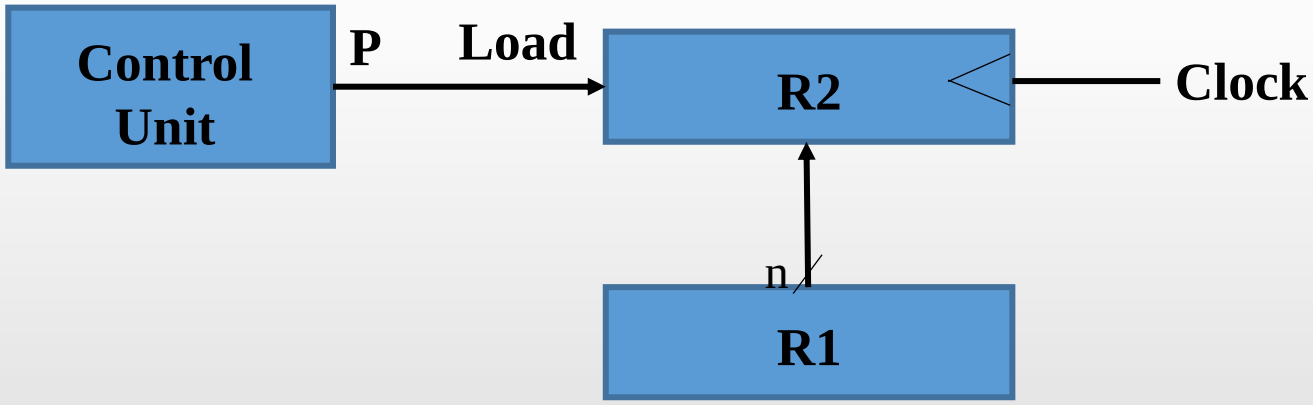


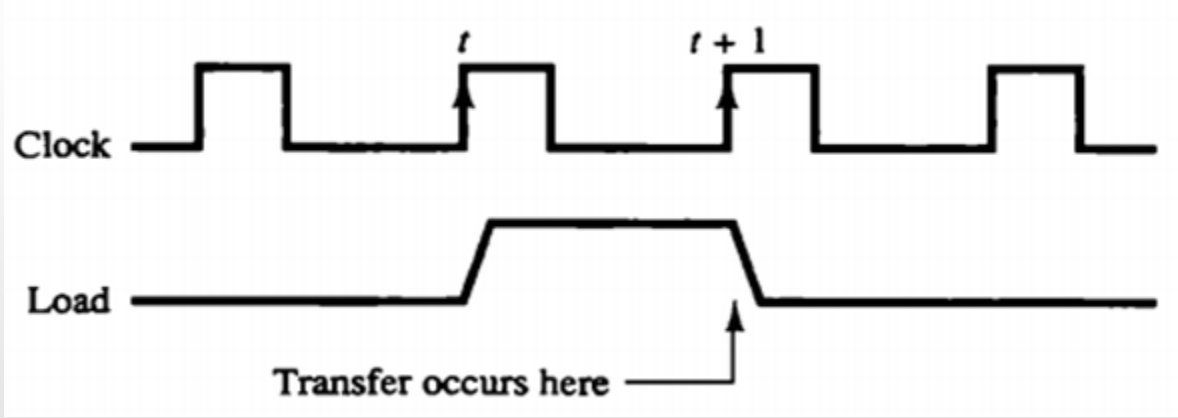
Fig: Block diagram of a register

*** **Example:** Represent the following statement by register transfer statement and show the block diagram with its timing diagram for this statement.
If (P = 1) then (R2 ← R1)

Sol:
P: R2 ← R1



(a) Block diagram



(b) Timing Diagram

*** **Example:** Represent the following statement by register transfer statement and show the block diagram for this statement.
If (y = 1 and T = 1) then (R2 ← R1, R1 ← R2)

Sol:
yT: R2 ← R1, R1 ← R2

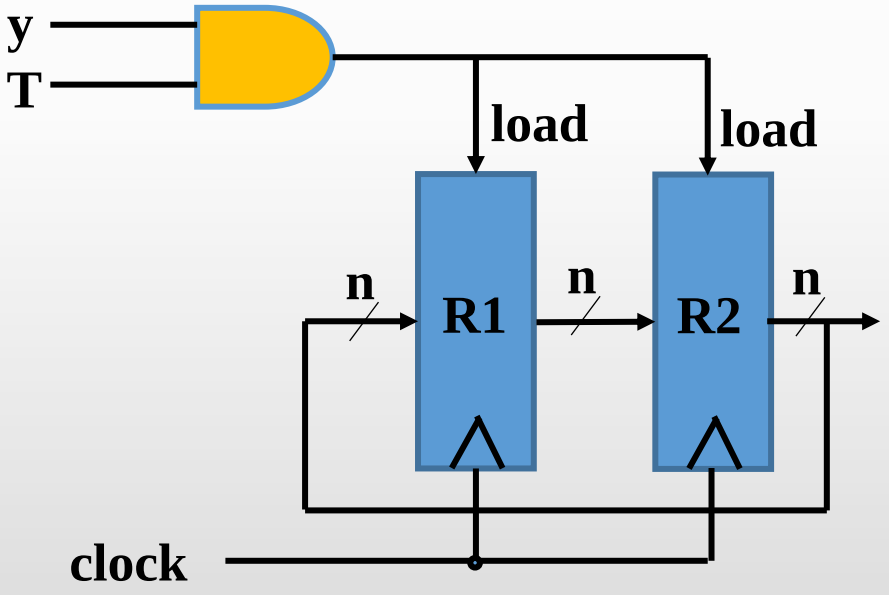


Fig: Logic diagram

Bus and Memory Transfers

Bus Transfer

- 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.

Bus ← C, R1 ← Bus

- If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

R1 ← C

- For example, the bus system for 4-bit four registers by using 1-bit multiplexers with its function table are as shown in the following.
- For 4-bits four registers,
  - **Number of multiplexers = Number of bits in register**
  - **Number of input lines in multiplexer = Number of registers**

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

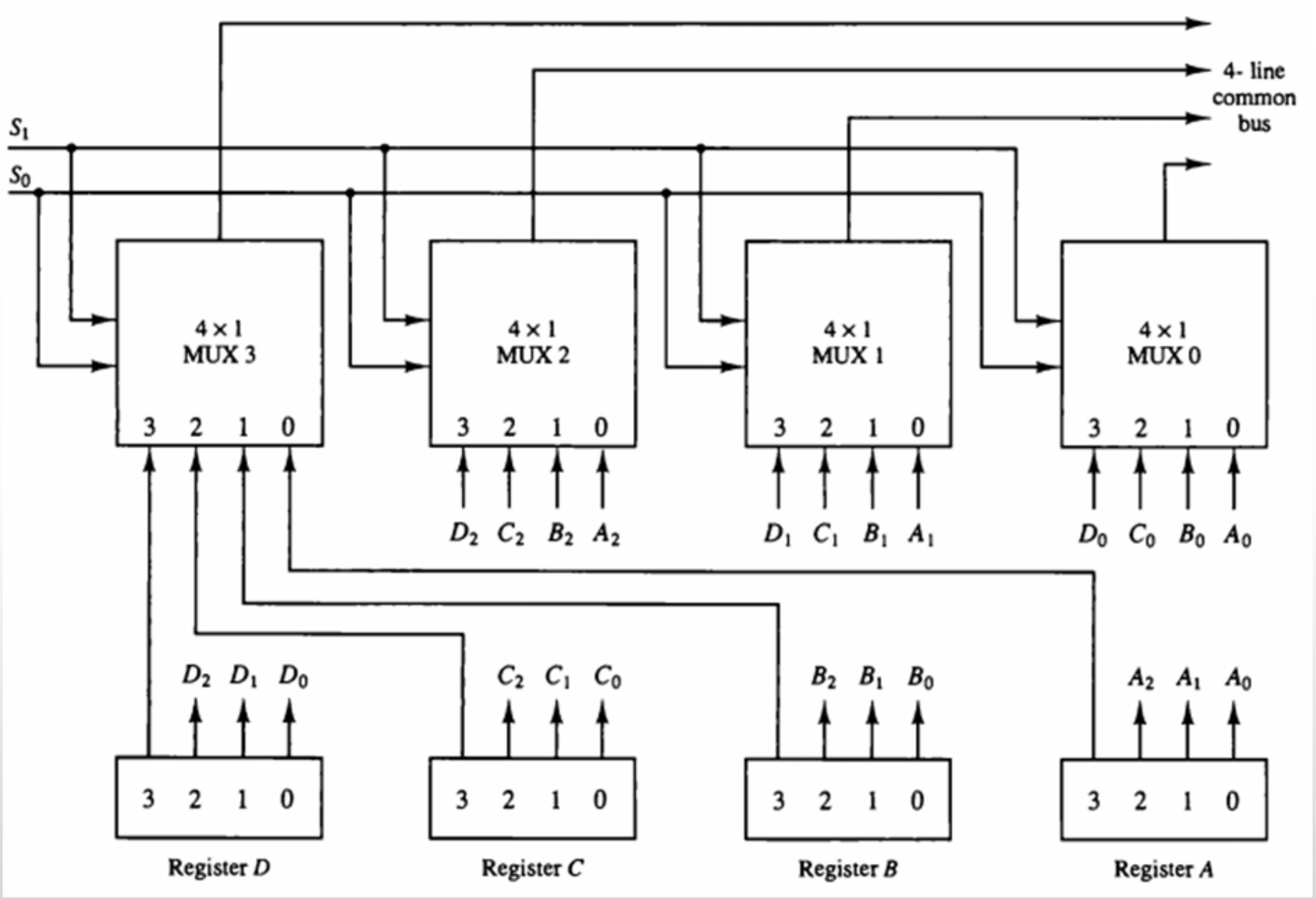


Fig: Bus system for 4-bit four registers

*** **Example:** Design the circuit for register transfers that data of 8-bit four registers R1, R2, R3 and R4 are connected to 8-bit 4x1 multiplexers to the inputs of fifth register R5. The required transfers are dictated by four timing variables T_0 through T_3 as follows.

T_0: R5←R1

T_1: R5←R2

T_2: R5←R3

T_3: R5←R4

Sol:

T_3	T_2	T_1	T_0	S_1	S_0	Y	Load
0	0	0	0	x	x	0	0
0	0	0	1	0	0	R1	1
0	0	1	0	0	1	R2	1
0	1	0	0	1	0	R3	1
1	0	0	0	1	1	R4	1

$S_1 = T_2 + T_3$

$S_0 = T_1 + T_3$

$Load = T_0 + T_1 + T_2 + T_3$

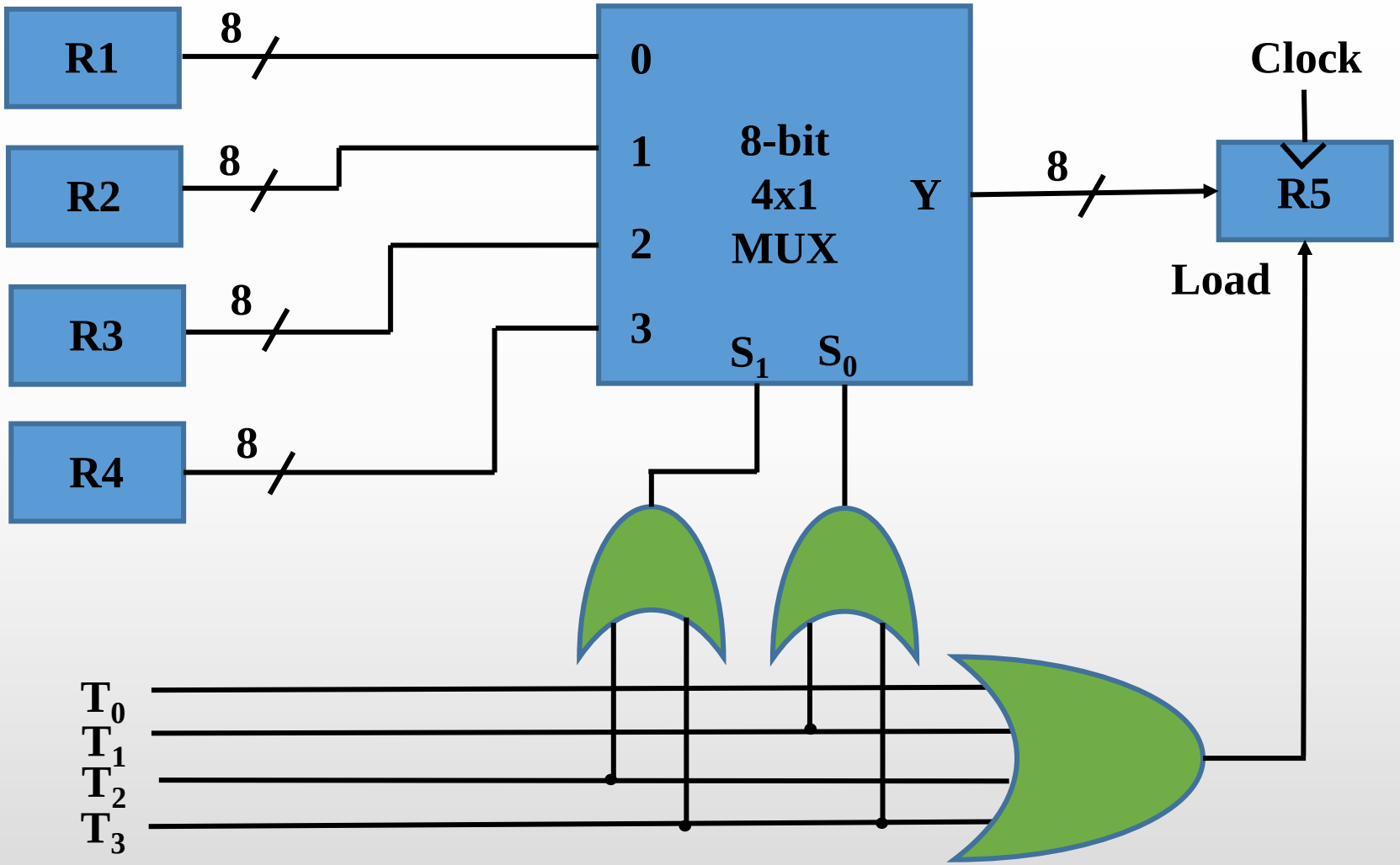
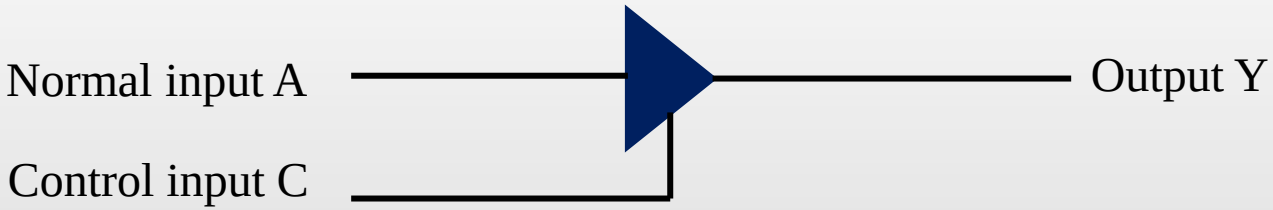


Fig: Block diagram for register transfers

Three-State Bus Buffers

- 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.
- Two of the states are signals equivalent to logic 1 and 0 as in conventional gate.
- The third state is a high-impedance state that behaves like an open circuit.
- The three-state gates may perform any conventional logic such as AND or NAND.



Graphic symbols for three-state buffer

- A bus line for 1-bit four registers can be designed by using decoder and three state-buffers.

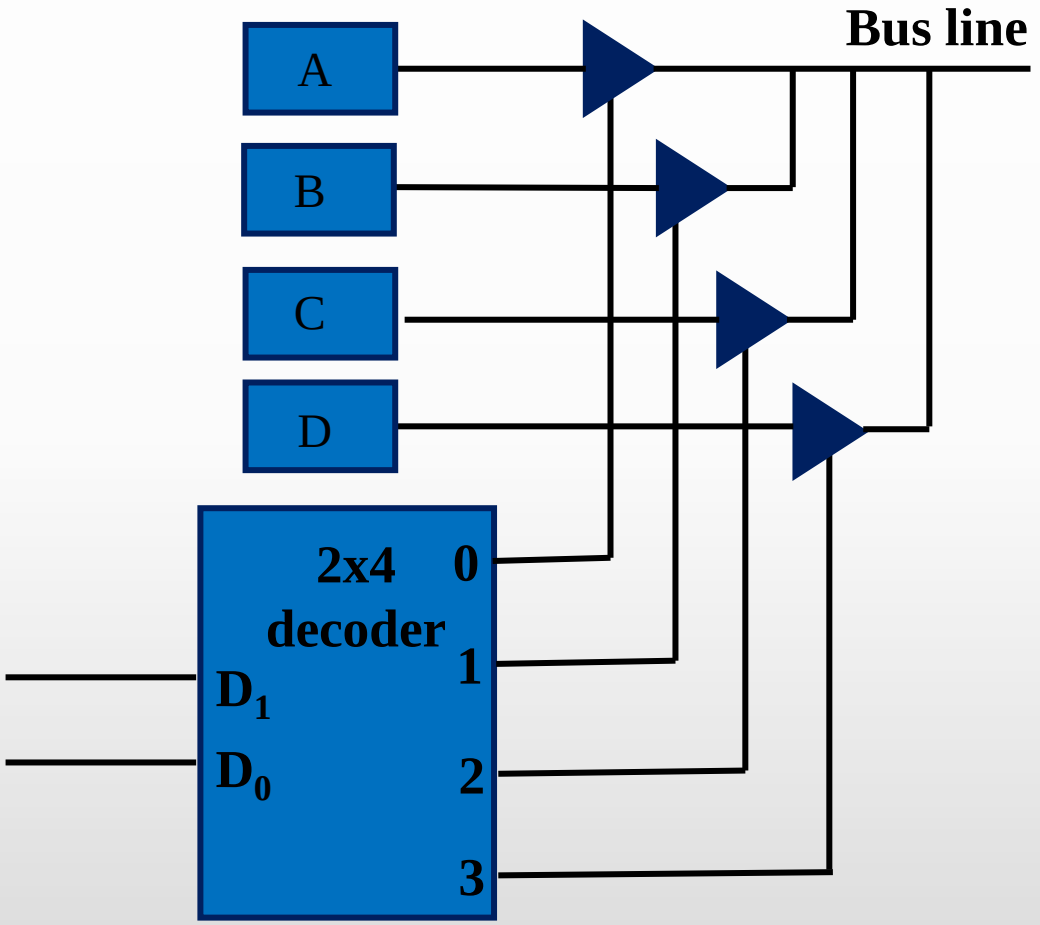


Fig: Bus line for 1-bit four registers with three state-buffers

Memory Transfer

- The transfer of information from a memory word to the outside environment is called a read operation.

Read: DR ← M[AR]

- The transfer of new information to be stored into the memory is called a write operation.

Write: M[AR] ← DR

- Symbolic designation of arithmetic microoperations are
  - R3 ← R1 + R2
  - R3 ← $\overline{R1}$ –R2
  - R2 ← $\overline{R2}$
  - R2 ← R2 + $\overline{1}$
  - R3 ← R1 + R2 + 1
  - R1 ← R1 +1
  - R1 ← R1 - 1

Arithmetic Microoperations

Arithmetic Microoperations

- The most basic arithmetic microoperations for basic computer are described in the following table.

Symbolic designation	Description
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 – R2	Contents of R1 minus R2 transferred t oR3
R2 ← R2’	Complement the content of R2 (1’s complement)
R2 = R2’ + 1	2’s complement the content of R2 (negate)
R3 ← R1 + R2’ + 1	R1 plus 2’s complement of R2 transferred to R3 (subtract)
R1 ← R1 + 1	Increment the content of R1 by one
R1 ← R1 -1	Decrement the content of R1 by one

Binary Adder

- An ***n***-bit binary adder requires ***n*** full-adder.
- A **4**-bit binary adder is designed by combining **1**-bit full adders.

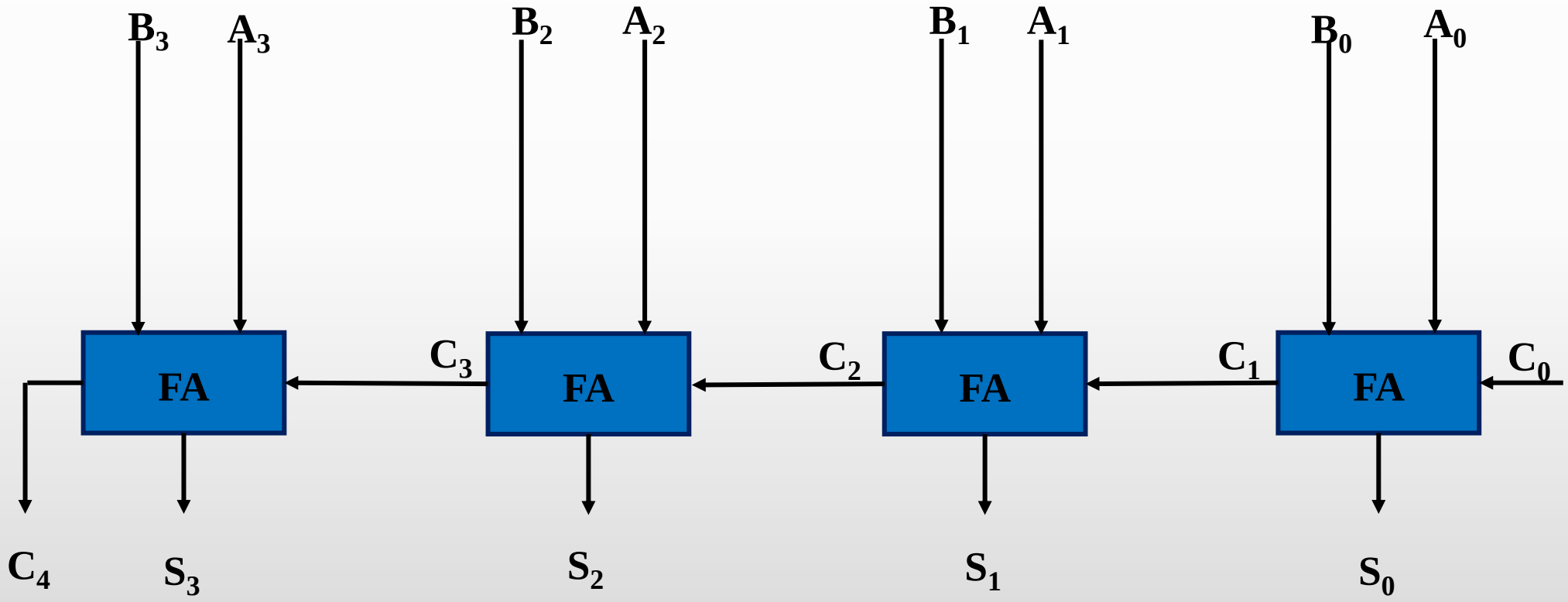


Fig: Block diagram of 4-bit binary adder

Binary Adder-Subtractor

- The subtraction of binary numbers xan be computed by means of complements.
- The 4-bit binary adder–subtractor is designed to compute (A+B) and (A-B) using 1-bit full adders and required gates.
- 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 **(A+B)**.
- When **M = 1**, the circuit becomes a subtractor **(A-B)**.
- Each exclusive-OR gate receives input **M** and one of the inputs of **B**.

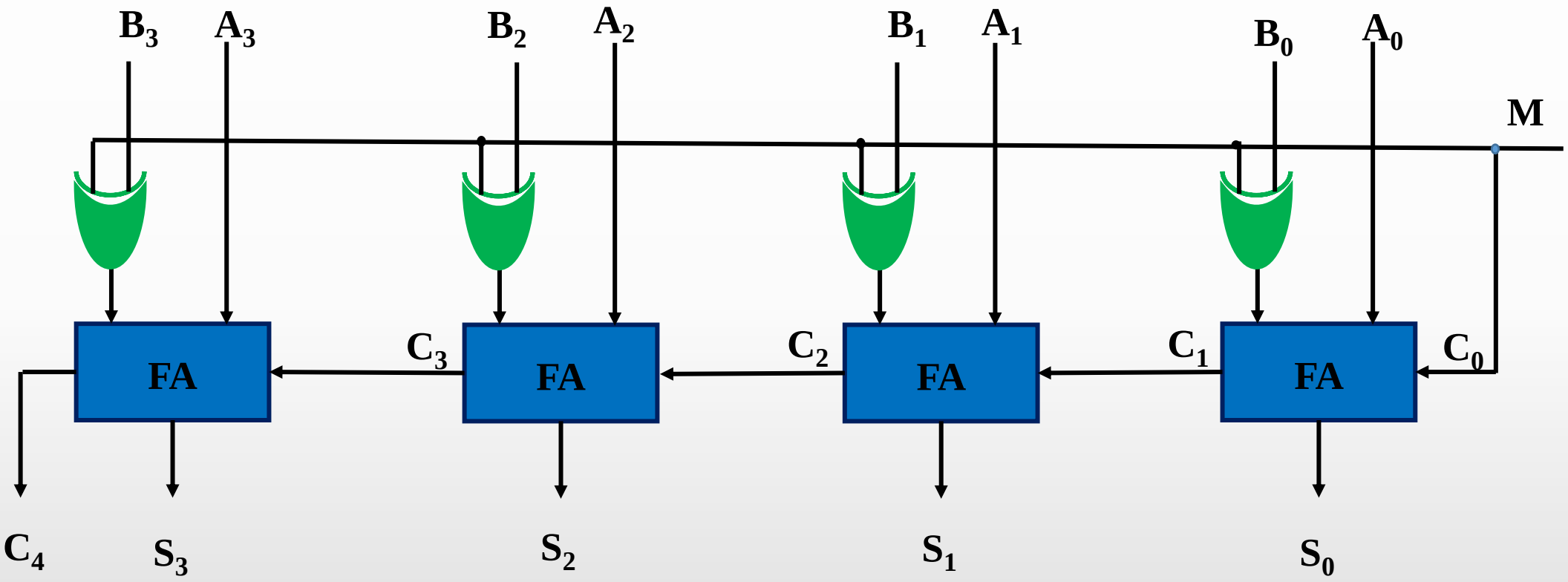


Fig: Block diagram of binary adder-subtractor

For example,

A=1100 => 12

B=1001 => 9

1’com for B=> 0110

2’com for B=> 0110

$$\begin{array}{r} +\quad 1 \\ \hline 0111 \end{array}$$

For A+ B => 1100

$$\begin{array}{r} 1001 \\ \hline 10101 \end{array}$$

=> 21

For A-B => -B= $\overline{B}$ +1 = 2’com for B

=> A+$\overline{B}$+1

=> 1100

$$\begin{array}{r} +\ 0111 \\ \hline 10011 \end{array}$$

=> 3

Binary Incrementer

- A **binary incrementer** operates the increment microoperation that adds one to a number in a register.

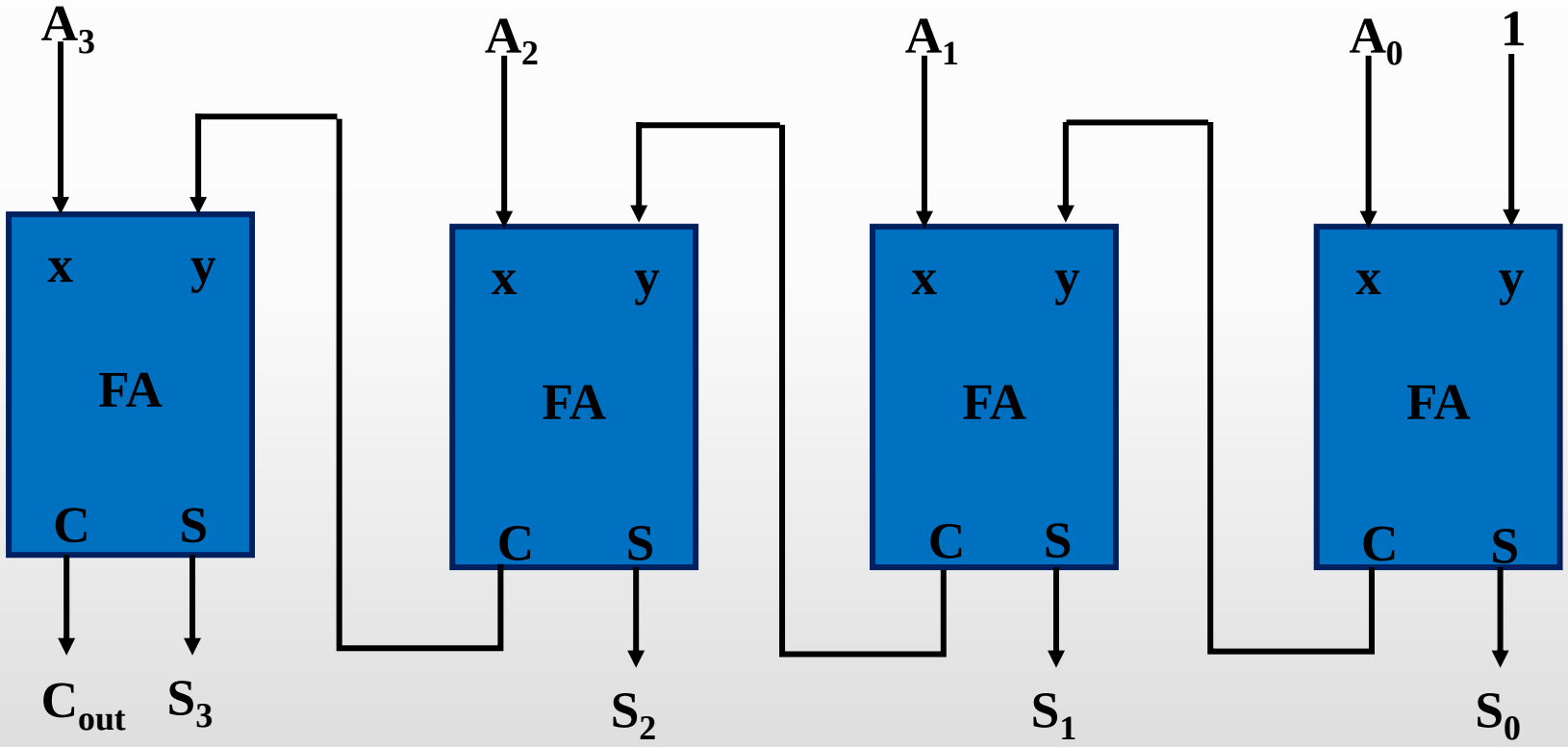


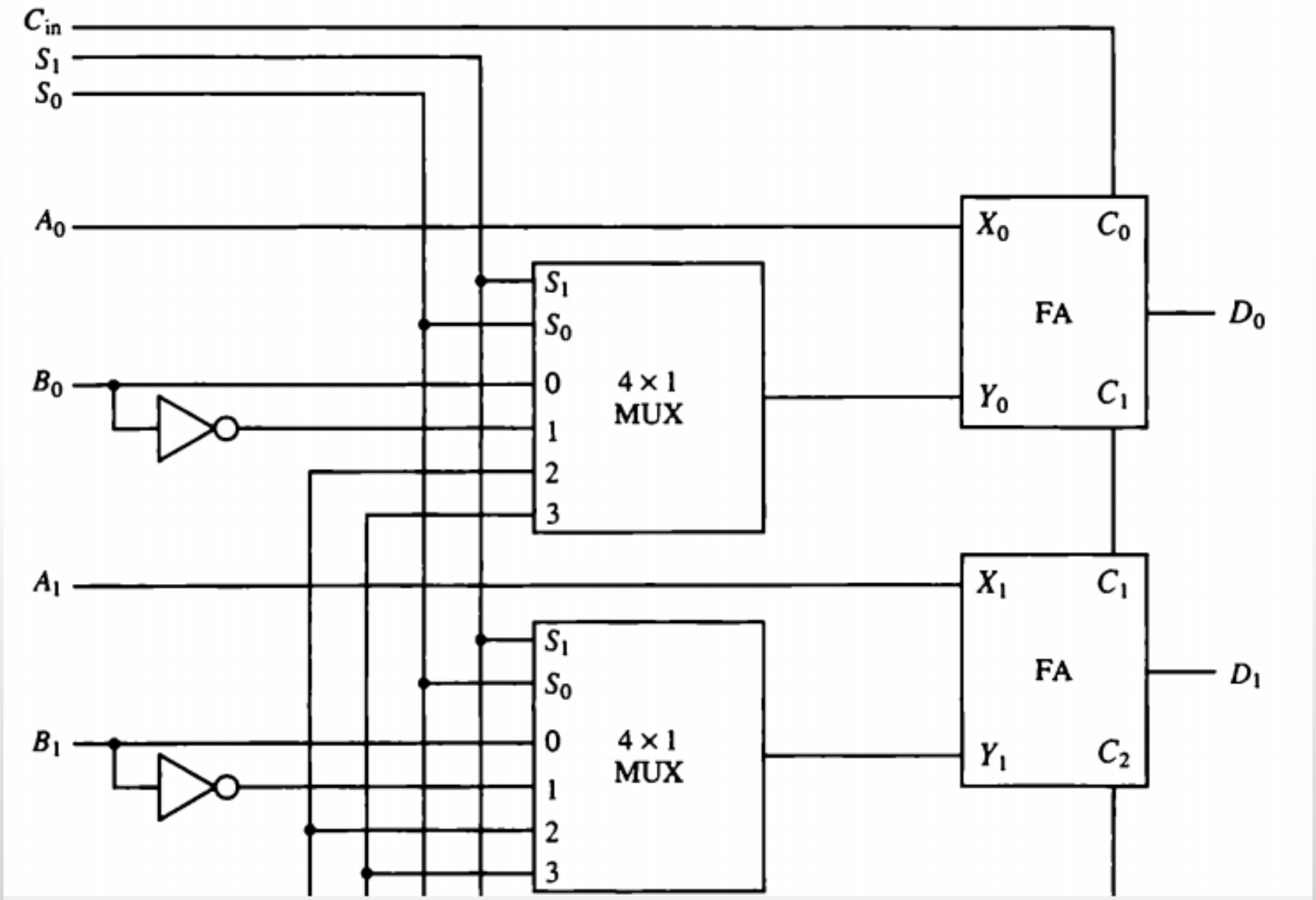
Fig: Block diagram of binary incrementer

Arithmetic Circuit

- A **4-bit arithmetic circuit** can operate arithmetic microoperations with its function table.
- 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.
- When S_1S_0= 00, if C_{in} = 0, D = A + B, if C_{in} = 1, output D = A + B + 1.
- When S_1S_0= 01, if C_{in} = 0, D = A - B - 1, if C_{in} = 1, output D = A - B.
- When S_1S_0= 10, if C_{in} = 0, D = A + 0 = A, if C_{in} = 1, output D = A + 1.
- When S_1S_0= 11, if C_{in} = 0, D = A -1, if C_{in} = 1, output D = A -1 + 1 = A .

Arithmetic Circuit Function table

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


C_{in}
S_1
S_0
A_0
B_0
A_1
B_1
S_1
S_0
0
1
2
3
4 × 1
MUX
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
FA
Y_1
C_2
D_1

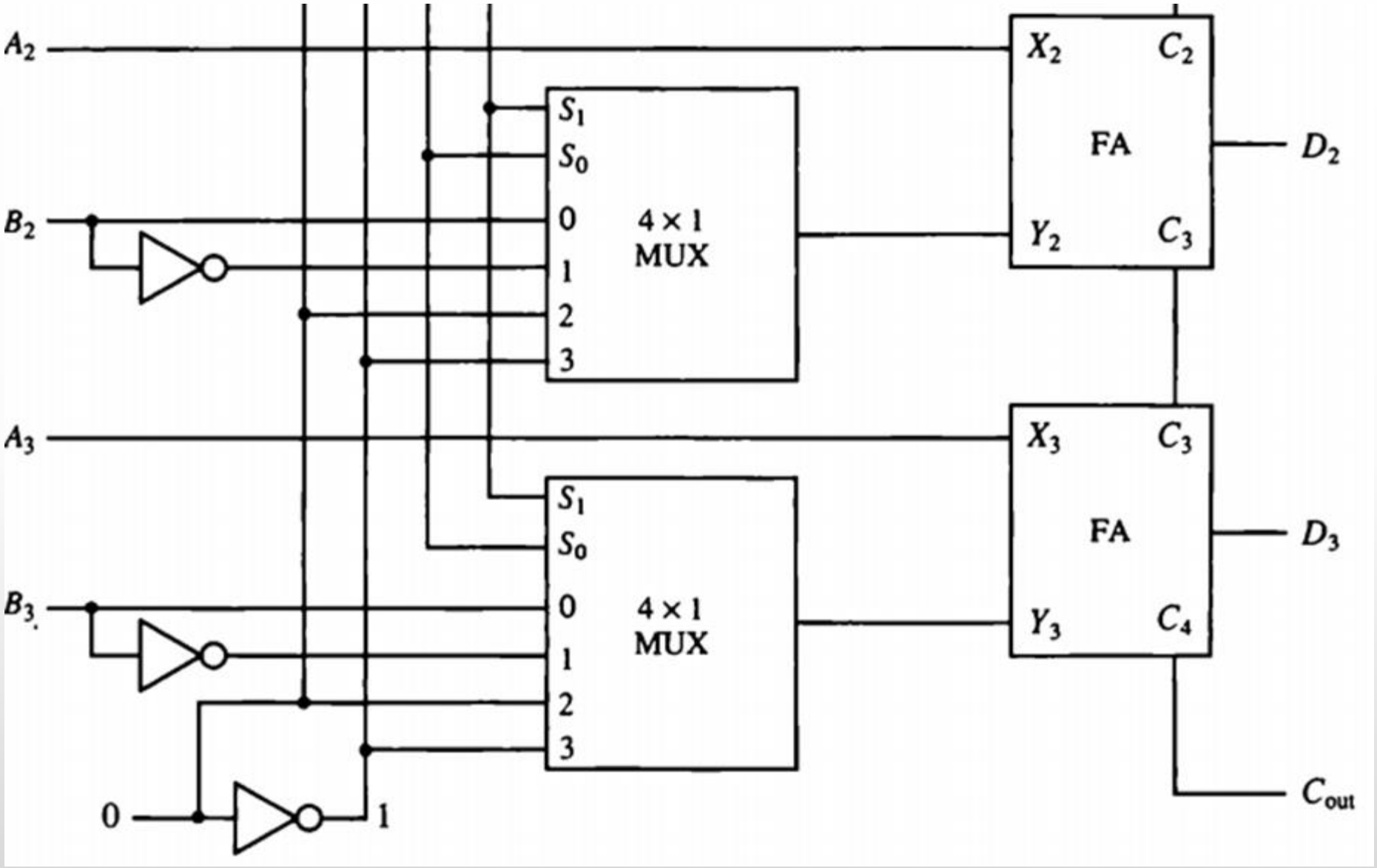


Fig: Block diagram of 4-bit arithmetic circuit.

Logic Microoperations

Logic Microoperations

- Logic microoperations specify binary operations for strings of bits stored in registers.
- For example, exclusive-OR microoperation is performed as follows,

P: R1 ← R1 ⊕ R2

1010 Content of Rl
1100 Content of R2
0110 Content of Rl after P = 1

Sixteen Logic Microoperartions

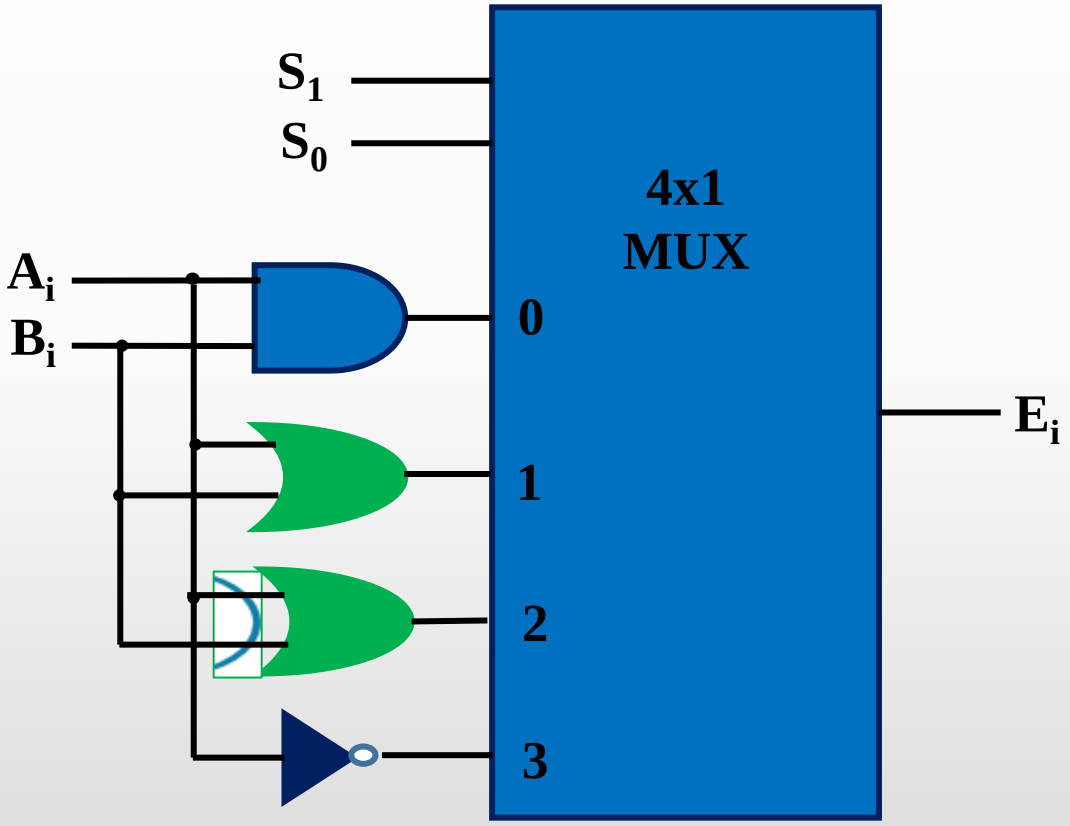
Boolean Function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow A \oplus B$	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

***Example:** Design a logic circuit that can operate 4 logic microoperations (AND, OR, exclusive-OR, NOT) with its function table.

Sol:

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

Function Table



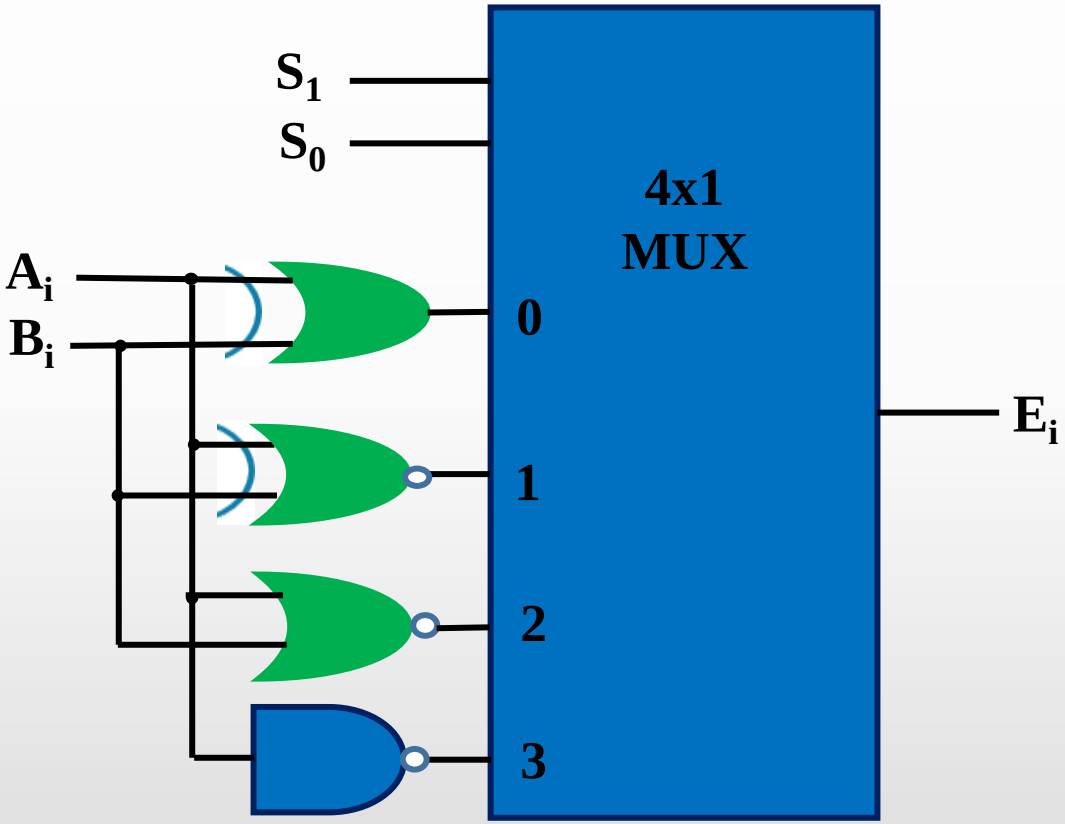
Logic Diagram

***Example:** Design a logic circuit that can operate 4 logic microoperations (EX-OR, EX-NOR, NOR, NAND) with its function table.

Sol:

S_1	S_0	Output	Operation
0	0	$E=A \oplus B$	XOR
0	1	$E=\overline{A \oplus B}$	X-NOR
1	0	$E=\overline{A \vee B}$	NOR
1	1	$E=\overline{A \wedge B}$	NAND

Function Table



Logic Diagram

The following logic microoperations are used in digital system.

Selective-set operation

- Selective-set operation set to 1 the bits in A only where there are corresponding 1's in B.

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

Selective-complement operation

- Selective-complement operation complements bits in A only where there are 1's in B.

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

Selective-clear operation

- Selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B.

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

Mask operation

- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B.

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

Clear operation

- All 0's result if the two numbers are equal.
- The content in register B is inserted by copying the content of register A.

1010	A
1010	B
0000	A

Insert operation

The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.

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

```
0000 1010  A before
1001 0000  B (insert)
-----------
1001 1010  A after intersection
```

Shift Microoperations

Shift Microoperations

Shift microoperations are used for serial transfer of data. There are three types of shifts: logical, circular, and arithmetic.

- A **logical shift** is one that transfers 0 through the serial input. Symbols shl and shr are for logical shift-left and shift-right microoperations.
- The **circular shift** or **rotate operation** circulates the bits of the register around the two ends without loss of information.
- An **arithmetic shift** is a microoperation 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.

Examples of Shift Microoperations

Symbolic designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left R
R ← ashr R	Arithmetic shift-right R

Arithmetic shift right

Hardware Implementation of Shifter

- A 4-bit combinational circuit shifter can be designed by using multiplexers with its function table.
- A shifter with ***n*** data inputs and outputs requires ***n*** multiplexers.
- When the select line **S =1**, the input data are shifted left.
- When the select line **S = 0**, the input data are shifted right.
- The two serial inputs can be controlled by another multiplexer to provide three possible types of shift.

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

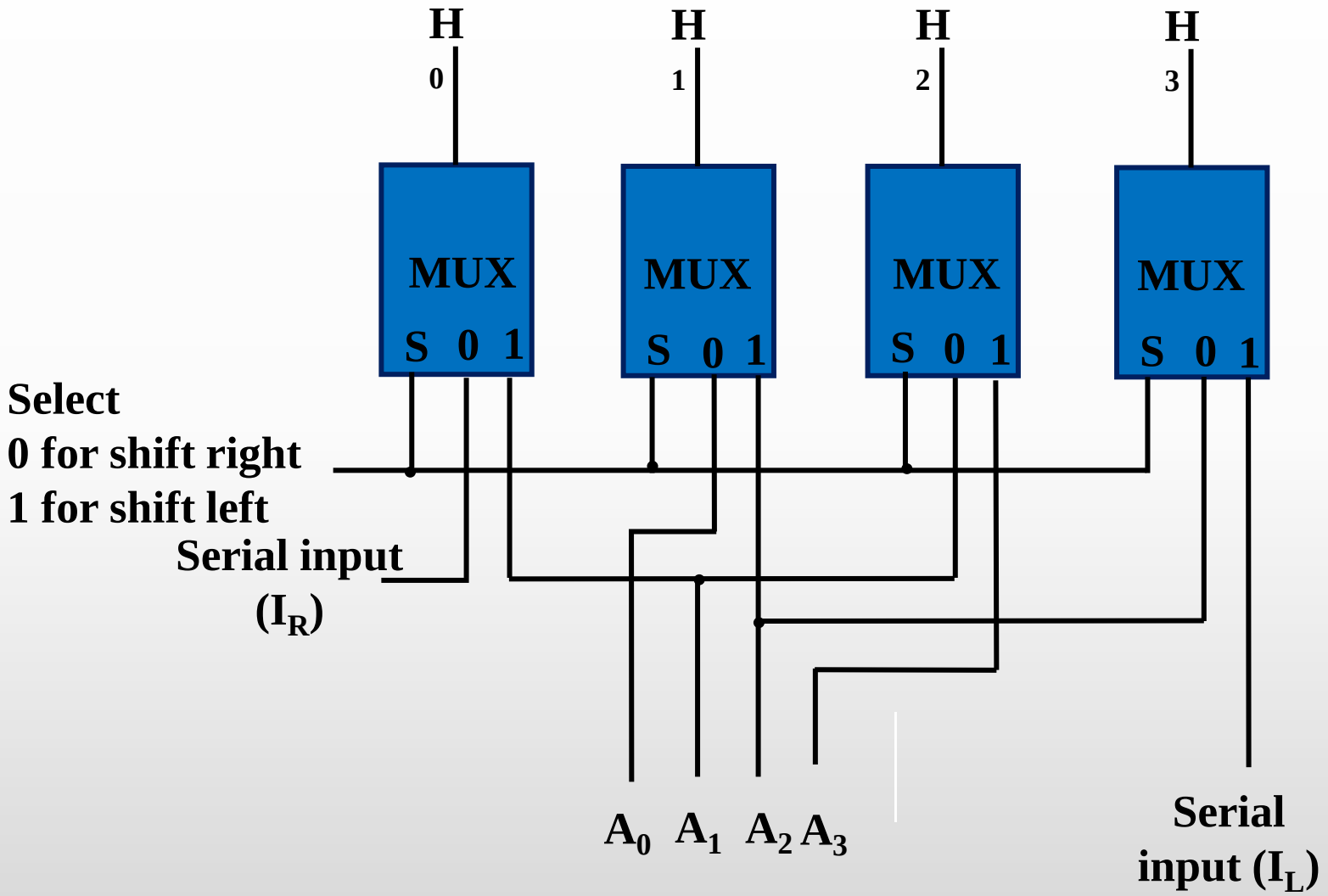


Fig: 4-bit combinational circuit shifter

Arithmetic Logic Shift Unit

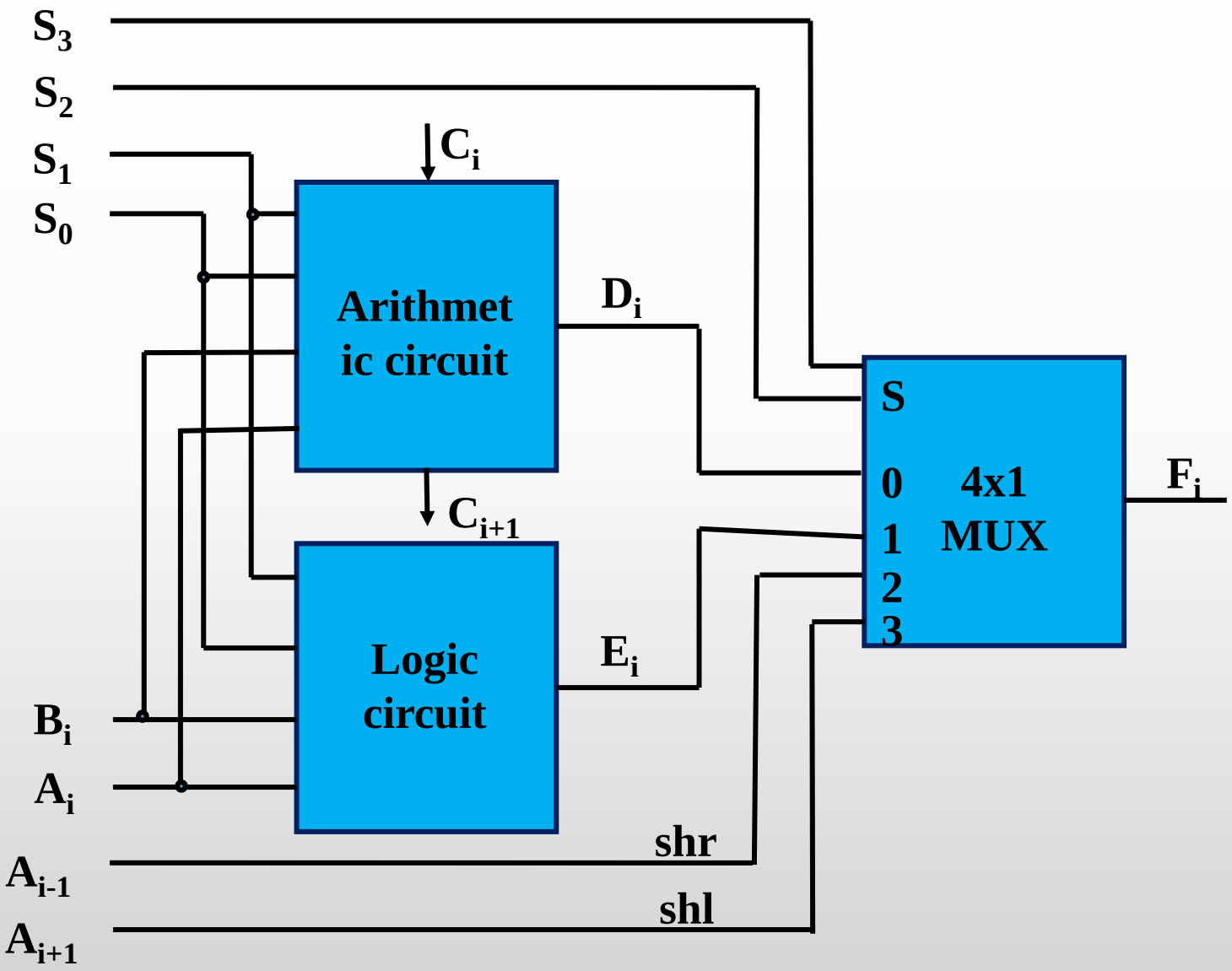
Arithmetic Logic Shift Unit

- Computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, 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 shift microopeiations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.

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	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 Logic Shift Unit

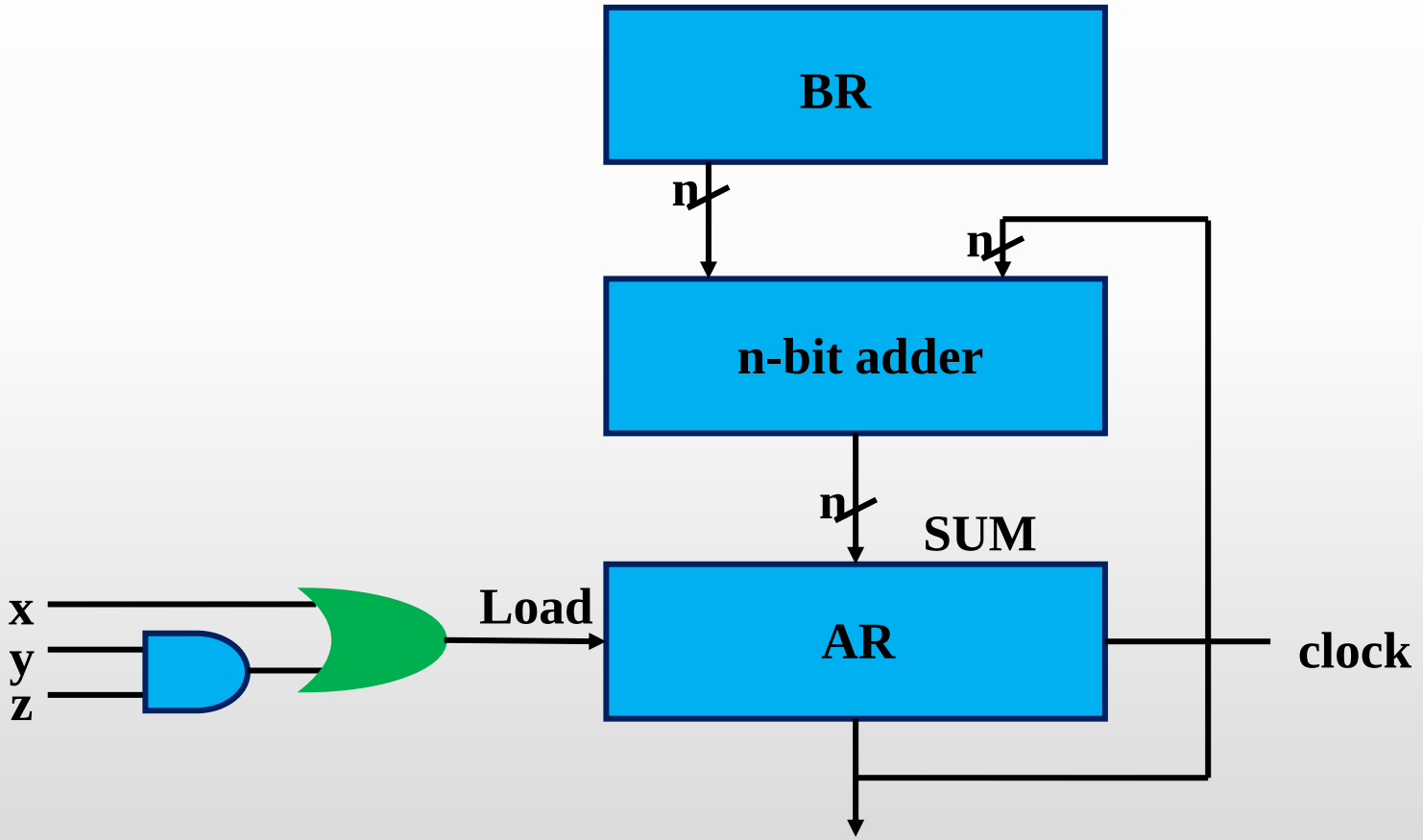


Exercises

Ex:1. Draw the block diagram for the hardware that implements the following statements:

x + yz: AR ← AR + BR

Sol:



Ex:2. 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 the execution of the following sequence of microoperations.

(a) AR ← AR + BR Add BR to AR
(b) CR ← CR ^ DR, BR ← BR + 1 AND DR to CR, increment BR
(c) AR ← AR – CR Subtract CR from AR

Sol:

(a)
AR = 11110010
BR = 11111111 (+)

AR = 11110001
After calculation,
AR = 11110001 BR = 11111111 CR = 10111001 DR = 11101010

(b)

CR = 10111001 BR = 11111111

DR = 11101010 (AND) +1

CR = 10101000 BR = 00000000

After calculation,

AR = 11110001 BR = 00000000 CR = 10101000 DR = 11101010

(d)

AR = 11110001

CR = 10101000 (-)

AR = 01001001

After calculation,

AR = 01001001 BR = 00000000 CR = 10101000 DR = 11101010

Ex:3. An 8-bit register contains the binary value 10011100. What is the register value after an arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow. Also determine the decimal values after shifting.
Sol:

R = 10011100 = -100
Arithmetic shift right : **11001110** = **-50**
Arithmetic shift left : **00111000**
(In arithmetic shift left, if the leftmost bit is changed, the overflow occurs and the result in register is incomplete.)
After an arithmetic shift left, overflow occurs because a negative number changed to positive.

Ex:4. Starting from an initial value of R = 11011101, determine the sequence of binary values in R after a logical shift-left, followed by a circular shift-right, followed by a logical shift-right and a circular shift-left.

Sol:

R = 11011101

Logical shift left : **10111010**

Circular shift right: **01011101**

Logical shift right : **00101110**

Circular shift left : **01011100**

Thank You

Lecture for Next Week

- **Basic Computer Organization and Design**
  - **Instruction Codes**
  - **Computer Registers**
  - **Computer Instructions**
  - **Timing and Control**
  - **Instruction Cycle……**