Register Transfer Language

In this lecture, we will study

i. Microoperations

ii. Register Transfer Language and Register Transfer

iii. Bus and Memory Transfer

iv. Arithmetic, Logical, and Shift Microoperations

v. ALU

Chapter Exercises

Microoperations and Register Transfer Language

A Digital System

A digital system is an interconnection of digital hardware modules that accomplish a specific task. These modules are constructed from components like registers, decoders, arithmetic elements, and control logic.

Microoperations

Digital modules are best defined by the **REGISTERS** they contain and the **OPERATIONS** that can be performed on the data stored in these registers.

The operations executed on the data stored in registers are called **MICROOPERATIONS.**

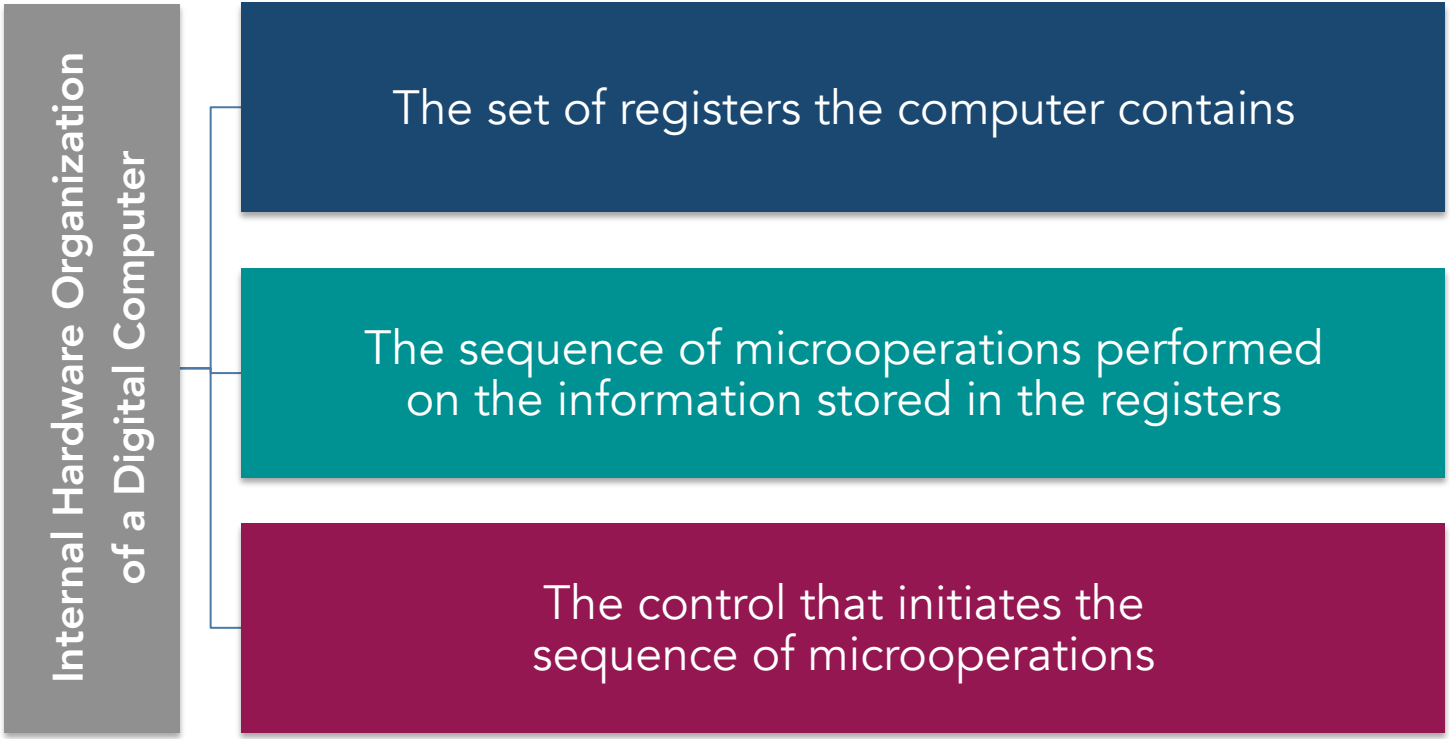
Microoperations

So, a microoperation is an elemantary operation performed on the information stored in one or more registers.

'Shift', '*load'*, '*clear'*, '*count'* are all examples of microoperations.

A *bidirectional shift register*, for instance, is capable of performing the 'shift right' and 'shift left' microoperations. Similarly, a *counter with parallel load* is capable of performing 'increment' and 'load' microoperations.

Now that we know what a microoperation is, we can define the internal hardware organization of a digital computer by specifying:



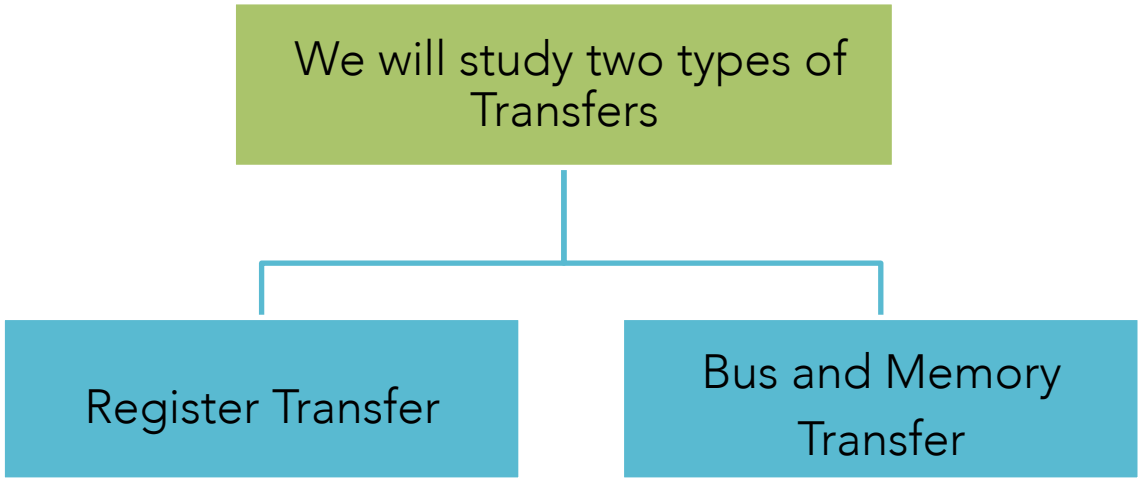
Register Transfer Language

It is possible to specify the sequence of microoperations by explaining every operation in words, but that would usually involve very lengthy descriptive explanations. It is usually a much better idea to use some sort of symbolic representation to describe the various microoperations.

The symbolic notation used to describe the microoperation transfers among registers is called **REGISTER TRANSFER LANGUAGE.**

Register Transfer Language

The term ‘transfer’ in Register Transfer Language implies the availability of hardware logic circuits that can perform the stated microoperation and ***transfer*** the result of the operation to the same or another register.



Registers

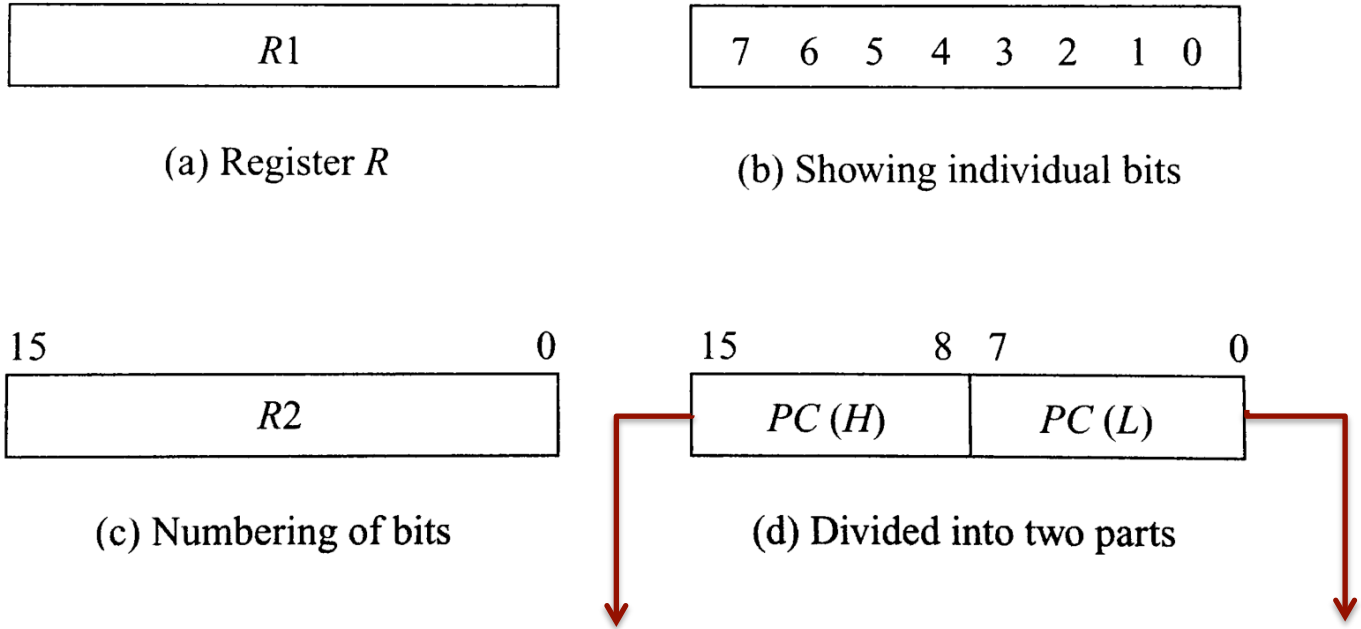
Registers: How are they represented?

Computer registers are designated by capital letters (sometimes followed by numerals), which denote the function of the register.

For instance, a register that holds the memory location of data that needs to be accessed is called a *Memory Address Register*, and it is designated as **MAR**.

A *Program Counter* is designated as **PC**
An *Instruction Register* is designated as **IR**, and so forth.

Block Diagram of a Register



Register Transfer

Register Transfer

Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

For instance, the statement

$$R2 \leftarrow R1$$

denotes the transfer of the content of register R1 to register R2. In other words, it designates the REPLACEMENT of the content of R2 by the content of R1.

Register Transfer

Normally, we want transfer to occur only under a predetermined control condition.

This can be shown by the means of an *if-then* statement.

If (P=1) *then* (R2←R1)

Where P is the control signal generated in the control section.

Control Function

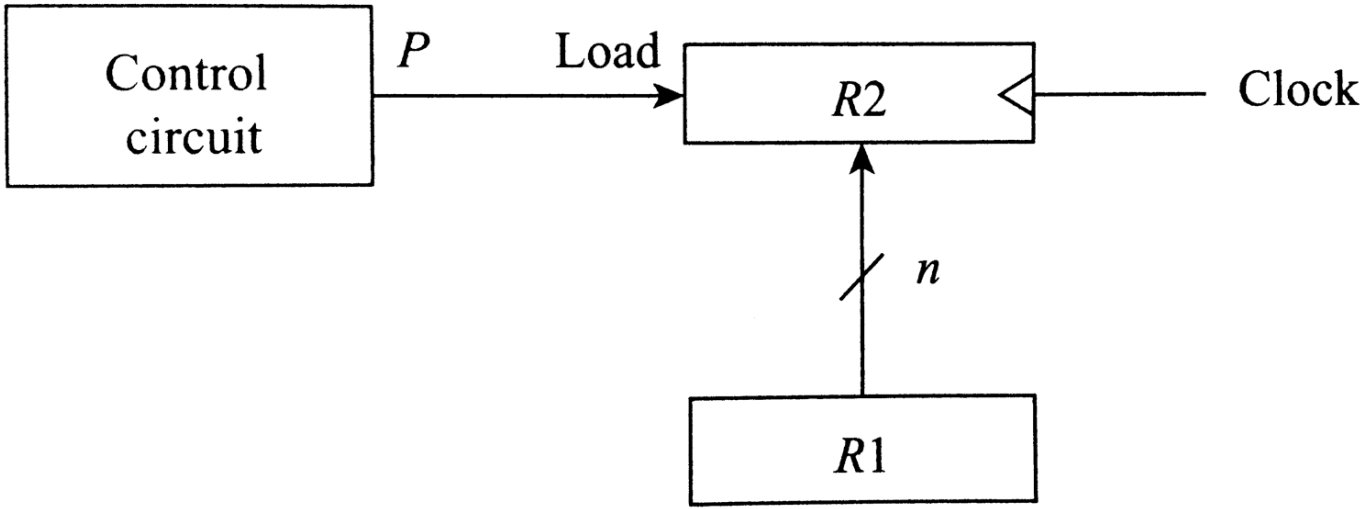
Sometimes, it is convenient to separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variable that is equal to 1 or 0.

$$P: R2 \leftarrow R1$$

The control condition is terminated with a colon. It symbolizes the requirement that the transfer of the content of R1 into R2 be executed only when P=1.

Block Diagram of a Typical Register Transfer Operation

When the control variable P=1, n bits of data are transferred from R1 to R2.



Basic Symbols of the Register Transfer Notation

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

Exercise

What does the following statement represent?

T: R2 ← R1, R3 ← R2

Bus Transfer

Common Bus System

In a typical digital computer with multiple registers, paths must be provided to transfer information from one register to another.

If separate lines of communication are used between each and every register, the number of wires will be excessive.

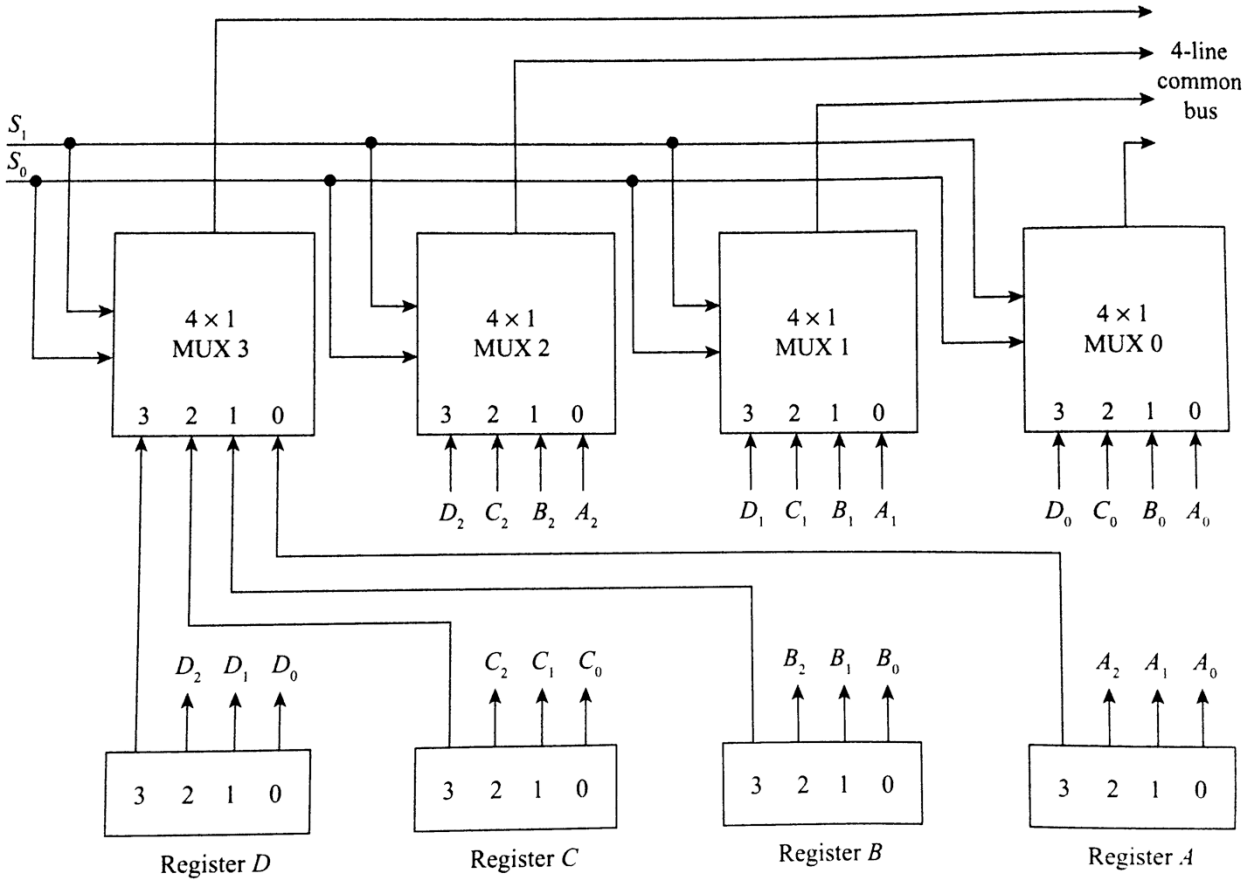
It is more efficient to use a COMMON BUS SYSTEM.

Common Bus

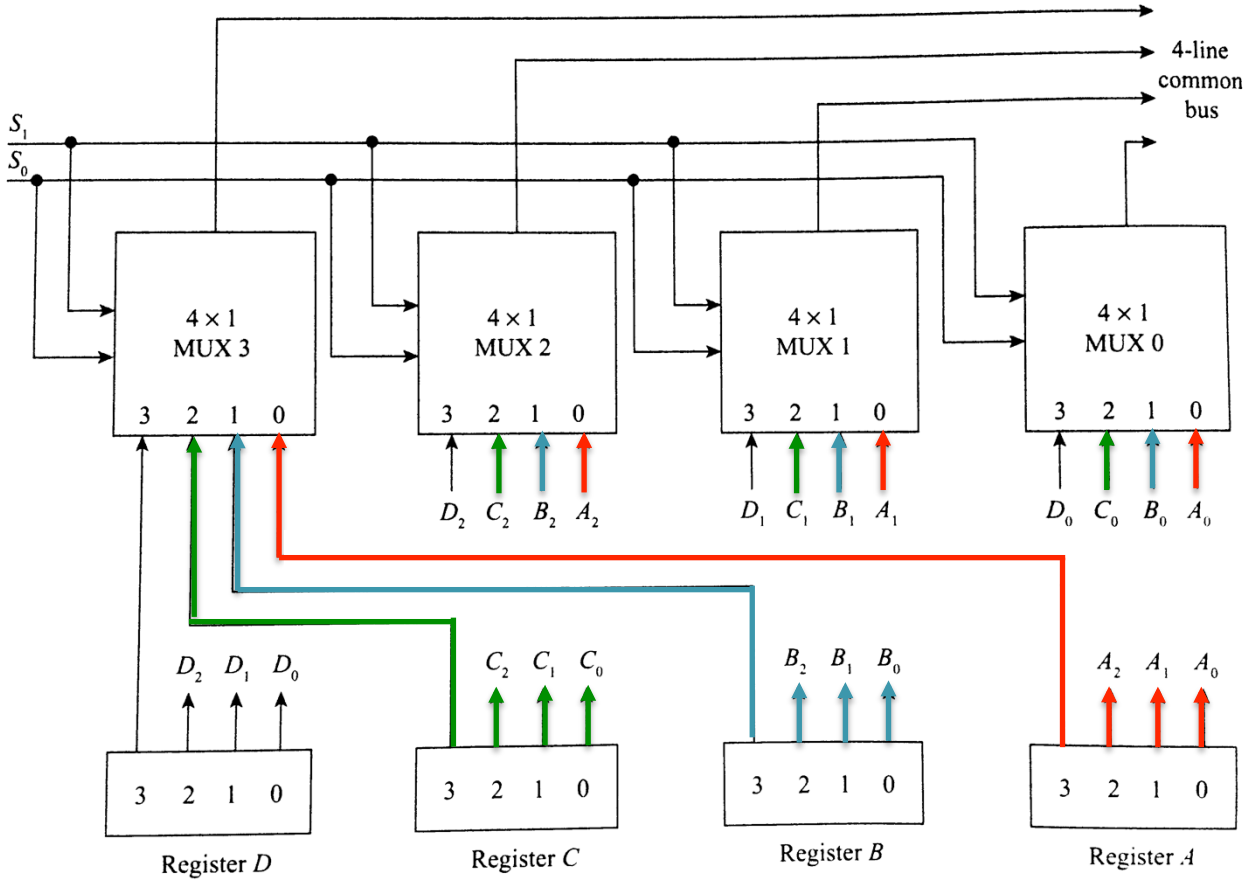
A bus consists of a set of common lines – one for each bit of a register – through which binary information is transferred one at a time.

A control signal determines which register is selected by the bus during a particular register transfer.

Bus System for Four Registers



Bus System for Four Registers



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

Function Table

Bus–Based Register Transfer

When the bus is included, register transfer is symbolized as follows:

T: BUS ← C, R1 ← BUS

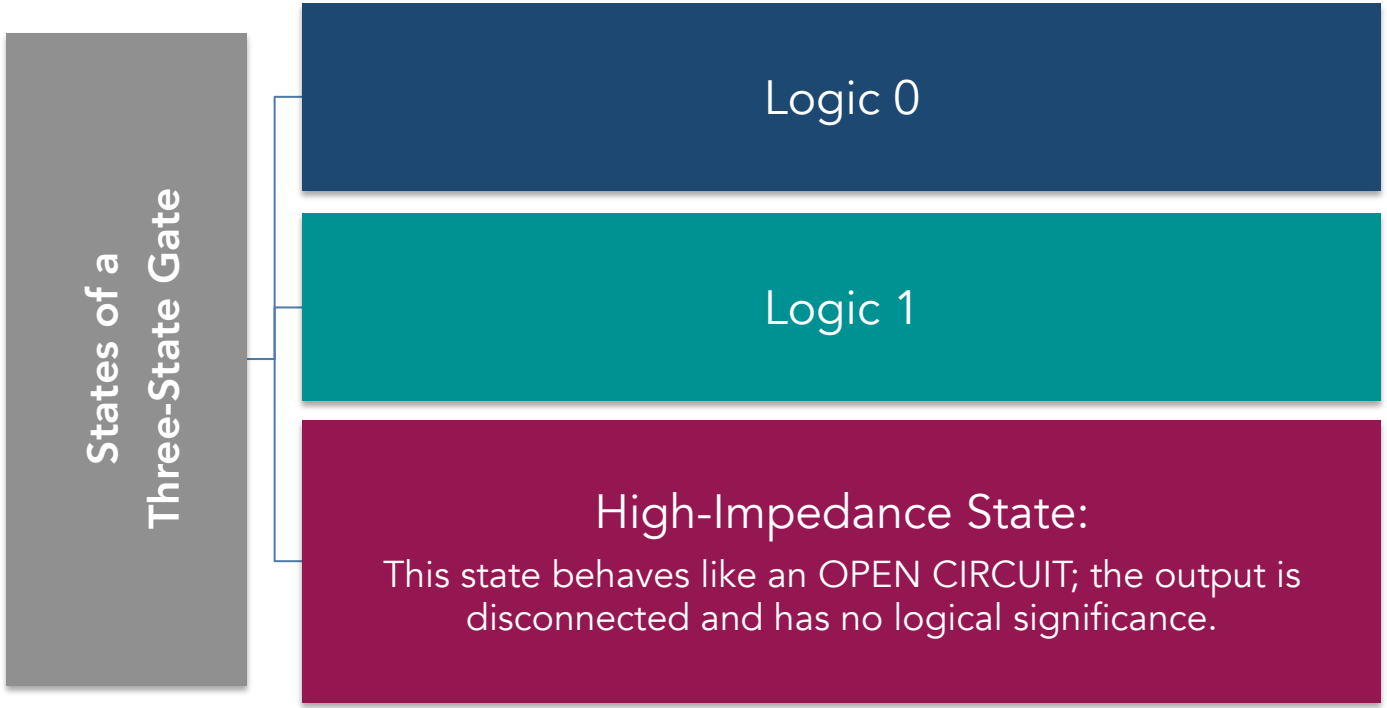
This statement implies that the contents of register C are placed on the bus and the contents of the bus are then loaded into register R1 by activating the load control input T.

If the bus is known to exists in the system, the previous statement can be simply written as:

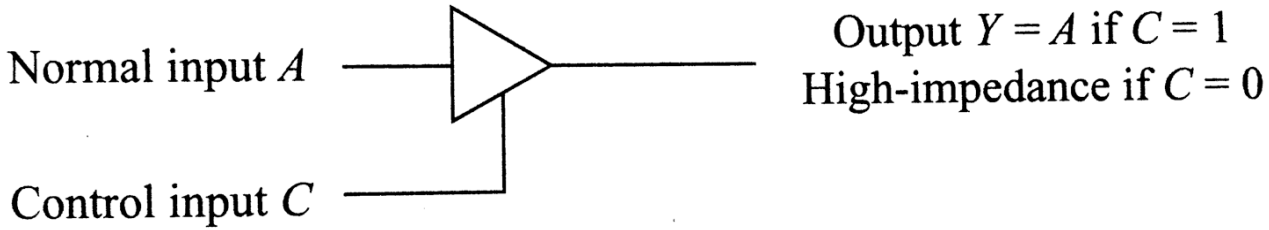
R1 ← C

Three–State Bus Buffers

A bus system can be constructed with three-state gates instead of multiplexers.



Graphical Symbol of a Three–State Buffer



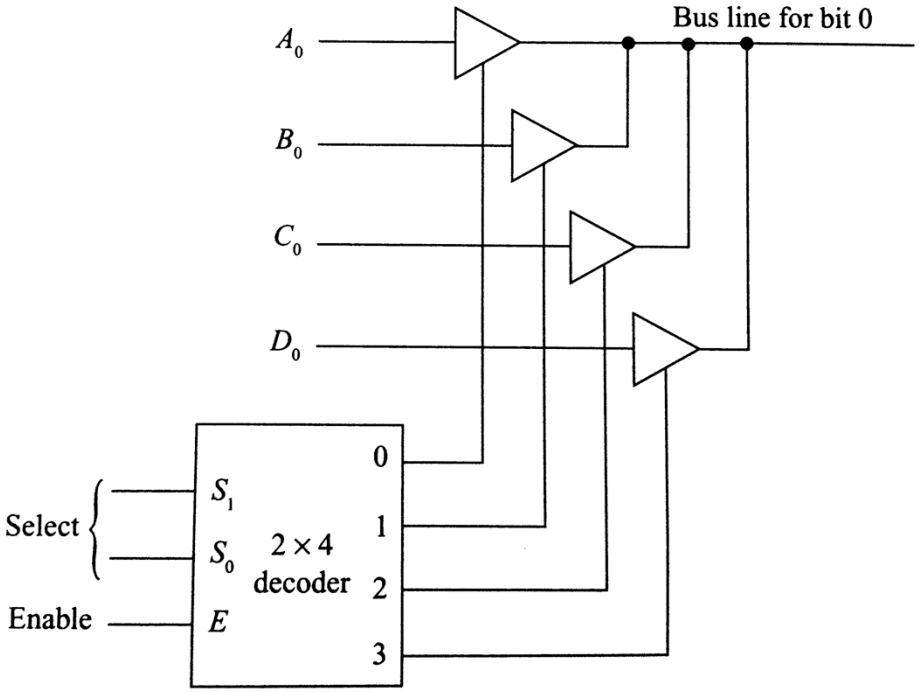
Unlike regular buffer gates, a three-state buffer has an additional control input C;

C determines the output state of the buffer.

When C=1, the output is enabled and the gate behaves like a conventional buffer.

When C=0, the output is disabled and the gate goes into a high-impedance state.

Bus Line With Three–State Buffers



Only one of the four three-state buffers should be active at any given time. One way to achieve that is to use a decoder.

When Enable input of the decoder is 0, all of its outputs will be 0, i.e., all the three-state buffers will be inactive. When E=1, any one of the three-state buffers will become active, depending on the values of the Select inputs.

Exercise

Can we build a bus system using a gate other than the buffer?

If yes, how? If not, why?

Memory Transfer

Memory Transfer Operations

READ OPERATION

Transfer of information from a memory word to the outside environment

WRITE OPERATION

Transfer of new information to be stored into the memory

Symbolic Representation of a READ Operation

Consider a memory unit that receives the address from a register, called the address register (AR), and the data is to be transferred into another register called the data register (DR).

The READ operation can be stated as follows:

Read: DR ← M[AR]

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

Symbolic Representation of a WRITE Operation

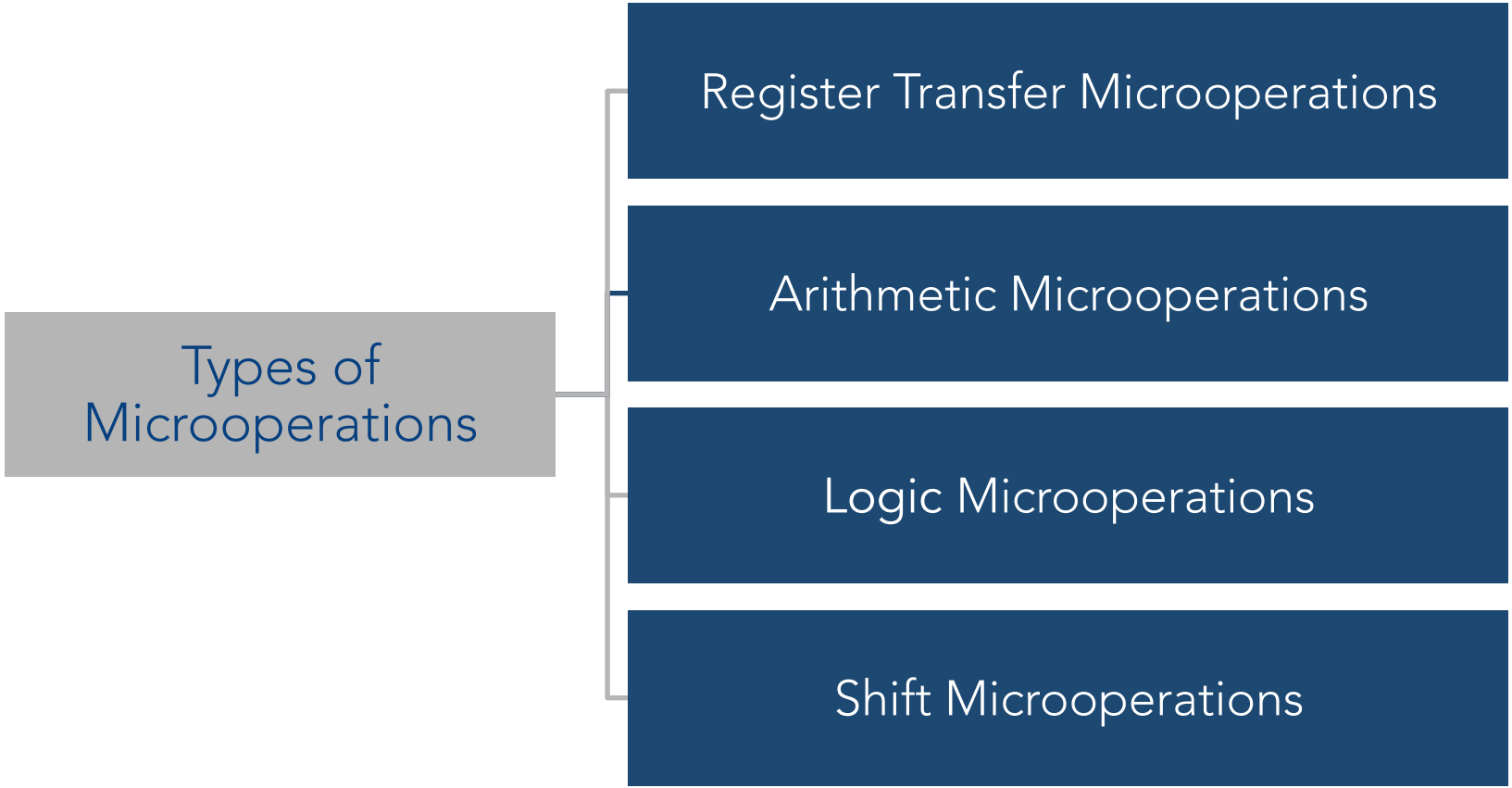
A write operation transfers the contents of a register (say R1) to a memory word M selected by the address present in the address register (say AR).

The WRITE operation can be stated symbolically as :

Write: M[AR] ← R1

This causes a transfer of information from R1 into the memory word M selected by the address in AR.

Microoperations

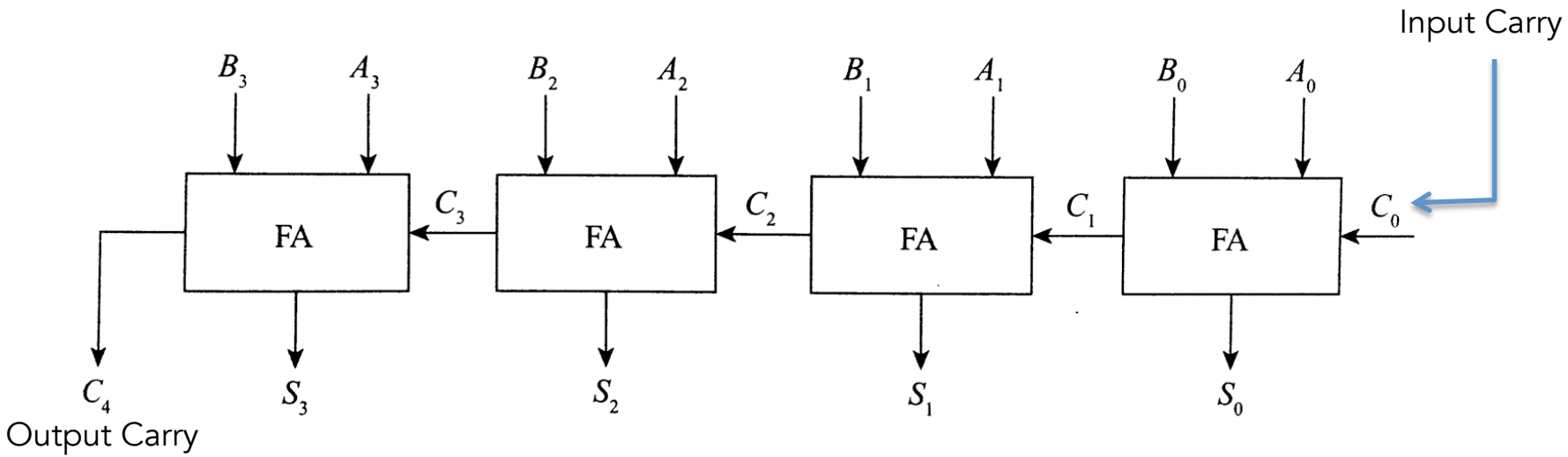

Types of Microoperations
Register Transfer Microoperations
Arithmetic Microoperations
Logic Microoperations
Shift Microoperations

Arithmetic Microoperations

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

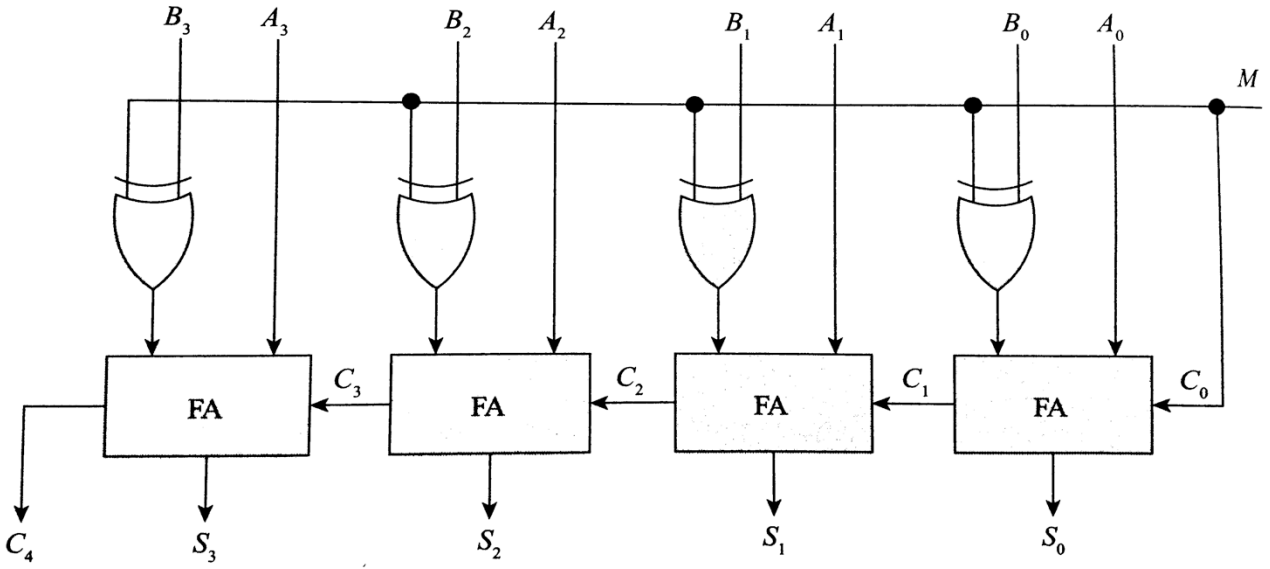
4–Bit Binary Adder

Bits A_0, A_1, A_2, and A_3 come from one register (such as R1) and
Bits B_0, B_1, B_2, and B_3 come from another register (such as R2).



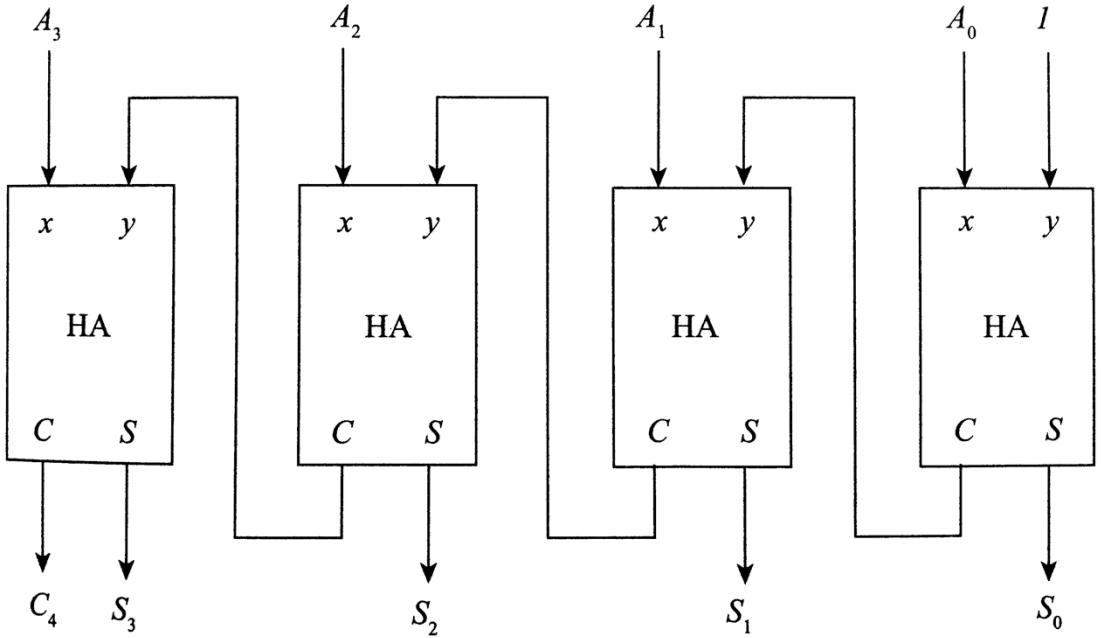
Bits S_0, S_1, S_2, and S_3 together form the required sum.

Binary Adder–Subtractor



M is the mode of the circuit
When M=0, the circuit becomes an adder
When M=1, the circuit becomes a subtractor

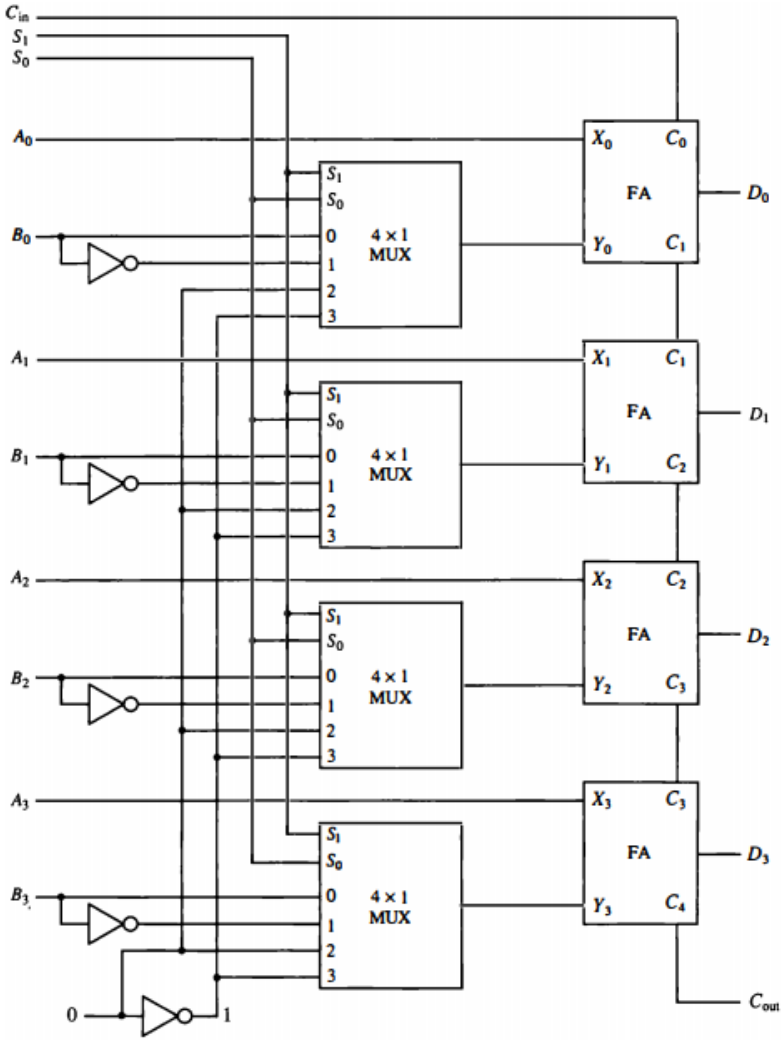
4–Bit Binary Incrementer



The increment microoperation adds 1 to a number in a register.

Arithmetic Circuit

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



Logic Microoperations

Logic microoperations specify binary operations that can be performed on strings of bits stored in registers.

These operations consider each bit of the register separately and treat them as binary variables,
which is why, they are considered to be bit-wise operations.

Logic Microoperations

For instance, the XOR microoperation with the contents of two registers R1 and R2 is symbolized by the statement

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

The XOR microoperation is performed on the individual bits of the registers, provided that the control variable P = 1.

Logic Microoperations: Example

Let the contents of R1 be 1010 and the contents of R2 be 1100.

The XOR microoperation **P: R1 ← R1 ⊕ R2**

symbolizes the following operation:

1010	Contents of R1
1100	Contents of R2
0110	Contents of R1 after P=1

List of Logic Microoperations

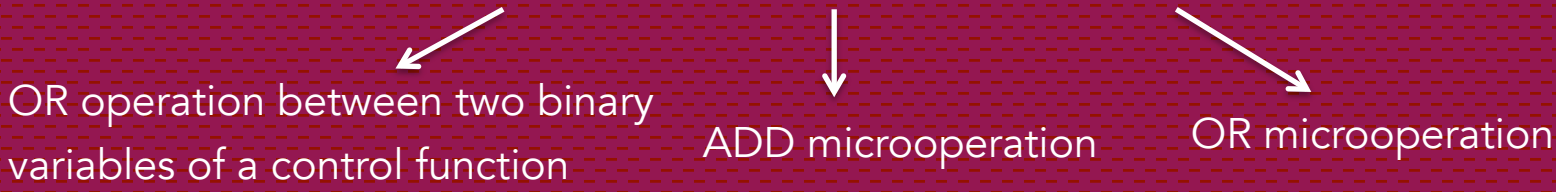
Boolean function	Microoperation	Name	Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear	$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_1 = xy$	$F \leftarrow A \wedge B$	AND	$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$		$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_3 = x$	$F \leftarrow A$	Transfer A	$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$		$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_5 = y$	$F \leftarrow B$	Transfer B	$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR	$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_7 = x + y$	$F \leftarrow A \vee B$	OR	$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Important Note

When the symbol + occurs in a microoperation, it will denote an arithmetic plus. But when it occurs in a control (or Boolean) function, it will denote an OR operation.

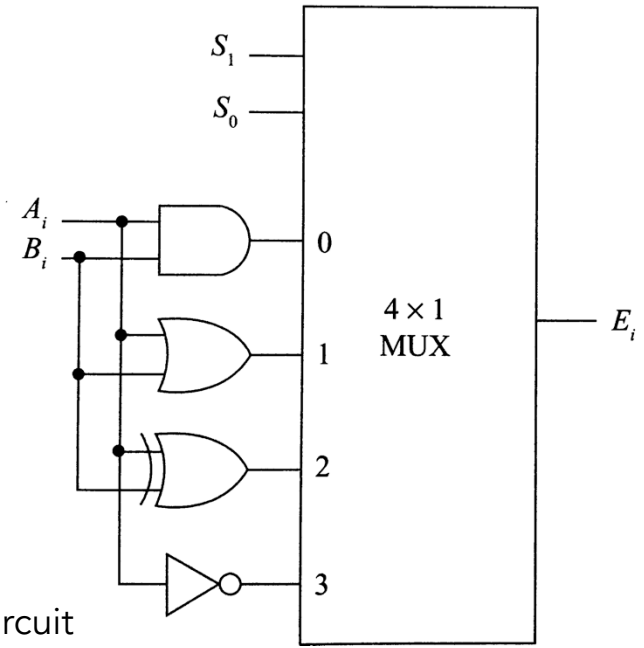
For instance, in the following statement:

P + Q: R1 ← R2 + R3, R4 ← R5 v R6



Hardware Implementation of Logic Microoperations

Although there are 16 logic microoperations, most computers use only 4 – AND, OR, XOR, and COMPLEMENT – from which, all others can be derived.



Logic Circuit

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

Function Table

Some Applications of Logic Microoperations

Logic microoperations are very useful for manipulating individual bits or portions of a word stored in a register.

Selective-Set Operation: This operation sets to 1 the bits in register A where there are corresponding 1s in register B. For instance,

1 0 1 0	A (before)
1 1 0 0	B (logical operand)
1 1 1 0	A (after selective-set)

Some Applications of Logic Microoperations

Selective-Complement Operation: This operation complements the bits in register A where there are corresponding 1s in register B. It does not affect bit positions that have 0s in B. For instance,

1 0 1 0	A (before)
1 1 0 0	B (logical operand)
0 1 1 0	A (after selective-complement)

Some Applications of Logic Microoperations

Selective-Clear Operation: This operation clears to 0 the bits in A only when there are corresponding 1s in B. For instance,

1 0 1 0	A (before)
1 1 0 0	B (logical operand)
0 0 1 0	A (after selective-clear)

The corresponding microoperation is $\mathbf{A} \leftarrow \mathbf{A} \wedge \overline{\mathbf{B}}$

Some Applications of Logic Microoperations

Mask Operation: This operation is similar to the selective-clear operation, except that the bits of A are cleared to 0 only when there are corresponding 0s in B. For instance,

1 0 1 0	A (before)
1 1 0 0	B (logical operand)
1 0 0 0	A (after masking)

Some Applications of Logic Microoperations

Insert Operation: This 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. For instance, suppose A contains 8 bits: 0110 1010. To replace the four leftmost bits by 1001, we first mask the four unwanted bits:

0 1 1 0 1 0 1 0	A (before)
0 0 0 0 1 1 1 1	B (mask)
0 0 0 0 1 0 1 0	A (after masking)

And then inserting the new value:

0 0 0 0 1 0 1 0	A (before)
1 0 0 1 0 0 0 0	B (insert)
1 0 0 1 1 0 1 0	A (after insertion)

Some Applications of Logic Microoperations

Clear Operation: This operation compares the words in A and B and produces an all 0 result if the two numbers are equal.

(This is done using the XOR microoperation.)

1 0 1 0	A
1 0 1 0	B
0 0 0 0	$A \leftarrow A \oplus B$

Shift Microoperations

Shift microoperations are used for serial transfer of data.

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

There are three types of shifts:

> Logical

> Circular

> Arithmetic

Logical Shift

Logical shift transfers 0 through the serial input.

For instance, the microoperation

R1 ← shl R1

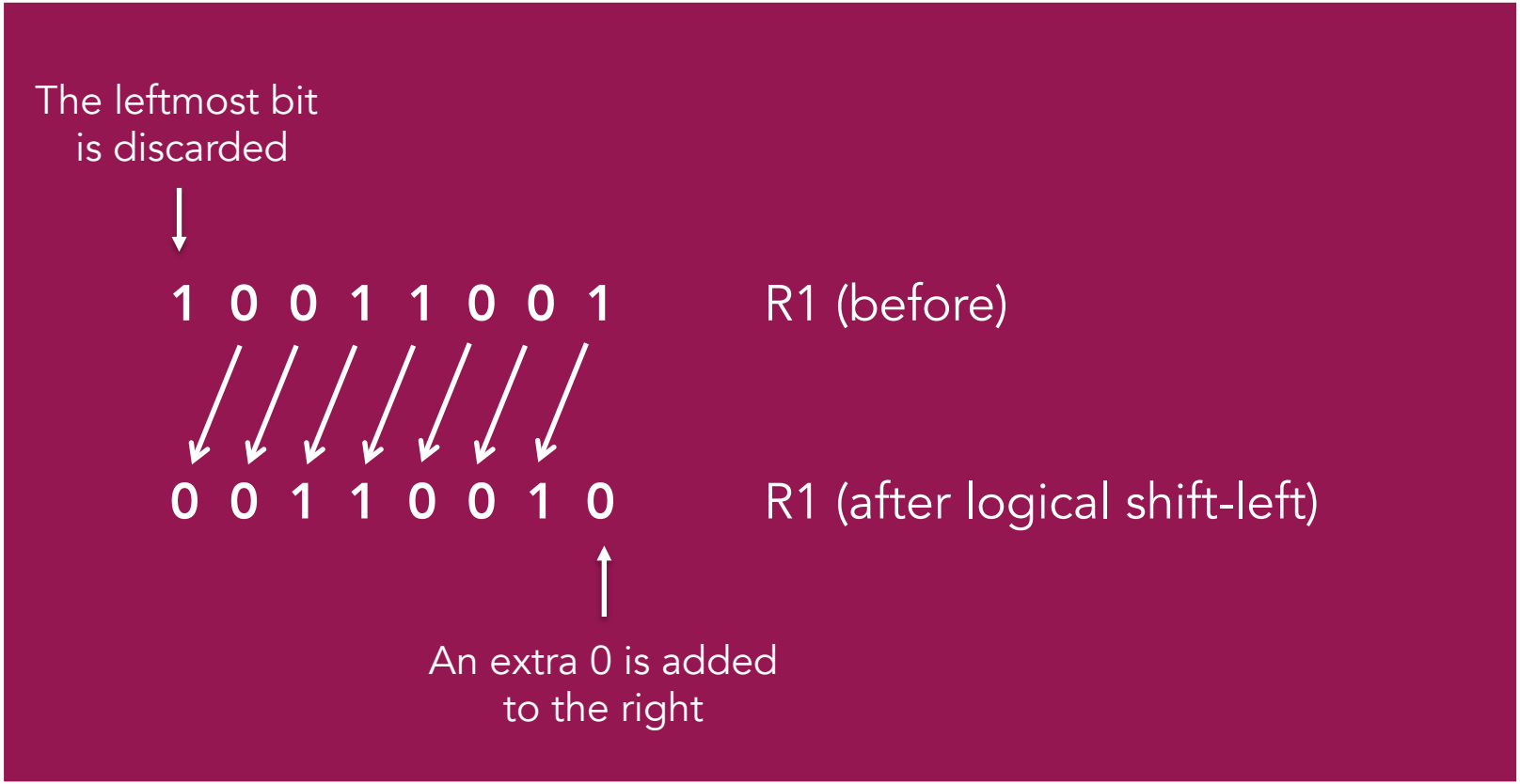
indicates a 1-bit shift to the left of the contents of register R1

and

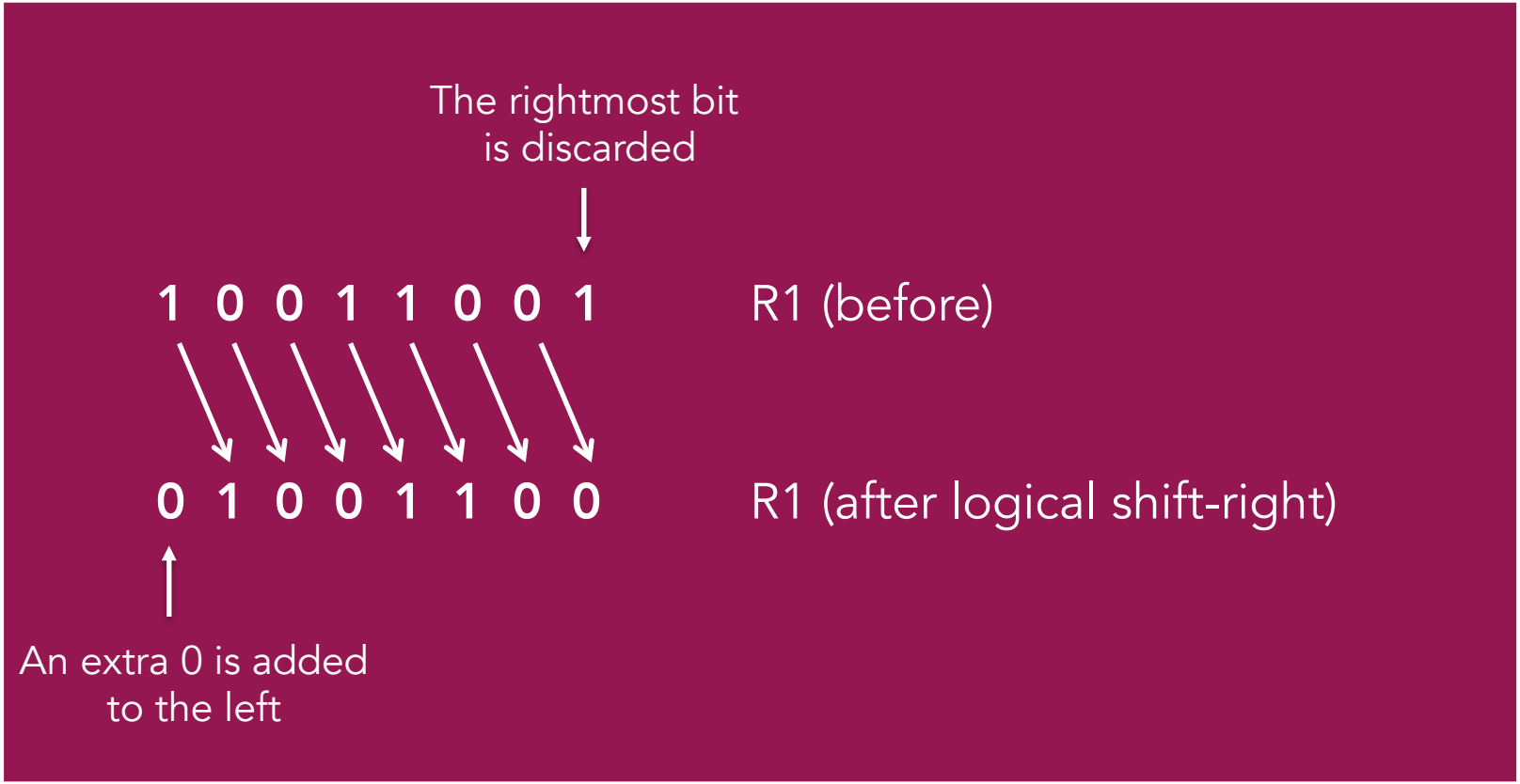
R2 ← shr R2

indicates a 1-bit shift to the right of the contents of register R2.

Example of Logical Shift-Left



Example of Logical Shift-Right



Circular Shift aka Rotation Operation

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

The microoperation

R1 ← cil R1

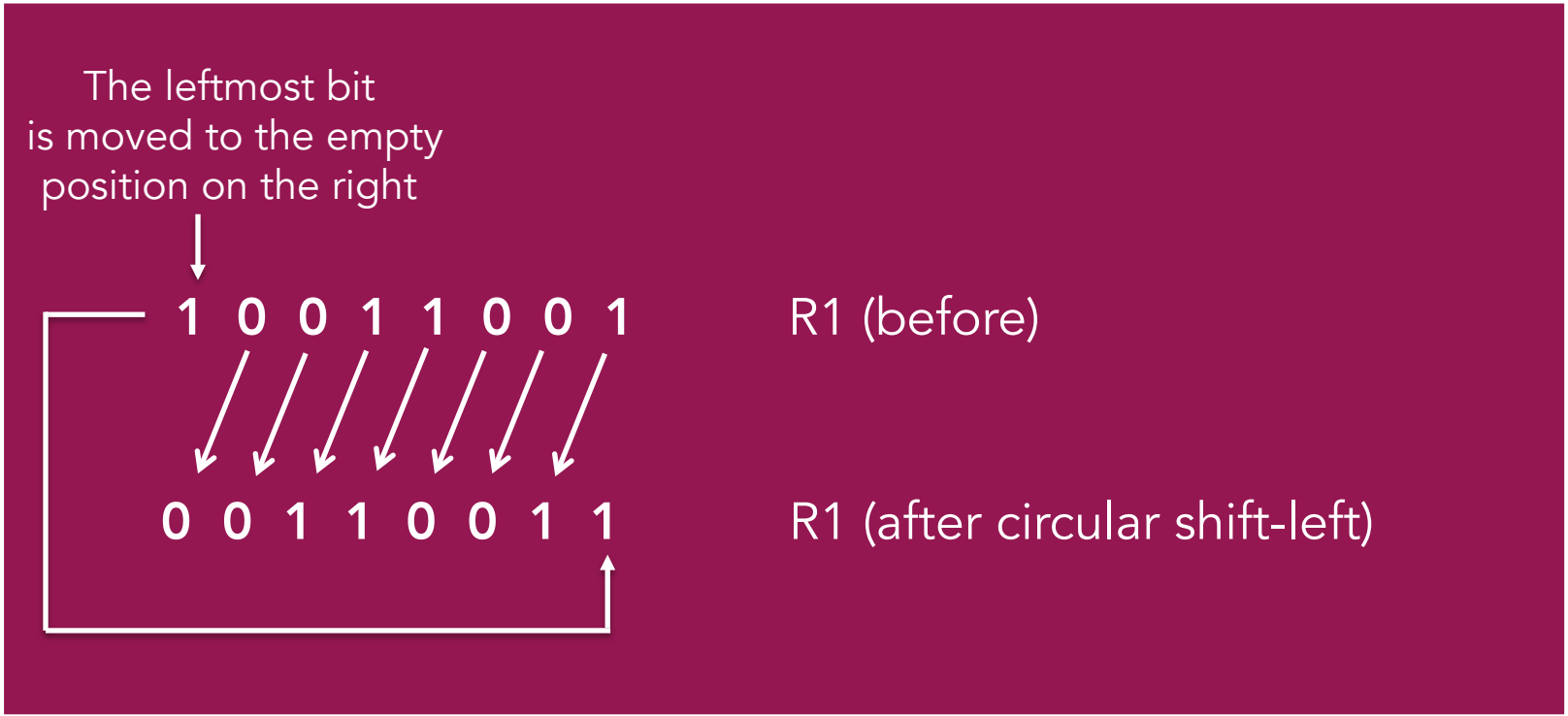
indicates a 1-bit circular shift to the left of the contents of R1

and

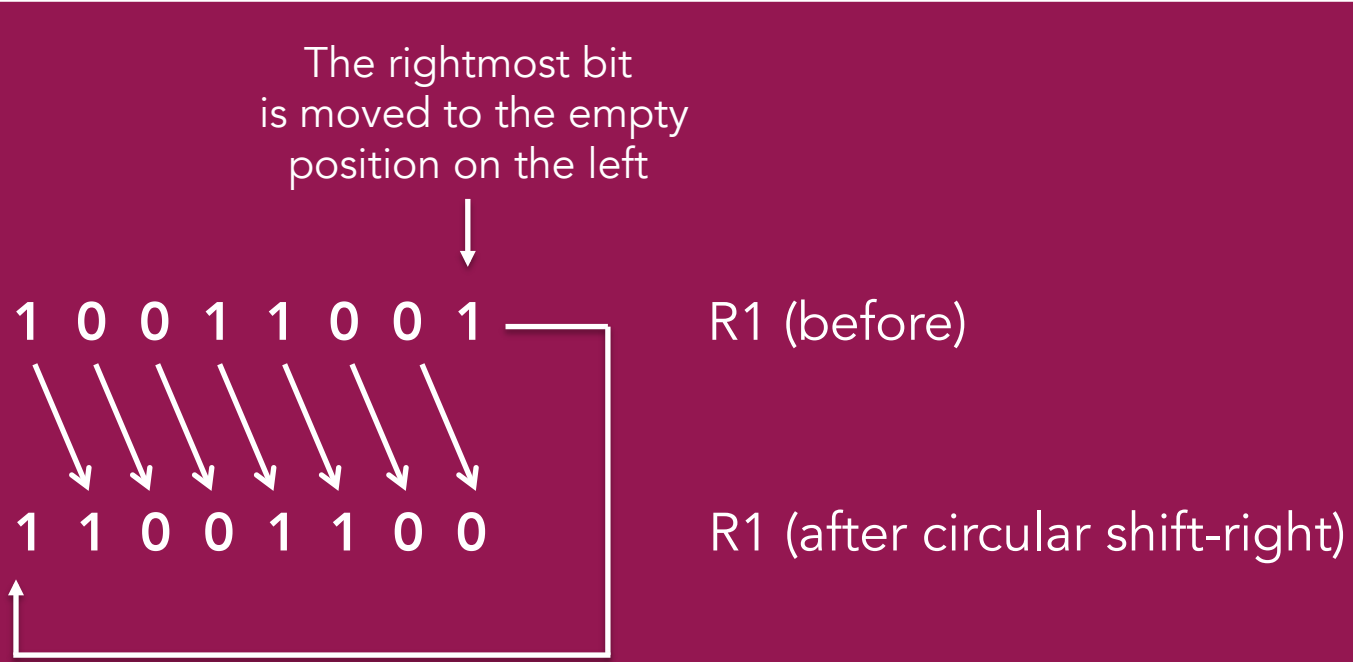
R2 ← cir R2

indicates a 1-bit circular shift to the right of contents of R2.

Example of Circular Shift-Left



Example of Circular Shift-Right



Arithmetic Shift

This shifts a **signed binary number** to the left or right.

The microoperation

R1 ← ashl R1

multiples a signed binary number by 2

and

R2 ← ashr R2

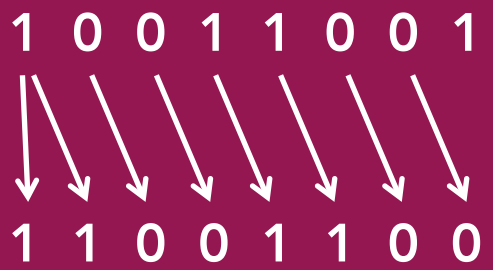
divides a signed binary number by 2.

The sign bit (i.e., the leftmost bit) must remain unchanged.

Example of Arithmetic Shift-Right

The leftmost bit is the sign bit, and it must remain unchanged.

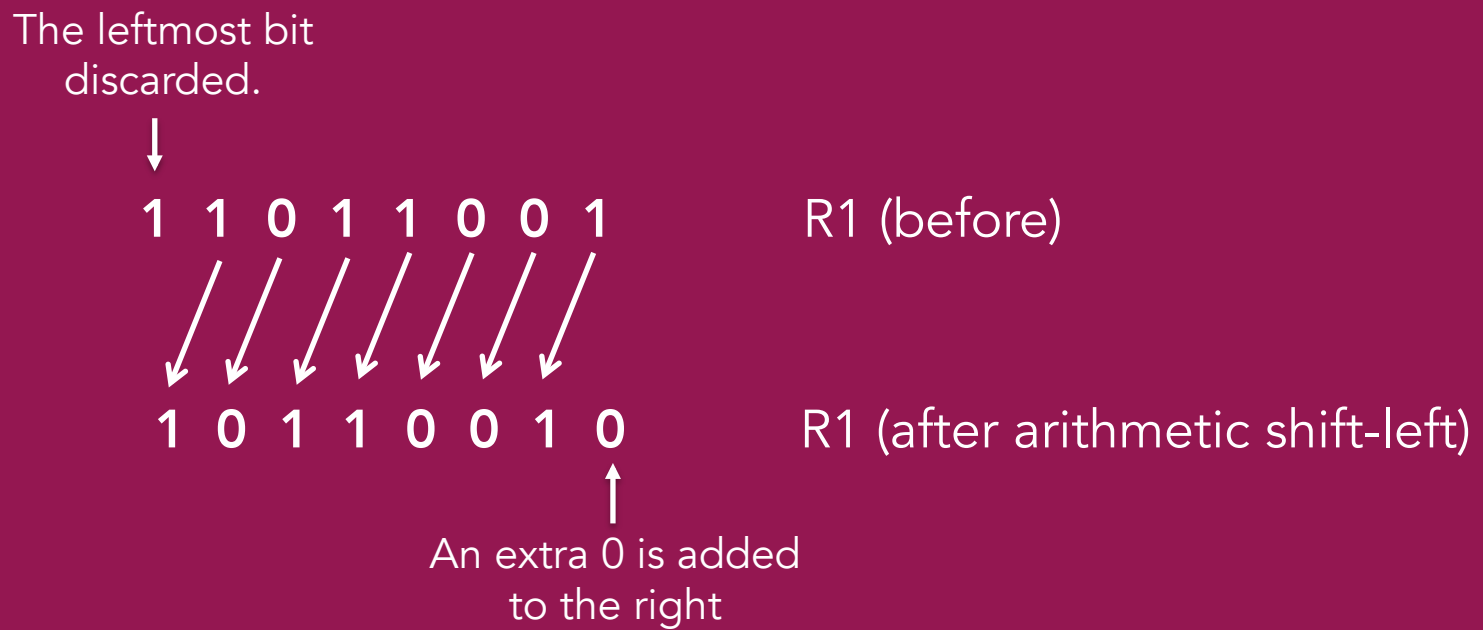
The rightmost bit is discarded



R1 (before)

R1 (after arithmetic shift-right)

Example of Arithmetic Shift-Left



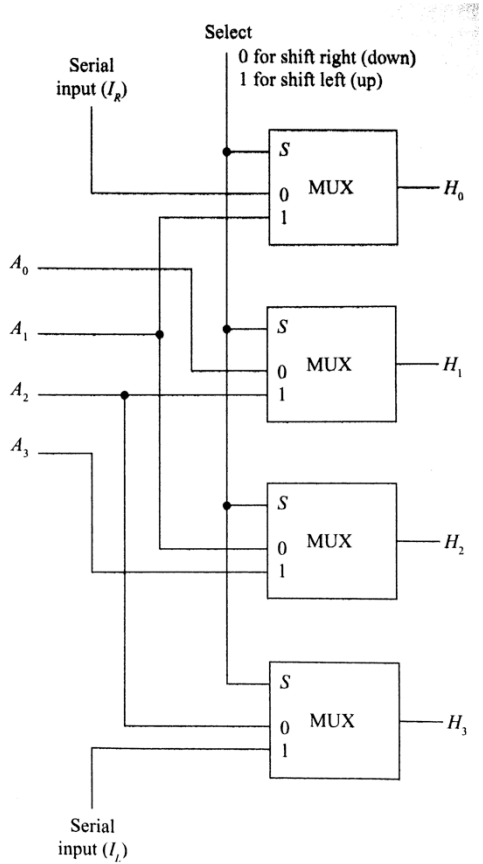
Caveats of Arithmetic Shift-Left

Arithmetic shift-left can cause the sign bit to be lost, which means that a number, which was originally negative, could become positive after the shift, and vice versa. This may result in the generation of erroneous results.

The lost sign bit is generally treated as "overflow". In other words, if an arithmetic shift-left operation causes the sign of the given number to change, overflow occurs.

1 0 0 1 1 0 0 1	R1 (negative number)
0 0 1 1 0 0 1 0	R1 (becomes positive after the shift)

Hardware Implementation of a Circuit Shifter



4-bit Combinational Circuit Shifter

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

Function Table

Exercise

Which category of microoperations does not manipulate the actual information content of a register?

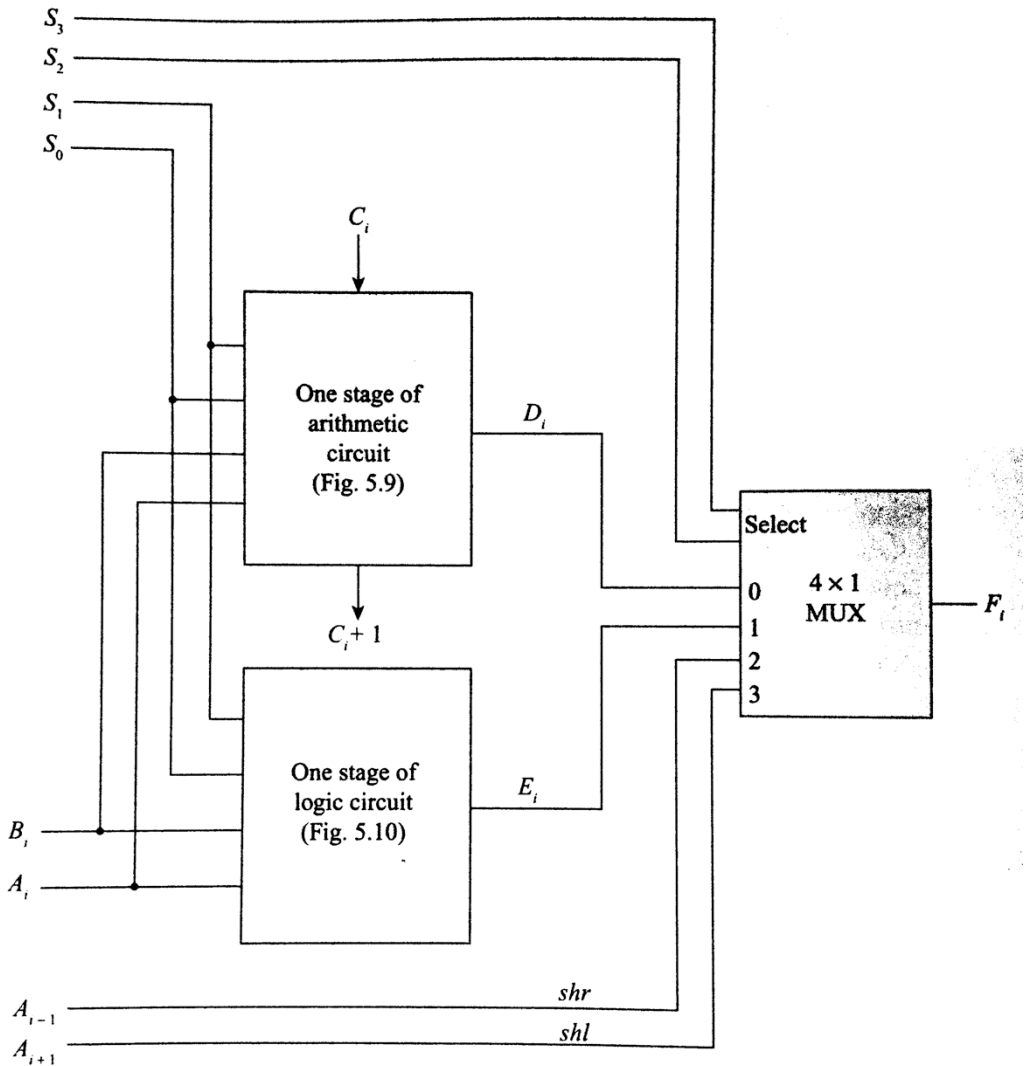
Arithmetic Logic Shift Unit

Arithmetic Logic Shift Unit

The arithmetic, logic, and shift circuits that we examined previously can all be combined into one single Arithmetic Logic Shift Unit with common selection variables.

That way, a single circuit can perform all these operations.

Arithmetic Logic Shift Unit



Function Table for Arithmetic Logic Shift Unit

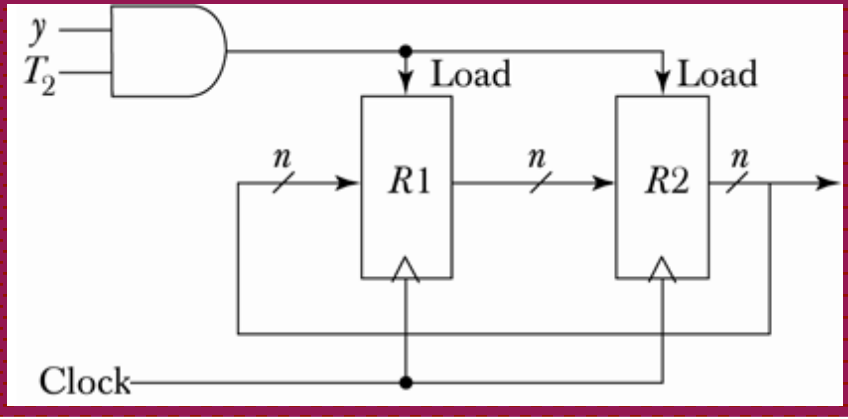
Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
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	$F = \text{shr } A$	Shift right A into F
1	1	X	X	X	$F = \text{shl } A$	Shift left A into F

Exercises

1. Show the block diagram of the hardware that implements the following register transfer statement:

yT2: R2 ← R1, R1 ← R2

Solution:



2. The outputs of four registers R0, R1, R2, and R3 are connected through 4-to-1 line multiplexer to the inputs of the fifth register, R5. Each register is 8 bits long. The required transfers are dictated by four timing variables T_0 to T_3 as follows:

T_0: R5 ← R0

T_1: R5 ← R1

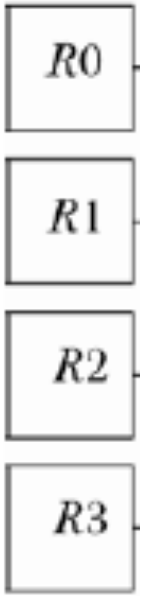
T_2: R5 ← R2

T_3: R5 ← R3

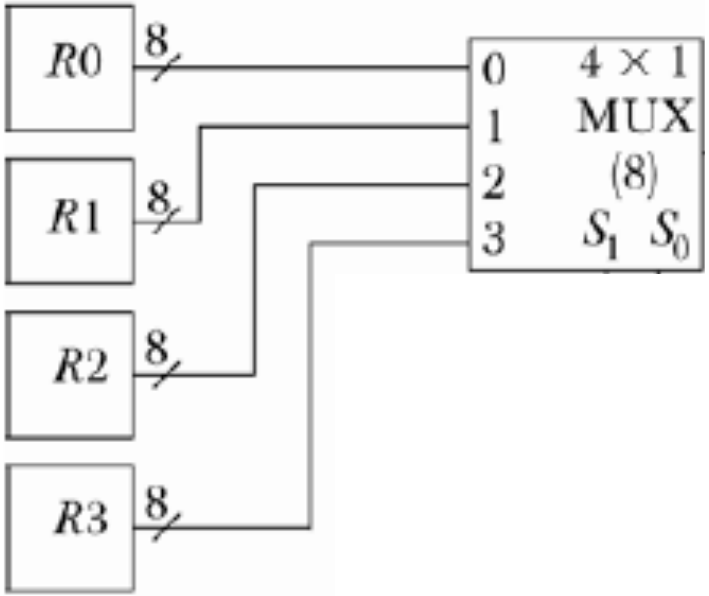
The timing variables are mutually exclusive. Draw a block diagram showing the hardware implementation of the register transfers.

Include the necessary connections from the four timing variables to the selection inputs of the mux and to the load input of register R5.

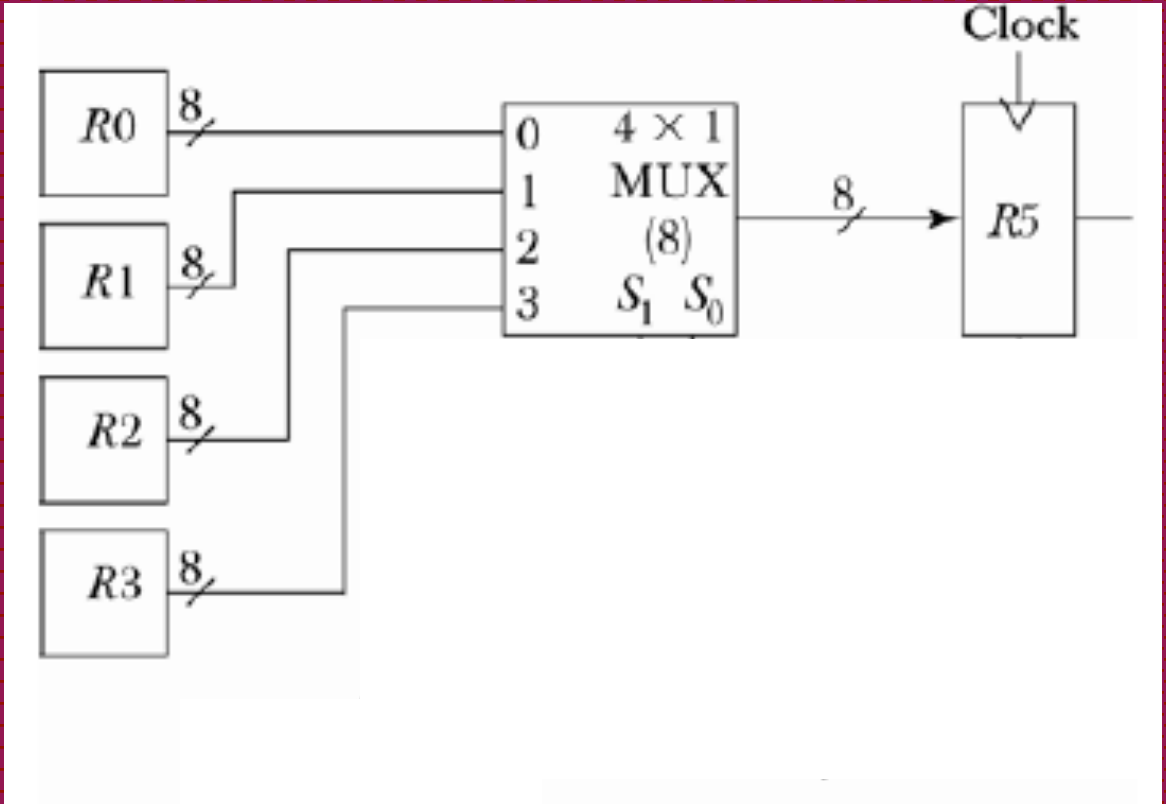
Solution: Step 1



Solution: Step 2



Solution: Step 3



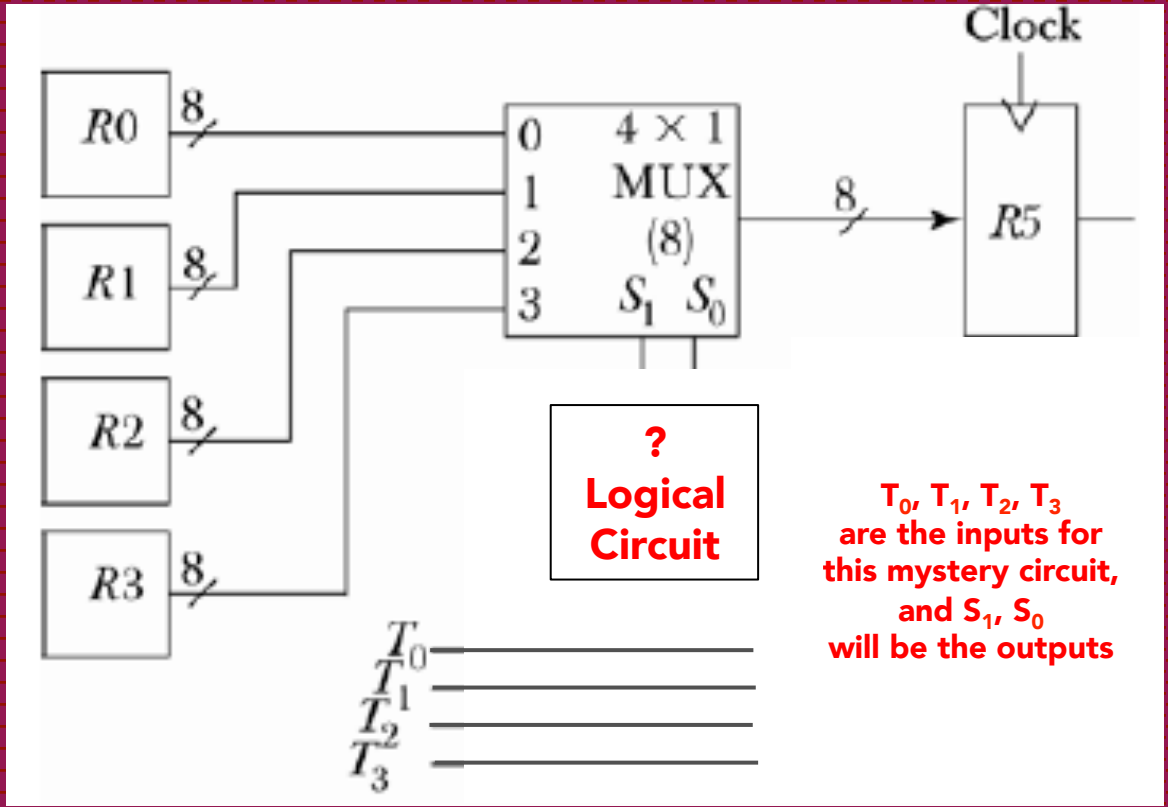
Solution: Step 4

The two selection inputs of the mux (i.e., S_1 and S_0), will help determine which of the four registers will be able to load their contents into R5.

Each register transfer is further controlled by a timing signal (at T_0, R0 will transfer its contents to R5. At T_1, R1 will transfer its contents to R5. And so on).

So now, we need to find a way to convert the four timing signals (T_0, T_1, T_2, and T_3) into the mux's selection inputs S_1 and S_0.

Solution: Step 4



Solution: Step 4

Our mux has four inputs (one from each of the four registers). One of these inputs will become the output, depending on the combination of S_1 and S_0.

S_1	S_0	The input from which register will become the output	Which timing signal which allow this register to transfer data
0	0	R0	T_0
0	1	R1	T_1
1	0	R2	T_2
1	1	R3	T_3

Solution: Step 4

Inputs to the mystery circuit

Outputs of the mystery circuit

The previous truth table can be written as:

Timing Signals T_0 T_1 T_2 T_3	S_1	S_0	Output of the mux
0 0 0 0	X	X	-
1 0 0 0	0	0	R0
0 1 0 0	0	1	R1
0 0 1 0	1	0	R2
0 0 0 1	1	1	R3

Solution: Step 4

We have a truth table that provides the values of S_1 and S_0 corresponding to the values of T_0, T_1, T_2, and T_3.

Now we need to find simplified expressions that represent these relationships.

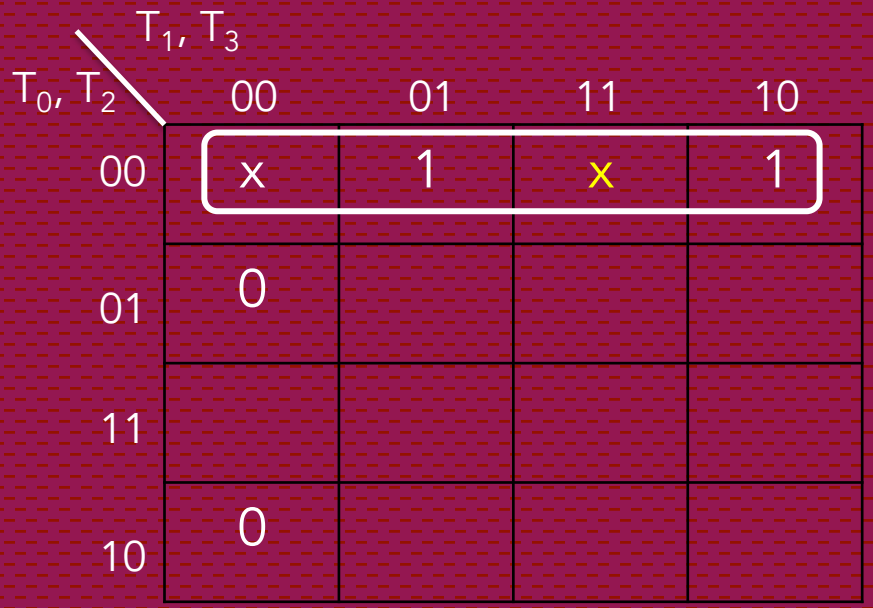
We'll use K-maps.

Solution: Step 5

K-Map for S_1

T_0, T_1 \ T_2, T_3	00	01	11	10
00	x	1		1
01	0			
11				
10	0			

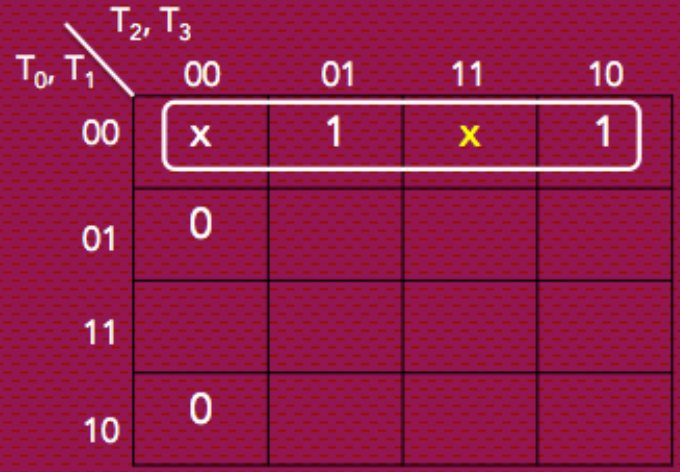
Solution: Step 5



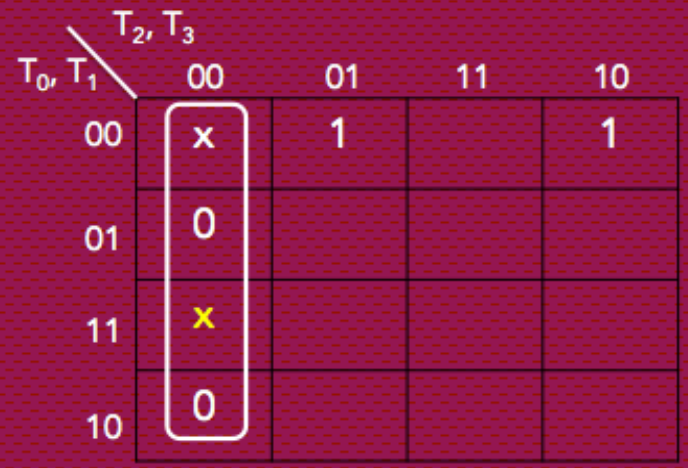
$$S_0 = T_0'T_2'$$

Solution: Step 5

Simplified K-Maps for S_1



OR



$S_1 = T_0'T_1'$

$S_1 = T_2+T_3$ (POS)

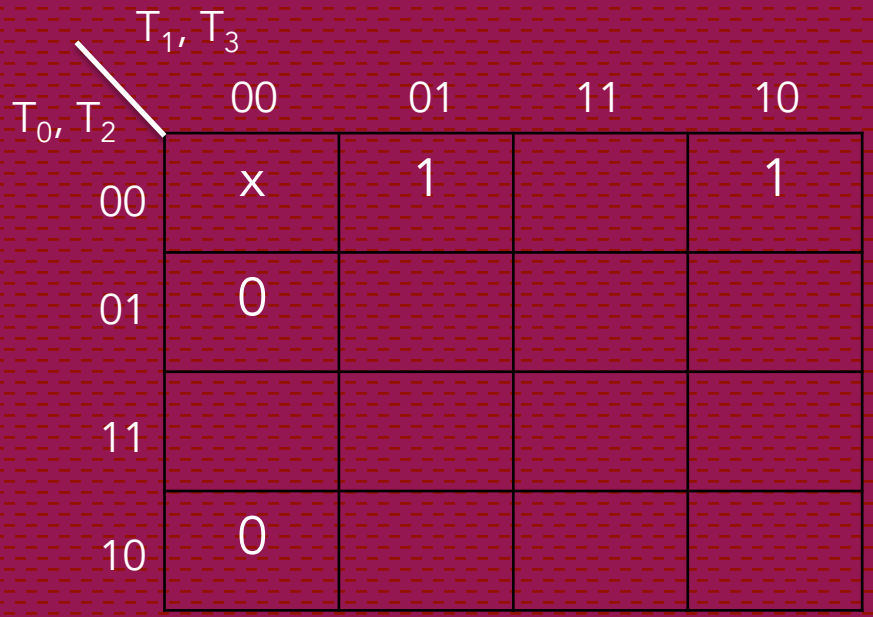
Solution: Step 6

K-Map for S_0

T_0, T_1 \ T_2, T_3	00	01	11	10
00	x	1		0
01	1			
11				
10	0			

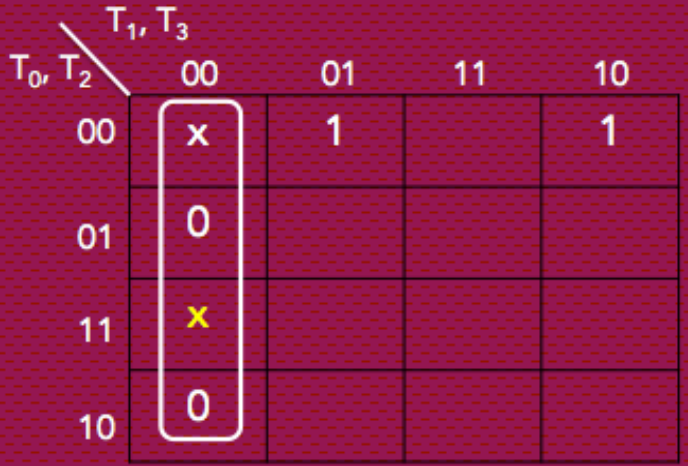
Solution: Step 6

Alternative K-Map for S_0



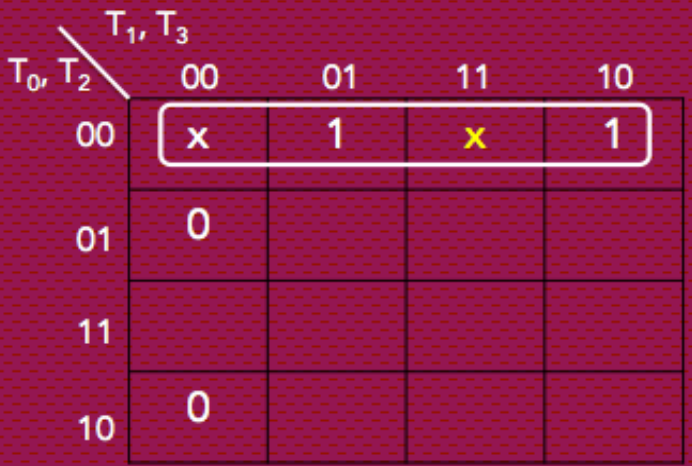
Solution: Step 5

Simplified Alternative K-Maps for S_0



$S_0 = T_1 + T_3$ (POS)

OR



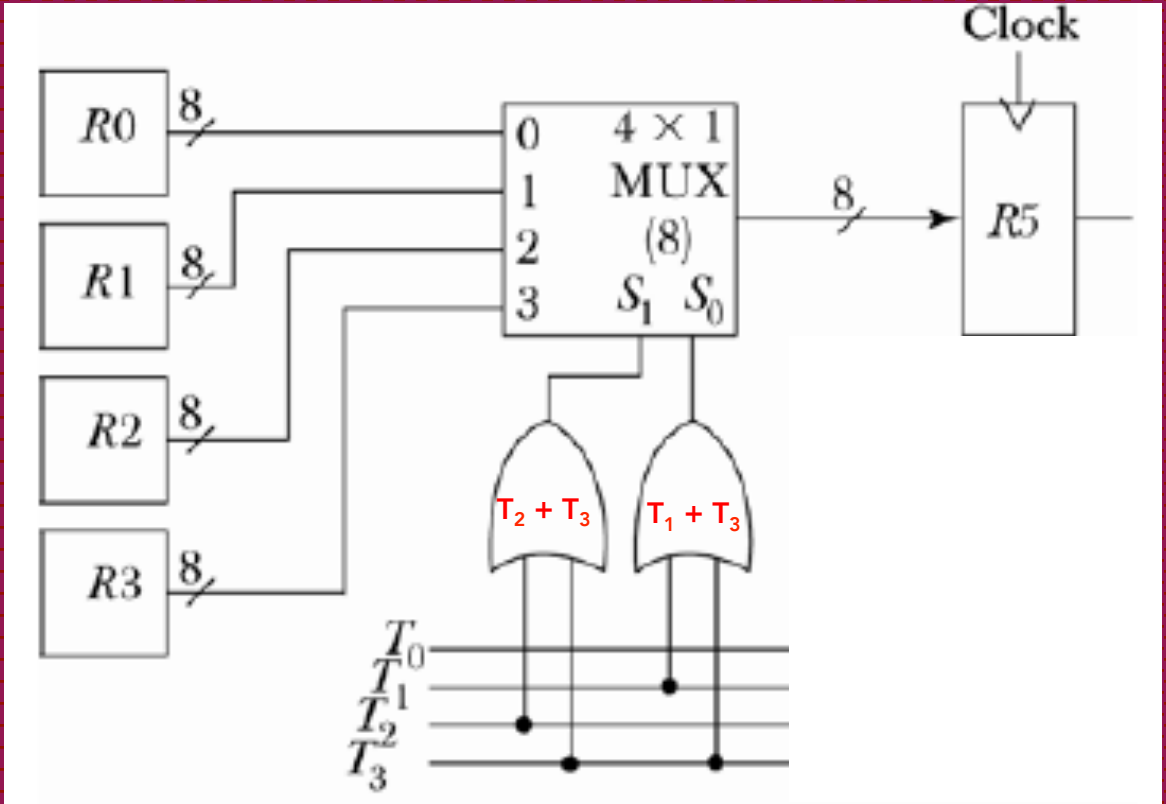
$S_0 = T_0'T_2'$

Solution: Step 5

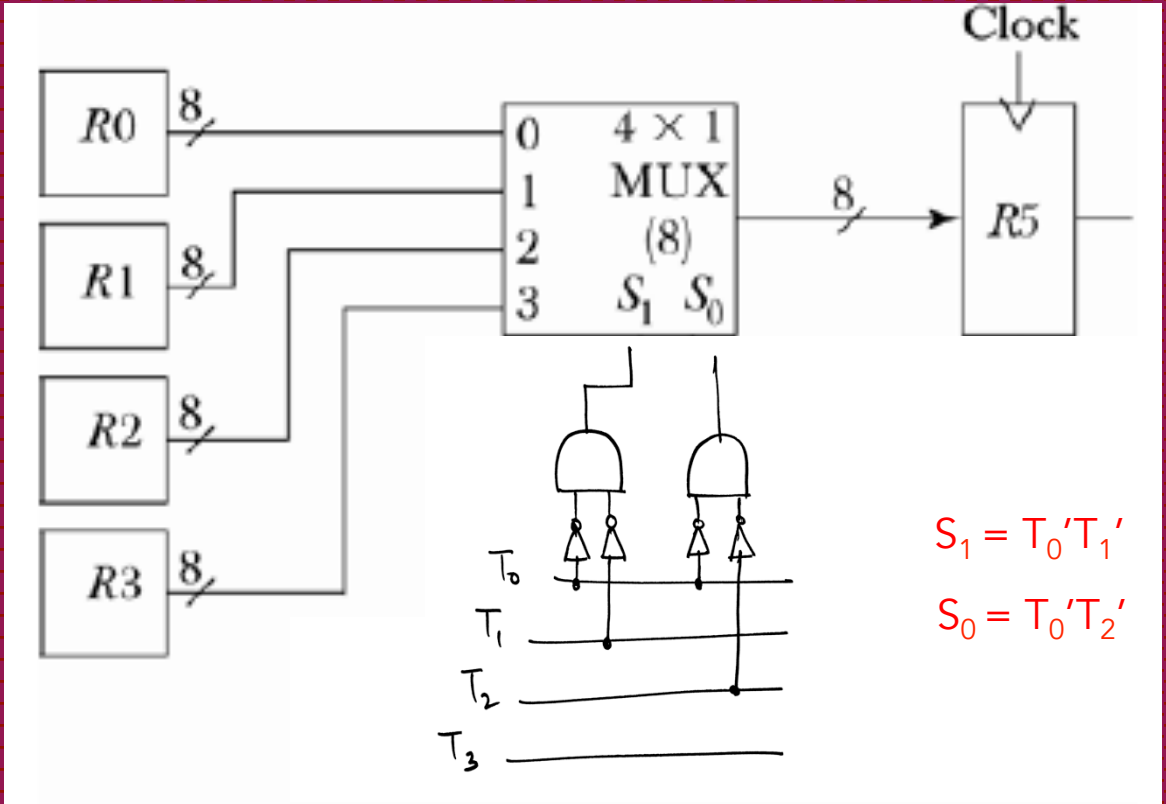
Crosscheck your answer

Timing Signals T_0 T_1 T_2 T_3	S_1	S_0	$S_1 = T_2+T_3$	$S_0 = T_1+T_3$
1 0 0 0	0	0	0 + 0 = 0	0 + 0 = 0
0 1 0 0	0	1	0 + 0 = 0	1 + 0 = 1
0 0 1 0	1	0	1 + 0 = 1	0 + 0 = 0
0 0 0 1	1	1	0 + 1 = 1	0 + 1 = 1

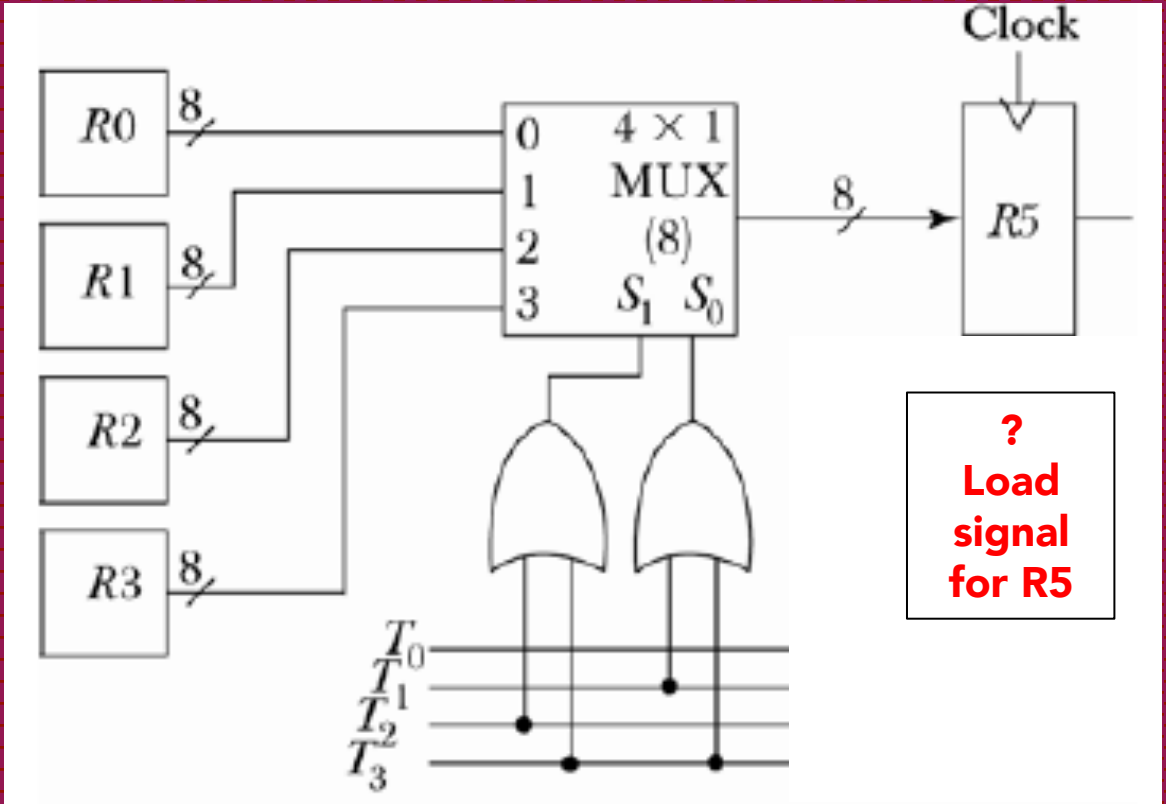
Solution: Step 6



Alternative Solution (Using SOP Simplification)



Solution: Step 7



Solution: Step 7

What about the load signal for R5?

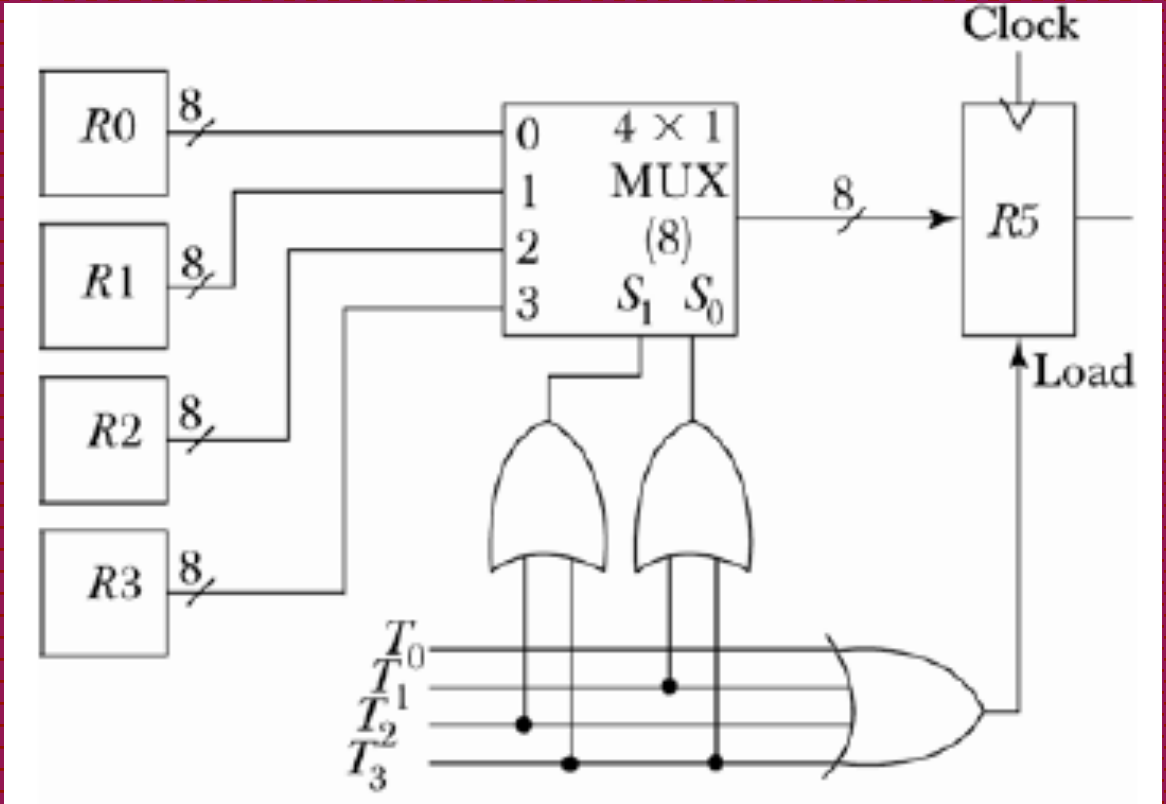
$$\text{Load} = T_0 + T_1 + T_2 + T_3$$

R5 must be connected to all the timing signals (since different timing signals enable different registers to transfer their contents to R5).

Also, only one of the timing signals will be 1 at any given time, and we want the load signal of R5 to be 1 for every timing signal.

So, we have to use an OR gate.

Solution: Step 7



3. Represent the following conditional control statement by two register transfer statements with control functions.

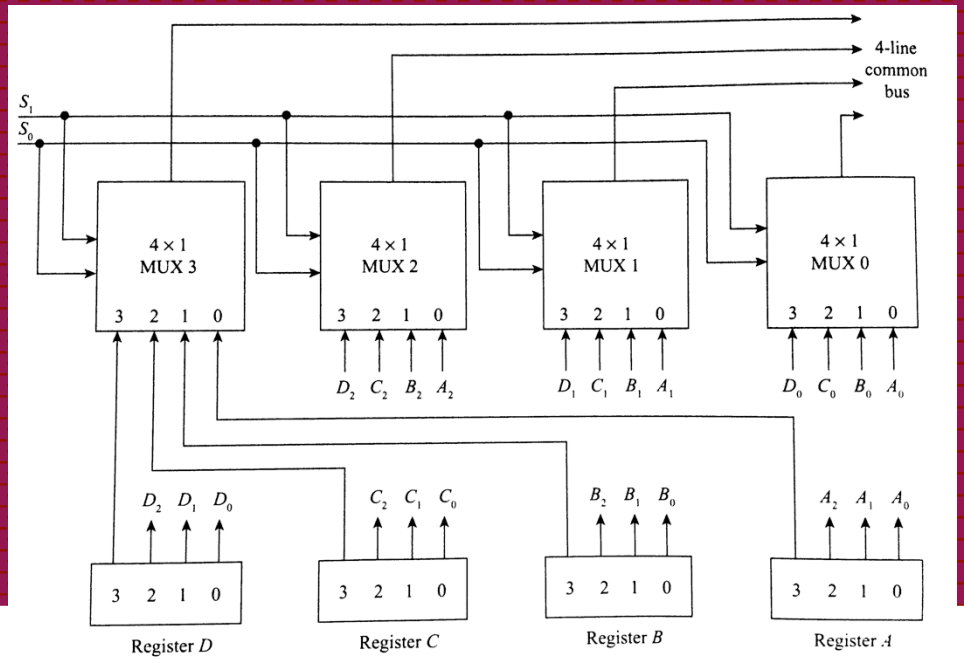
If (P=1) then (R1 ← R2) else if (Q=1) then (R1 ← R3)

Solution:

P: R1 ← R2

P′Q: R1 ← R3

4. What has to be done to the bus system in the following figure to be able to transfer information from any register to any other register? Specifically, show the connections that must be included to provide a path form the outputs of register C to the inputs of register A.

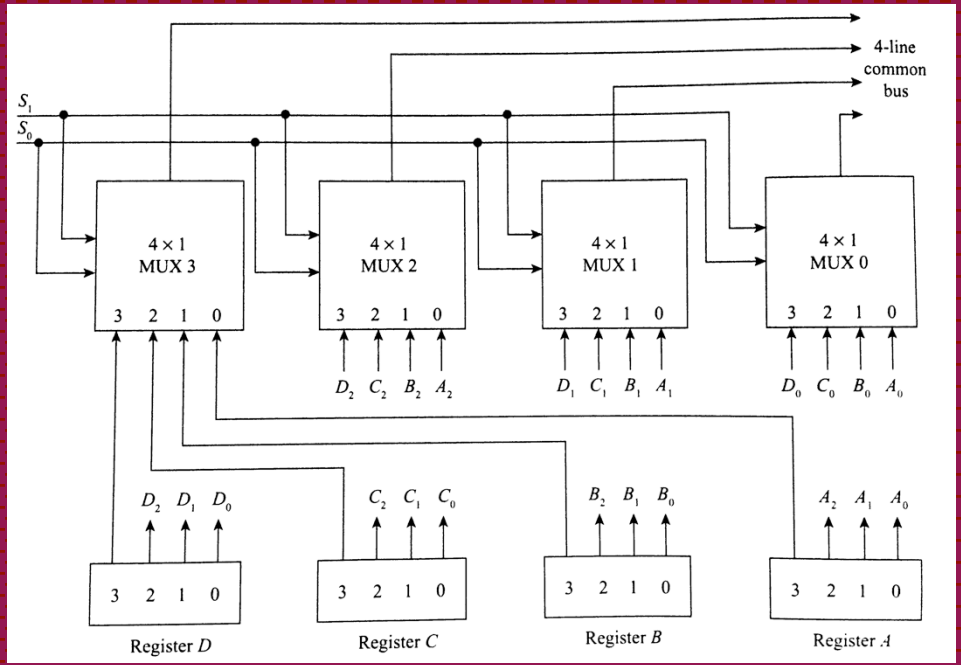


Solution:

Connect the 4-line common bus to the four inputs of each register.

Provide "load control" and

"clock" inputs for each register.



To transfer from C to A:

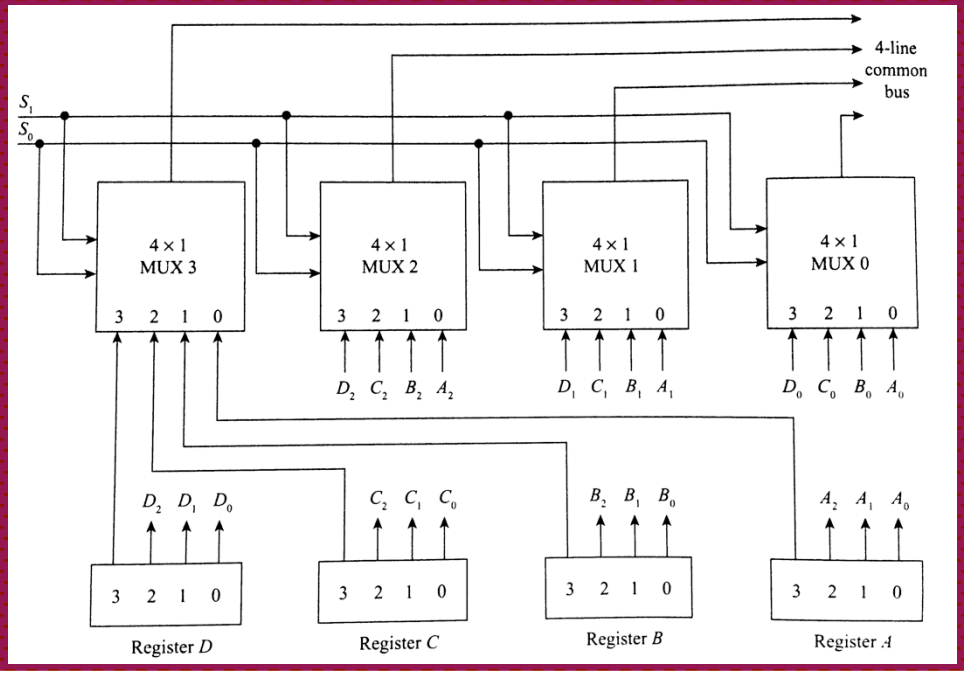
1. Apply $S_1S_0 = 10$

   (to select C for the bus.)

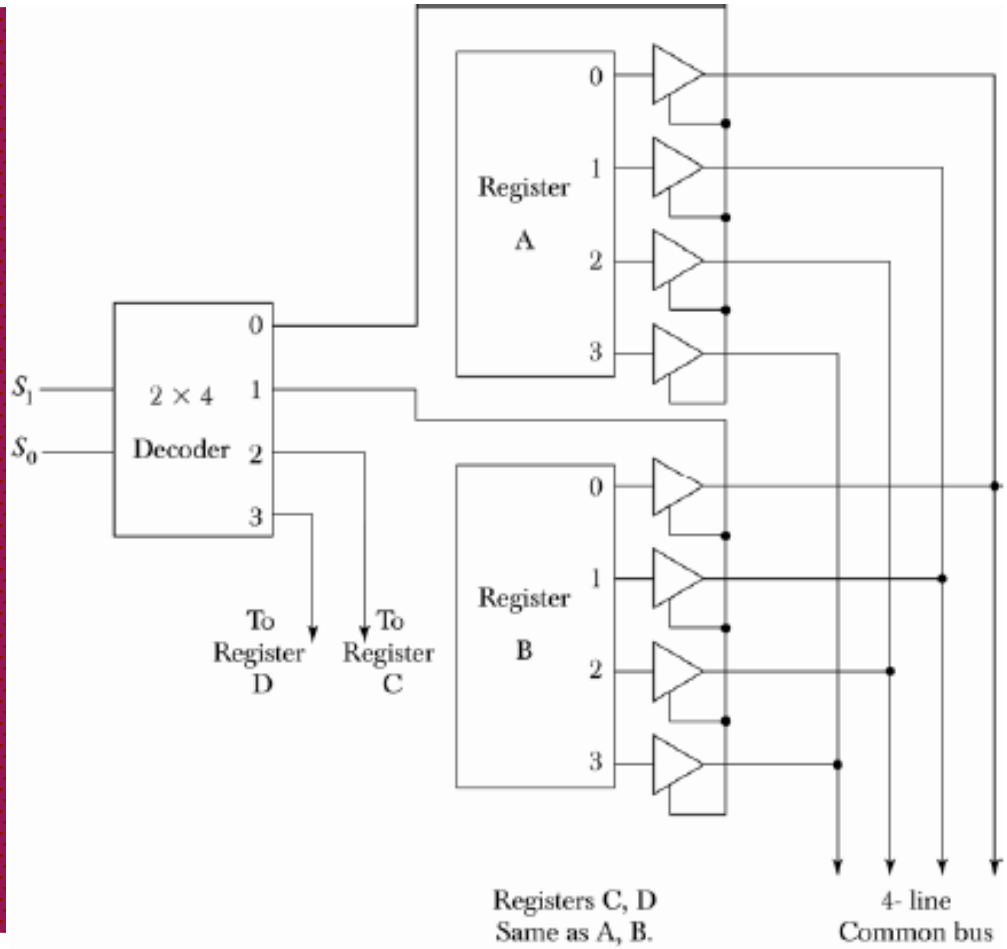
2. Enable the load input of A

3. Apply a clock pulse.

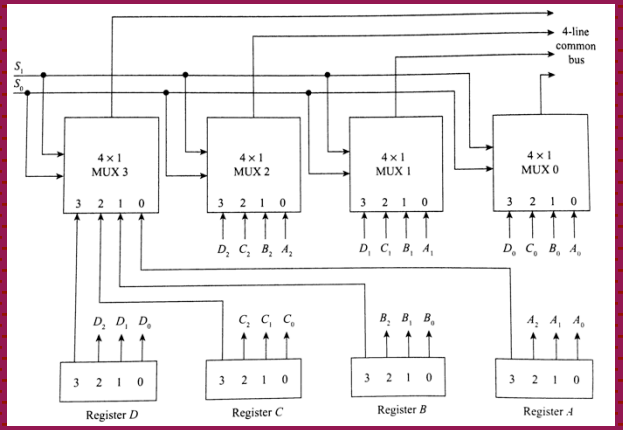
5. Draw a diagram for a bus system similar to the one shown in the following figure, but use three-state buffers and a decoder instead of the multiplexers.

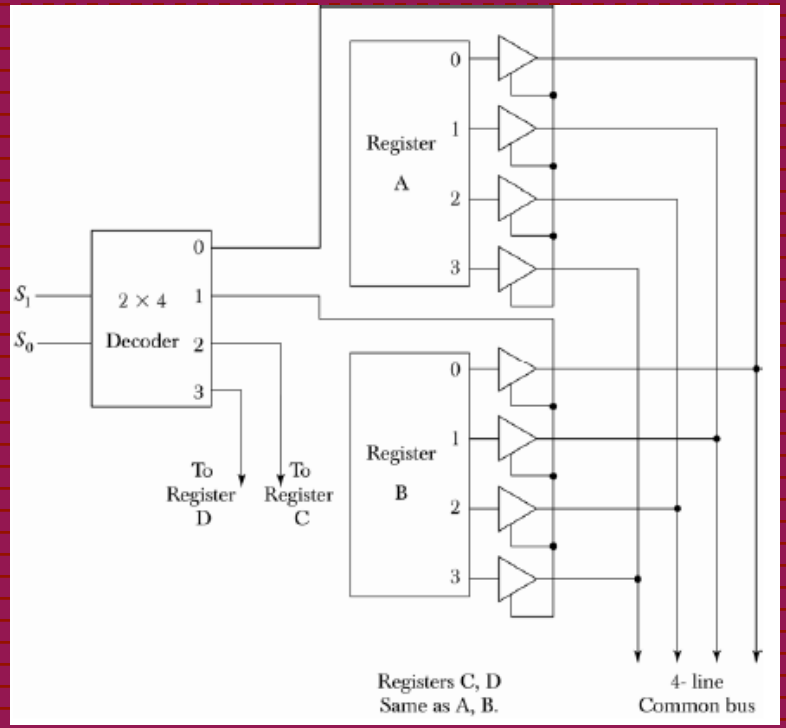


Solution:


Register
A
Register
B
0
1
2
3
S_1
S_0
2 × 4
Decoder
To
Register
D
To
Register
C
Registers C, D
Same as A, B.
4- line
Common bus

Solution:


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
Register D
Register C
Register B
Register A


S_1
S_0
2 × 4 Decoder
0
1
2
3
To Register D
To Register C
Register A
Register B
Registers C, D Same as A, B.
4- line Common bus

6. A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed using multiplexers.

a. What size of multiplexers are needed?

b. How many selection inputs are there in each multiplexer?

c. How many multiplexers are there in the bus system?

Solution:

a. What size of multiplexers are needed?

16 x 1

b. How many selection inputs are there in each multiplexer?

4 selection lines (because $2^4=16$)

c. How many multiplexers are there in the bus system?

32 (one for each bit of the registers).

7. The following transfer statement specify a memory. Explain the memory operation in each case.

a. R2 ←M[AR]

b. M[AR] ← R3

c. R5 ← M[R5]

Solution:

a. R2 ←M[AR]

Read the memory word specified by the address in AR into R2.

b. M[AR] ← R3

Write the contents of R3 into the memory word specified by the address in AR.

c. R5 ← M[R5]

Read the memory word specified by the address in R5 into R5. (This will overwrite the previous value of R5.)

8. Draw a block diagram of the hardware that implements the following register transfer statement:

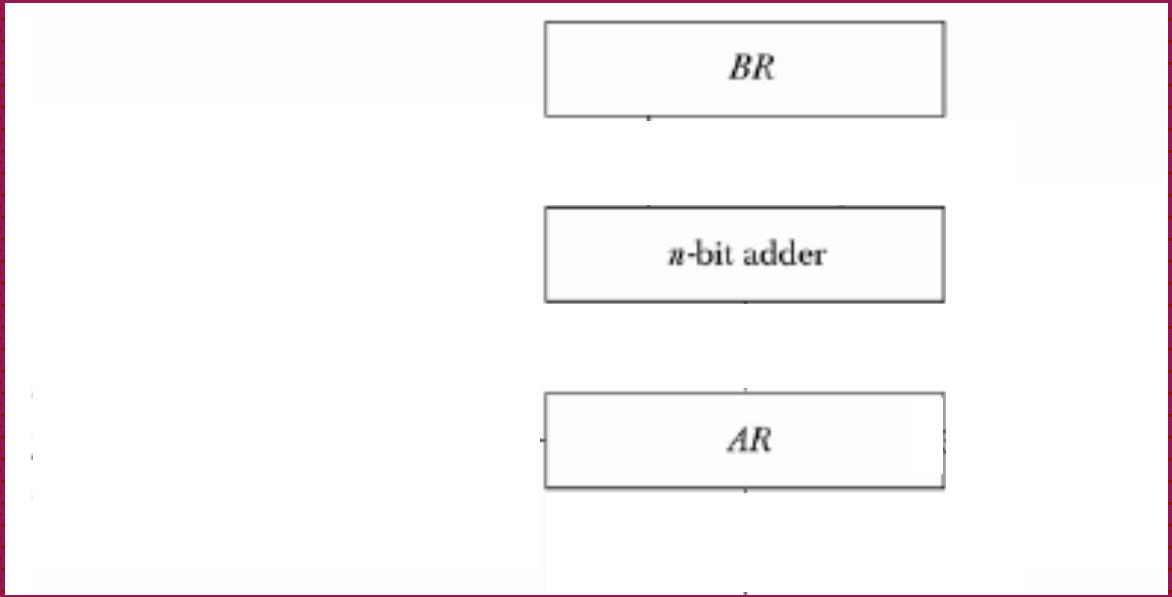
x+yz: AR ← AR+BR

AR and BR are two n-bit registers, and x, y, z are control variables. Include the logic gates for the control functions.

Remember that a + sign denotes an OR operation in a control or Boolean function but it represents an arithmetic plus in a microoperation.

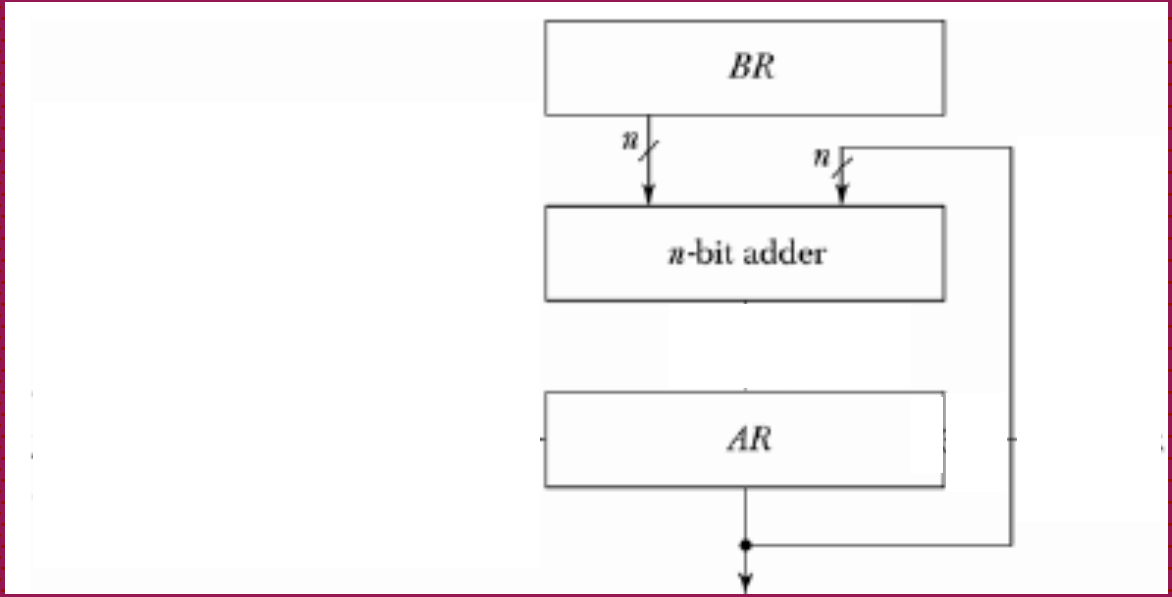
Solution: Step 1

x+yz: AR ← AR+BR



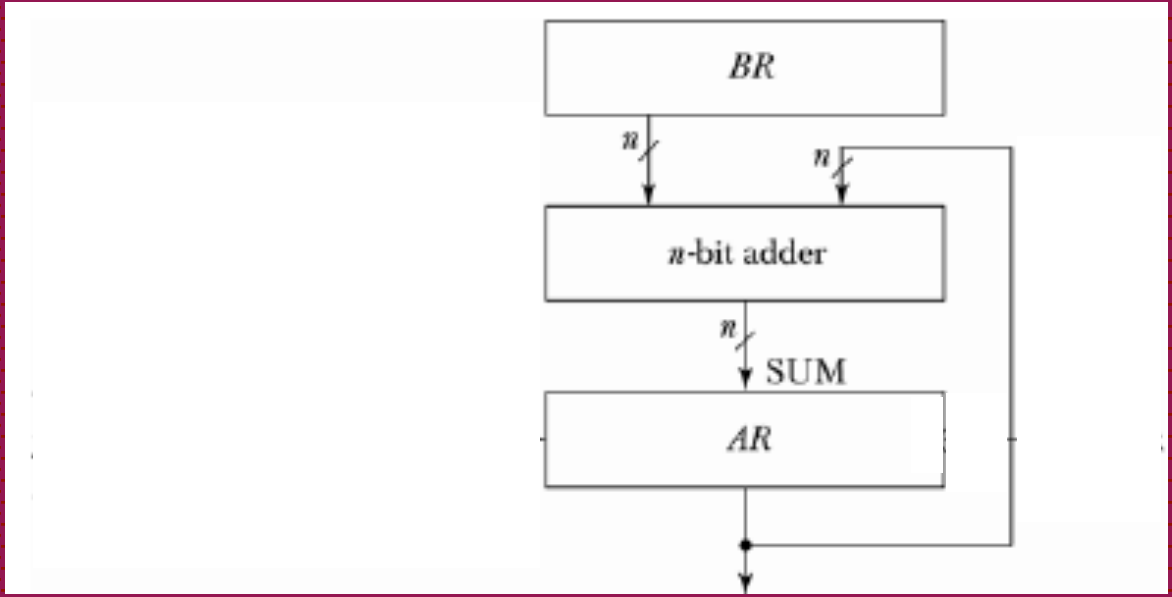
Solution: Step 2

x+yz: AR ← AR+BR



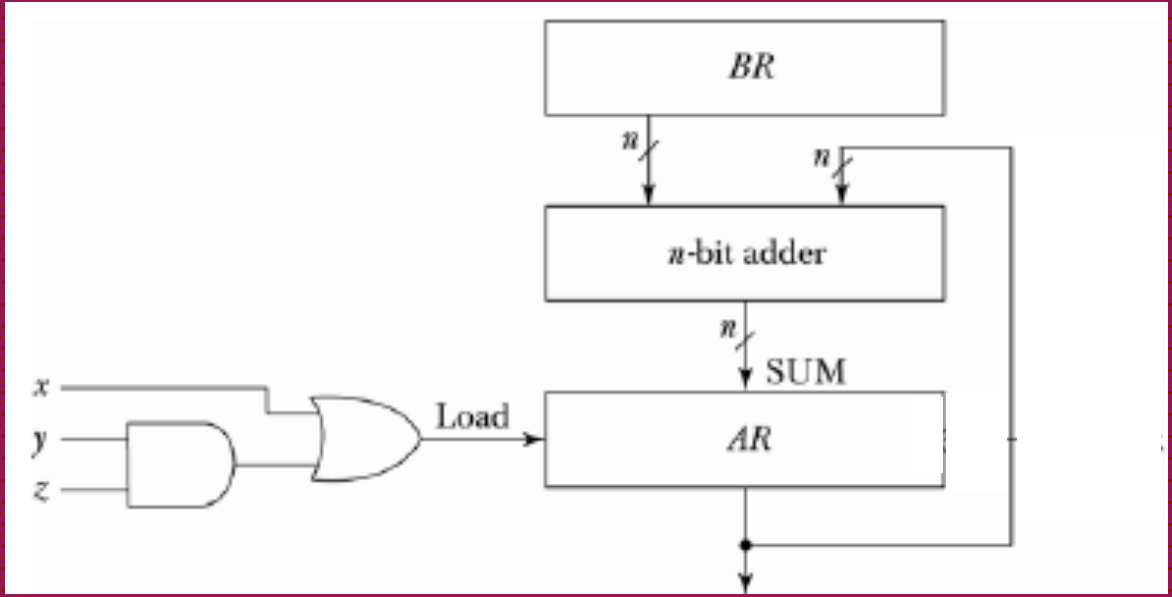
Solution: Step 3

x+yz: AR ← AR+BR



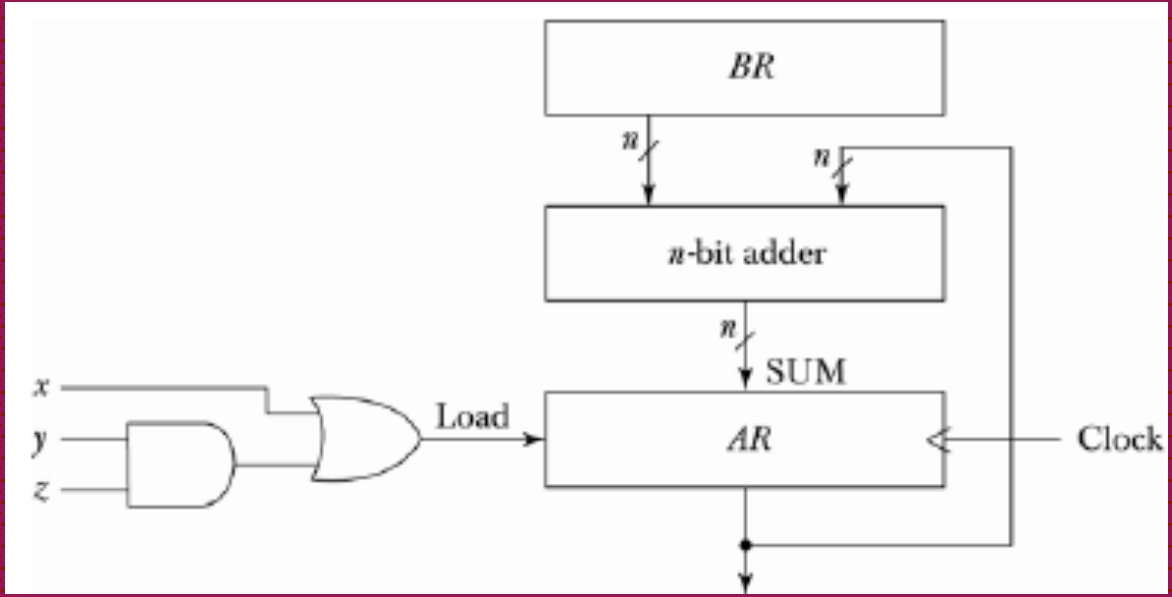
Solution: Step 4

x+yz: AR ← AR+BR



Solution: Step 5

x+yz: AR ← AR+BR



9. Show the hardware that implements the following statement:

$$xyT_0+T_1+ y'T_2: AR \leftarrow AR+1$$

Include the logic gates for the control function and a block diagram for a binary counter with a count enable input.

Solution: Step 1

$xyT_0+T_1+ y'T_2$: $AR \leftarrow AR+1$ → Binary Counter

Solution: Step 2

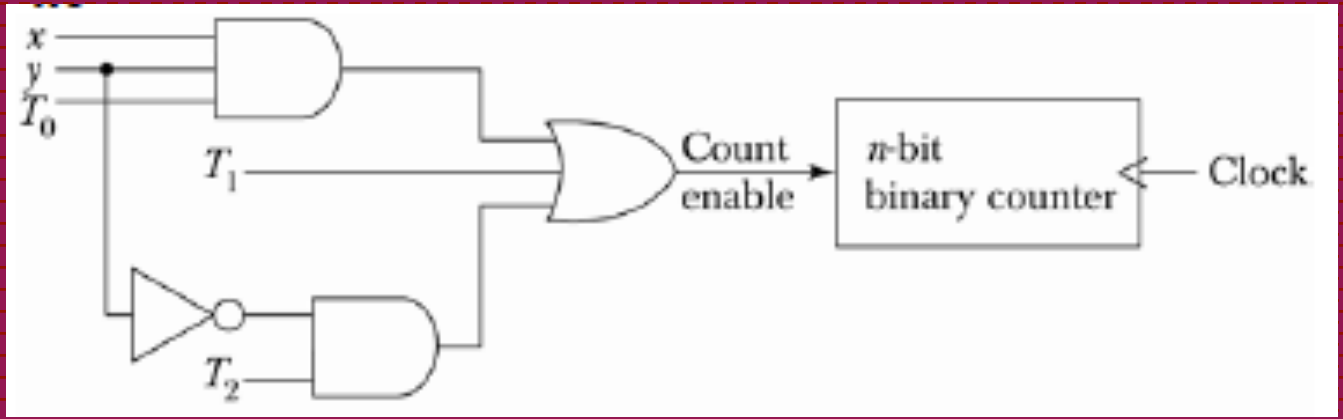
$xyT_0+T_1+ y'T_2$: AR ← AR+1

Solution: Step 3

$xyT_0+T_1+ y'T_2$: AR ← AR+1

Solution: Step 4

$xyT_0+T_1+ y'T_2$: AR ← AR+1



10 . Consider the following register transfer statements for two 4-bit registers R1 and R2:

xT: R1 ← R1+R2

x'T: R1 ← R2

Every time T=1, either the contents of R2 are added to the contents of R1 (if x=1), or the contents of R2 are transferred to R1 (if x=0). Draw a diagram showing the hardware implementation of the two statements. Use block diagrams for the registers, a 4-bit adder, and a quadruple 2-to-1 line mux that selects the inputs to R1. In the diagram, show how x and T select the inputs of the mux and the load input of R1.

Solution: Step 1

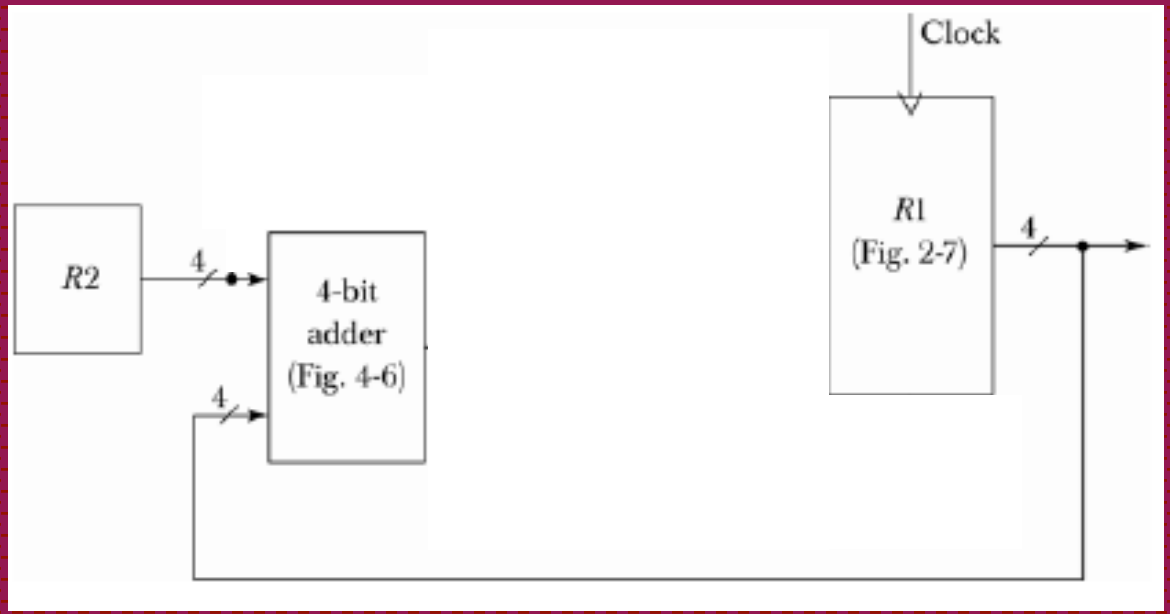
xT: R1 ← R1+R2

x'T: R1 ← R2

Solution: Step 2

xT: R1 ← R1+R2

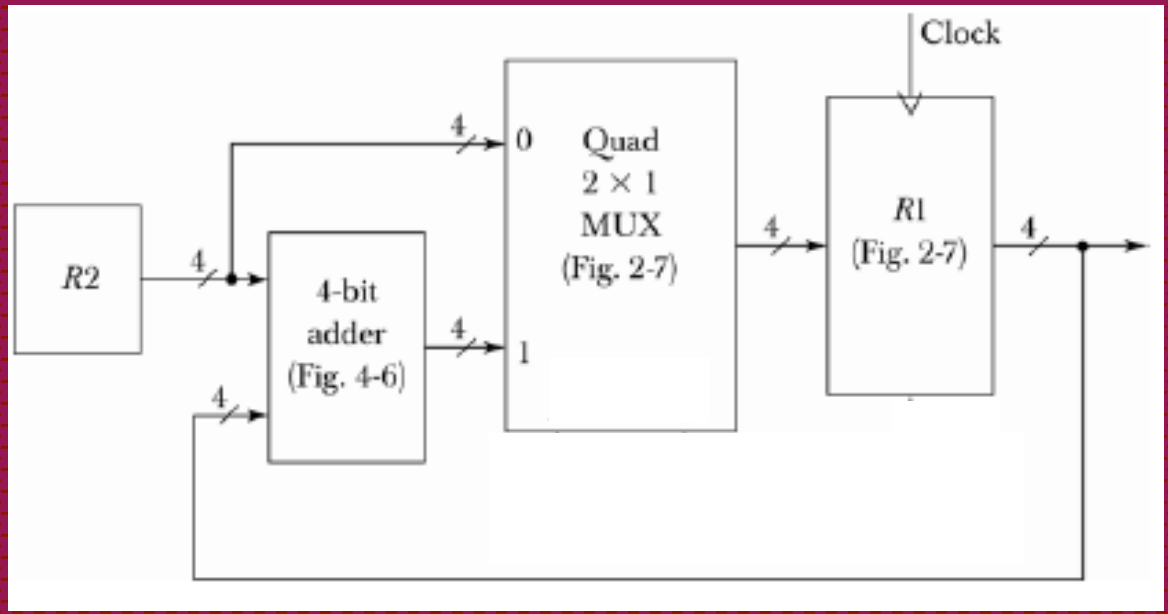
x'T: R1 ← R2



Solution: Step 3

xT: R1 ← R1+R2

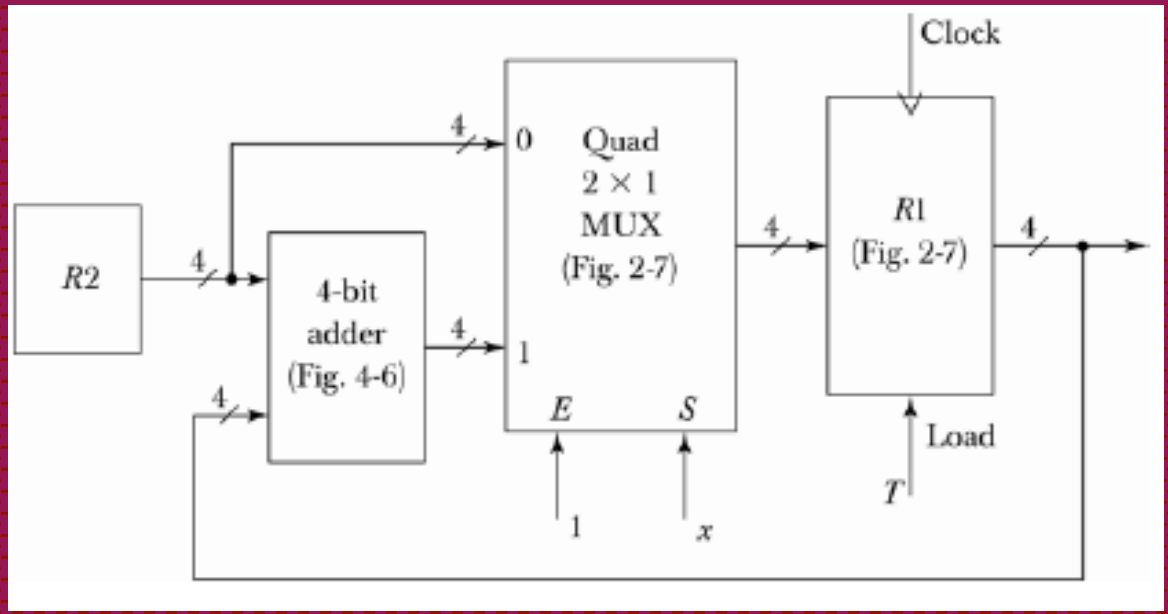
x'T: R1 ← R2



Solution: Step 4

xT: R1 ← R1+R2

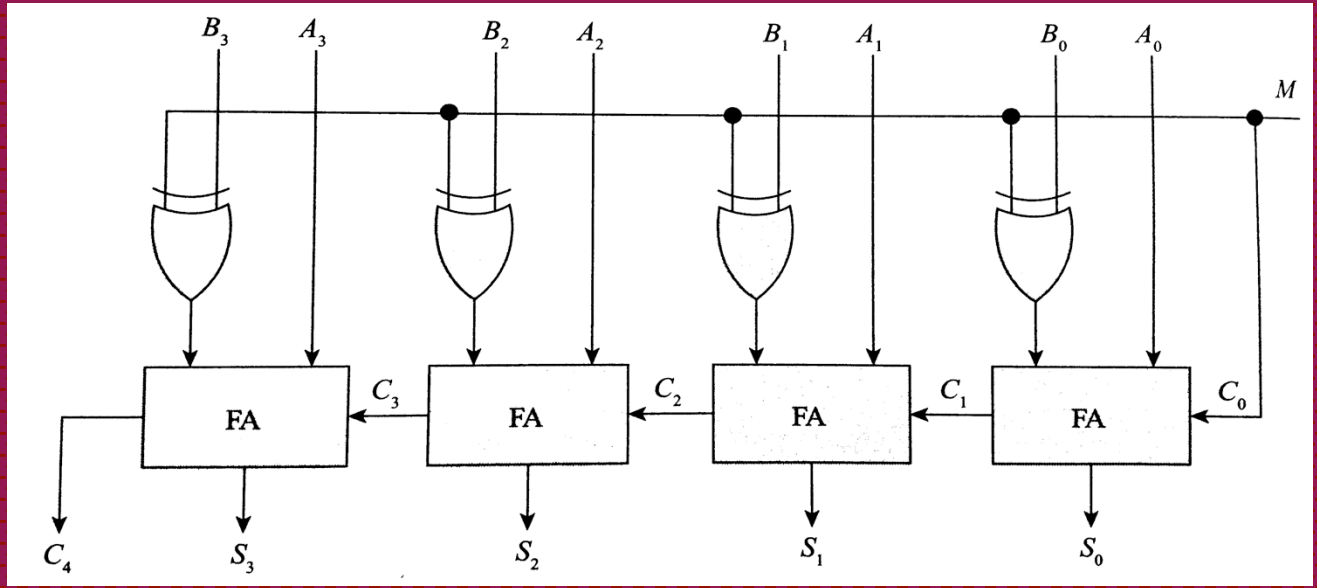
x'T: R1 ← R2



12. The adder-subtractor circuit shown below has the following values for input mode M and data inputs A and B. In each case, determine the values of the outputs S_3, S_2, S_1, S_0, and C_4.

M	A	B
0	0111	0110
0	1000	1001
1	1100	1000
1	0101	1010
1	0000	0001

Adder-Subtractor



Solution:

When mode M = 0, the circuit acts as an adder.

When M= 1, the circuit acts as a subtractor. (subtraction in 2's complement form)

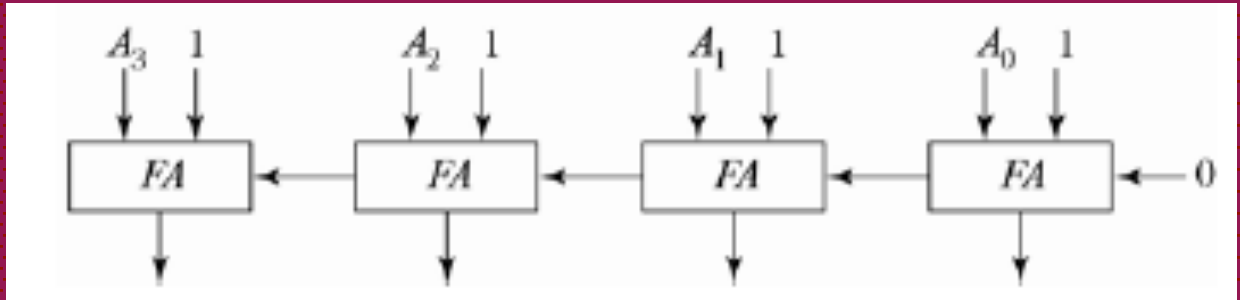
M	A	B	Sum/Difference	Carry (C_4)
0	0111	0110	0111 + 0110 = 1101	0
0	1000	1001	1000 + 1001 = 0001	1
1	1100	1000	1100 – 1000 = 0100	1
1	0101	1010	0101 – 1010 = 1011	0
1	0000	0001	0000 – 0001 = 1111	0

13. Design a 4-bit combinational circuit decrementer using four full-adders.

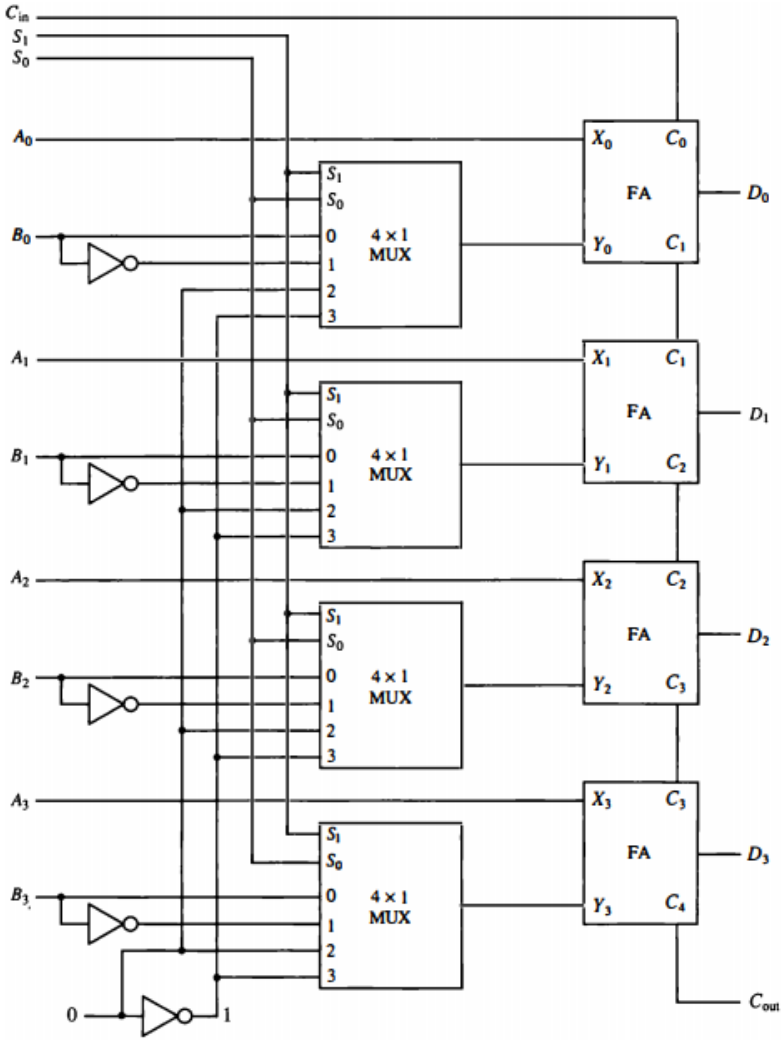
Solution: A decrementer operates by subtracting 1 from the given number, i.e., it adds (-1) to the number.

So, A – 1 = A+ (2's complement of 1) = A + 1111 ⟶

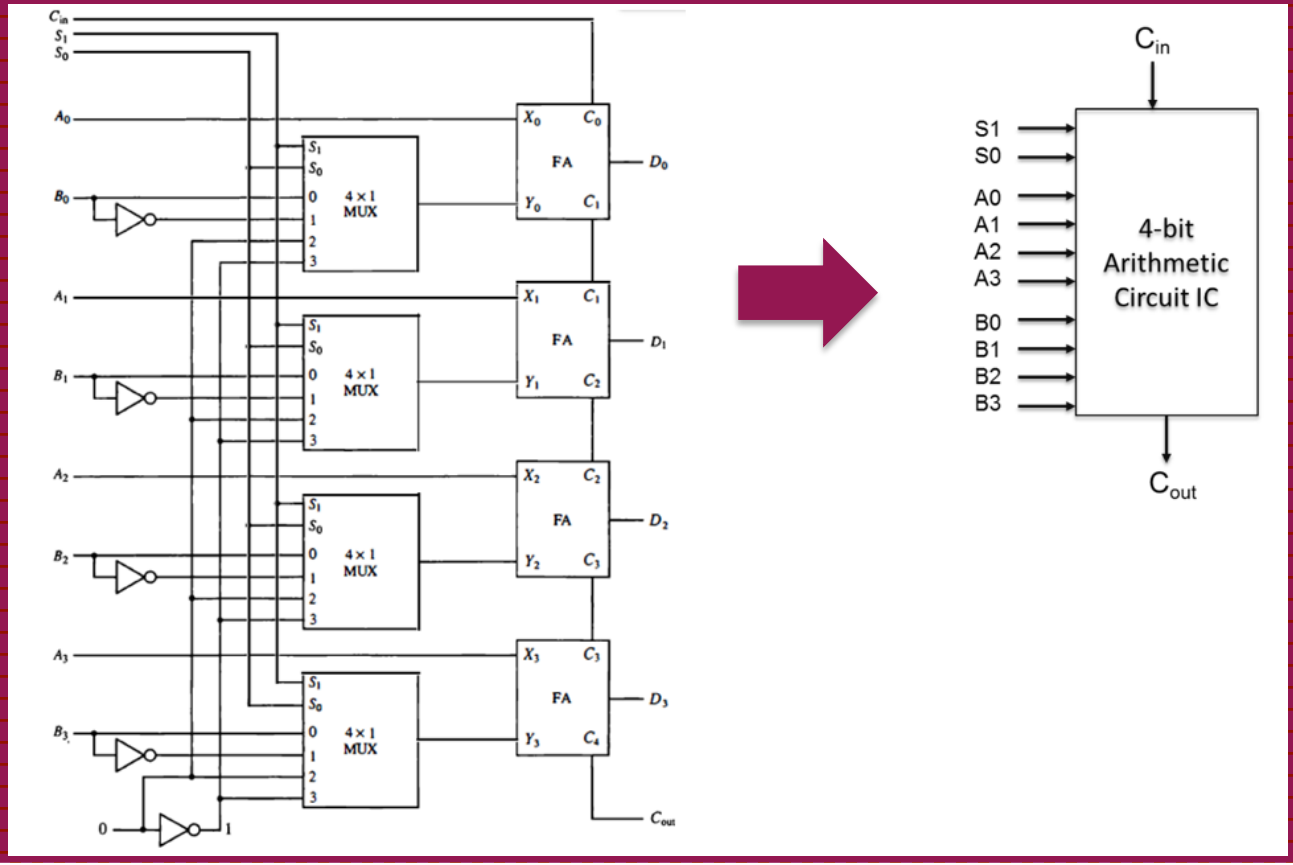
	A_3	A_2	A_1	A_0
+	1	1	1	1



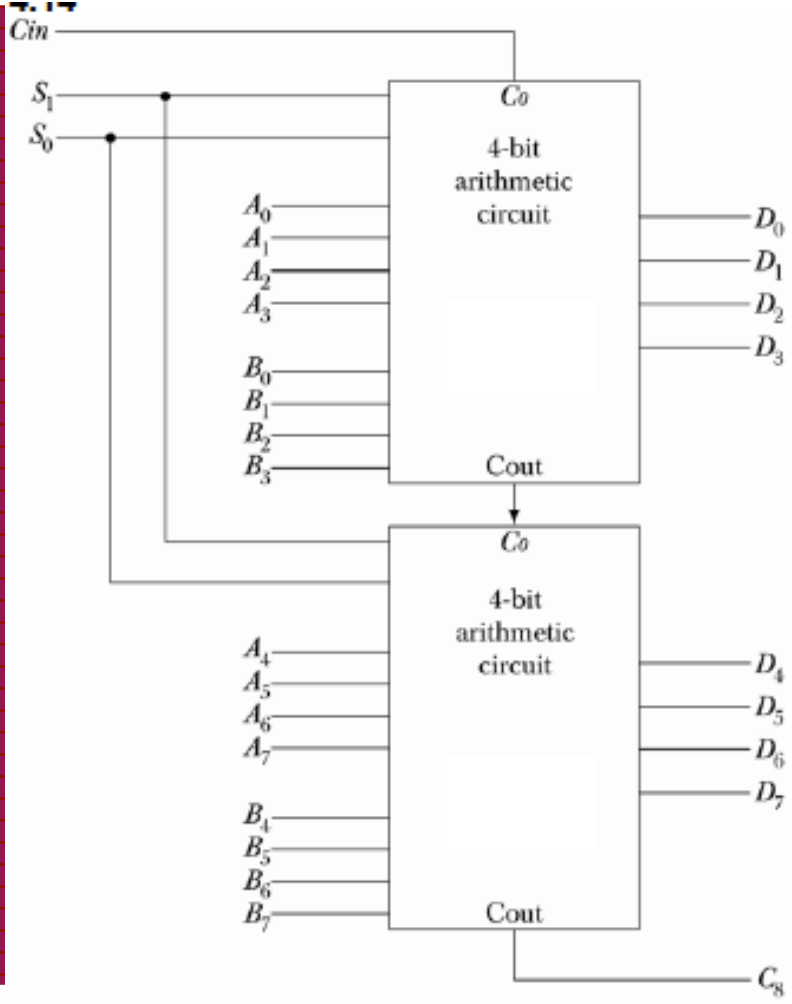
14. Assume that the following 4-bit arithmetic circuit is enclosed in a single IC package. Show the connections among two such ICs that form an 8-bit arithmetic circuit.



Solution:


C_{in}
S_1
S_0
A_0
B_0
A_1
B_1
A_2
B_2
A_3
B_3
0
1
4 × 1 MUX
FA
X_0 C_0 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}
S1
S0
A0
A1
A2
A3
B0
B1
B2
B3
4-bit Arithmetic Circuit IC
C_{in}
C_{out}

Solution:


Cin
S1
S0
Co
4-bit
arithmetic
circuit
A0
A1
A2
A3
B0
B1
B2
B3
D0
D1
D2
D3
Cout
Co
4-bit
arithmetic
circuit
A4
A5
A6
A7
B4
B5
B6
B7
D4
D5
D6
D7
Cout
C8

15. Design an arithmetic circuit with one selection variable S and two n-bit data inputs A and B. the circuit generates the following four arithmetic operations in conjunction with the input carry C_{in}. Draw a logic diagram for the first two stages.

S	$C_{in} = 0$	$C_{in} = 1$
0	D = A + B	D = A + 1
1	D = A – 1	D = A + B′ + 1

Solution: *Let's gather some clues about the kind of circuit we need to draw.*

S	$C_{in} = 0$	$C_{in} = 1$
0	D = A + B	D = A + 1
1	D = A – 1	D = A + B′ + 1

Essentially, we need to perform addition, so we have to use an ***adder****.*

There is also a carry, so we have to use a ***full-adder****.*

In each of the four operations, one of the operands is always A, and the second operand keeps changing. So, one of the inputs to the FA should be A and the second input should come from a ***multiplexer*** *(this will also take care of the selection variable).*

Solution: Step 1

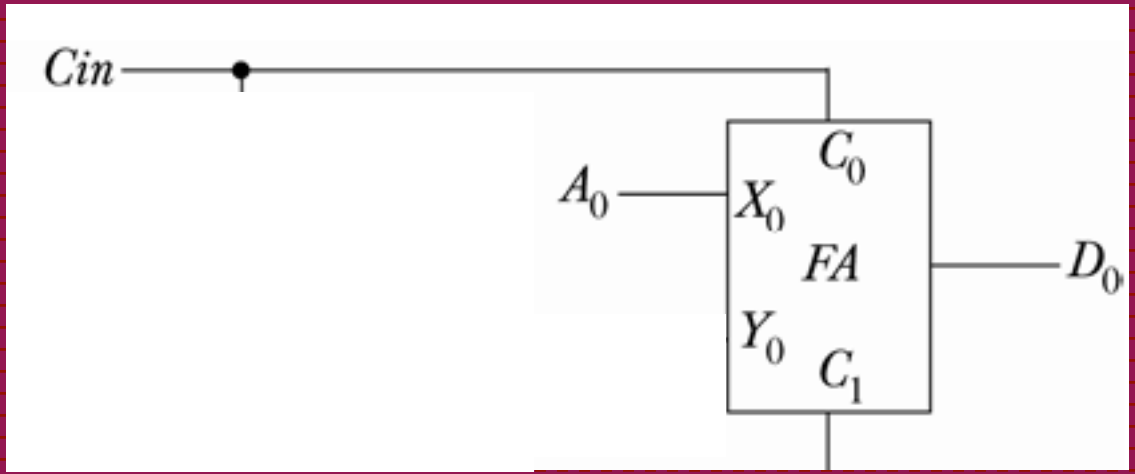
Let's convert the given table into a truth table for the new circuit.

S	$C_{in} = 0$	$C_{in} = 1$
0	D = A + B	D = A + 1
1	D = A – 1	D = A + B′ + 1

S	C_{in}	X	Y
0	0	A	B
0	1	A	0
1	0	A	1
1	1	A	B′

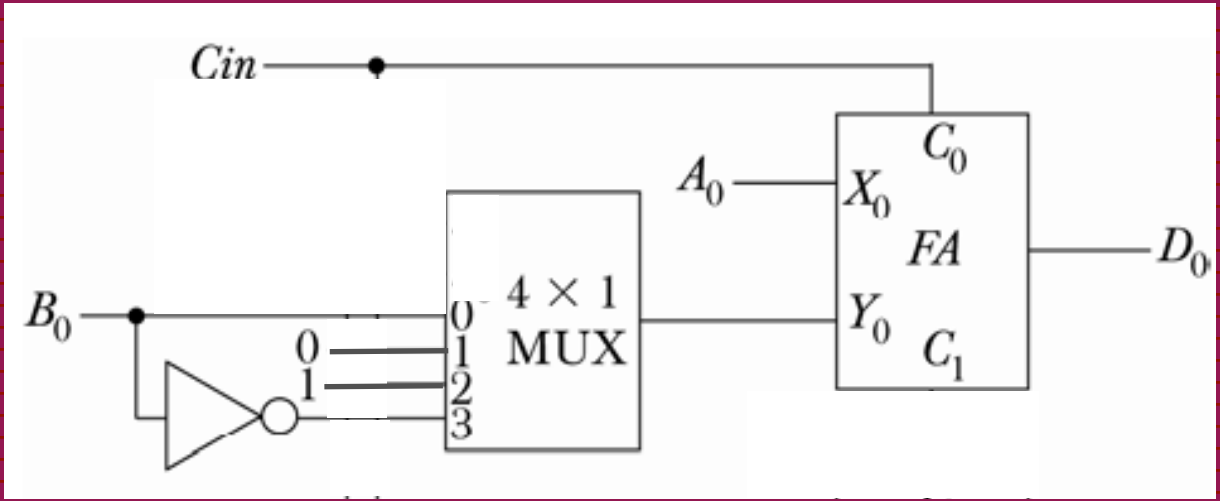
Solution: Step 2

S	C_{in}	X	Y
0	0	A	B
0	1	A	0
1	0	A	1
1	1	A	B′



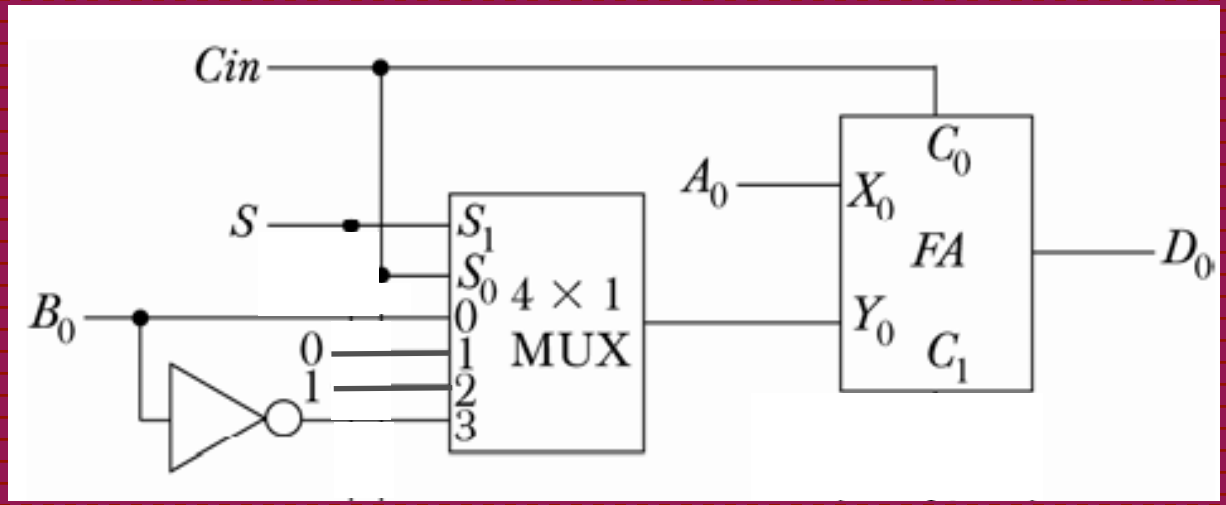
Solution: Step 3

S	C_{in}	X	Y
0	0	A	B
0	1	A	0
1	0	A	1
1	1	A	B′

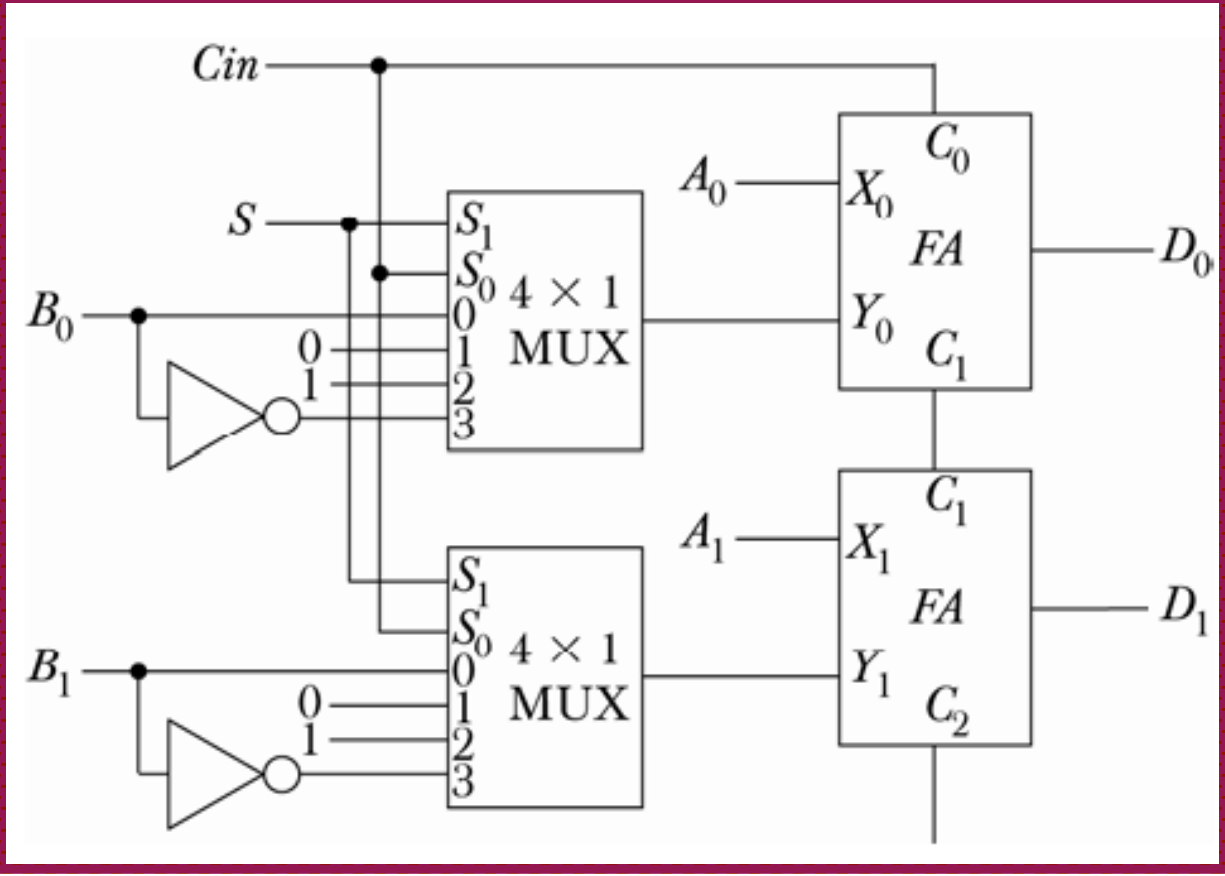


Solution: Step 4

S	C_{in}	X	Y
0	0	A	B
0	1	A	0
1	0	A	1
1	1	A	B′



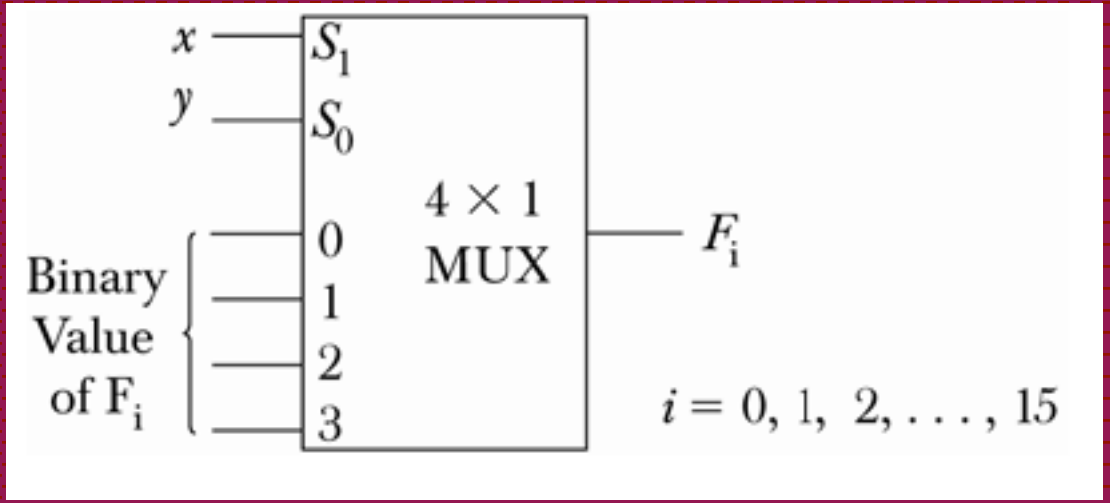
Solution: Step 5



16. Design a combinational circuit that selects and generates any of the 16 logic functions listed in the following table.

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Solution:



17. Design a digital circuit that performs the four logic operations (Exclusive–OR, Exclusive–NOR, NOR, and NAND). Use two selection variables and show the logic diagram for a typical stage.

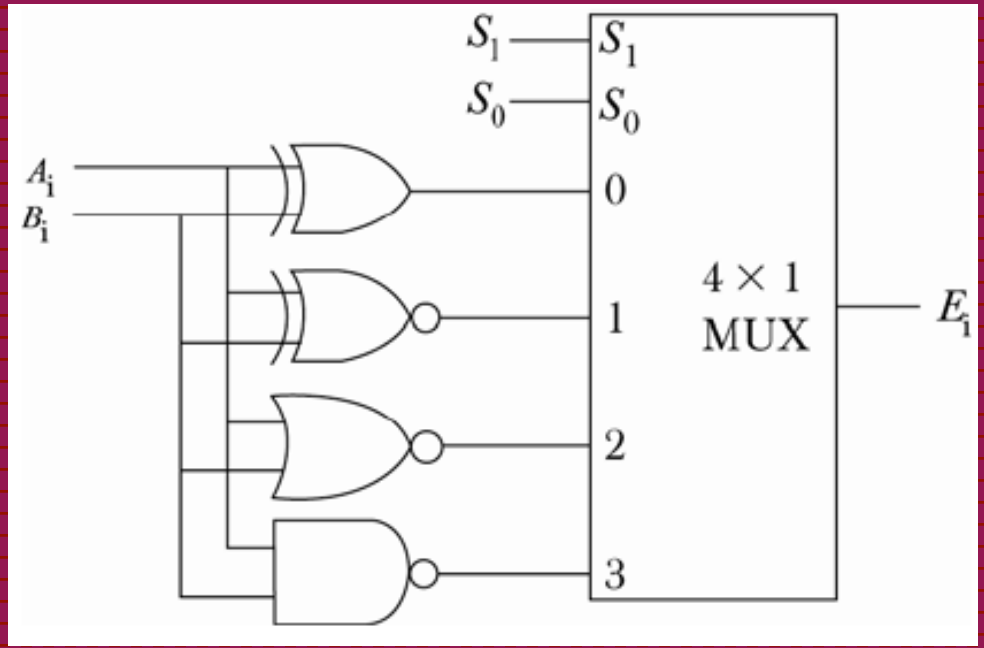
Solution: Step 1

Let's first generate a function table for the circuit.

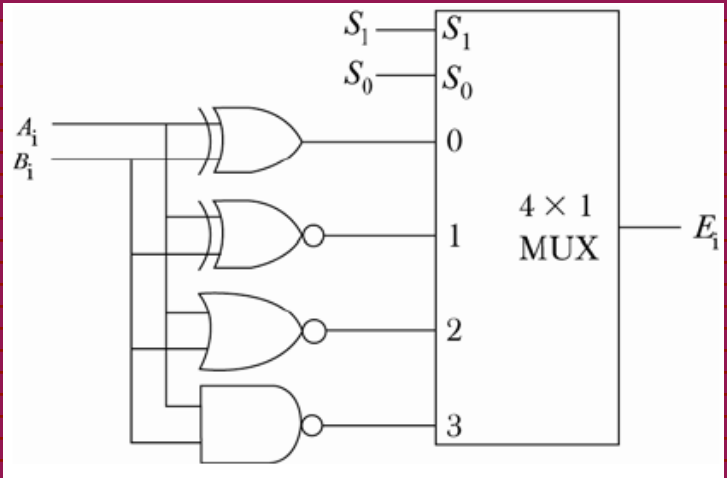
S_1	S_0	Function to be performed
0	0	**XOR**
0	1	**XNOR**
1	0	**NOR**
1	1	**NAND**

Solution: Step 2

Since we need to be able to generate four functions using two selection variables, we will have to use a 4x1 mux.



Solution



S_1	S_0	E_i
0	0	A_i XOR B_i
0	1	A_i XNOR B_i
1	0	A_i NOR B_i
1	1	A_i NAND B_i

18. Register A holds the 8-bit binary 11011001. Determine the logic microoperation and the value in the second register B that will cause the value in A to be changed to:

a. 01101101

b. 11111101

Solution:

a. XOR operation with B = 10110100, followed by register transfer.

	1 1 0 1 1 0 0 1	Value in A
⊕	1 0 1 1 0 1 0 0	Value in B
	0 1 1 0 1 1 0 1	Result in A after A ← **A ⊕ B**

b. OR operation with B = 11111101, followed by register transfer.

	1 1 0 1 1 0 0 1	Value in A
V	1 1 1 1 1 1 0 1	Value in B
	1 1 1 1 1 1 0 1	Result in A after A ← **A V B**

19. 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 and CR, increment BR)

c. AR ← AR - CR (Subtract CR from AR)

Solution:

a. ***AR ← AR + BR*** ***(Add BR to AR)***

AR = 1 1 1 1 0 0 1 0

BR = 1 1 1 1 1 1 1 1

AR+BR = 1 1 1 1 0 0 0 1

So, AR = 11110001, BR = 11111111, CR = 10111001, DR = 11101010

Solution:

b. CR ← CR ∧ DR, BR ← BR + 1 (AND DR and CR, increment BR)

CR = 1 0 1 1 1 0 0 1

DR = 1 1 1 0 1 0 1 0

CR∧DR = 1 0 1 0 1 0 0 0

BR = 1 1 1 1 1 1 1 1

\+ 1

BR + 1 = 0 0 0 0 0 0 0 0

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

Solution:

c. AR ← AR - CR (Subtract CR from AR)

AR = 1 1 1 1 0 0 0 1

CR = 1 0 1 0 1 0 0 0

AR - CR = 0 1 0 0 1 0 0 1

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

20. An 8-bit registers 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.

Solution:

R = 1 0 0 1 1 1 0 0

After Arithmetic Shift-Right 1 1 0 0 1 1 1 0

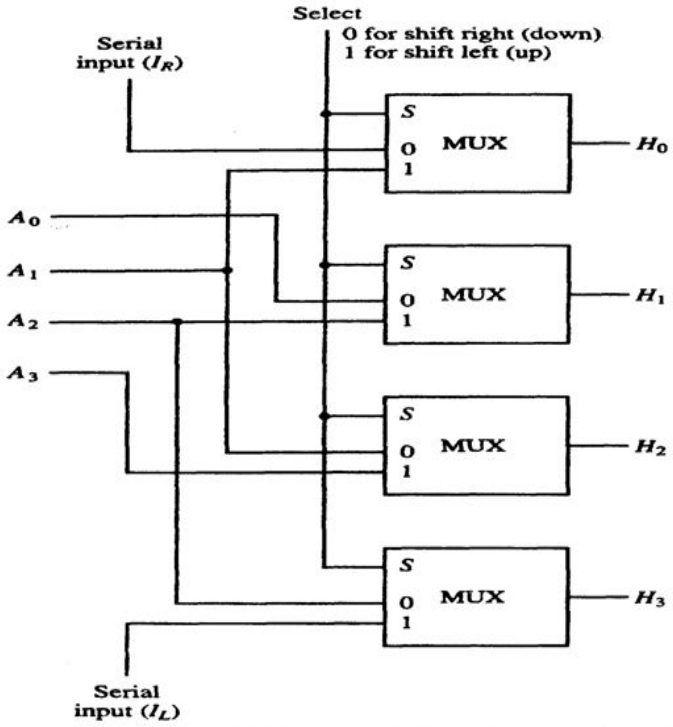
After Arithmetic Shift-Left 0 0 1 1 1 0 0 0 : overflow occurs because the number, which was originally negative, becomes positive after the shift.

21. Starting from the 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.

Solution:

	R = 1 1 0 1 1 1 0 1
After Logical Shift-Left	1 0 1 1 1 0 1 0
After Circular Shift-Right	0 1 0 1 1 1 0 1
After Logical Shift-Right	0 0 1 0 1 1 1 0
After Circular Shift-Left	0 1 0 1 1 1 0 0

22. What is the value of output H in the following figure if input A is 1001, S = 1, I_r = 1, and I_L = 0?



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

Solution: The given circuit is a 4-bit combinational circuit shifter.

S = 1 will cause it to perform a **left-shift**.

The given input is A = 1 0 0 1, i.e., $A_0 = 1$, $A_1 = 0$, $A_2 = 0$, and $A_3 = 1$.

Since the circuit will perform a left-shift, its output (according to the below mentioned function table) should be:

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

A_1 A_2 A_3 I_L

i.e.,

0 0 1 0

23. What is wrong with the following register transfer statements?

a. xT: AR ← $\overline{AR}$, AR ← 0

b. yT: R1 ← R2, R1 ← R3

c. zT: PC ← AR, PC ← PC + 1

Solution

a. xT: AR ← AR, AR ← 0

The two microoperations -Compliment and Clear - are supposed to be done at the same time on the same register, this is wrong coz these microoperations can't be implemented on the same register at the same time.

b. yT: R1 ← R2, R1 ← R3

Two different values cannot be transferred to the same register at the same time.

c. zT: PC ← AR, PC ← PC + 1

It is not possible to move a new value into a register (PC) and increment that value at the same time.