REGISTER TRANSFER AND MICROOPERATIONS

Slides prepared by:
Dr. Zubair Ahmad Shah
Department of Computer Science and Engineering
Islamic University of Science and Technology, Awantipora

REGISTER TRANSFER

- **Microoperation:** A microoperation is an elementary operation performed on the information stored in one or more registers.
  - e.g. load, clear, shift..

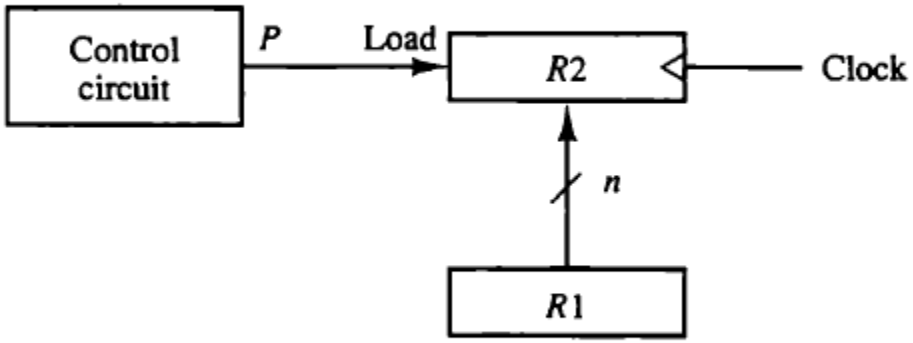
The internal hardware organization of a digital computer is best defined by specifying:

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.

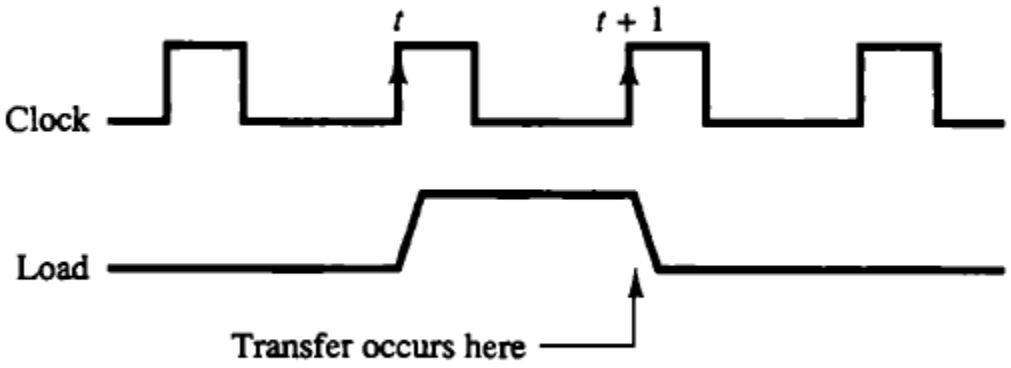
- **Register Transfer Language**: Symbolic notation used to describe microoperation transfers among registers.
  - e.g. **R2 <- R1** denotes transfer of the content of register R1 into register R2.
  - e.g. **if (P = 1) then (R2 <- R1)** denotes transfer of the content of register R1 into register R2 under a predetermined control condition (P is a control signal generated by Control Unit).

    Another way of representing the same: **P: R2 <- R1**

Transfer from $R1$ to $R2$ when $P = 1$.



(a) Block diagram



(b) Timing diagram

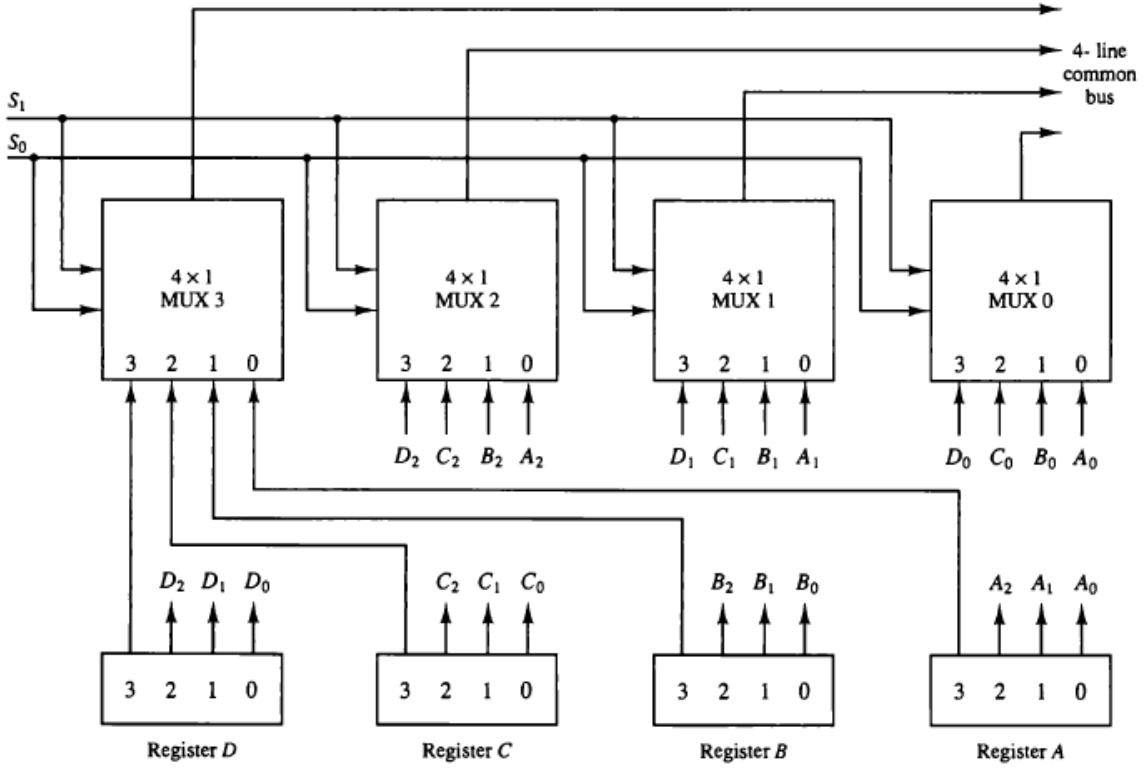
Basic Symbols for Register Transfers

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*

Bus Transfers:

- An efficient scheme for transferring information between registers in a multi-register configuration is a **common bus system**.
- Control signals will determine which register is selected by the bus during each particular register transfer.
- A common bus system can be constructed using:
  1) Multiplexers
  2) Three-State Buffers

Constructing Bus using Multiplexers



Bus system for four registers.

Function Table

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

- In general, for ***k*** registers of ***n*** bits each, ***n*** number of ***kx1*** size multiplexers will be needed to construct an ***n***-line common bus.

  => No. of multiplexers needed = No. of bits in each register

  Size of each multiplexer = Total no. of registers x 1

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.

QUESTION

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

a) How many multiplexers are there in the bus?

b) What size of multiplexers is needed?

c) How many selection lines are there in each multiplexer?

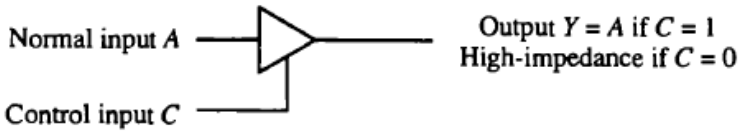
SOLUTION

a) 32

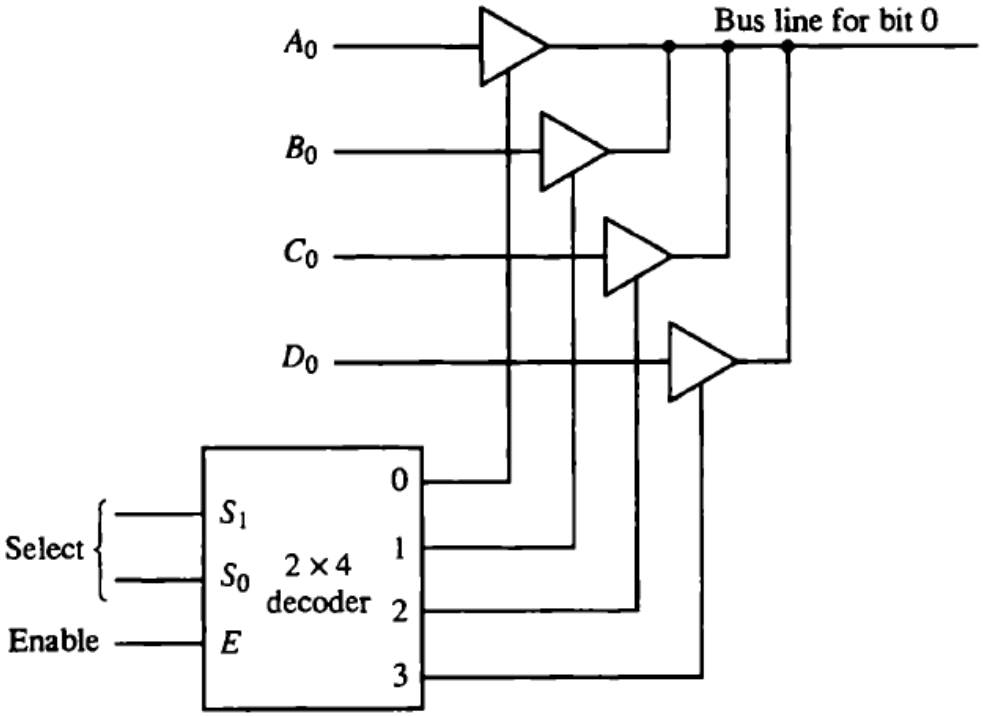
b) 16 x 1

c) 4

Constructing Bus using Three-State Buffers



Graphic symbol for three-state buffer.



Bus line with three state-buffers.

- **Memory Transfer Representation:**

DR <- M[AR] - Memory Read Operation

M[AR] <- R1 - Memory Write Operation

MICROOPERATIONS

- **Microoperation:** A microoperation is an elementary operation performed on the information stored in one or more registers.

- **Classification of Microoperations:**
  - Register transfer microoperations
  - Arithmetic microoperations
  - Logic microoperations
  - Shift microoperations

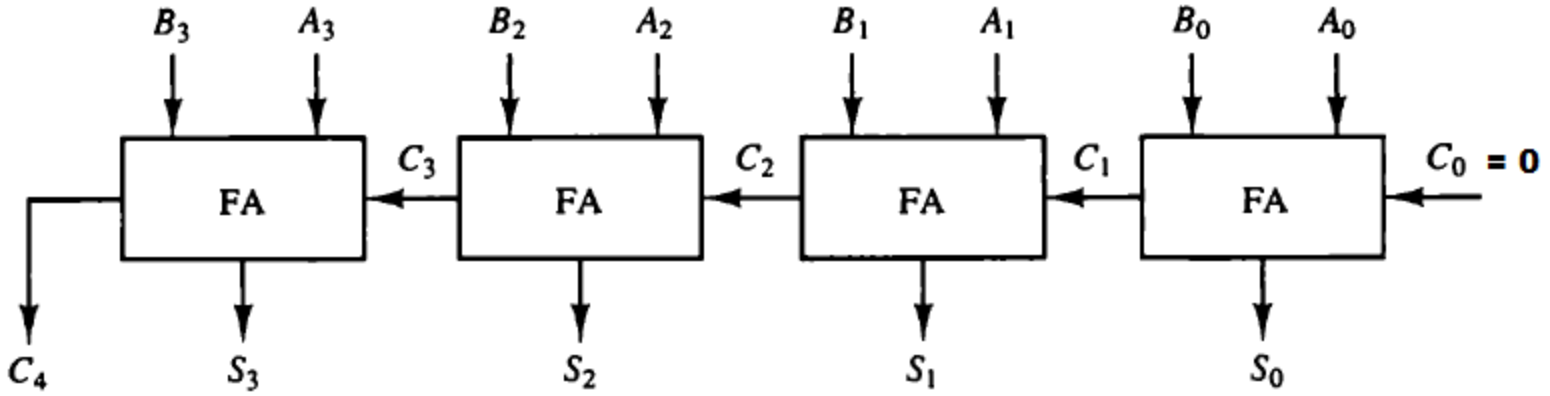
Arithmetic Microoperations

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

- To implement arithmetic microoperations with hardware, we need registers to hold the data and digital components to perform arithmetic operations.

Binary Adder: A digital circuit that performs arithmetic sum of two binary numbers of any length is called a binary adder.

```
   0111
+  0011
  ------
  0 1010
```



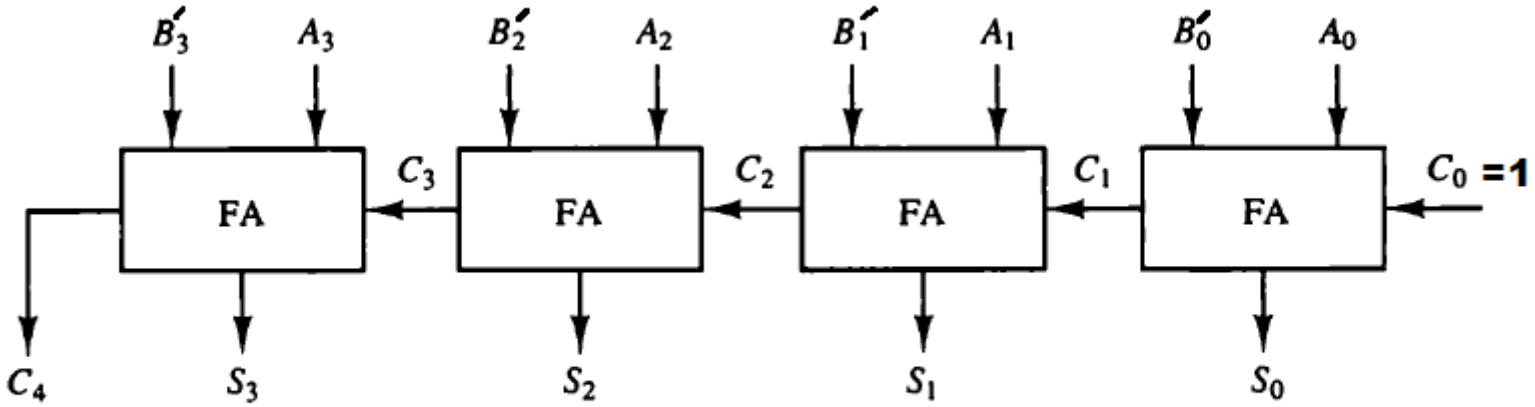
4-bit binary adder.

Binary Subtractor:

A – B = A + 2's complement of B

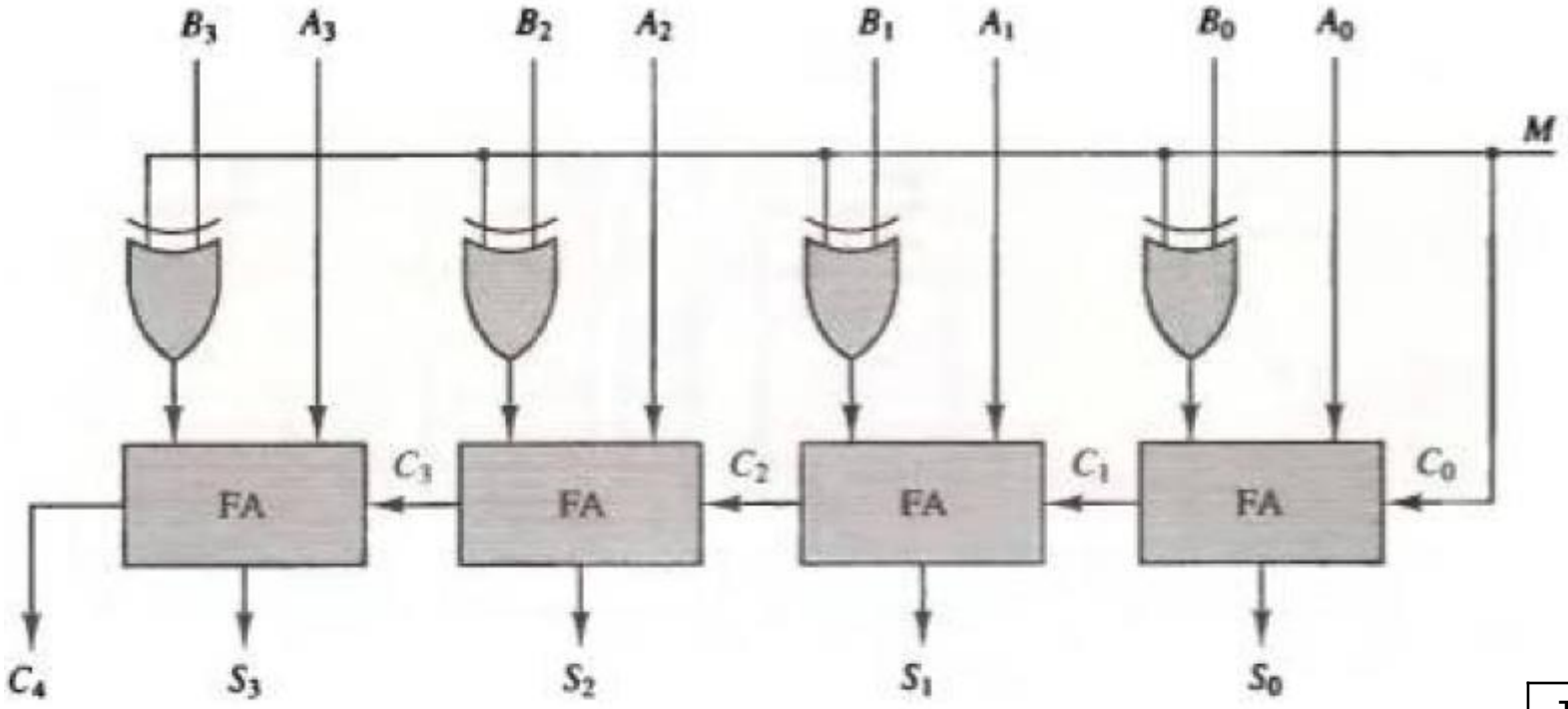
= A + 1's complement of B + 1

0111 – 0011 = 0111 + 1100 + 1 = 1 0100



4-bit binary subtractor.

Binary Adder-Subtractor:



4-bit adder-subtractor.

Truth Table of XOR Gate		
X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

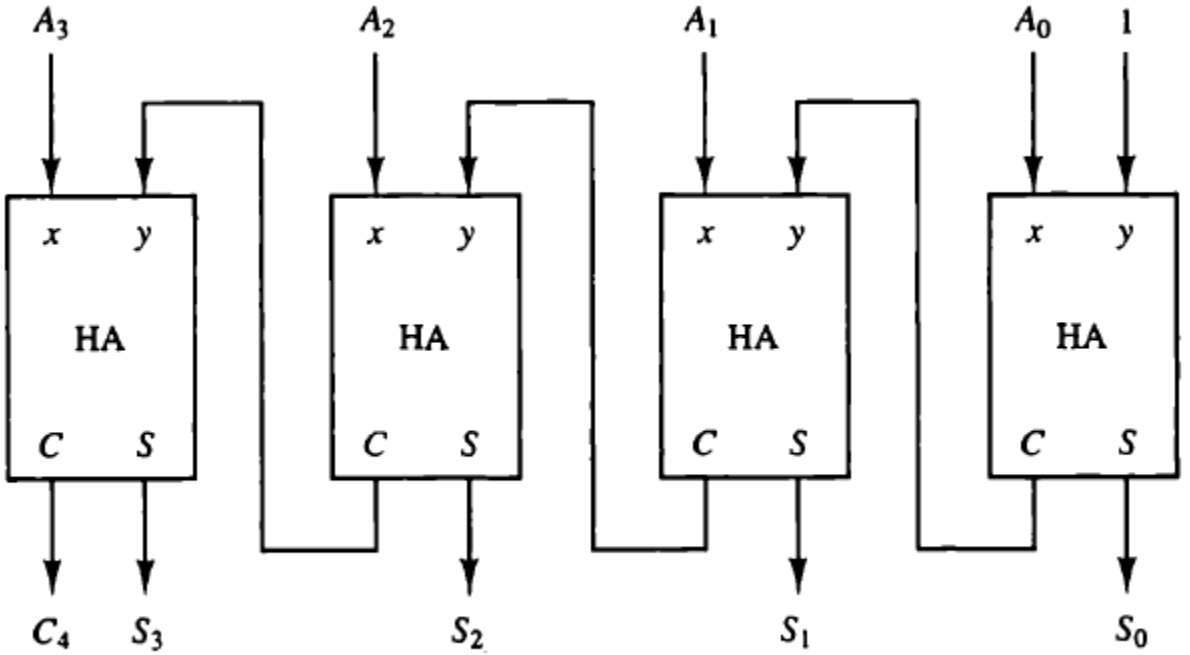
Binary Incrementer:

0111

\+ 0001

———

0 1000

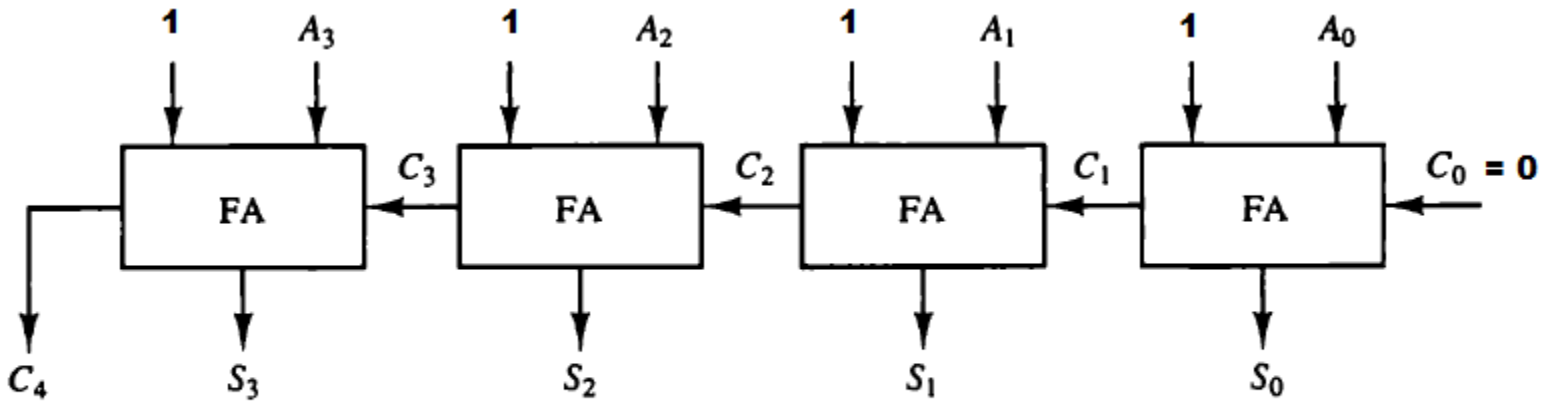


4-bit binary incrementer.

Binary Decrementer:

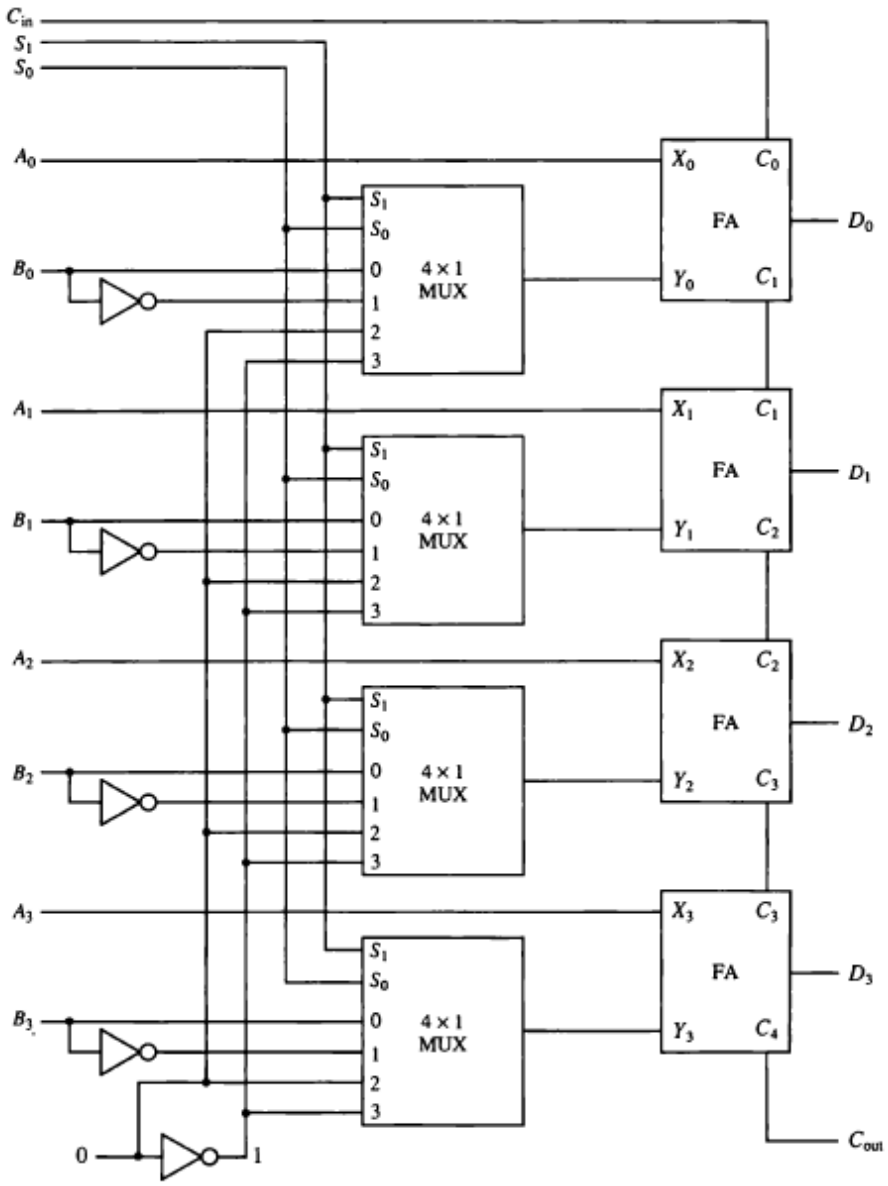
0111 – 0001 = 0111 + 2's complement of 0001

= 0111 + 1111

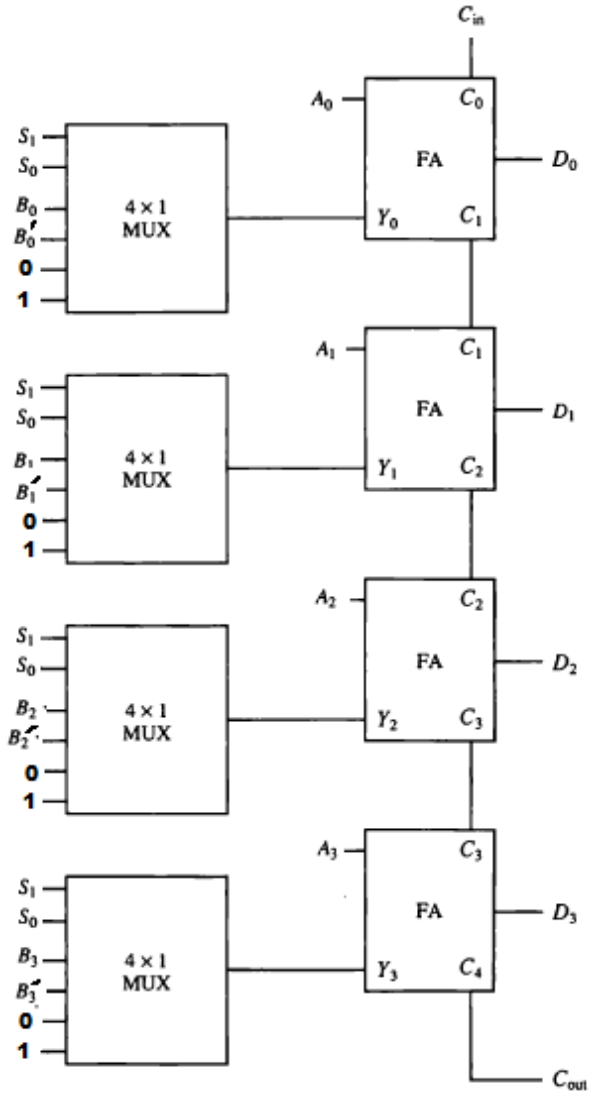


4-bit binary decrementer

Arithmetic Circuit:



4-bit arithmetic circuit.



Arithmetic Circuit Function Table

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

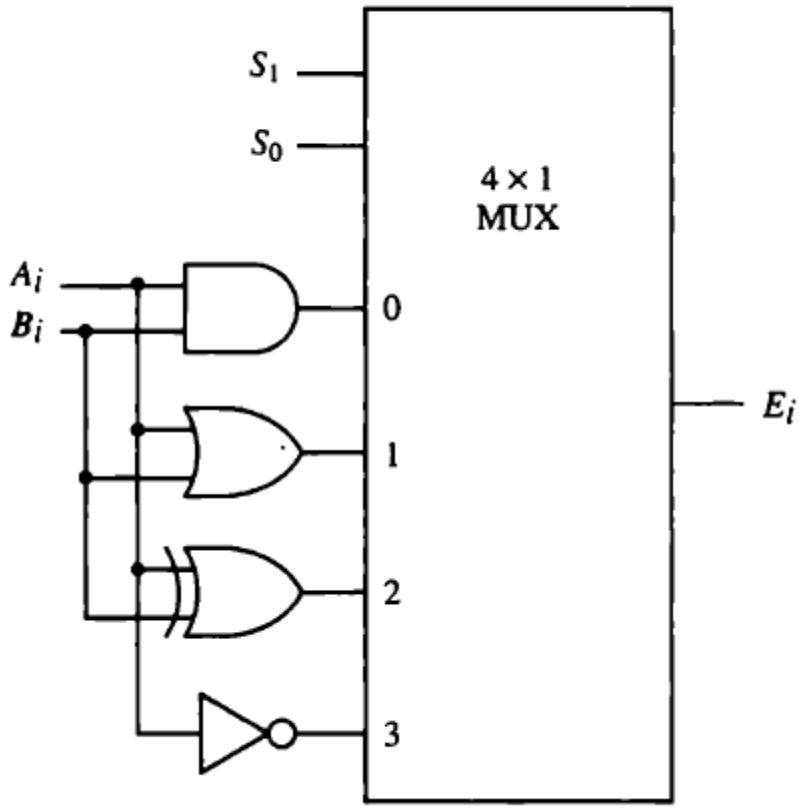
Truth Tables for 16 Functions of Two Variables

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

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

One stage of logic circuit.



(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

Some Applications of Logic Microoperations:

Say, Register A = 0011 and Register B = 0101

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

0011	A (before)
0101	B (logic operand)
0111	A (after)

=> Logic OR

2. ***Selective-Complement:*** This operation complements bits in A where there are corresponding 1s in B.

0011	A (before)
0101	B (logic operand)
0110	A (after)

=> Logic XOR

3. ***Selective-Clear***: This operation clears to 0 the bits in A where there are corresponding 1s in register B.

0011	A (before)
0101	B (logic operand)
0010	A (after)

=> Logic A.B’

4. ***Mask***: This operation clears to 0 the bits in A where there are corresponding 0s in register B.

0011	A (before)
0101	B (logic operand)
0001	A (after)

=> Logic AND

5. ***Insert***: 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.

0110 1010	A (before)
0000 1111	B (mask)
0000 1010	A (after masking)
=> Logic AND	

0000 1010	A (before)
1001 0000	B (insert)
1001 1010	A (after insertion)
=> Logic OR	

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

0011	A
0011	B
0000	A

=> Logic XOR

Shift Microoperations

Types of Shifts:

1. **Logical Shift**

   Logical Shift-left: 1010
   010_ 0100

   Logical Shift-right: 1010
   _101 0101

2. **Circular Shift**

   Circular Shift-left: 1010
   010_ 0101

   Circular Shift-right: 1010
   _101 0101

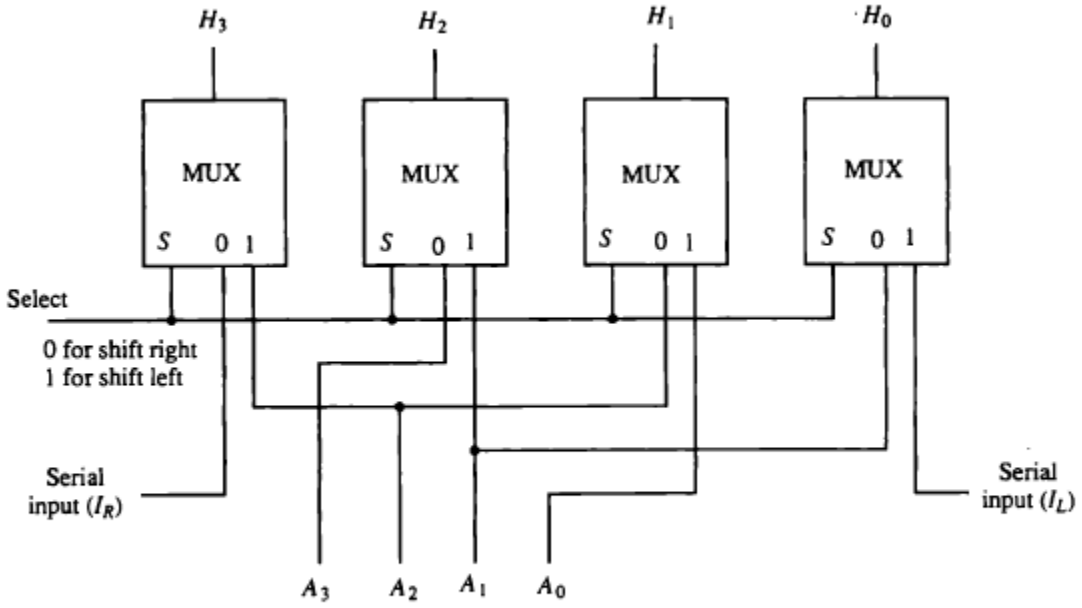
3. **Arithmetic Shift**

   Arithmetic Shift-left: 1010
   1 010 1 10_ 1100

   Arithmetic Shift-right: 1010
   1 010 1 _01 1001

Shift Microoperations

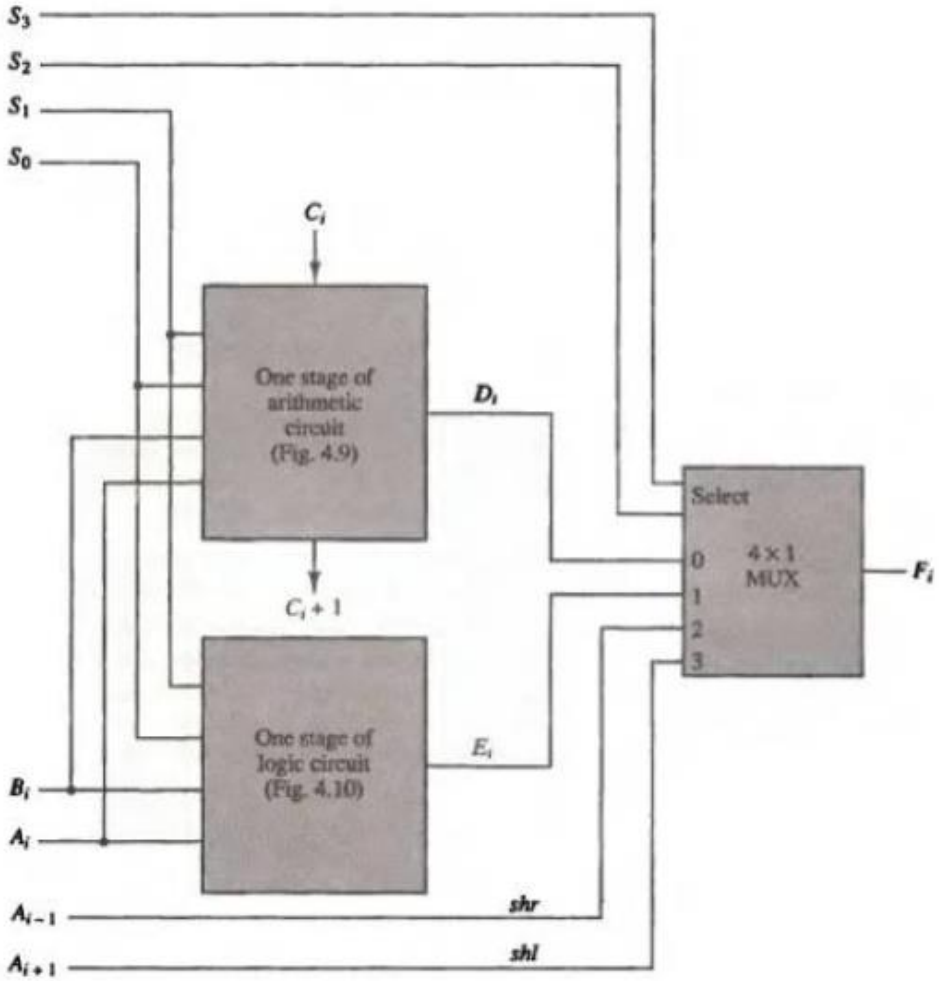
Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R



4-bit combinational circuit shifter.

Arithmetic Logic Shift Unit

One stage of 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	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

THANK YOU