Computer Architecture

Dr. Esti Stein
(Partly taken from Dr. Alon Schclar slides)

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Microoperations Types

Data transfer only

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

Changes data during transfer

Arithmetic Microoperations

Basic: addition, subtraction, increment, decrement & shift

Multiplication and division are not basic

Arithmetic Microoperations

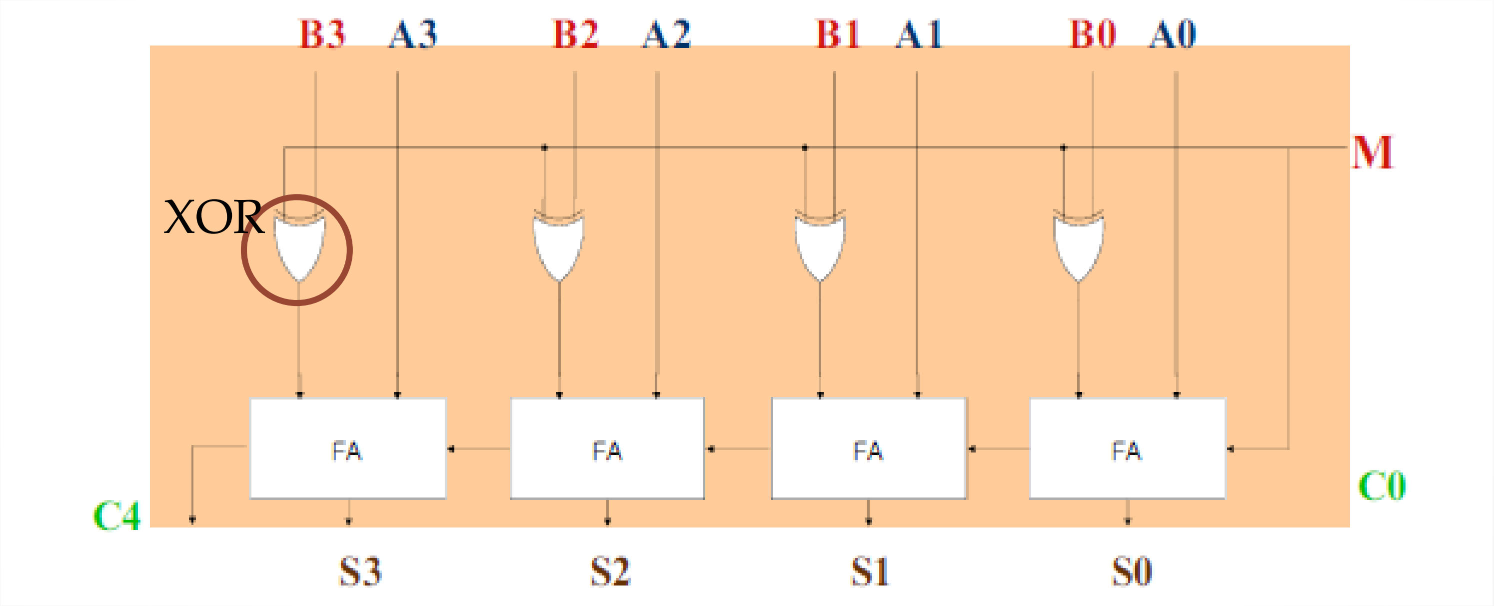
There are two identical operations

TABLE 4-3 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

Binary Adder-Subtractor

- M =0 : Adder $B \oplus M + C_0 = B \oplus 0 + 0 = B \Rightarrow S = A + B$
- M =1 : Subtractor $B \oplus M + C_0 \longrightarrow B \oplus 1 + 1 = B' + 1 = -B \Rightarrow S = A - B$



Binary Incrementer

Sequential circuit using a binary counter

Truth Table

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (toggles)

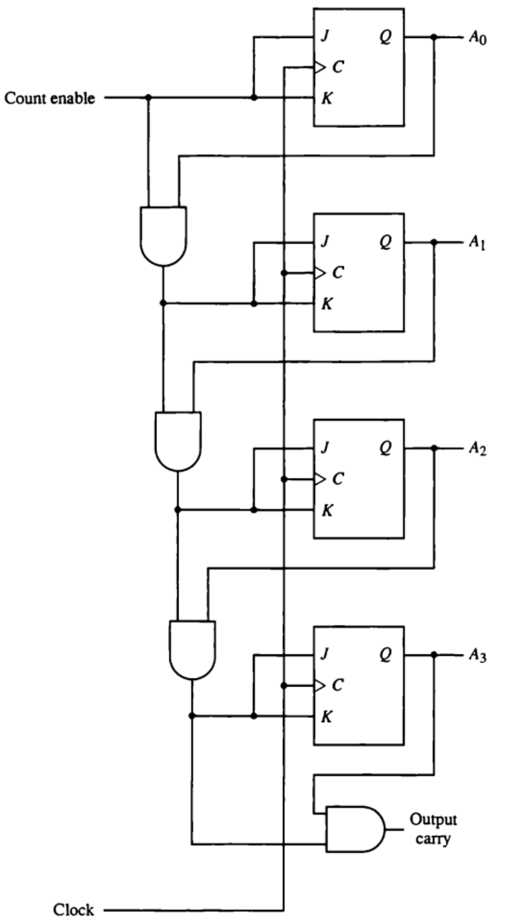
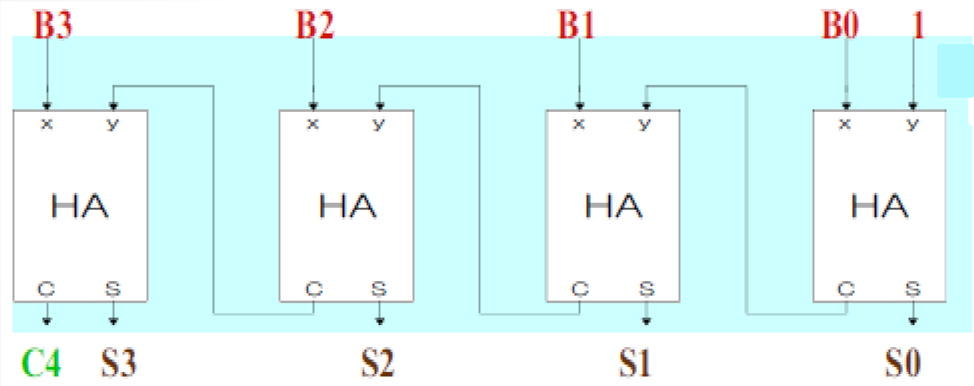


Figure 2-10 4-bit synchronous binary counter.

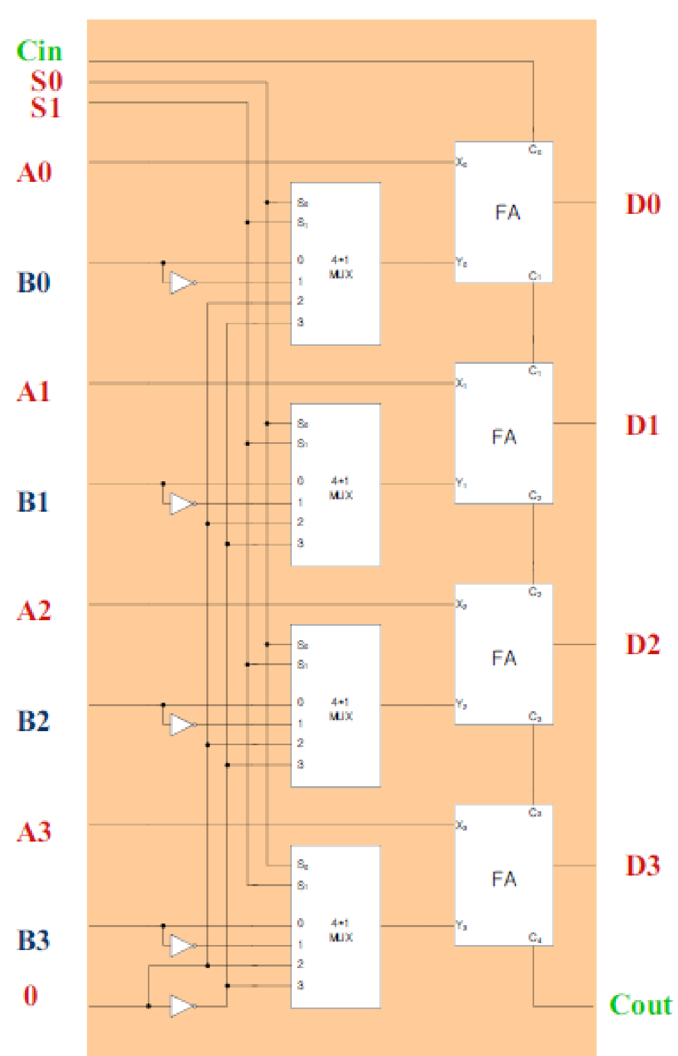
Combinational circuit using half adders



Arithmetic Circuit

TABLE 4-4 Arithmetic Circuit Function Table

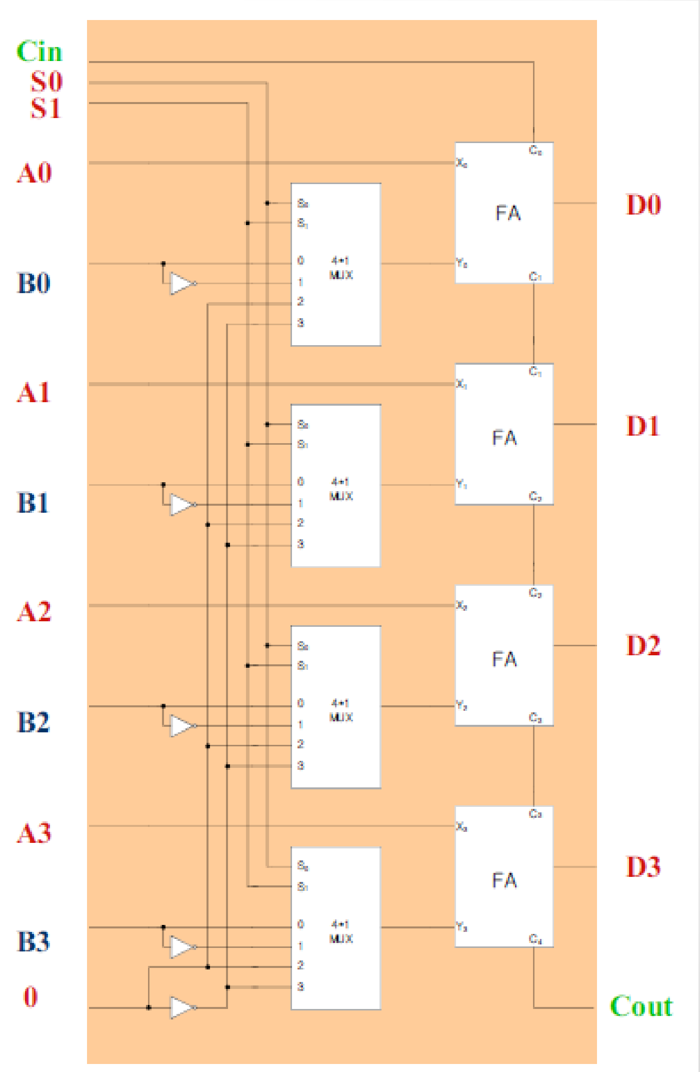
Select S_1	Select S_0	Select C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A



Arithmetic Circuit

TABLE 4-4 Arithmetic Circuit Function Table

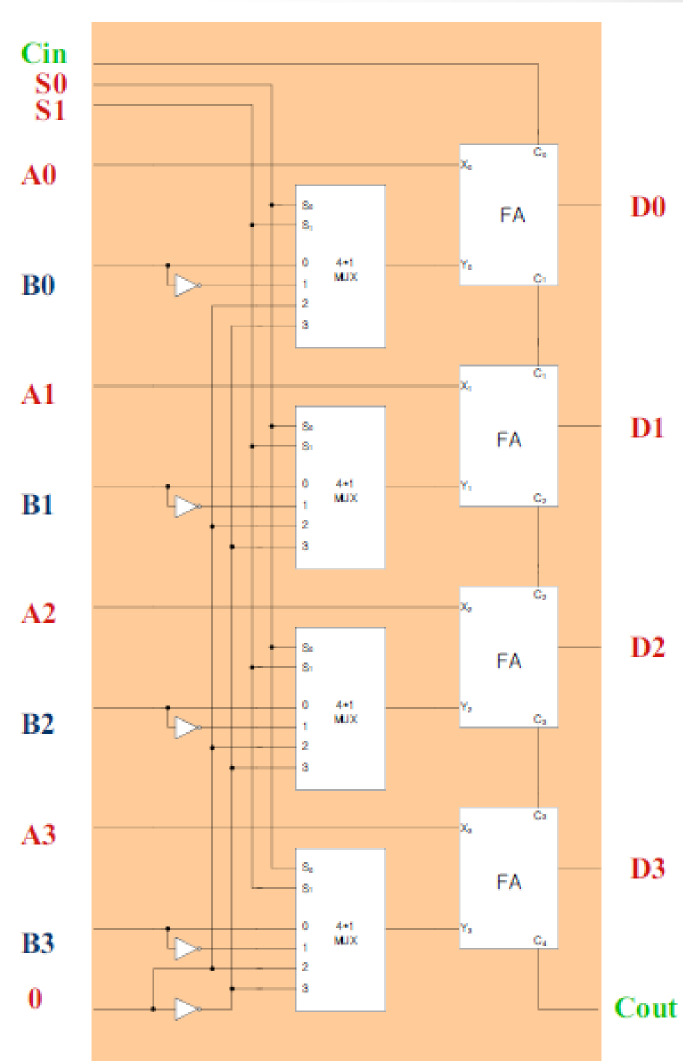
Select S_1	Select S_0	C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D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



Arithmetic Circuit

TABLE 4-4 Arithmetic Circuit Function Table

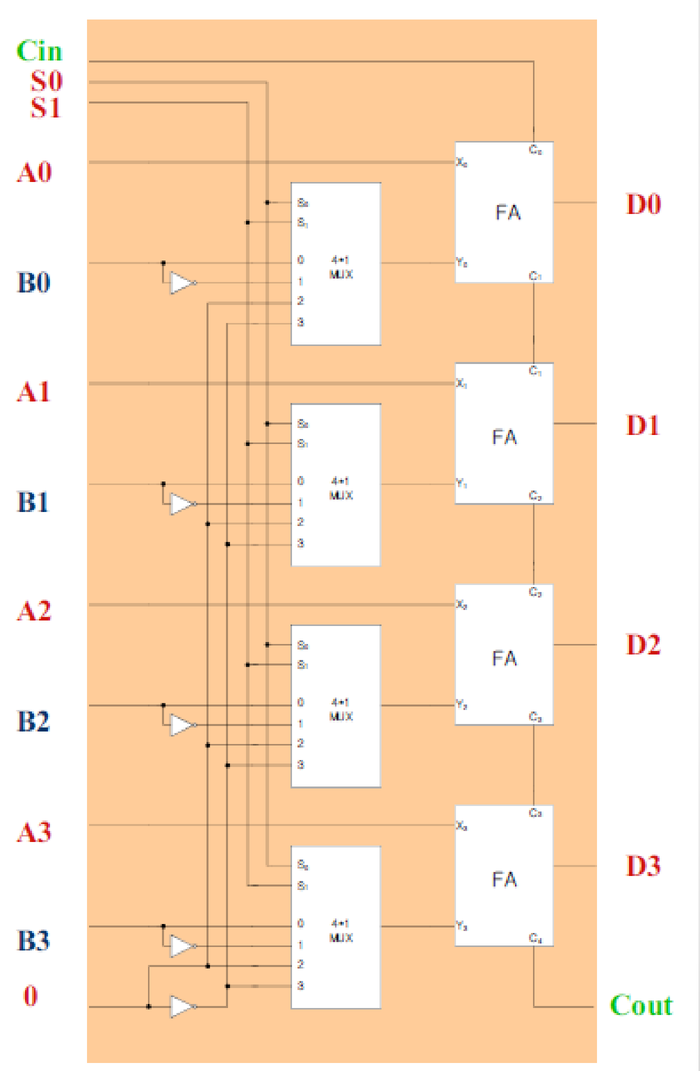
Select S_1	Select S_0	Select C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A



Arithmetic Circuit

TABLE 4-4 Arithmetic Circuit Function Table

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

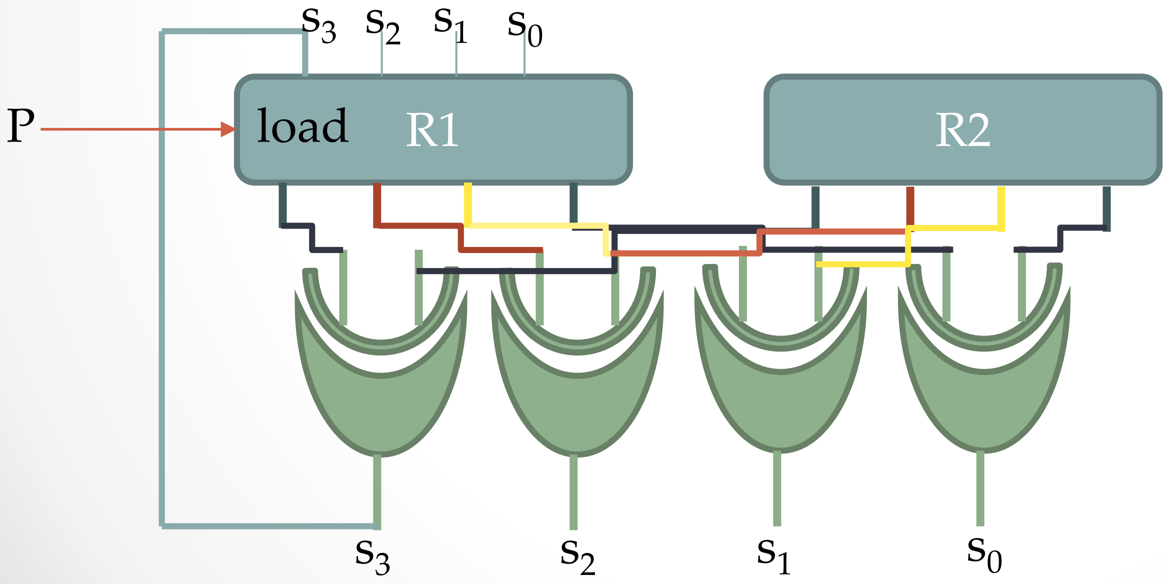


Logic Microoperations

Logic microoperations consider ***each bit of the register separately*** and treat them as binary variables

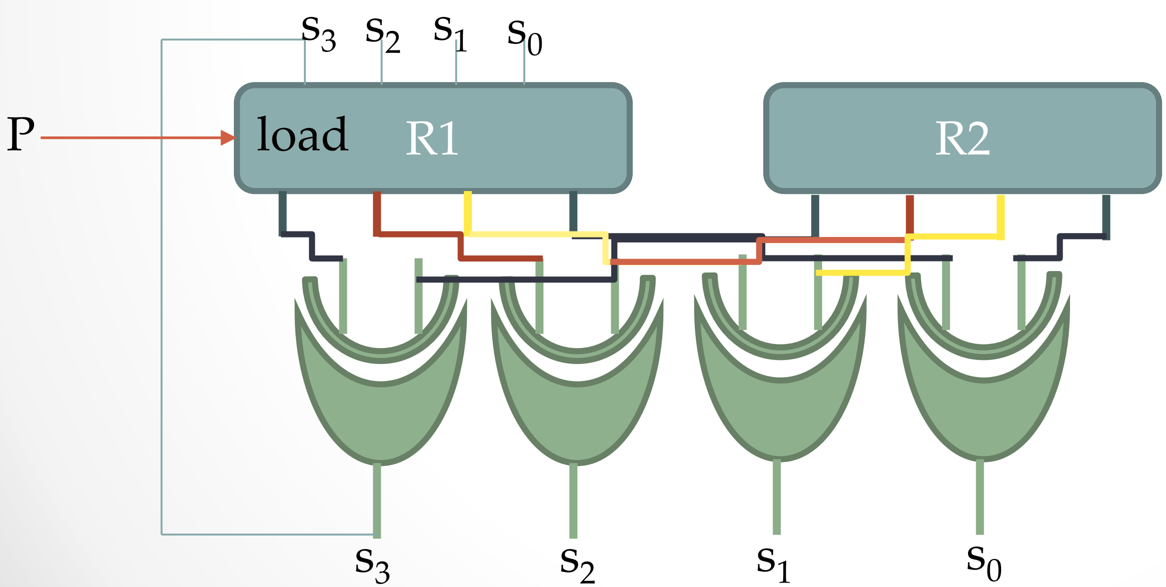
Example : XOR

if (P = = 1) then R1 $\leftarrow$ R1 $\oplus$ R2   or in RTL   P : R1 $\leftarrow$ R1 $\oplus$ R2



Logic Microoperations

R1 :	1 0 0 1
R2 :	1 1 0 0
R1 :	0 1 0 1
	s_3 s_2 s_1 s_0



Logic Microoperations

Special symbols in RTL :

Left side - condition

Or denoted by **+**
And denoted by **·**
complement denoted by **′**

:

Right side - operation

Or denoted by $\vee$
And denoted by $\wedge$
complement denoted by $\overline{}$
Arith. Plus denoted by **+**
Arith. Mult. denoted by *****

Example:

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

or   addition   or

Logic Microoperations

There are 16 different function that can be performed with two binary variables:

TABLE 4-5 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

$F \leftarrow 0$

Logic Microoperations

There are 16 different function that can be performed with two binary variables:

TABLE 4-5 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

$F \leftarrow xy$

Logic Microoperations

There are 16 different function that can be performed with two binary variables:

TABLE 4-5 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

$F \leftarrow xy'$

Logic Microoperations

There are 16 different function that can be performed with two binary variables:

TABLE 4-5 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

$F \leftarrow ?$

Logic Microoperations

There are 16 different function that can be performed with two binary variables:

TABLE 4-5 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

$F \leftarrow x \oplus y$

Logic Microoperations

TABLE 4-5 Truth Tables for 16

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

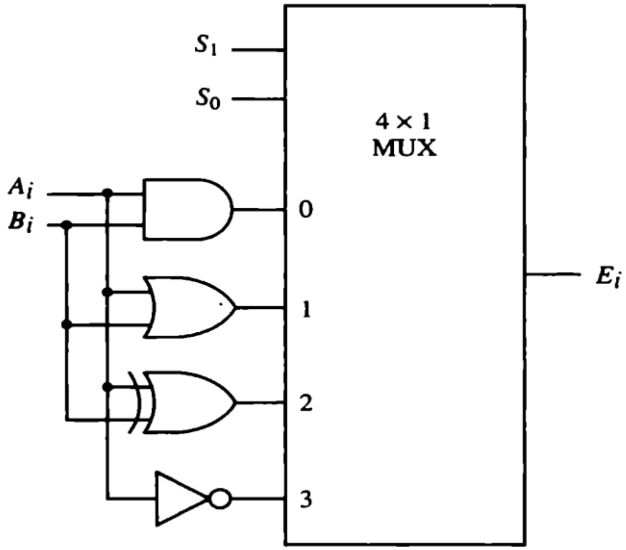
TABLE 4-6 Sixteen Logic Microoperations

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

LM – HW implementaion

- Logic gates are inserted for each bit or pair of bits in the registers.
- Most computers use only 4 operations: AND, OR, XOR and complement.
- All 12 others can be derived.

Figure 4-10 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

LM - applications

- Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register
- Used to **<u>change bit values</u>**, delete a group of bits, or insert new bit values

<u>Selective set:</u>

$A \leftarrow A \vee B$

Sets to **1** the bits in register A where there are corresponding 1's in register B.
It does not effect bit positions that have 0's in B.

$$
\begin{array}{ll}
A: & \boxed{1\,0}\,1\,0 \\
B: & \boxed{1\,1}\,0\,0 \quad \vee \\
\hline
A: & \boxed{1\,1}\,1\,0
\end{array}
$$

LM - applications

- Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register
- Used to **<u>change bit values</u>**, delete a group of bits, or insert new bit values

<u>Selective complement:</u>

$A \leftarrow A \oplus B$

Complements to **1** the bits in register A where there are corresponding 1's in register B.
It does not effect bit positions that have 0's in B.

A : 1 0 1 0
B : 1 1 0 0 $\oplus$

A : 0 1 1 0

LM - applications

- Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register
- Used to **<u>change bit values</u>**, delete a group of bits, or insert new bit values

<u>Selective clear:</u>

$A \leftarrow A \wedge \bar{B}$

Clears to **0** the bits in register A where there are corresponding 1's in register B.
It does not effect bit positions that have 0's in B.

A :	1	0	1	0	
B :	1	1	0	0	$\wedge$ not
A :	0	0	1	0	

LM - applications

- Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register
- Used to **<u>change bit values</u>**, delete a group of bits, or insert new bit values

<u>Selective mask:</u>

$A \leftarrow A \wedge B$

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.

A : 1 0 $\boxed{1\ 0}$
B : 1 1 $\boxed{0\ 0}$ ^
A : 1 0 $\boxed{0\ 0}$

LM - applications

Insert:

- The insert operation inserts a new value into a group of bits.
- This is done by :
  1. Masking the bits
  2. ORing them with the required value

We would like to change the two rightmost bits of A to 0 1

A :	1	0	1	0	
B :	1	1	0	0	$\wedge$
A :	1	0	0	0	

Clear the bits you want to change

A :	1	0	0	0	
B :	0	0	0	1	$\vee$
A :	1	0	0	1	

Set the bits to the desired pattern

LM - applications

- Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register
- Used to **<u>change bit values</u>**, delete a group of bits, or insert new bit values

<u>Clear operaion:</u>

$A \leftarrow A \oplus B$

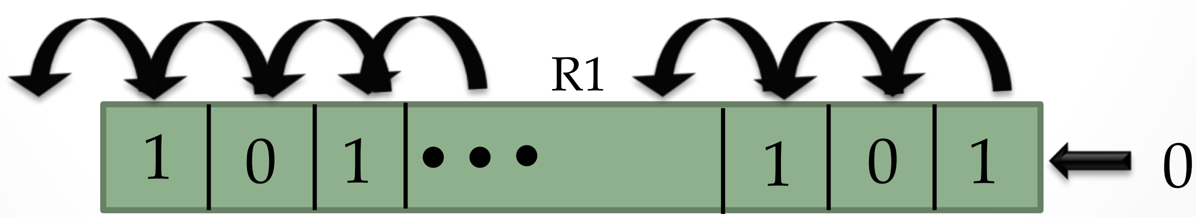
The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal.

$$\begin{array}{ll} A: & \boxed{1\,0}\,1\,0 \\ B: & \boxed{1\,0}\,0\,0 \quad \oplus \\ \hline A: & \boxed{0\,0}\,1\,0 \end{array}$$

Shift Microoperations

- Shift microoperations can be used for serial transfer of data.
- Three types of shift microoperation : ***Logical, Circular,*** and ***Arithmetic***

A ***logical shift*** transfers 0 through the serial input.
The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.
Shl – for shift left (insert 0 from rightmost side)



$$R1 \leftarrow shl\ R1$$

Shift Microoperations

- Shift microoperations can be used for serial transfer of data.
- Three types of shift microoperation : ***Logical, Circular,*** and ***Arithmetic***

A ***logical shift*** transfers 0 through the serial input.
The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.
Shl – for shift left (insert 0 from rightmost side)

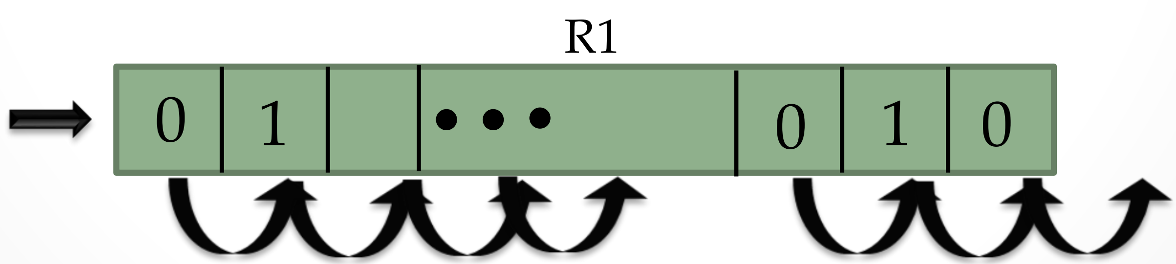
0	1	• • •	0	1	0

$$R1 \leftarrow \text{shl } R1$$

Shift Microoperations

- Shift microoperations can be used for serial transfer of data.
- Three types of shift microoperation : ***Logical, Circular,*** and ***Arithmetic***

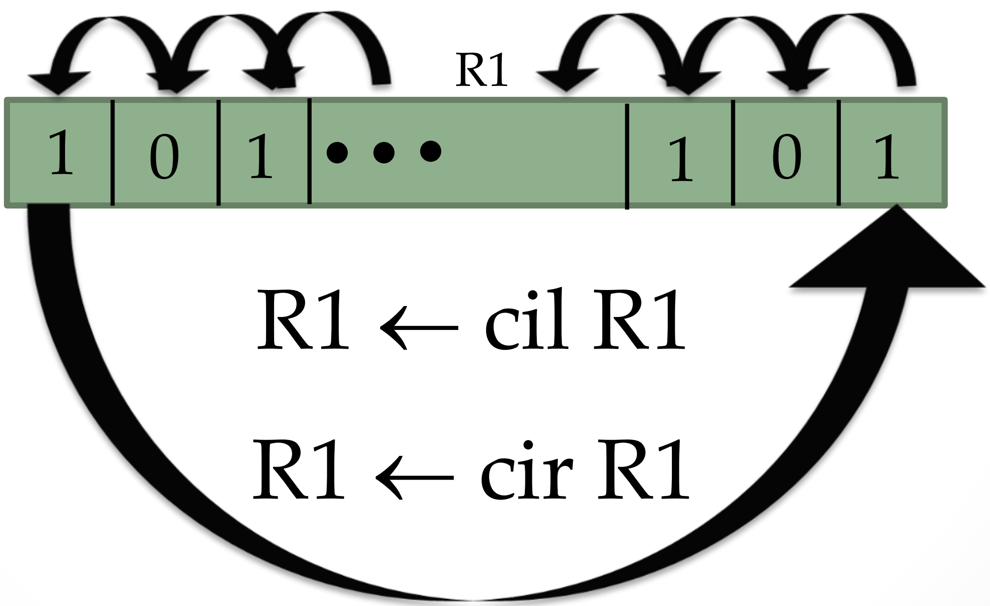
A ***logical shift*** transfers 0 through the serial input.
The bit transferred to the end position through the serial input is assumed to be **0** during a logical shift.
Shr – for shift right (insert 0 from leftmost side)



$$R1 \leftarrow \text{shr } R1$$

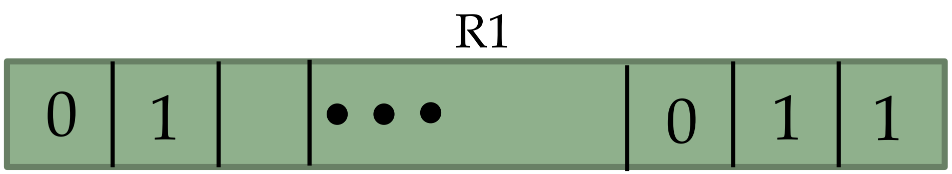
Circular shift

A ***circular shift*** circulates bits of the register around the two ends without loss of any bit.



Circular shift

A ***circular shift*** circulates bits of the register around the two ends without loss of any bit.



$$R1 \leftarrow \text{cil } R1$$

Arithmetic shift

- An ***arithmetic shift*** shifts a signed binary number to the left or right.
- An arithmetic shift-left (ashl) multiplies a signed binary number by 2.
- An arithmetic shift-right (ashr) divides a signed binary number by 2.
- Arithmetic shifts must leave the sign bit unchanged.

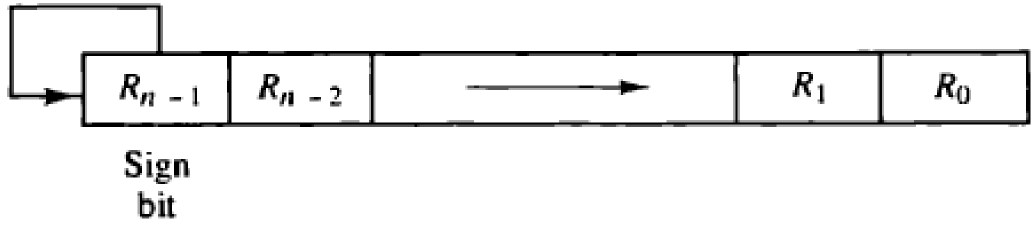
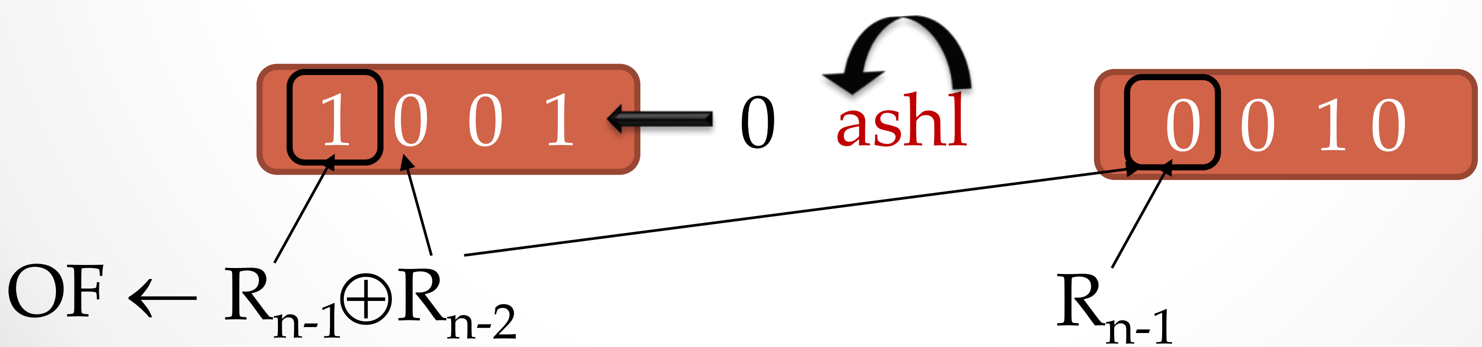


Figure 4-11 Arithmetic shift right.

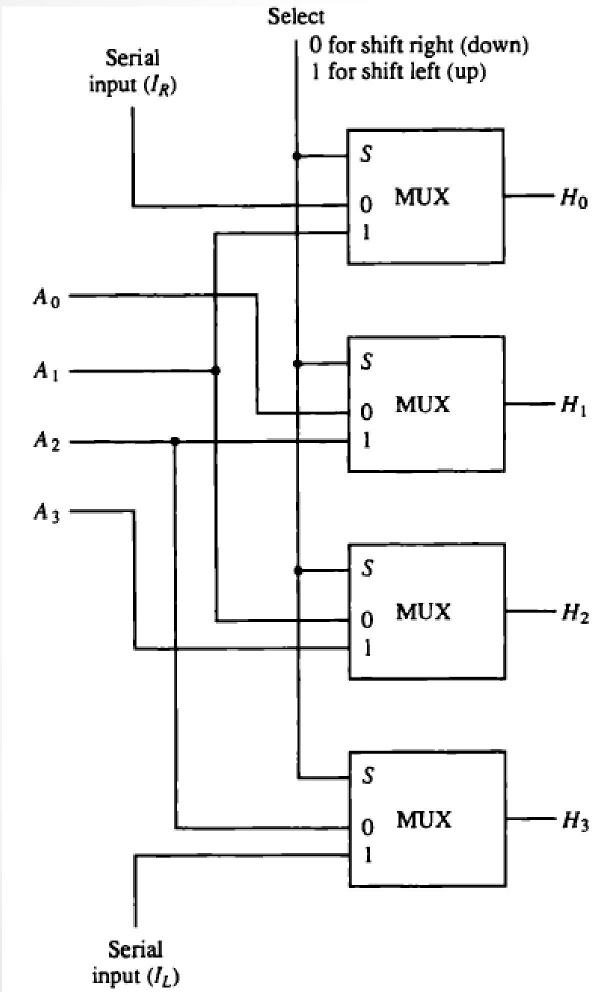
Arithmetic shift-left

- The arithmetic shift-left insert 0 into R_0, and shifts all other bits to the left.
- R_{n-2} becomes R_{n-1} and R_{n-1} lost and becomes the sign bit.
- A sign reversal occurs (overflow) if the bit in R_{n-1} changes its value after shift.

Example: a 4 bit signed R : 1 0 0 1 (= -7)



Shift – HW implementaion

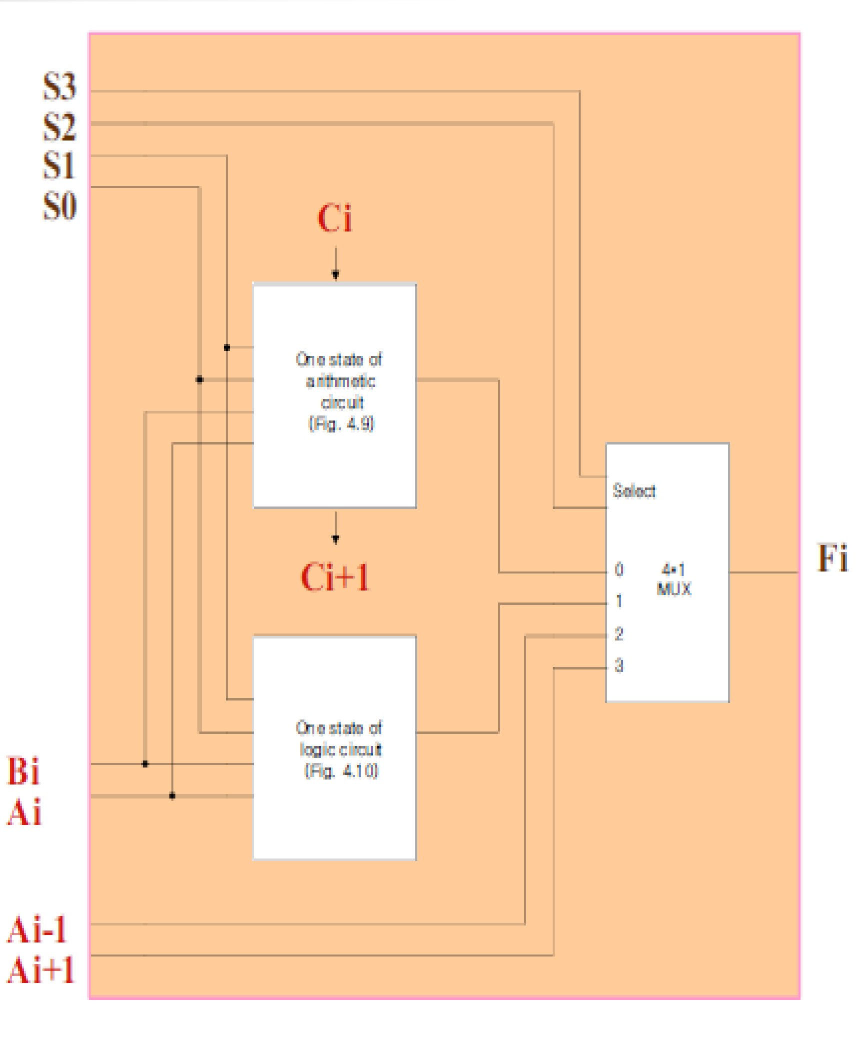


Function table

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

- When S=0, the input data is shifted right (down in the diagram).
- When S=1, the input data is shifted left (up in the diagram).
- A shifter with n data inputs, requires n multiplexers.

ALU - Arithmetical Logical Unit



- The contents of specific source registers are placed in the input of the ALU.
- The ALU performs the operation, and the result is then transferred to a destination register.
- The ALU is a combinational circuit, therefore the data transferred from source to destination through the ALU is performed during one clock pulse period.

ALU - Arithmetical Logical Unit

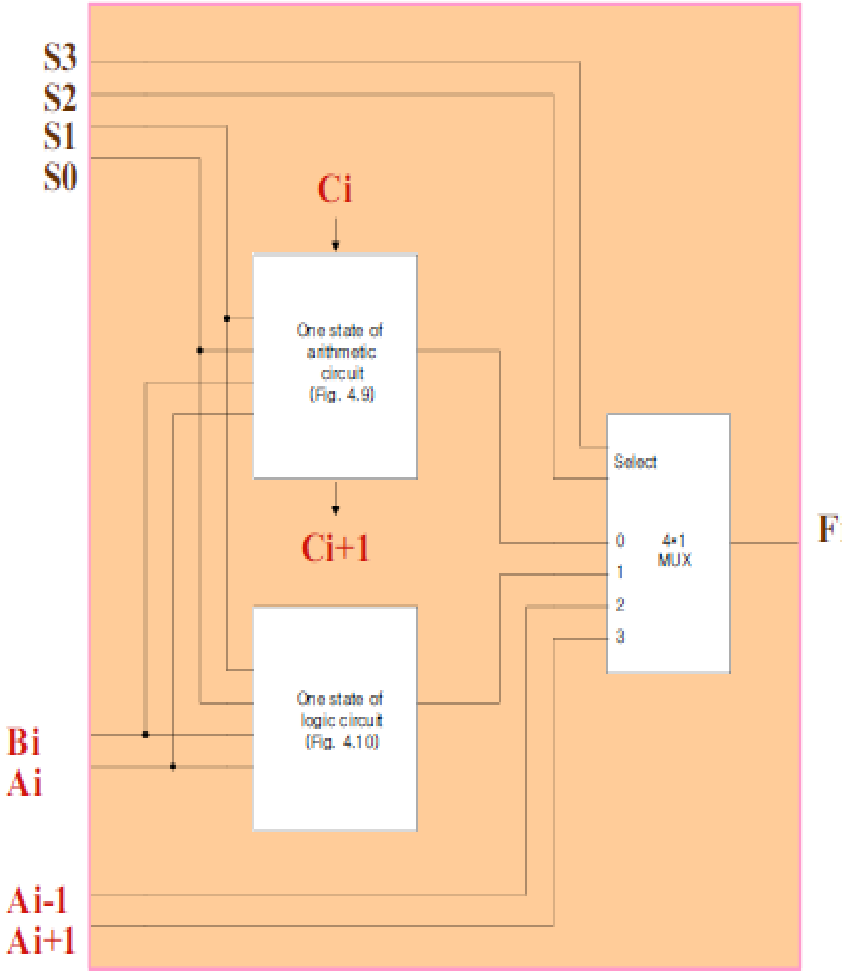


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

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

ALU - Arithmetical Logical Unit

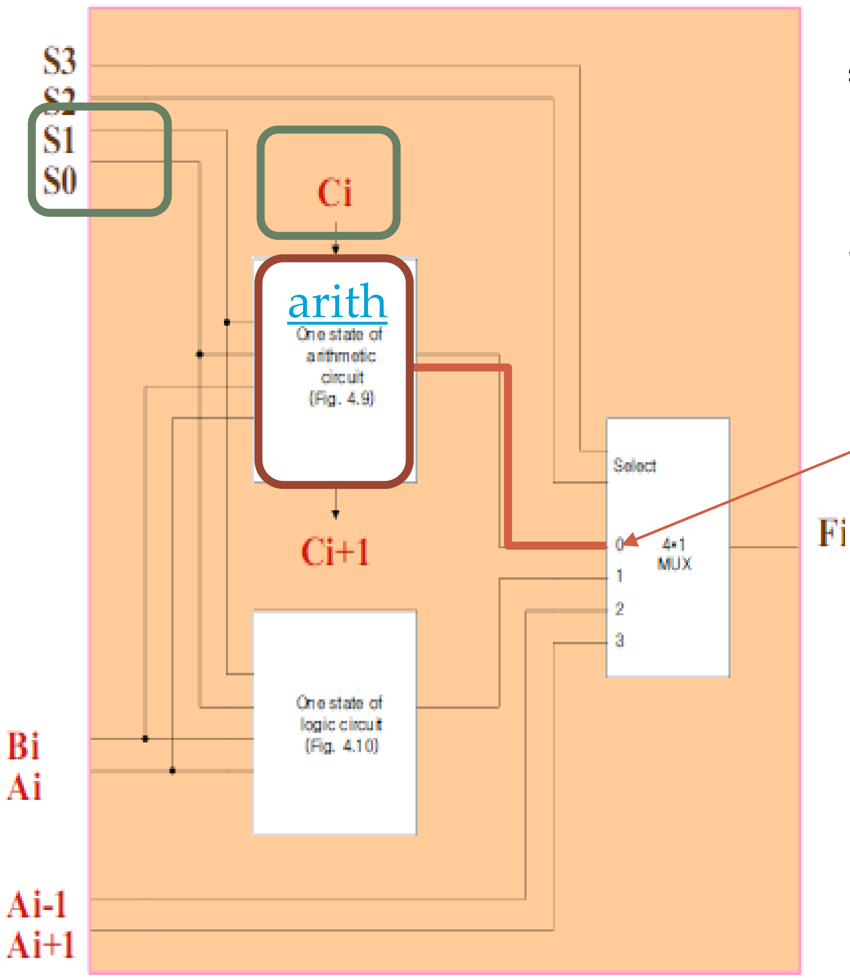


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

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

ALU - Arithmetical Logical Unit

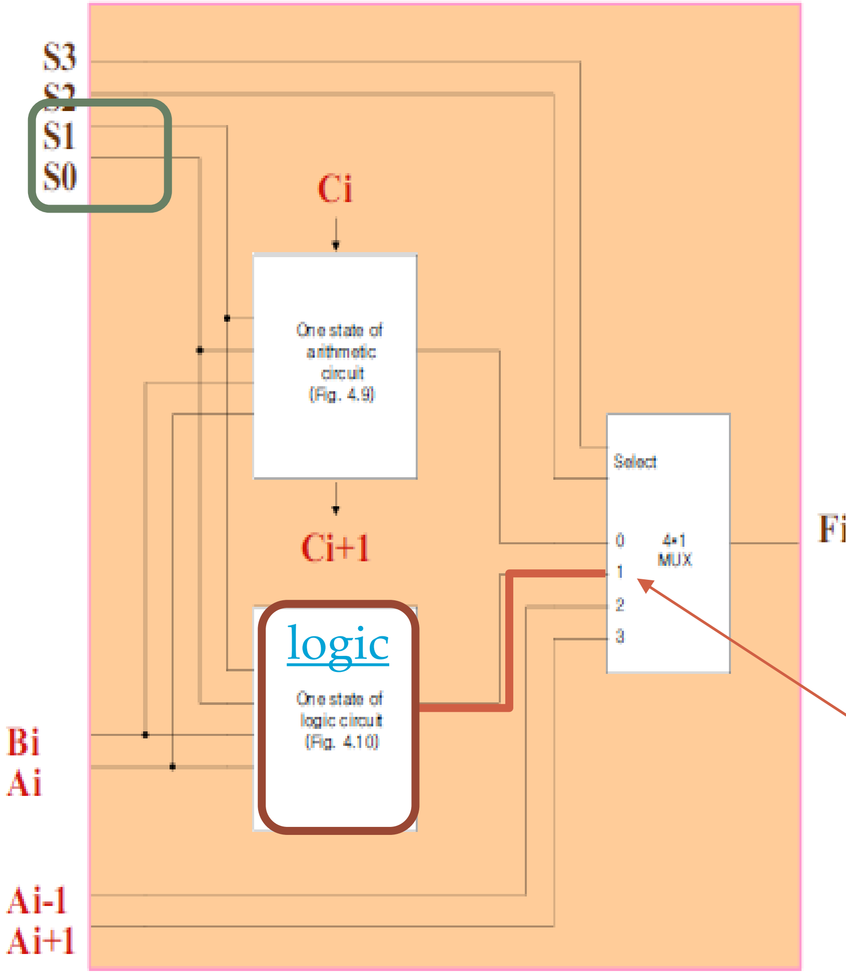


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

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

ALU - Arithmetical Logical Unit

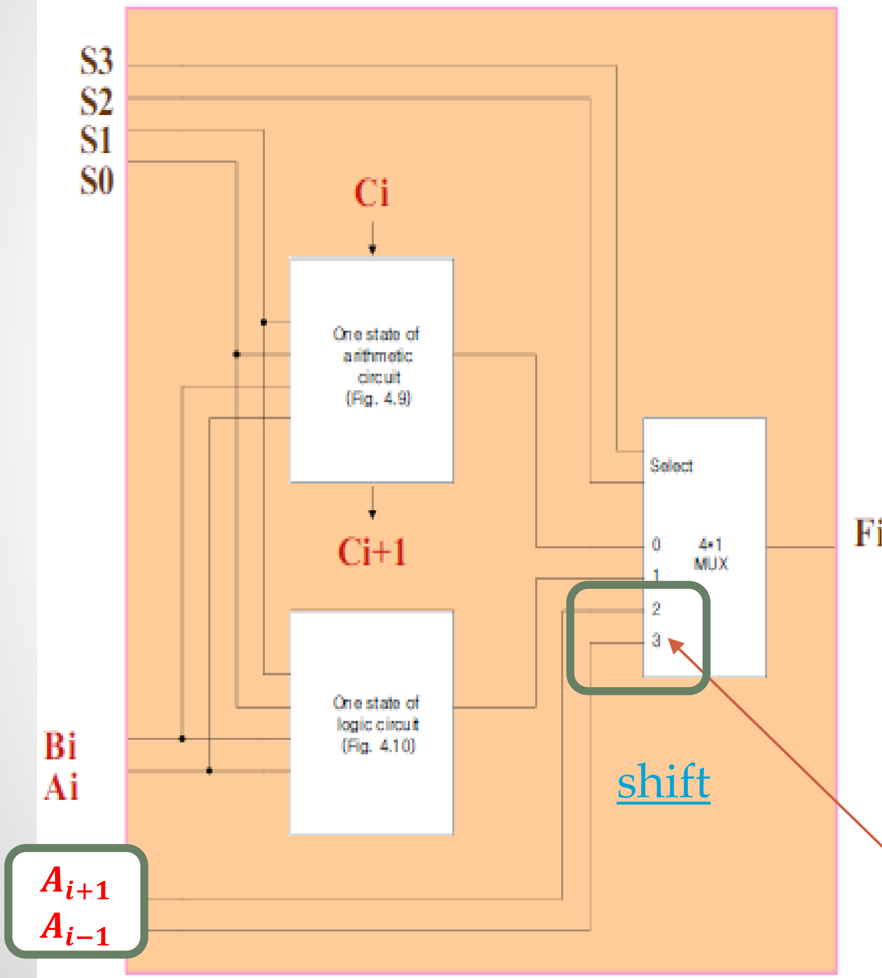


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

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