Computer System Architecture

DR. Howida Youssry

Arithmetic Microoperations

A Microoperation is an elementary operation performed with the data stored in registers.

Usually, it consist of the following 4 categories:

- Register transfer: transfer data from one register to another
- Arithmetic microoperation
- Logic microoperation
- Shift microoperation

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 (subtract)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one

Multiplication and division are not basic arithmetic operations

Multiplication : R0 = R1 * R2

Division : R0 = R1 / R2

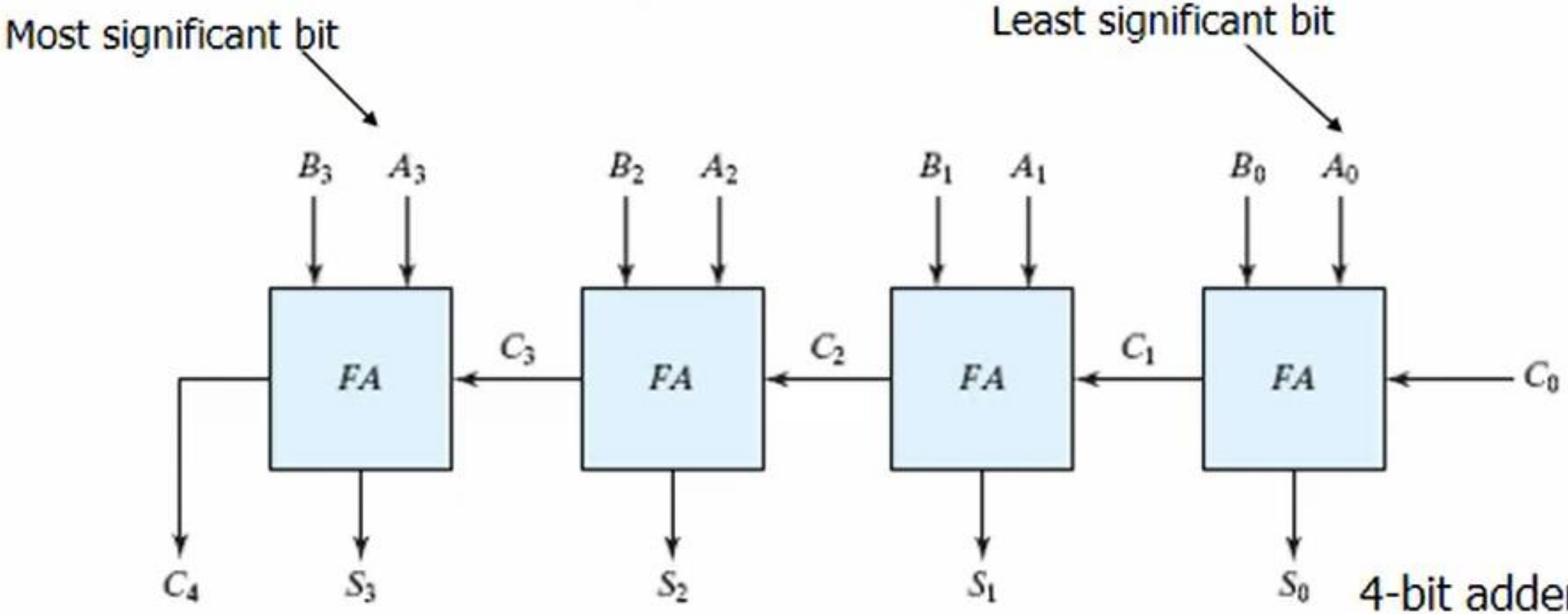
Arithmetic Microoperations

- A single circuit does both arithmetic addition and subtraction depending on control signals.

- • **Arithmetic addition**:
- R3 ← R1 + R2 (Here + is not logical OR. It denotes addition)

Binary Adder

- To implement add microoperation with a *n-bit* binary adder:
  - *n* full adders connected in cascade, with
  - the output carry from each full adder connected to the input carry of the next full adder in chain



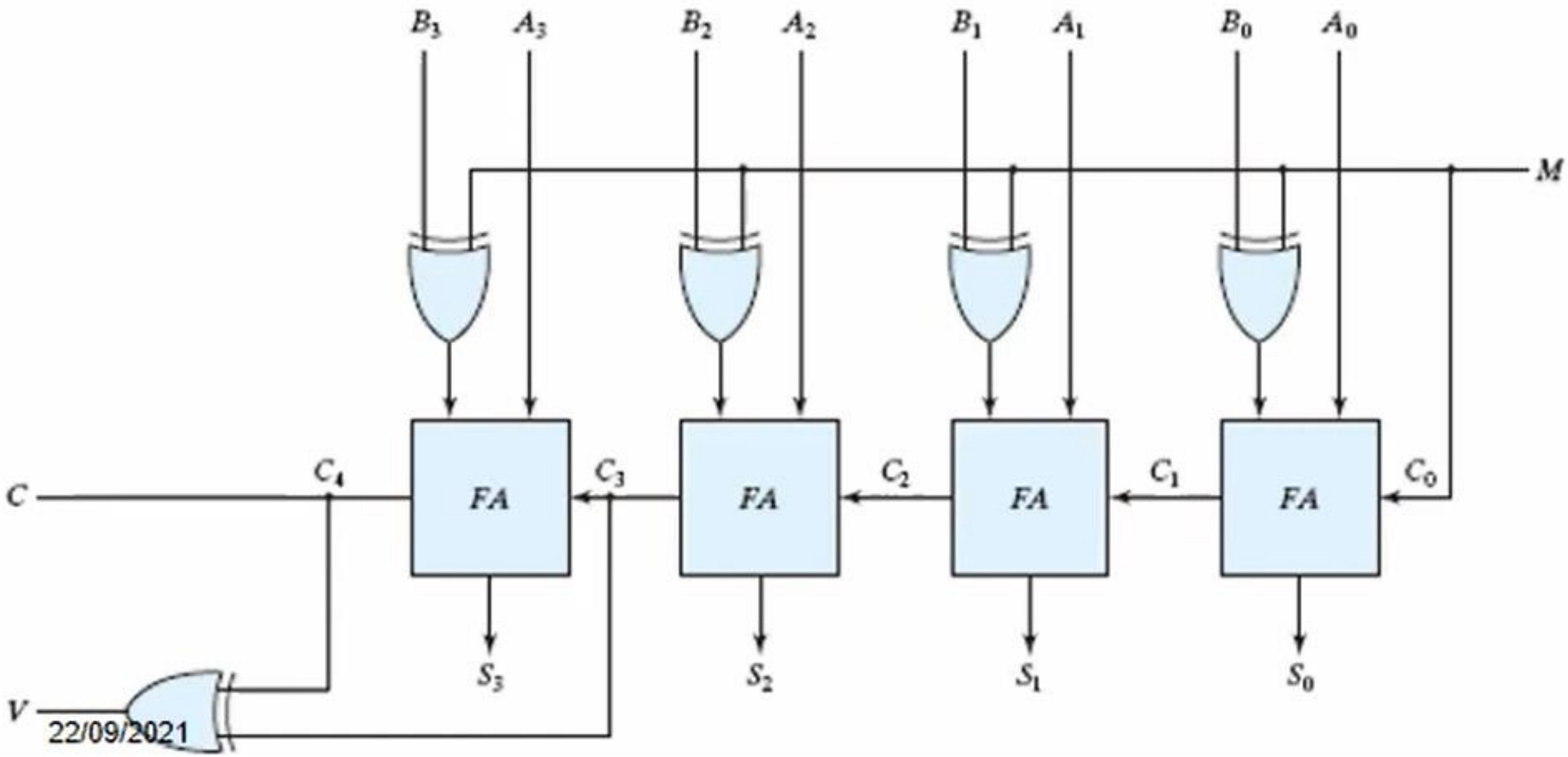
4-bit adder

Arithmetic Microoperations

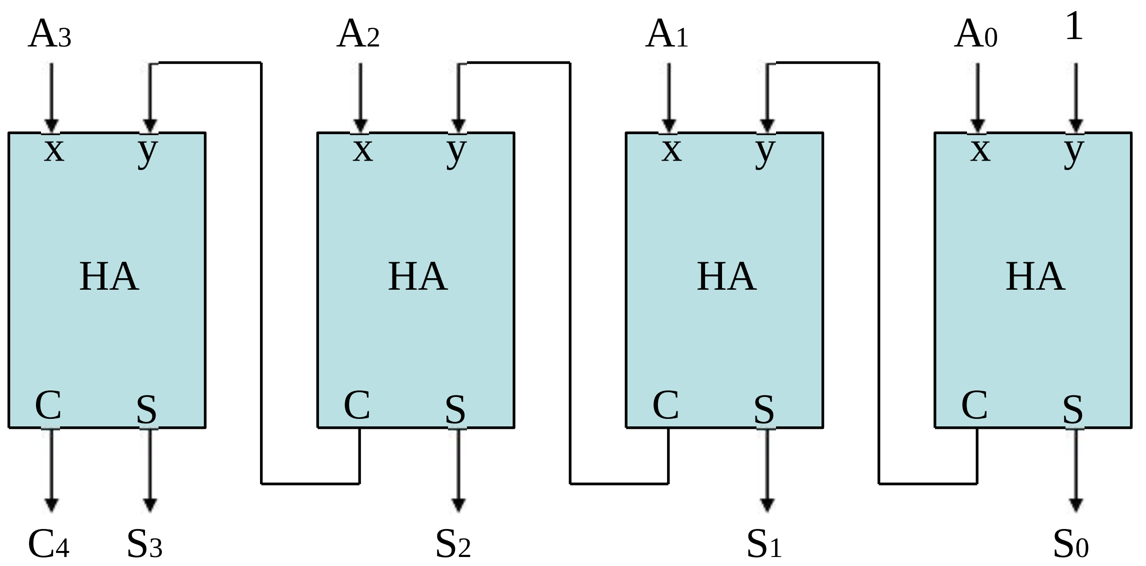
- **Arithmetic subtraction**:
- $R3 \leftarrow R1 + R2' + 1$
- where R2 is the 1's complement of R2.
- Adding 1 to the one's complement is equivalent to taking the 2's complement of R2 and adding it to R1.

Binary Adder-Subtractor

- M=0 → Adder (B⊕0=B and C_0=0)
- M=1 → Subtractor (B⊕1=B' and C_0=1)



Binary Incrementor



4-bit Binary Incrementer

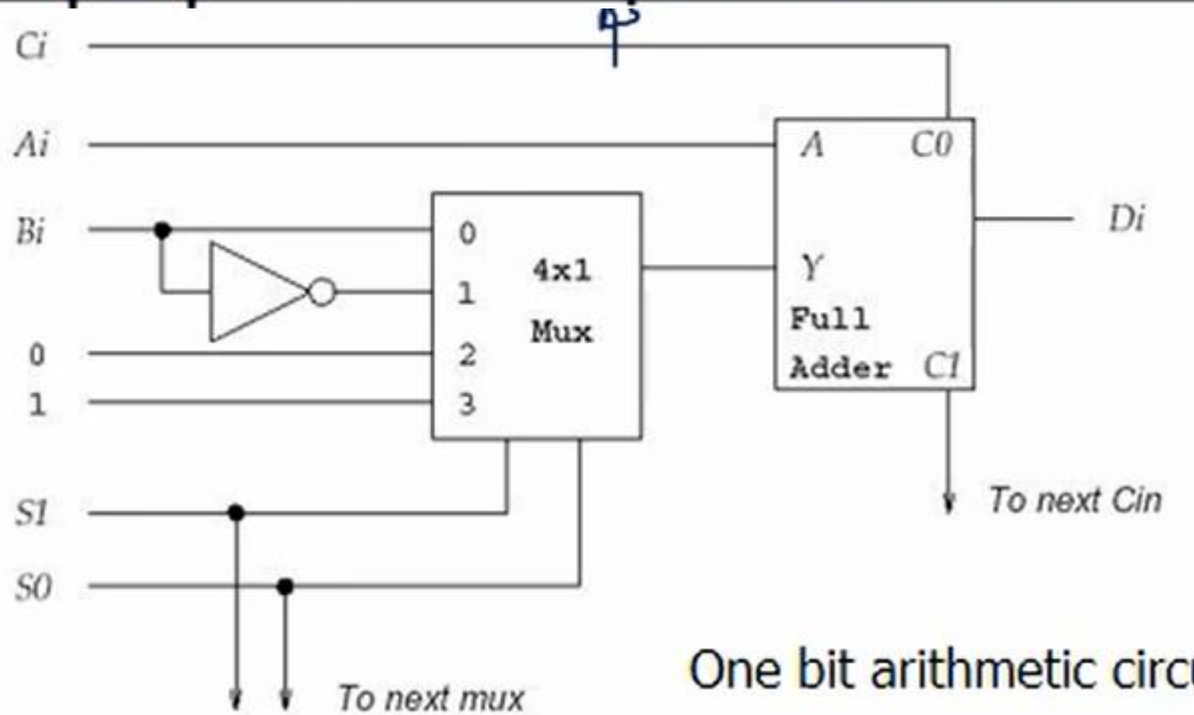
BINARY ADDER-SUBTRACTOR

- • M = 0: Note that B XOR 0 = B. This is exactly the same as the binary adder with carry in C0 = 0.

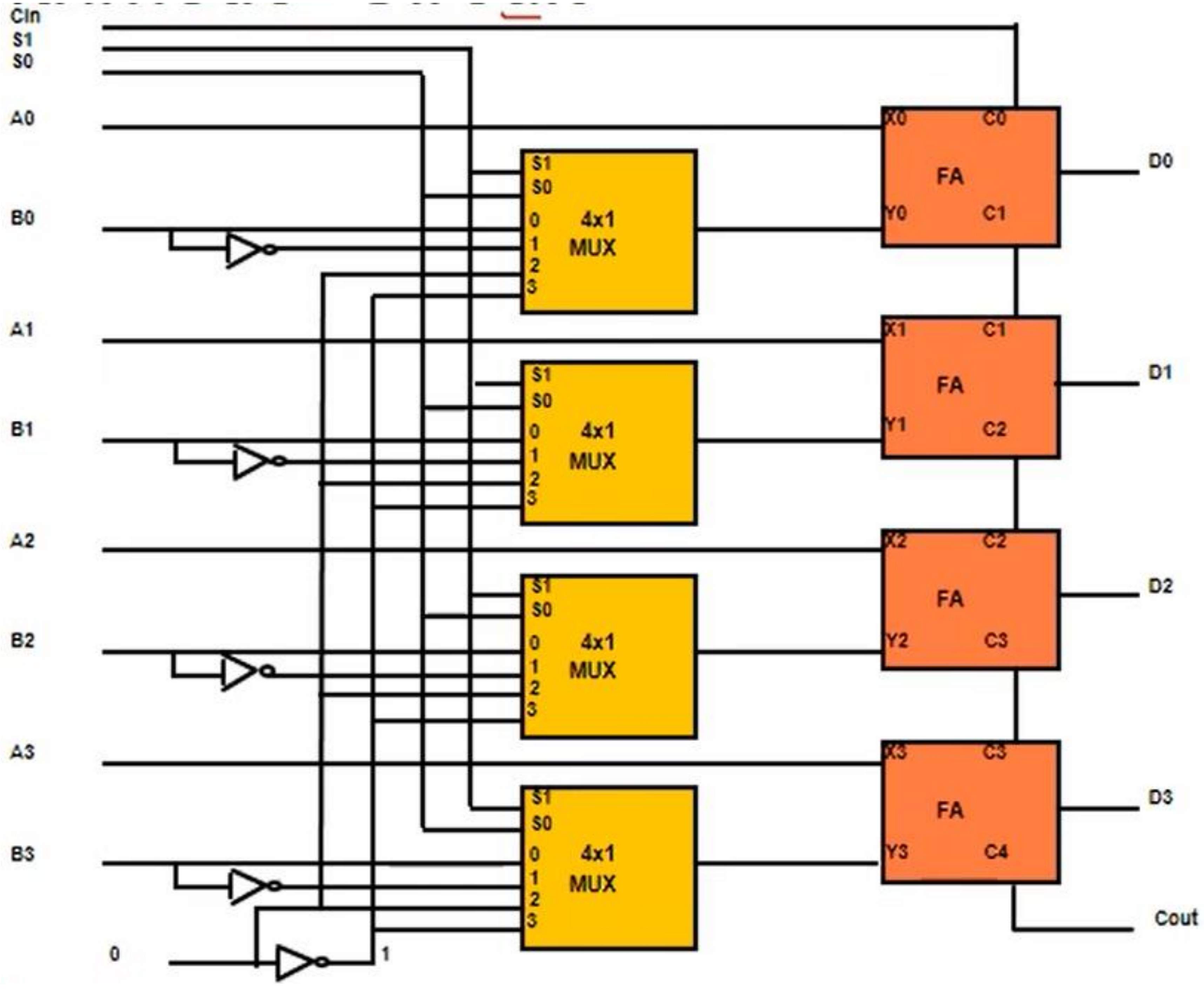
  - M = 1: Note that B XOR 1 = B (flip all B bits). The outputs of the XOR gates are thus the 1's complement of B.
  - M = 1 also provides a carry in 1. The entire operation is: A + B′ + 1.

Arithmetic Circuit

S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B’	D = A + B’	Subtract with borrow
0	1	1	B’	D = A + 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



One bit arithmetic circuit


CIn
S1
S0
A0
B0
A1
B1
A2
B2
A3
B3
0
1
S1
S0
0
1
2
3
4x1
MUX
X0
C0
Y0
C1
FA
X1
C1
Y1
C2
X2
C2
Y2
C3
X3
C3
Y3
C4
D0
D1
D2
D3
Cout

Logic Microoperations

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

$P: R1 \leftarrow R1 \oplus R2$

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

  - Manipulating the **bits** stored in a register

Symbolic designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1's complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

Logic Microoperations

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

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

$$\begin{array}{rl} 1010 & \text{Content of R1} \\ +\ 1100 & \text{Content of R2} \\ \hline 0110 & \text{Content of R1 after P=1} \end{array}$$

Special Symbols

- $\vee$ Used for denote OR
- $\wedge$ Used for denote AND

Example:

$$P+Q : R1 \leftarrow R2+R3 \text{ , } R4 \leftarrow R5 \vee R6$$

OR Operation   Add microoperation   OR microoperation

More Logic Microoperation

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 4-5. Truth Table for 16 Functions of Two Variables

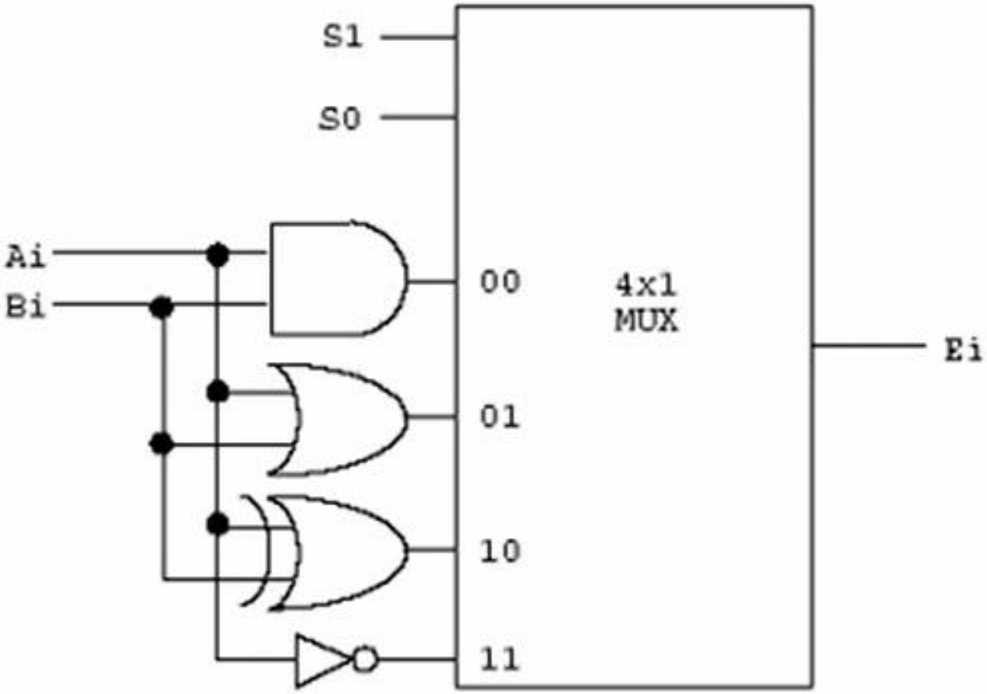
Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR
$F_7 = x+y$	$F \leftarrow A \vee B$	OR

Boolean function	Microoperation	Name
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Compl-B
$F_{11} = x+y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Compl-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

TABLE 4-6. Sixteen Logic Microoperations

HARDWARE IMPLEMENTATION OF LOGIC MICROOPERATIONS

- However, most systems only implement four of these
  - AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), Complement/NOT
- The others can be created from combination of these



S_1	S_0	E_i	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

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

Selective-set $A \leftarrow A \vee B$

- The selective-set operation 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

1010 A before
1100 B(Logic Operand)
1110 A After

Selective-set

Selective-complement $A \leftarrow A \oplus B$

» The selective-complement operation complements bits in A where there are

corresponding 1's in B. It does not effect bit positions that have 0's in B

1010	A before
1100	B(Logic Operand)
0110	A After

Selective-complement

Selective-clear $A \leftarrow A \wedge \overline{B}$

The 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

Selective-clear

Selective-mask $A \leftarrow A \wedge B$

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

1010	A before
0011	B(Logic Operand)
0010	A After

Selective-mask

Insert

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

1) Mask

0110	1010	A before
0000	1111	B mask
0000	1010	A after mask

2) OR

0000	1010	A before
1001	0000	B insert
1001	1010	A after insert

Clear

0110	A
0110	B
0000	A after clear

Clear $A \leftarrow A \oplus B$

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

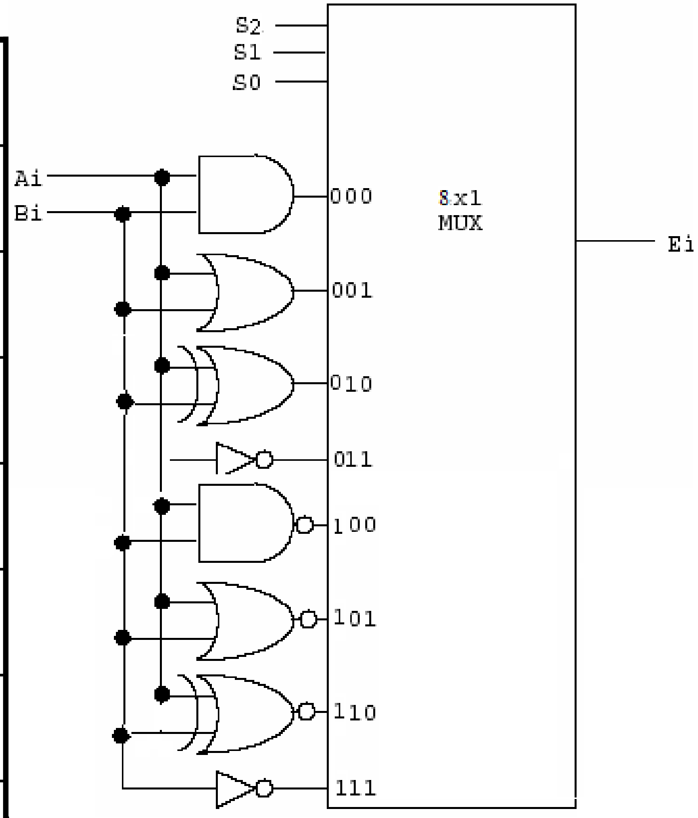
Logic Microoperations Hardware Implementation

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function
- Most computers use only four (AND, OR, XOR, and NOT) from which all others can be derived.

Example

- Extend the previous logic circuit to accommodate XNOR, NAND, NOR, and the complement of the second input.

S2	S1	S0	Output	Operation
0	0	0	$\mathbf{X} \wedge \mathbf{Y}$	AND
0	0	1	$\mathbf{X} \vee \mathbf{Y}$	OR
0	1	0	$\mathbf{X} \oplus \mathbf{Y}$	XOR
0	1	1	$\overline{A}$	Complement A
1	0	0	$\overline{(\mathbf{X} \wedge \mathbf{Y})}$	NAND
1	0	1	$\overline{(\mathbf{X} \vee \mathbf{Y})}$	NOR
1	1	0	$\overline{(\mathbf{X} \oplus \mathbf{Y})}$	XNOR
1	1	1	$\overline{B}$	Complement B



Homework 1

- Design a multiplexer to select one of the 16 functions.

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
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR
$F_7 = x+y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Compl-B
$F_{11} = x+y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Compl-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

TABLE 4-6. Sixteen Logic Microoperations

Shift Microoperations

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

Shift Microoperations

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

TABLE 4-7. Shift Microoperations

Logical Shift

- 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 (***Zero inserted***)

Logical Shift Example

1. **Logical shift**: Transfers 0 through the serial input.

R1 ← shl R1 Logical shift-left

R2 ← shr R2 Logical shift-right

(Example) Logical shift-left

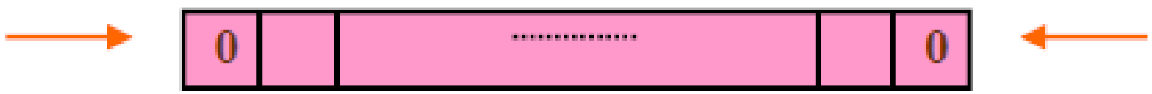
10100011 → 01000110

(Example) Logical shift-right

10100011 → 01010001

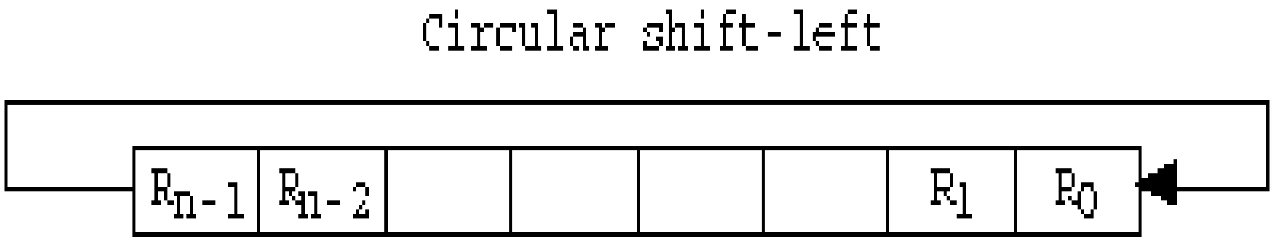
$R1 \leftarrow shl\ \ R1$

$R2 \leftarrow shr\ R2$



Circular Shift

- The ***circular shift*** circulates the bits of the register around the two ends without loss of information



Circular Shift Example

`Circular shift-left` $R1 \leftarrow cil \ \ R1$

`Circular shift-right` $R2 \leftarrow cir \ R2$

(Example) Circular shift-left

`10100011 is shifted to 01000111`

(Example) Circular shift-right

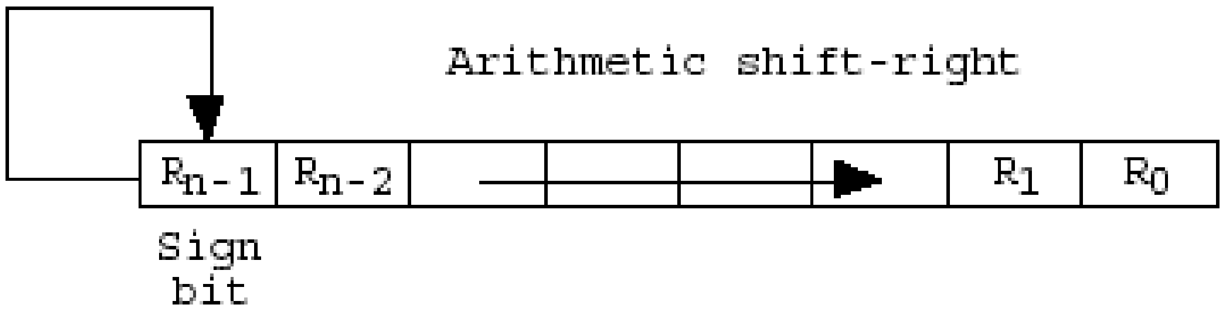
`10100011 is shifted to 11010001`

Arithmetic Shift

- An ***arithmetic shift*** 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
- In arithmetic shifts the sign bit receives a special treatment

Arithmetic Shift Right

- Arithmetic right-shift: Rn-1 remains unchanged;
- Rn-2 receives Rn-1, Rn-3 receives Rn-2, so on.
- For a negative number, 1 is shifted from the sign bit to the right. A negative number is represented by the 2's complement. The sign bit remained unchanged.



Arithmetic Shift Right

- Arithmetic Shift Right :
  - Example 1

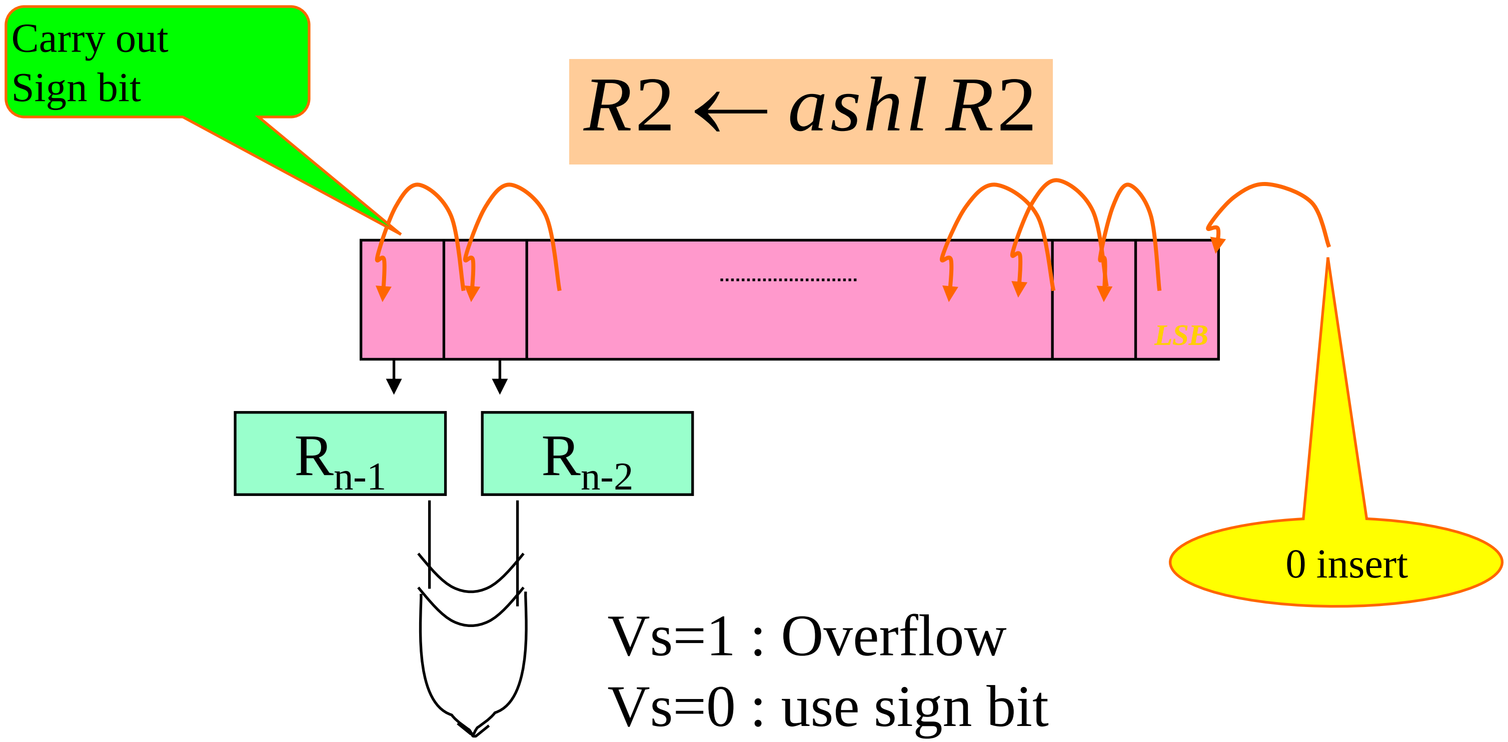
    0100 (4) → 0010 (2)
  - Example 2

    1010 (-6) → 1101 (-3)

Arithmetic Shift Left

The operation is same with Logic shift-left

The only difference is you need to check overflow problem (Check BEFORE the shift)



Arithmetic Shift Left

- Arithmetic Shift Left :
  - Example 1

    0010 (2) → 0100 (4)
  - Example 2

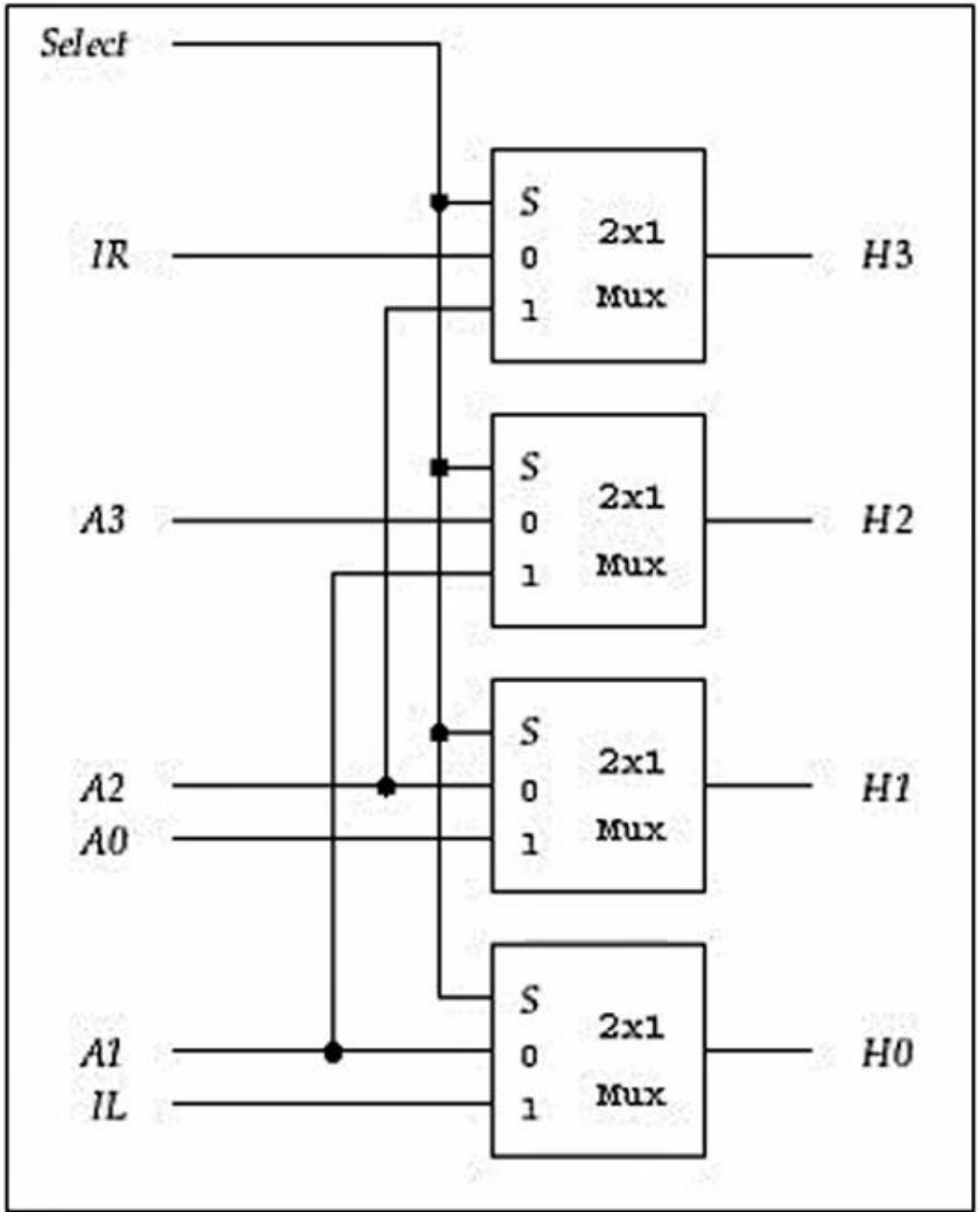
    1110 (-2) → 1100 (-4)

Arithmetic Shift Left

- Arithmetic Shift Left :
  - Example 3

    0100 (4) → 1000 (overflow)
  - Example 4

    1010 (-6) → 0100 (overflow)



S	H_3	H_2	H_1	H_0
0	I_R	A_3	A_2	A_1
1	A_2	A_1	A_0	I_L

- S=0, shift right
- S=1, shift left

Error in the book

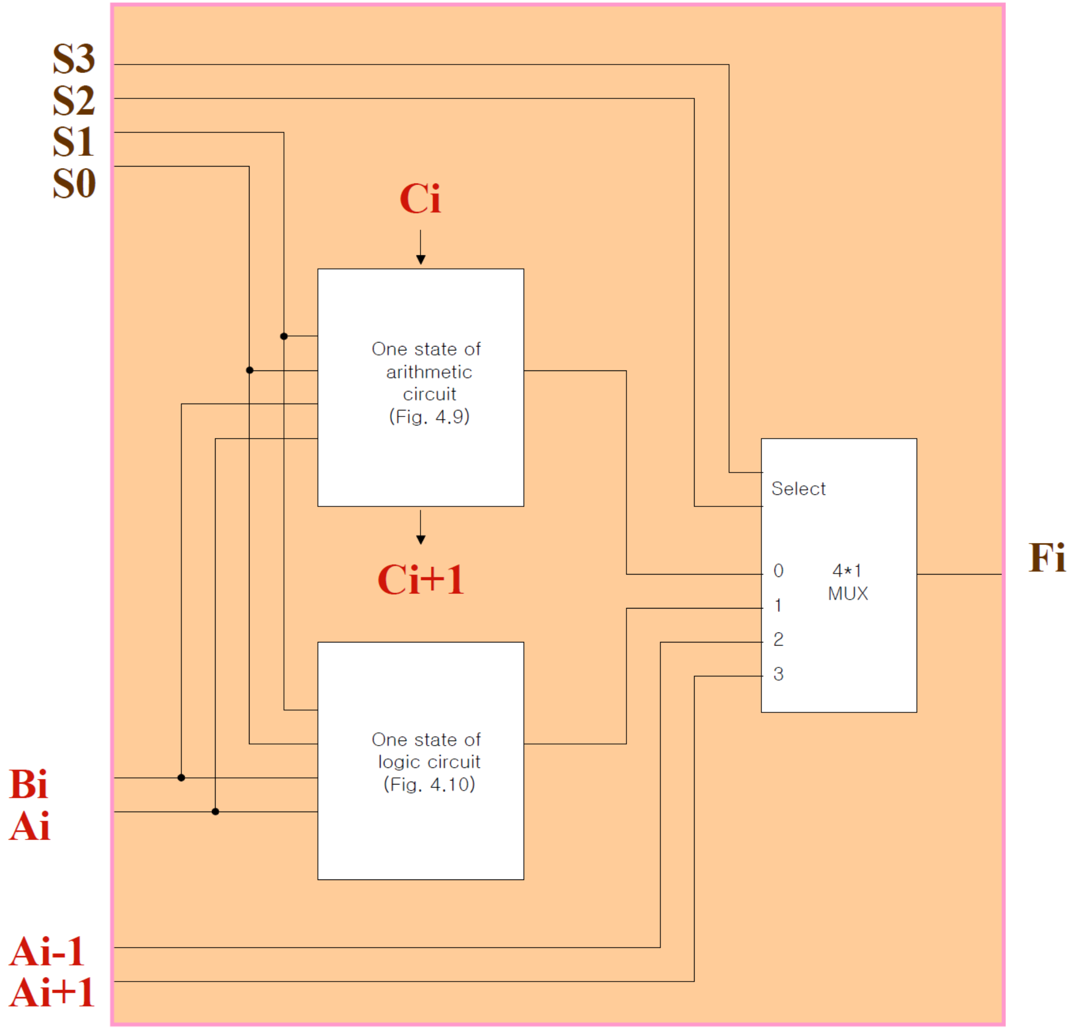
Example

- example: 011011

SHL	110110
SHR	001101
CiL	110110
CiR	101101
ASHL	Overflow
ASHR	001101

Arithmetic Logic Shift Unit

One stage of 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