Logic Microoperations

Tarunpreet Bhatia
CSED, Thapar University

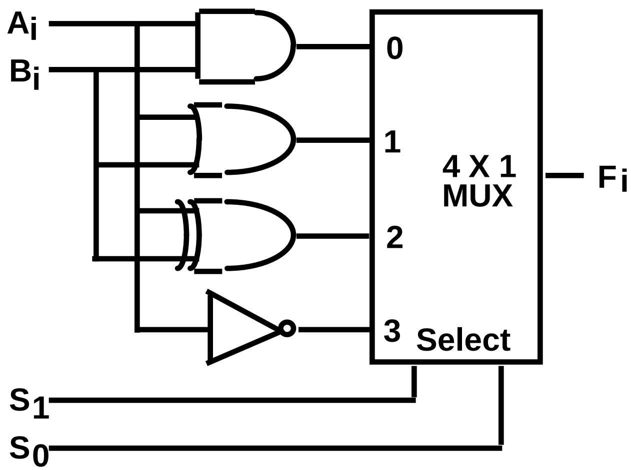
LIST OF LOGIC MICROOPERATIONS

- **List of Logic Microoperations**
  - **- 16 different logic operations with 2 binary variables.**
  - **- n binary variables $\rightarrow 2^{2^n}$ functions**

- **Truth tables for 16 functions of 2 variables and the corresponding 16 logic micro-operations**

x y	0 0 1 1 0 1 0 1	*Boolean Function*	*Micro-Operations*	*Name*
	0 0 0 0	F0 = 0	$F \leftarrow 0$	Clear
	0 0 0 1	F1 = xy	$F \leftarrow A \wedge B$	AND
	0 0 1 0	F2 = xy'	$F \leftarrow A \wedge B'$	
	0 0 1 1	F3 = x	$F \leftarrow A$	Transfer A
	0 1 0 0	F4 = x'y	$F \leftarrow A' \wedge B$	
	0 1 0 1	F5 = y	$F \leftarrow B$	Transfer B
	0 1 1 0	F6 = $x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
	0 1 1 1	F7 = x + y	$F \leftarrow A \vee B$	OR
	1 0 0 0	F8 = (x + y)'	$F \leftarrow (A \vee B)'$	NOR
	1 0 0 1	F9 = $(x \oplus y)'$	$F \leftarrow (A \oplus B)'$	Exclusive-NOR
	1 0 1 0	F10 = y'	$F \leftarrow B'$	Complement B
	1 0 1 1	F11 = x + y'	$F \leftarrow A \vee B$	
	1 1 0 0	F12 = x'	$F \leftarrow A'$	Complement A
	1 1 0 1	F13 = x' + y	$F \leftarrow A' \vee B$	
	1 1 1 0	F14 = (xy)'	$F \leftarrow (A \wedge B)'$	NAND
	1 1 1 1	F15 = 1	$F \leftarrow$ all 1's	Set to all 1's

HARDWARE IMPLEMENTATION OF LOGIC MICROOPERATIONS



Function table

S_1 S_0	Output	μ-operation
0 0	$F = A \wedge B$	AND
0 1	$F = A \vee B$	OR
1 0	$F = A \oplus B$	XOR
1 1	$F = A'$	Complement

APPLICATIONS OF LOGIC MICROOPERATIONS

- Logic microoperations can be used to manipulate individual bits or a portions of a word in a register
- Consider the data in a register A. In another register, B, is bit data that will be used to modify the contents of A
    - ➢ Selective-set $A \leftarrow A + B$
    - ➢ Selective-complement $A \leftarrow A \oplus B$
    - ➢ Selective-clear $A \leftarrow A \bullet B'$
    - ➢ Mask (Delete) $A \leftarrow A \bullet B$
    - ➢ Clear $A \leftarrow A \oplus B$
    - ➢ Insert $A \leftarrow (A \bullet B) + C$
    - ➢ Compare $A \leftarrow A \oplus B$

SELECTIVE SET

- In a selective set operation, the bit pattern in B is used to *set* certain bits in A

$$
\begin{array}{ll}
1\,1\,0\,0 & A_t \\
1\,0\,1\,0 & B \\
\hline
1\,1\,1\,0 & A_{t+1} \qquad (A \leftarrow A + B)
\end{array}
$$

- If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value.
- OR microoperation can be used to selectively set bits of a register.

SELECTIVE COMPLEMENT

- In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A

$$\begin{array}{rl} 1\,1\,0\,0 & A_t \\ 1\,0\,1\,0 & B \\ \hline 0\,1\,1\,0 & A_{t+1} \end{array} \qquad (A \leftarrow A \oplus B)$$

- If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged.
- The exclusive-OR microoperation can be used to selectively complement bits of a register.

SELECTIVE CLEAR

- In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A

$$
\begin{array}{ll}
1\,1\,0\,0 & A_t \\
1\,0\,1\,0 & B \\
\hline
0\,1\,0\,0 & A_{t+1} \quad (A \leftarrow A \cdot B')
\end{array}
$$

- If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged

MASK OPERATION

- In a mask operation, the bit pattern in B is used to *clear* certain bits in A

$$
\begin{array}{ll}
1\ 1\ 0\ 0\ A_t & \\
\underline{1\ 0\ 1\ 0\ B\qquad} & \\
1\ 0\ 0\ 0\ A_{t+1} & (A \leftarrow A \cdot B)
\end{array}
$$

- If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged

CLEAR OPERATION

- In a clear operation, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A.

$$
\begin{array}{ll}
1\,1\,0\,0 & A_t \\
1\,0\,1\,0 & B \\
\hline
0\,1\,1\,0 & A_{t+1} \quad (A \leftarrow A \oplus B)
\end{array}
$$

- It compares words in A and B and produces all 0's result if the two numbers are equal. This operation is achieved by XOR microoperation.

INSERT OPERATION

- An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged
- This is done as
  - A mask operation to clear the desired bit positions, followed by
  - An OR operation to introduce the new bits into the desired positions

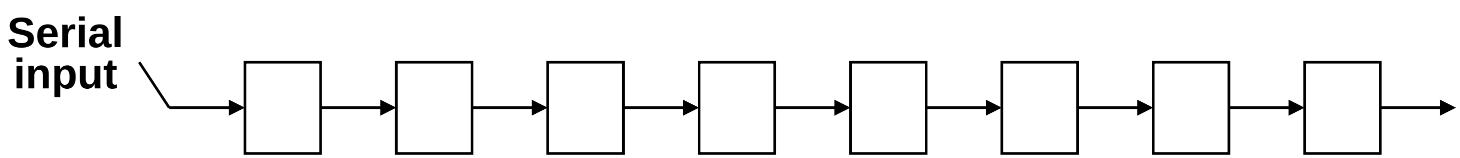
- Example
  - Suppose you wanted to introduce 1010 into the low order four bits of A: 1101 1000 1011 0001 A (Original)
    1101 1000 1011 1010 A (Desired)

  - ```
    1101 1000 1011 0001          A (Original)
    1111 1111 1111 0000          Mask
    1101 1000 1011 0000          A (Intermediate)
    0000 0000 0000 1010          Added bits
    ----------------------------------------------
     1101 1000 1011 1010         A (Desired)
    ----------------------------------------------
    ```
    ```

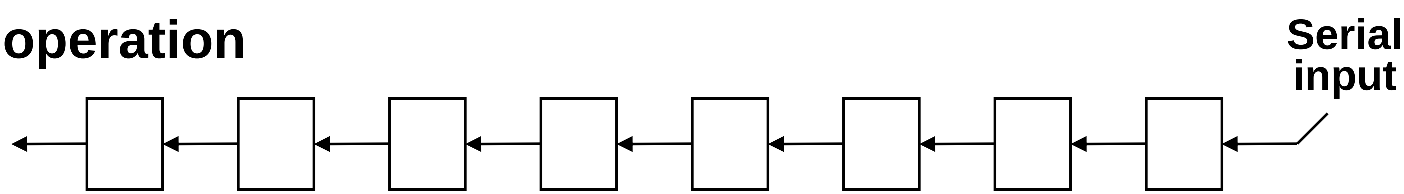
SHIFT MICROOPERATIONS

- Shift microoperations are used for serial transfer of data.
- The information transferred through the serial input determines the type of shift. There are three types of shifts
  - *Logical shift*
  - *Circular shift*
  - *Arithmetic shift*

- **A right shift operation**

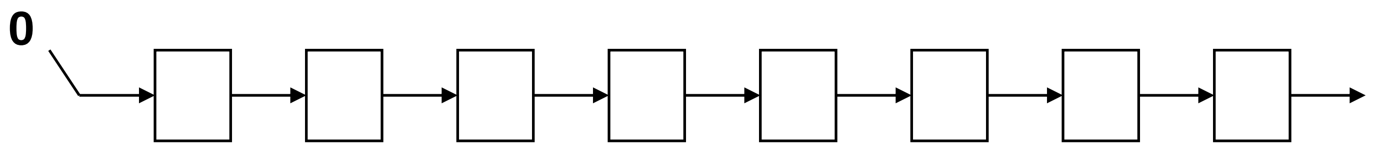


- **A left shift operation**

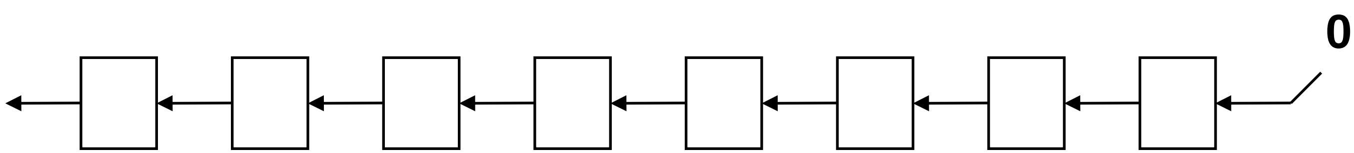


LOGICAL SHIFT

- In a logical shift the serial input to the shift is a 0.
- A right logical shift operation:



- A left logical shift operation:

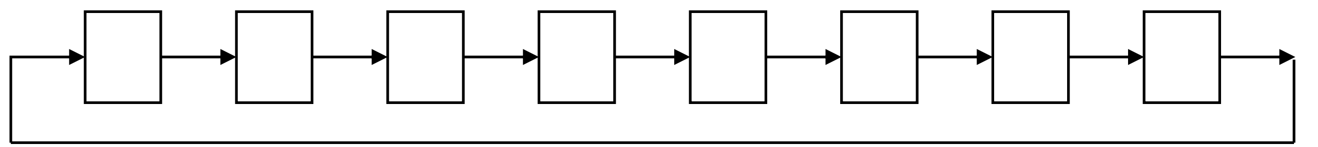


- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right
  - Examples:
    - R2 $\leftarrow$ *shr* R2
    - R3 $\leftarrow$ *shl* R3

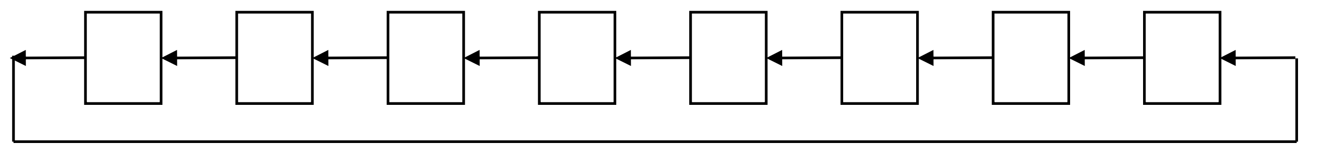
CIRCULAR SHIFT

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.

- A right circular shift operation:



- A left circular shift operation:

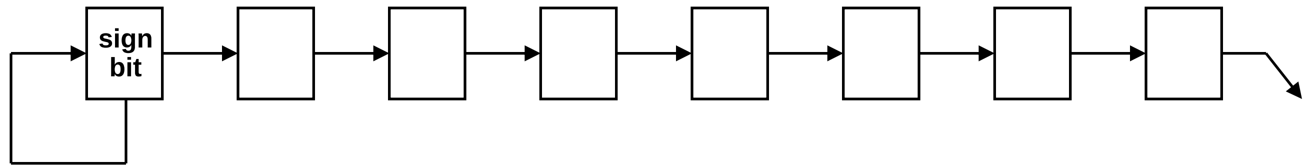


- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right
  - Examples:
    - R2 $\leftarrow$ *cir* R2
    - R3 $\leftarrow$ *cil* R3

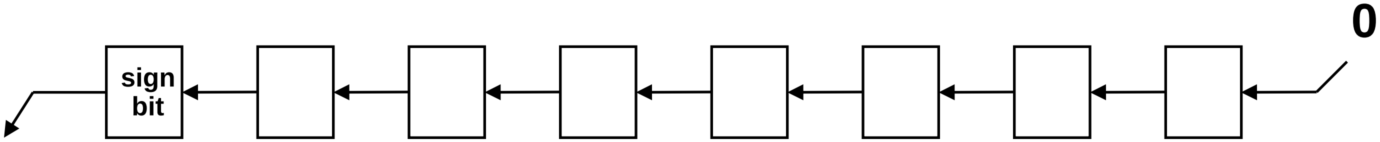
ARITHMETIC SHIFT

- An arithmetic shift is meant for signed binary numbers (integer)
- An arithmetic left shift multiplies a signed number by two
- An arithmetic right shift divides a signed number by two
- The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division

- A right arithmetic shift operation:

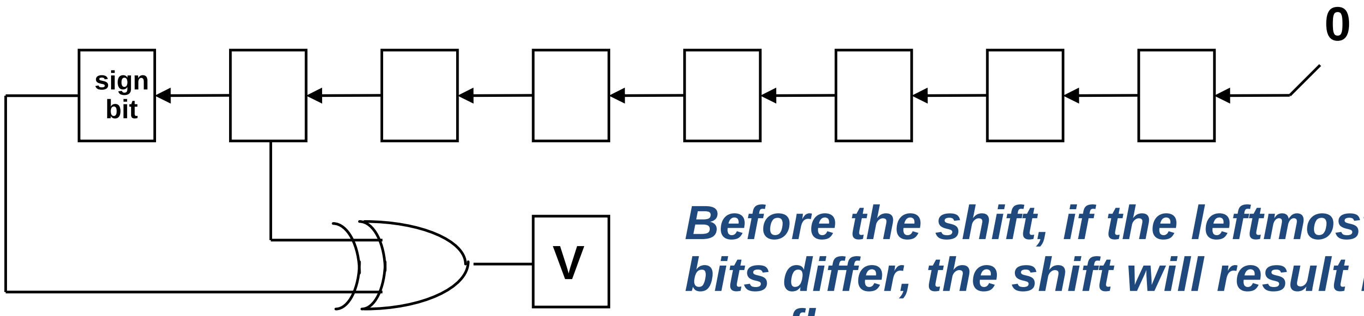


- A left arithmetic shift operation:



ARITHMETIC SHIFT

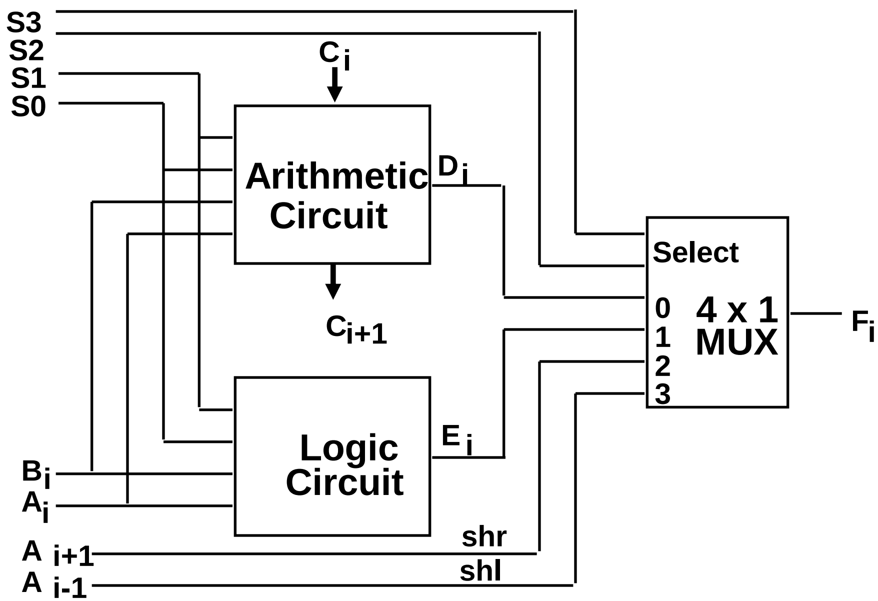
- An left arithmetic shift operation must be checked for the overflow



Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- In a RTL, the following notation is used
  - *ashl* for an arithmetic shift left
  - *ashr* for an arithmetic shift right
  - Examples:
    - R2 $\leftarrow$ *ashr* R2
    - R3 $\leftarrow$ *ashl* R3

ARITHMETIC LOGIC SHIFT UNIT



S3	S2	S1	S0	Cin	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 + B’	Subtract with borrow
0	0	1	0	1	F = A + B’+ 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
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 = A’	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

Example

Register A holds 8-bit operand 11011001. Determine the operand B and logic microoperation to be performed in order to change the value of A to

a) 01101101

b) 11111101

$$\begin{array}{l} A = \ 11011001 \\ \underline{B = \ 10110100 \oplus} \\ A' = 01101101 \quad A' \leftarrow A \oplus B \end{array}$$

$$\begin{array}{l} A = \ 11011001 \\ \underline{B = \ 11111101} \text{ (OR)} \\ A' = 11111101 \ A' \leftarrow A \vee B \end{array}$$