Overview

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Micro-operations
- **Logic Micro-operations**
- **Shift Micro-operations**
- **Arithmetic Logic Shift Unit**

Logic Micro operations

- **Logic microoperation**
  - Logic microoperations consider ***each bit of the register separately*** and treat them as binary variables
    - » *exam)* $P: R1 \leftarrow R1 \oplus R2$

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

  - Special Symbols
    - » Special symbols will be adopted for the logic microoperations ***OR(∨), AND(∧),*** and ***complement(a bar on top)***, to distinguish them from the corresponding symbols used to express Boolean functions
    - » *exam)* $P + Q: R1 \leftarrow R2 + R3,\ R4 \leftarrow R5 \vee R6$

      Logic OR (the + in P + Q)

      Arithmetic ADD (the + in R2 + R3)

- **List of Logic Microoperation**
  - Truth Table for 16 functions for 2 variables : ***Tab. 4-5***
  - 16 Logic Microoperation : ***Tab. 4-6***
- **Hardware Implementation**
  - 16 microoperation → Use only 4(AND, OR, XOR, Complement)

    ∵ All other Operation can be derived
  - One stage of logic circuit

Logic Microoperations

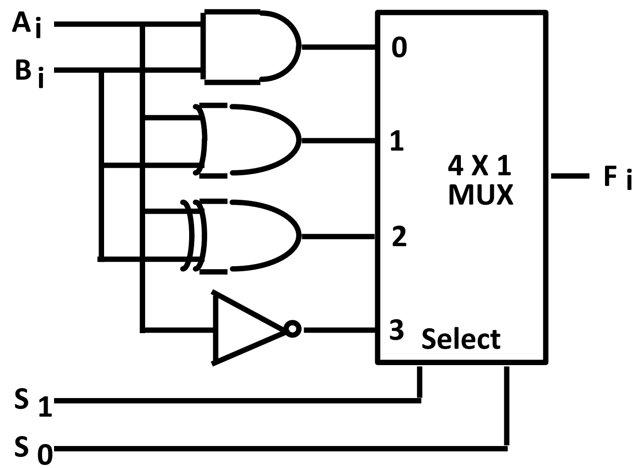
X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	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



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** $A \leftarrow A \oplus B$

Applications of Logic Microoperations

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

1 1 0 0 A_t

1 0 1 0 B

1 1 1 0 A_{t+1} $(A \leftarrow A + B)$

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

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

1 1 0 0 A_t

1 0 1 0 B

0 1 1 0 A_{t+1} $(A \leftarrow A \oplus B)$

If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged

Applications of Logic Microoperations

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

1 1 0 0	A_t	
1 0 1 0	B	
0 1 0 0	A_{t+1}	$(A \leftarrow A \cdot B')$

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

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

1 1 0 0	A_t	
1 0 1 0	B	
1 0 0 0	A_{t+1}	$(A \leftarrow A \cdot B)$

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

Applications of Logic Microoperations

5. 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

1 1 0 0	A_t
1 0 1 0	B
0 1 1 0	A_{t+1} $(A \leftarrow A \oplus B)$

Applications of Logic Microoperations

6. **An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged**

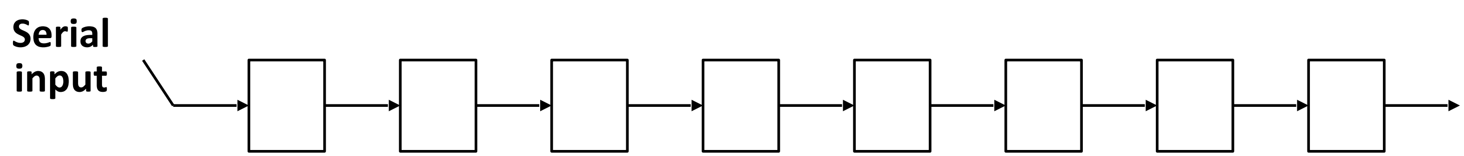
   This is done as

   - **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)
       ```
       ```

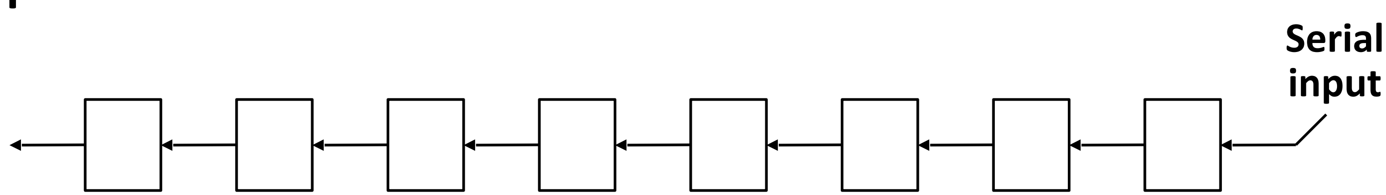
Shift Microoperations

- **There are three types of shifts**
  - ***Logical shift***
  - ***Circular shift***
  - ***Arithmetic shift***
- **What differentiates them is the information that goes into the serial input**

- **A right shift operation**



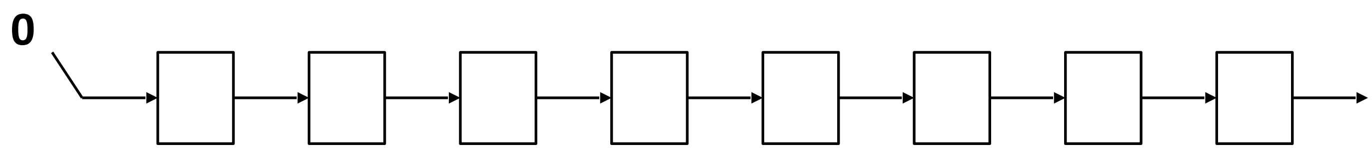
- **A left shift operation**



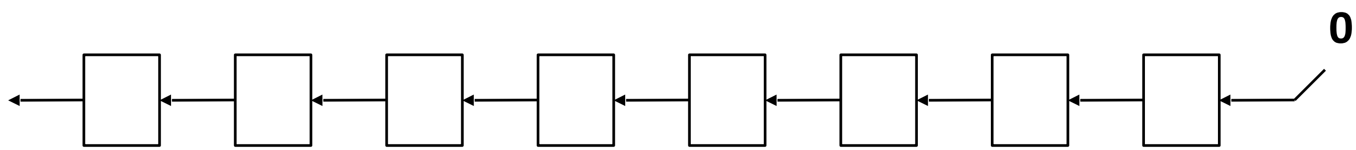
Logical Shift

- **In a logical shift the serial input to the shift is a 0.**

- **A right logical shift operation:**



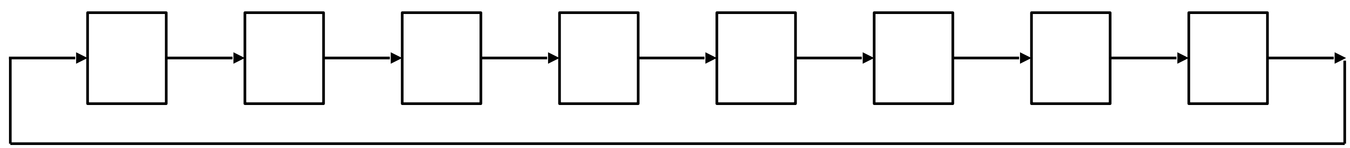
- **A left logical shift operation:**



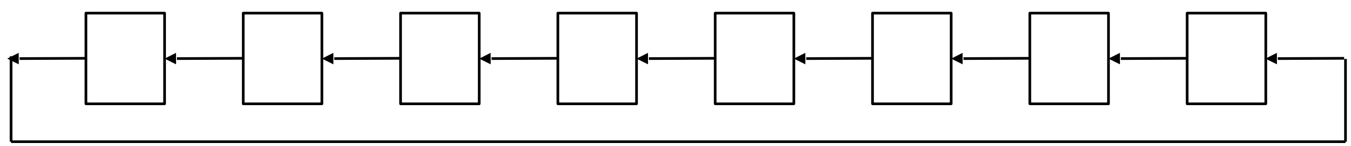
- **In a Register Transfer Language, the following notation is used**
  - ***shl*** **for a logical shift left**
  - ***shr*** **for a logical shift right**
  - **Examples:**
    - **R2 ← *shr* R2**
    - **R3 ← *shl* R3**

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:**

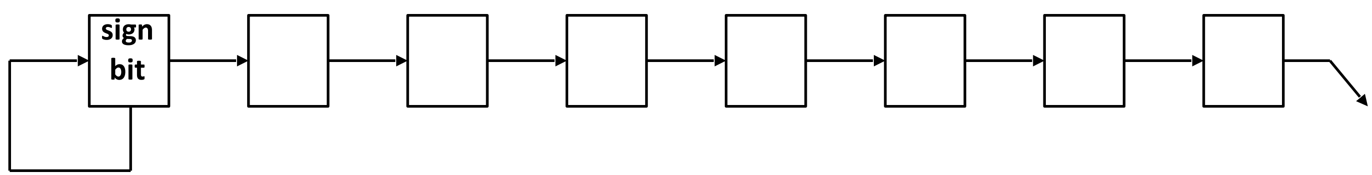


- **In a RTL, the following notation is used**
  - ***cil*** **for a circular shift left**
  - ***cir*** **for a circular shift right**
  - **Examples:**
    - **R2 ← *cir* R2**
    - **R3 ← *cil* R3**

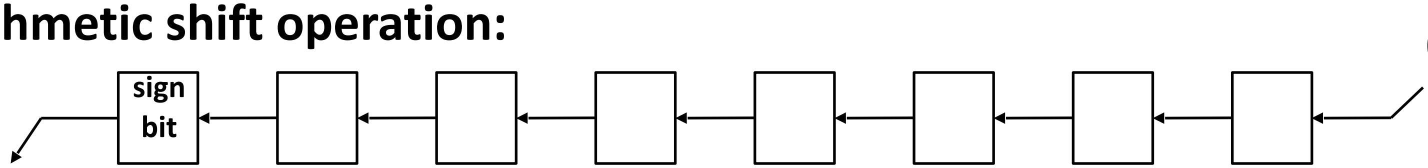
Arithmetic Shift

- **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**
- **Sign bit : 0 for positive and 1 for negative**
- **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:**

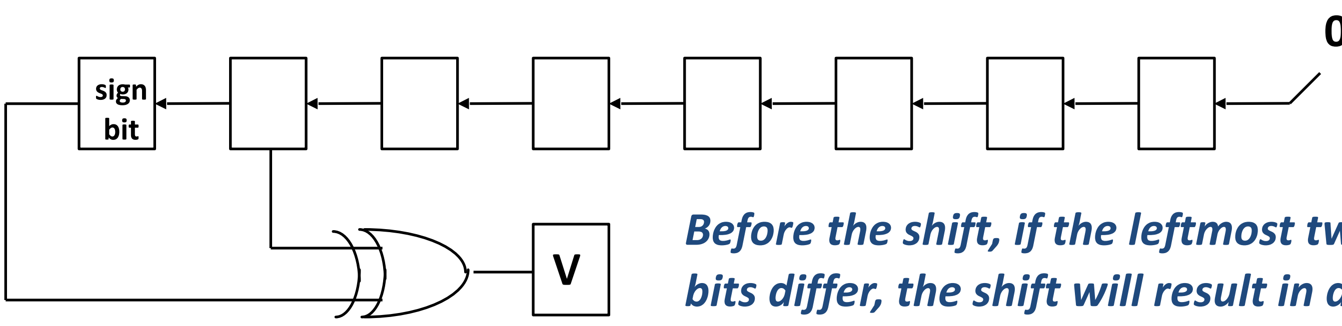


- **A left arithmetic shift operation:**



Arithmetic Shift

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

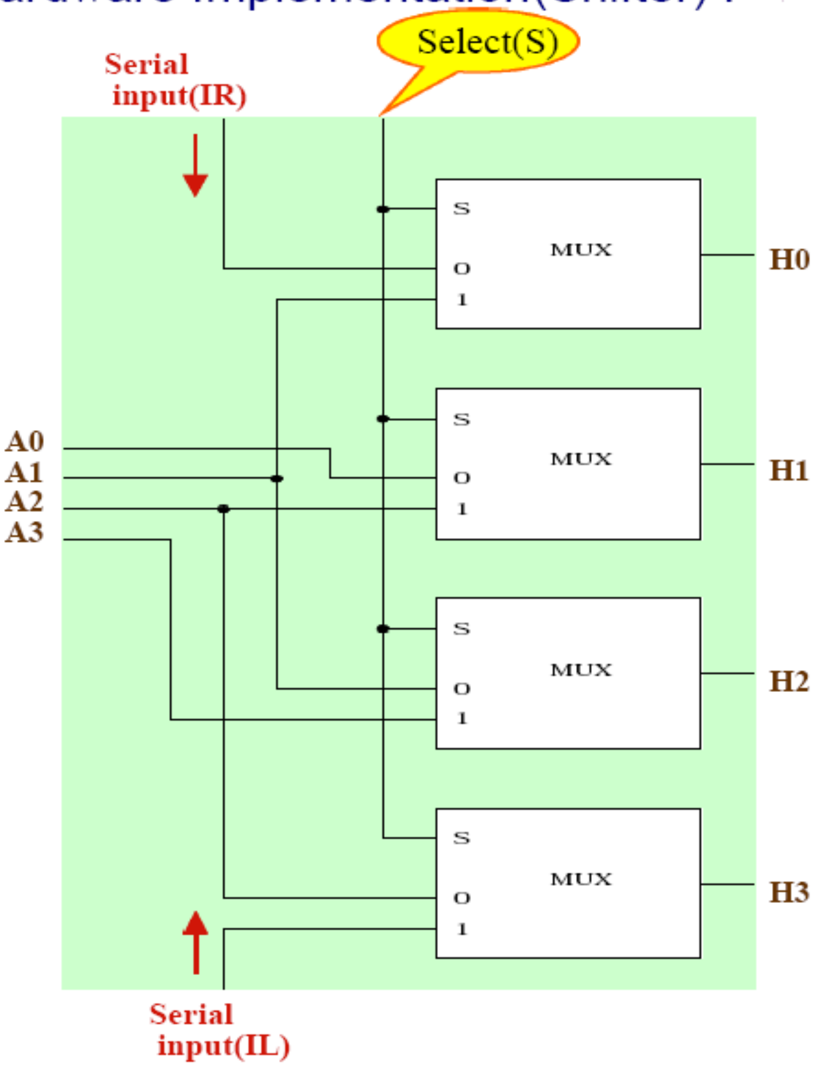


Before the shift, if the leftmost two bits differ, the shift will result in an overflow

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

Hardware Implementation of Shift Microoperation

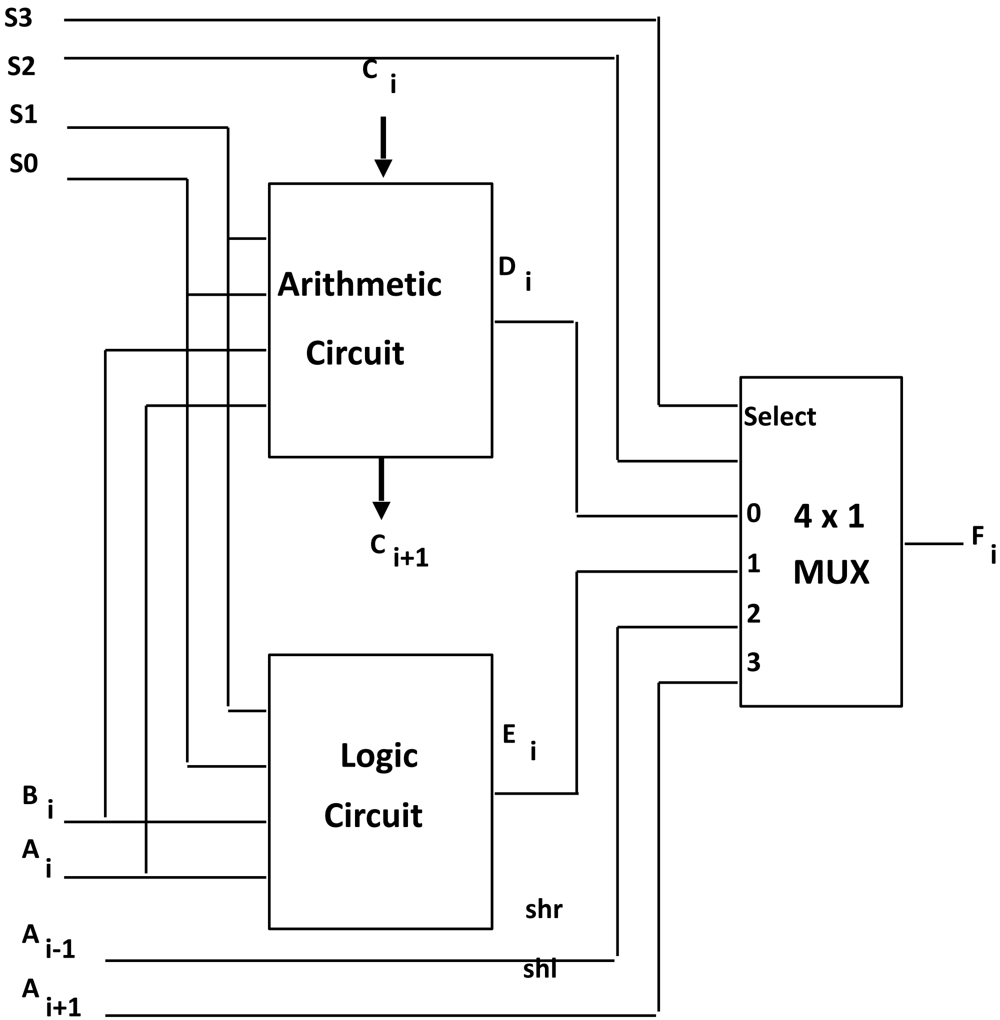
- Hardware Implementation(Shifter) :



Function Table

Select	output			
S	H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

Arithmetic Logic and Shift Unit



s2	s3	Operation
0	0	Arithmetic
0	1	Logical
1	0	Shr
1	1	shl

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