Overview

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

Register Transfer Language

- **Combinational and sequential circuits can be used to create simple digital systems.**
- **These are the low-level building blocks of a digital computer.**
- **Simple digital systems are frequently characterized in terms of**
  - **the registers they contain, and**
  - **the operations that are performed on data stored in them**
- **The operations executed on the data in registers are called <u>micro-operations</u> e.g. shift, count, clear and load**

Register Transfer Language

Internal hardware organization of a digital computer :

- **Set of registers and their functions**
- **Sequence of microoperations performed on binary information stored in registers**
- **Control signals that initiate the sequence of micro-operations (to perform the functions)**

Register Transfer Language

- **Rather than specifying a digital system in words, a specific notation is used, Register Transfer Language**
- **The symbolic notation used to describe the micro operation transfer among register is called a register transfer language**
- **For any function of the computer, the register transfer language can be used to describe the (sequence of) micro-operations**
- **Register transfer language**
  - **A symbolic language**
  - **A convenient tool for describing the internal organization of digital computers in concise/precise manner.**
  - **Can also be used to facilitate the design process of digital systems.**

Register Transfer Language

- **Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)**
- **Often the names indicate function:**

MAR	**- memory address register**
PC	**- program counter**
IR	**- instruction register**

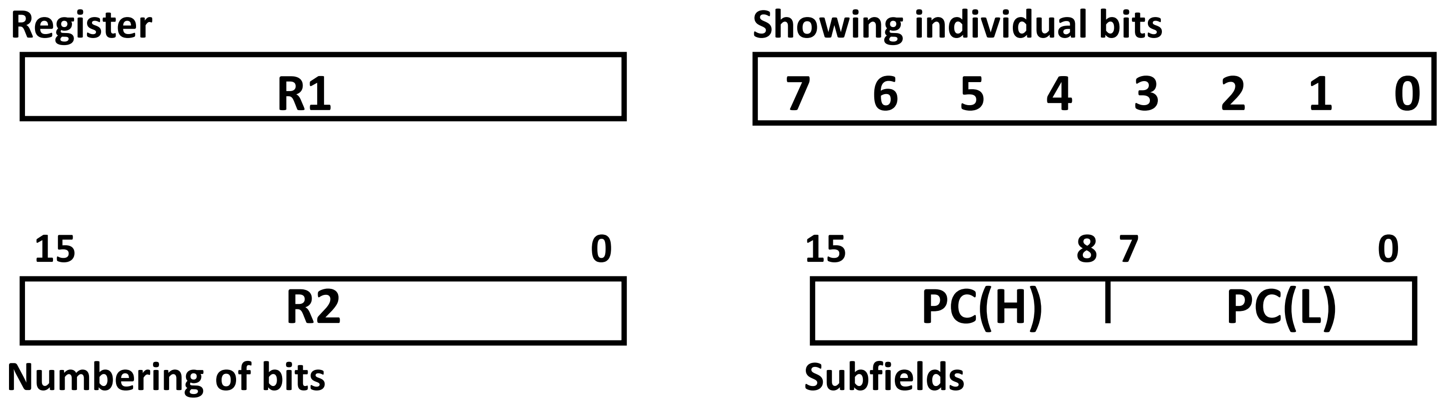
- **Registers and their contents can be viewed and represented in *various ways***
  - **A register can be viewed as a single entity:**

Register Transfer Language

- **Designation of a register**

  - a register
  - portion of a register
  - a bit of a register

- **Common ways of drawing the block diagram of a register**



Register Transfer Language

- **Copying the contents of one register to another is a register transfer**
- **A register transfer is indicated as**

$R2 \leftarrow R1$

- ➢ **In this case the contents of register R1 are copied (loaded) into register R2**
- ➢ **A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse**
- ➢ **Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2**

Register Transfer Language

- **A register transfer such as**

  R3 $\leftarrow$ R5

  Implies that the digital system has

  - **the data lines from the source register (R5) to the destination register (R3)**
  - **Parallel load in the destination register (R3)**
  - **Control lines to perform the action**

Control Functions

- **Often actions need to only occur if a certain condition is true**
- **This is similar to an "if" statement in a programming language**
- **In digital systems, this is often done via a *control signal*, called a *control function***
  - **If the signal is 1, the action takes place**
- **This is represented as:**

P: R2 ← R1

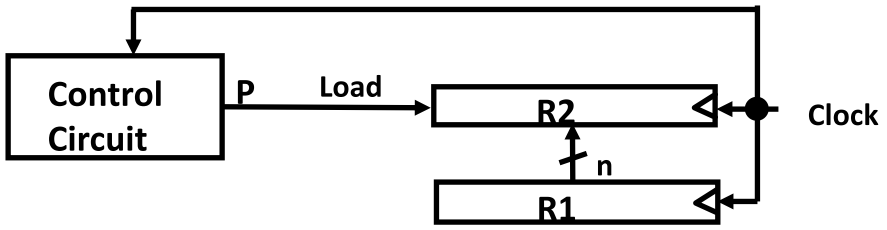
Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

Hardware Implementation of Controlled Transfers

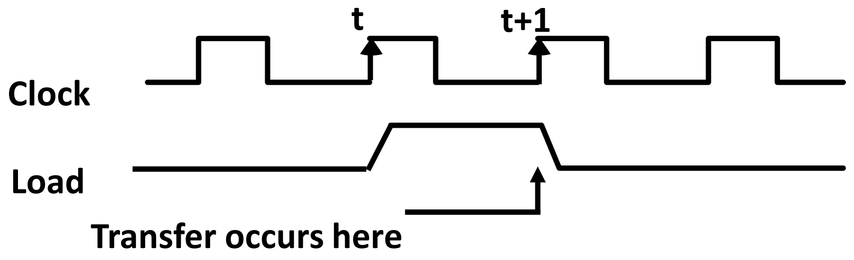
Implementation of controlled transfer

P: R2 ← R1

Block diagram



Timing diagram



- ➢ **The same clock controls the circuits that generate the control function and the destination register**
- ➢ **Registers are assumed to use *positive-edge-triggered* flip-flops**

Basic Symbols in Register Transfer

Symbols	Description	Examples
Capital letters & Numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	R2 $\leftarrow$ R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A $\leftarrow$ B, B $\leftarrow$ A

Overview

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

Connecting Registers - Bus Transfer

- **In a digital system with many registers, it is impractical to have data and control lines to directly allow each register to be loaded with the contents of every possible other registers**

- **To completely connect n registers → n(n-1) lines**
- **O(n^2) cost**
  - **This is not a realistic approach to use in a large digital system**

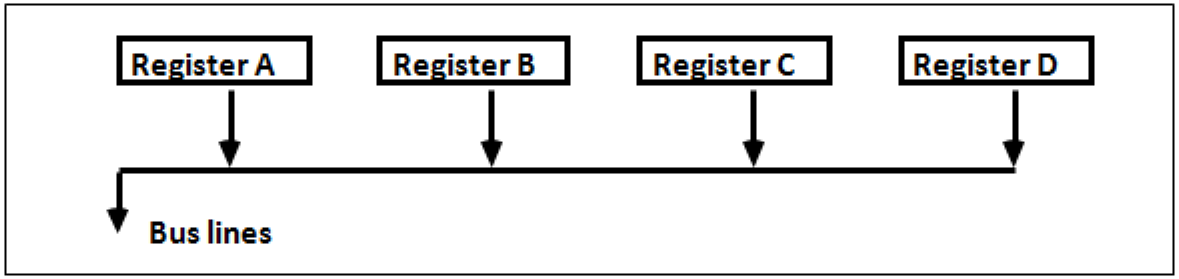
- **Instead, take a different approach**
- **Have one centralized set of circuits for data transfer – the bus**

- **BUS STRUCTURE CONSISTS OF SET OF COMMON LINES, ONE FOR EACH BIT OF A REGISTER THROUGH WHICH BINARY INFORMATION IS TRANSFERRED ONE AT A TIME**

- **Have control circuits to select which register is the source, and which is the destination**

Connecting Registers - Bus Transfer

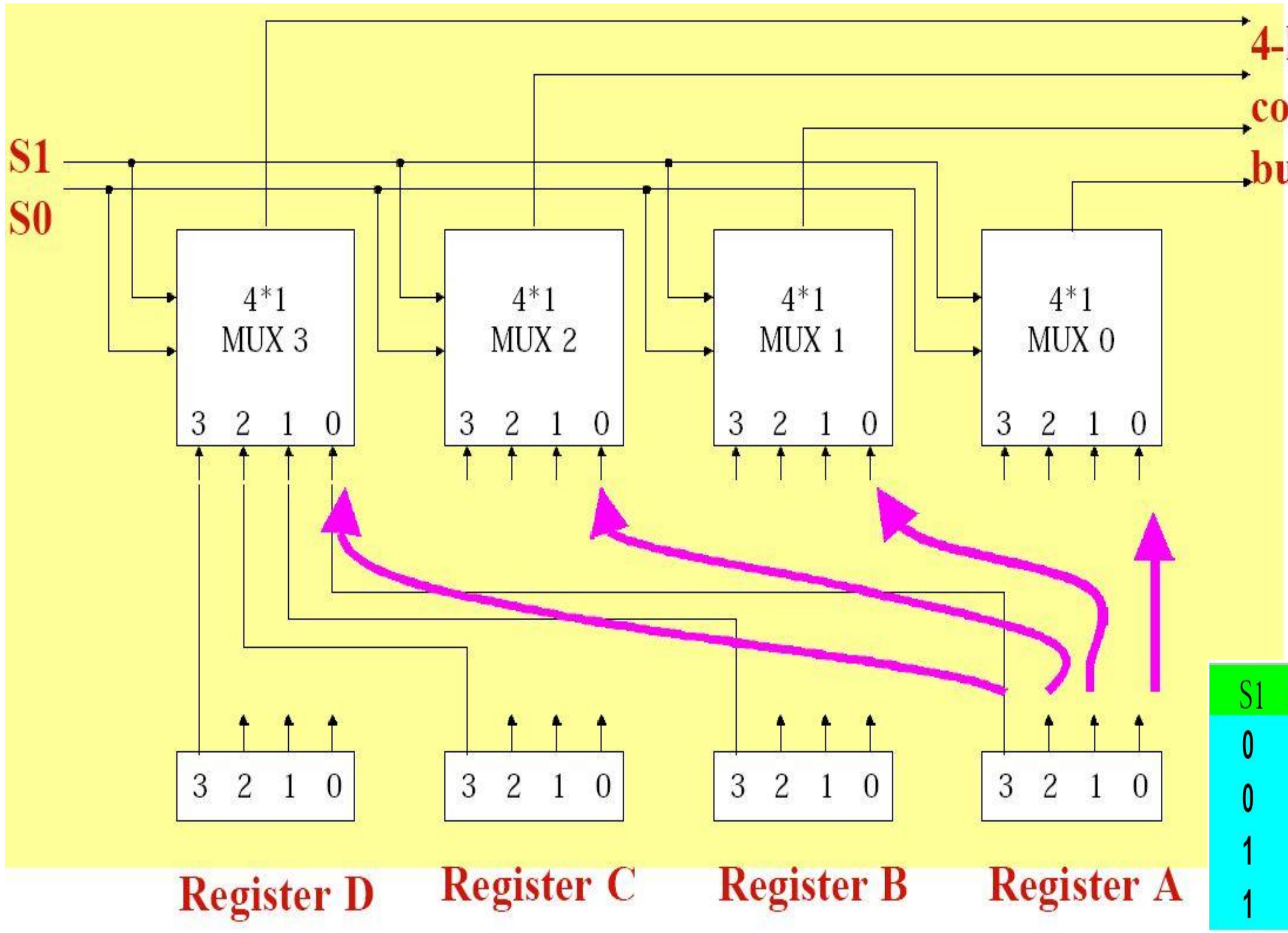
From a register to bus: BUS $\leftarrow$ R



- **One way of constructing common bus system is with multiplexers**
- **Multiplexer selects the source register whose binary information is kept on the bus.**

- **Construction of bus system for 4 register (Next Fig)**
  - **4 bit register X 4**
  - **four 4X1 multiplexer**
  - **Bus selection S0, S1**

Connecting Registers - Bus Transfer



S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Connecting Registers - Bus Transfer

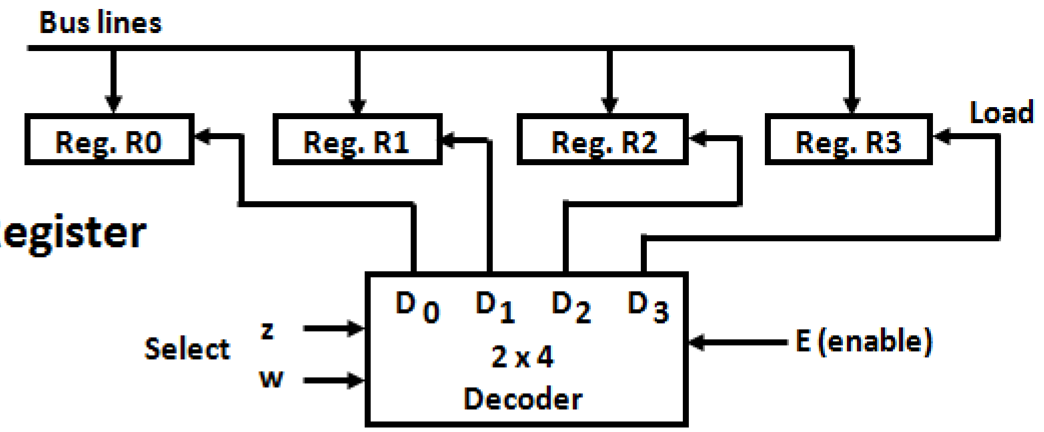
- For a bus system to multiplex k registers of n bits each
  - No. of multiplexer = n
  - Size of each multiplexer = k x 1

- **Construction of bus system for 8 register with 16 bits**
  - **16 bit register X 8**
  - **Sixteen 8X1 multiplexer**
  - **Bus selection S0, S1, S2**

Connecting Registers - Bus Transfer

From a Bus to Register

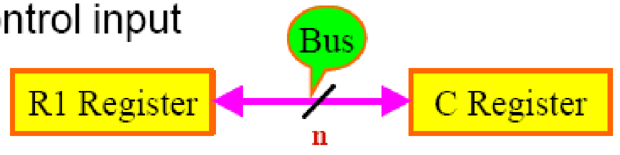
R$\leftarrow$ Bus



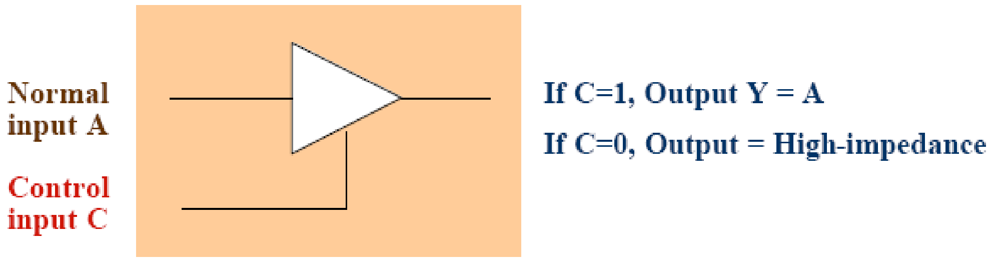
Connecting Registers - Bus Transfer

- **Bus Transfer**
  - The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input

$$\left.\begin{array}{l} Bus \leftarrow C, \;\; R1 \leftarrow Bus \\ R1 \leftarrow C \end{array}\right\} =$$

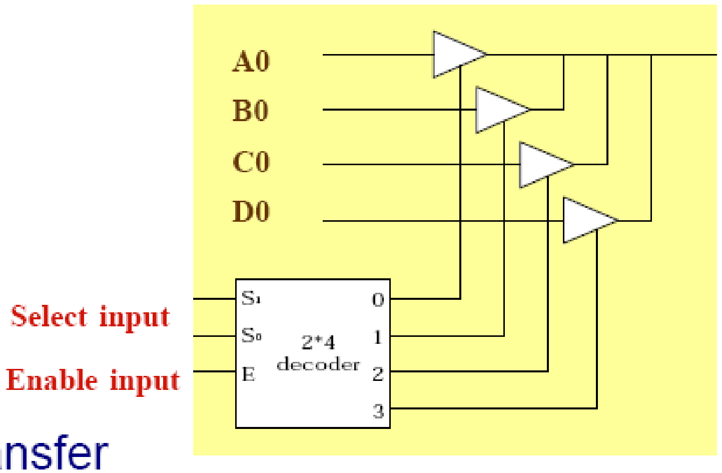


- **Three-State Bus Buffers**
  - A bus system can be constructed with ***three-state gates*** *instead of* ***multiplexers***
  - Tri-State : 0, 1, High-impedance(***Open circuit***)
  - Buffer
    - » A device designed to be inserted between other devices to match impedance, to prevent mixed interactions, and ***to supply additional drive or relay capability***
    - » Buffer types are classified as inverting or noninverting
  - Tri-state buffer gate : Fig. 4-4
    - » When control input =1 : The output is enabled(output Y = input A)
    - » When control input =0 : The output is disabled(output Y = high-impedance)



Connecting Registers - Bus Transfer

- The construction of a bus system with tri-state buffer : *Fig.*
  - The outputs of four buffer are connected together to form a single bus line(Tri-state buffer
  - No more than one buffer may be in the active state at any given time(2 X 4 Decoder
  - To construct a common bus for 4 register with 4 bit : Fig.



AR: Address Reg.
DR: Data Reg.
M : Memory Word(Data)

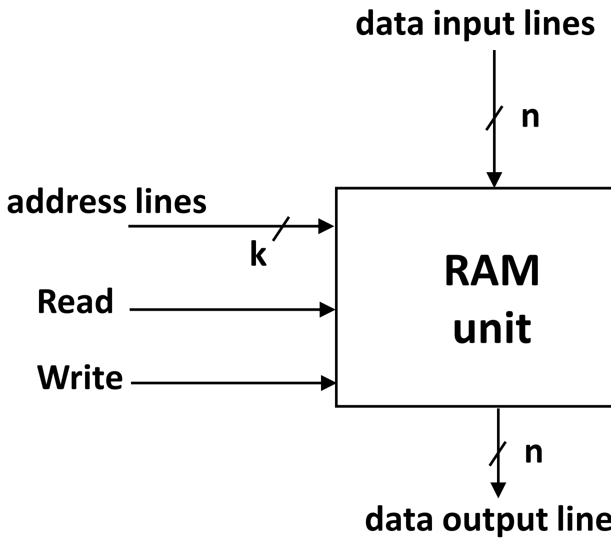
READ : $DR \leftarrow M[AR]$

WRITE : $M[AR] \leftarrow R1$

- Memory Transfer
  - Memory read : A transfer information into DR from the memory word M selected by the address in AR
  - Memory Write : A transfer information from R1 into the memory word M selected by the address in AR

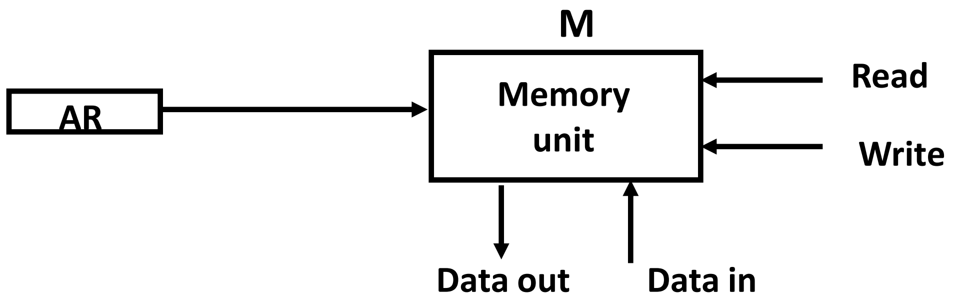
Memory - RAM

- **Memory (RAM) can be thought as a sequential circuits containing some number of registers**
- **Memory stores binary information in groups of bits called words**
- **These registers hold the *words* of memory**
- **Each of the r registers is indicated by an *address***
- **These addresses range from 0 to r-1**
- **Each register (word) can hold n bits of data**
- **Assume the RAM contains r = 2^k words. It needs the following**
  1. **n data input lines**
  2. **n data output lines**
  3. **k address lines**
  4. **A Read control line**
  5. **A Write control line**



Memory Transfer

Memory is usually accessed in computer systems by putting the desired address in a special register, the Memory Address Register (MAR, or AR)



Memory Read

- **To read a value from a location in memory and load it into a register, the register transfer language notation looks like this:**

$R1 \leftarrow M[AR]$

- **This causes the following to occur**
  1. **The contents of the MAR get sent to the memory address lines**
  2. **A Read (= 1) gets sent to the memory unit**
  3. **The contents of the specified address are put on the memory's output data lines**
  4. **These get sent over the bus to be loaded into register R1**

Memory Write

- **To write a value from a register to a location in memory looks like this in register transfer language:**

  $M[AR] \leftarrow R1$

- **This causes the following to occur**
  1. **The contents of the MAR get sent to the memory address lines**
  2. **A Write (= 1) gets sent to the memory unit**
  3. **The values in register R1 get sent over the bus to the data input lines of the memory**
  4. **The values get loaded into the specified address in the memory**

SUMMARY OF R. TRANSFER MICROOPERATIONS

$A \leftarrow B$	**1.Transfer content of reg. B into reg. A**
$AR \leftarrow DR(AD)$	**2.Transfer content of AD portion of reg. DR into reg. AR**
$A \leftarrow$ constant	**3.Transfer a binary constant into reg. A**
$ABUS \leftarrow R1, R2 \leftarrow ABUS$	**4.Transfer content of R1 into bus A and, at the same time, transfer content of bus A into R2**
AR	**5.Address register**
DR	**6.Data register**
M[R]	**7.Memory word specified by reg. R**
M	**8.Equivalent to M[AR]**
$DR \leftarrow M$	**9.Memory *read* operation: transfers content of memory word specified by AR into DR**
$M \leftarrow DR$	**10.Memory *write* operation: transfers content of DR into memory word specified by AR**

MICROOPERATIONS

Computer system microoperations are of four types:

- **Register transfer microoperations**
- **Arithmetic microoperations**
- **Logic microoperations**
- **Shift microoperations**

Arithmetic MICROOPERATIONS

- **The basic arithmetic microoperations are**
  - **Addition**
  - **Subtraction**
  - **Increment**
  - **Decrement**

- **The additional arithmetic microoperations are**
  - **Add with carry**
  - **Subtract with borrow**
  - **Transfer/Load**
  - **etc. …**

Summary of Typical Arithmetic Micro-Operations

$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow R2'$	Complement the contents of R2
$R2 \leftarrow R2' + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + R2' + 1$	subtraction
$R1 \leftarrow R1 + 1$	Increment
$R1 \leftarrow R1 - 1$	Decrement

Binary Adder

- 4-bit Binary Adder : ***Fig. 4-6***
  - Full adder = 2-bits sum + previous carry
  - Binary adder = the arithmetic sum of two binary numbers of any length
  - c_0(input carry), c_4(output carry)

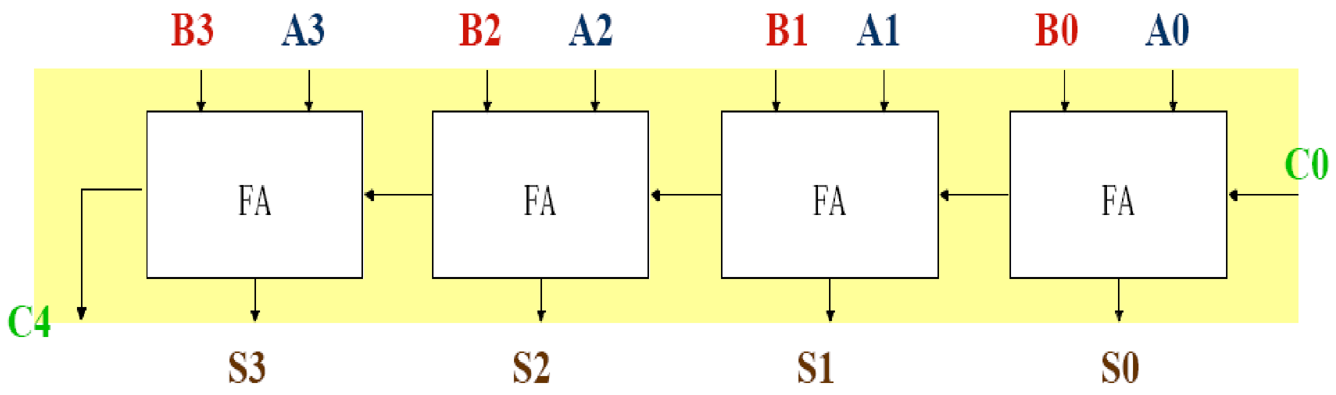
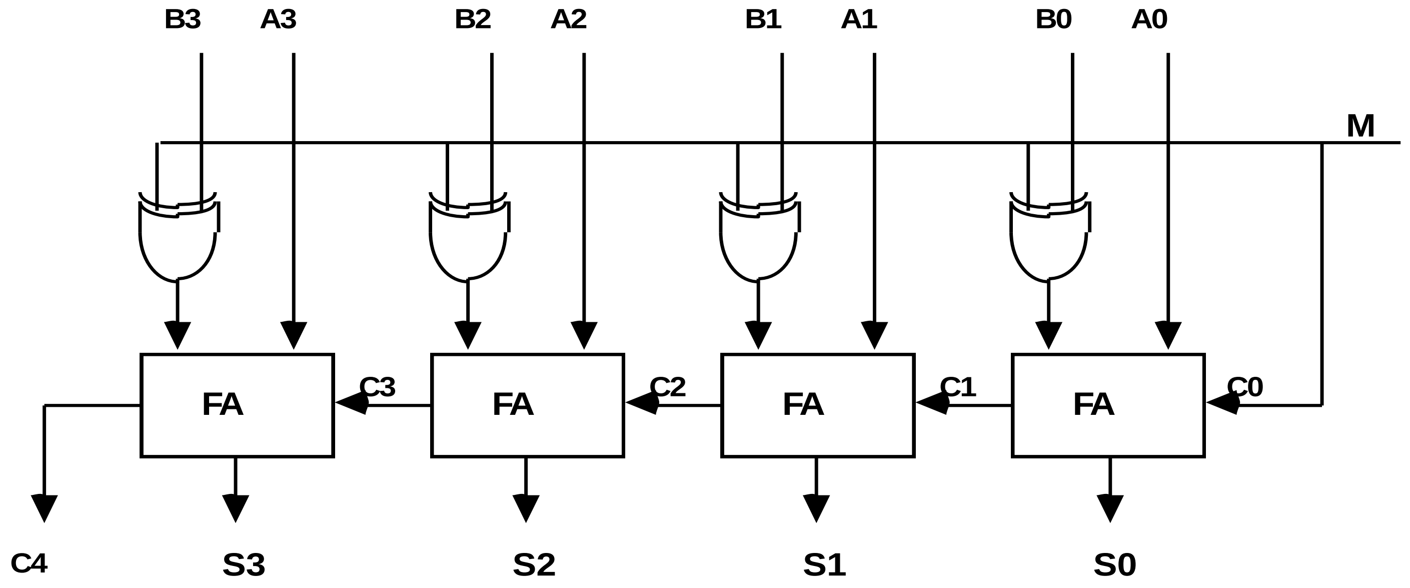


Figure 4-6. 4-bit binary adder

Binary Adder-Subtractor

Binary Adder-Subtractor

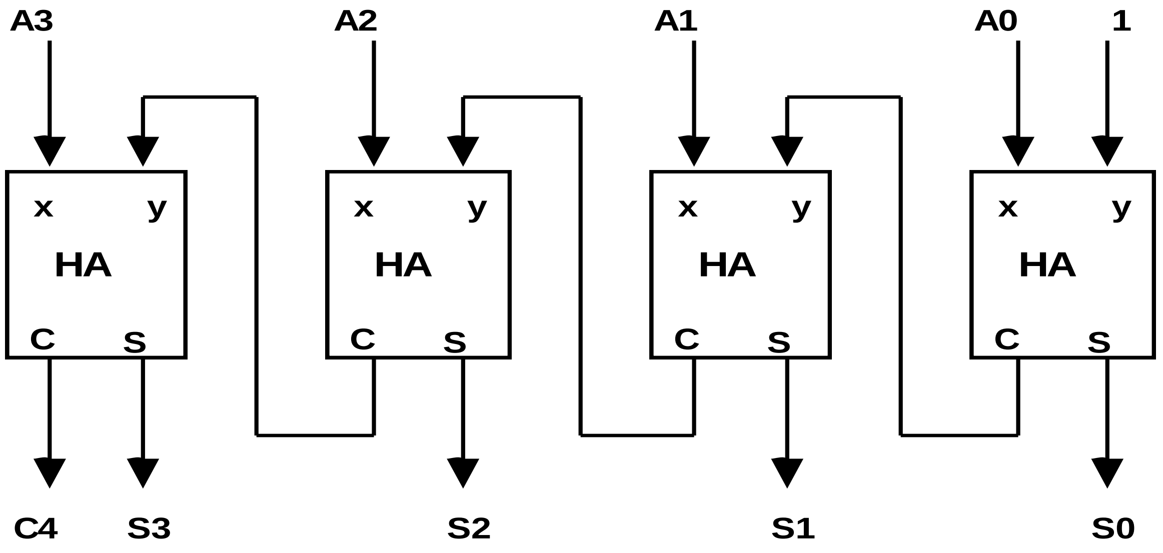


- Mode input M controls the operation
  - M=0 ---- adder
  - M=1 ---- subtractor

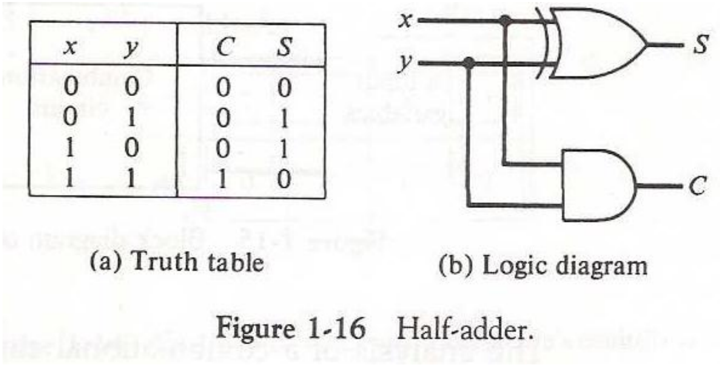
NOTE:
- B xor 0 = B
- B xor 1 = B'

Binary Incrementer

Binary Incrementer



Assume example of 0110 as input and Output must be 0111

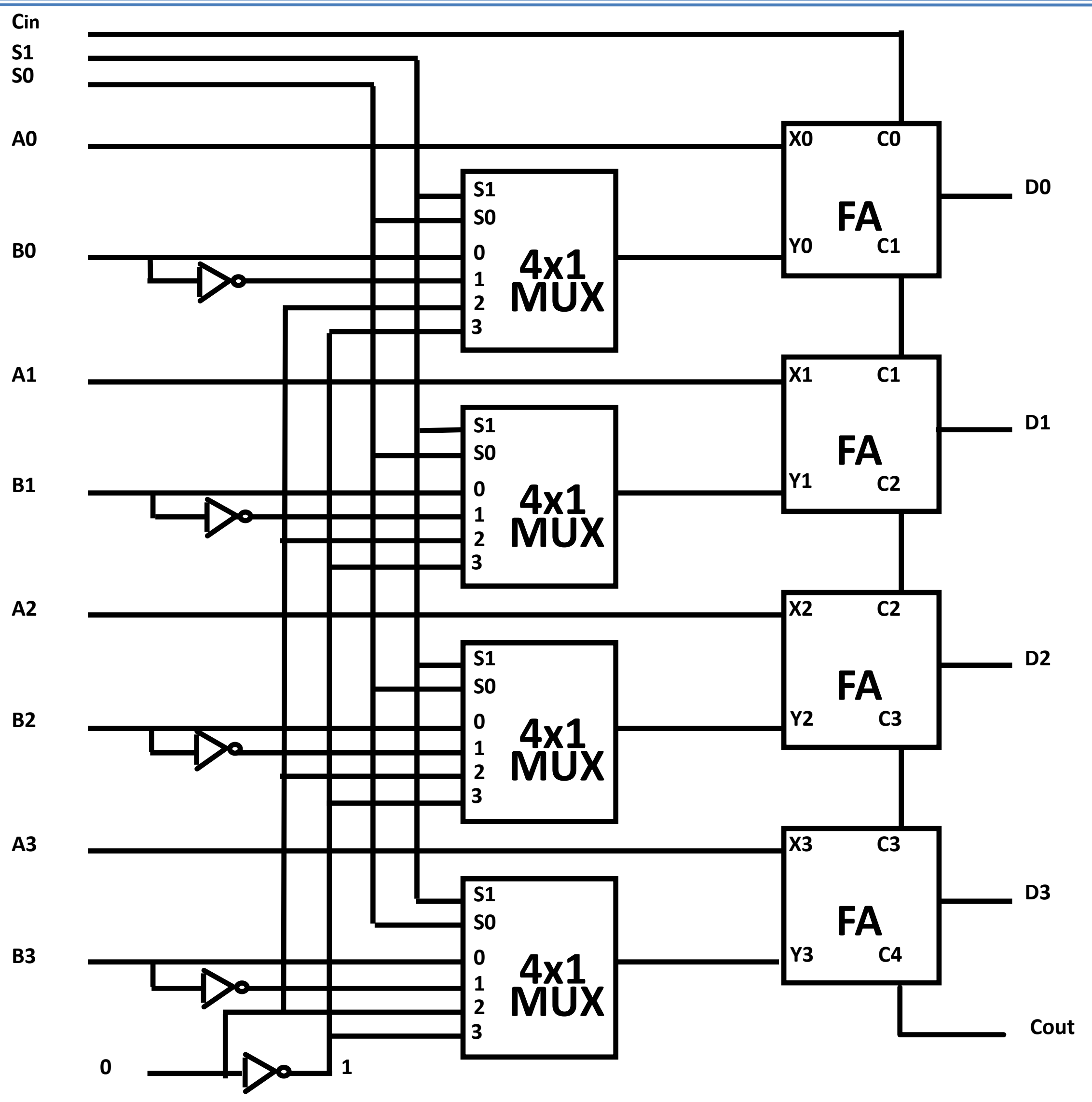


x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table (b) Logic diagram

Figure 1-16 Half-adder.

Arithmetic Circuits



Select		Input		Output
S_1	S_0	C_{in}	Y	$D=A+Y+C_{in}$
0	0	0	B	D=A+B
0	0	1	B	D=A+B+1
0	1	0	B'	D=A+B'
0	1	1	B'	D=A+B'+1
1	0	0	0	D=A
1	0	1	0	D=A+1
1	1	0	1	D=A-1
1	1	1	1	D=A

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

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

  - Special Symbols
    - » 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

Arithmetic ADD

- List of Logic Microoperation
  - Truth Table for 16 functions for 2 variables : ***Tab. 4-5***
  - 16 Logic Microoperation : ***Tab. 4-6***

∵ All other Operation can be derived

- Hardware Implementation
  - 16 microoperation → Use only 4(AND, OR, XOR, Complement)
  - One stage of logic circuit

Logic Microoperations

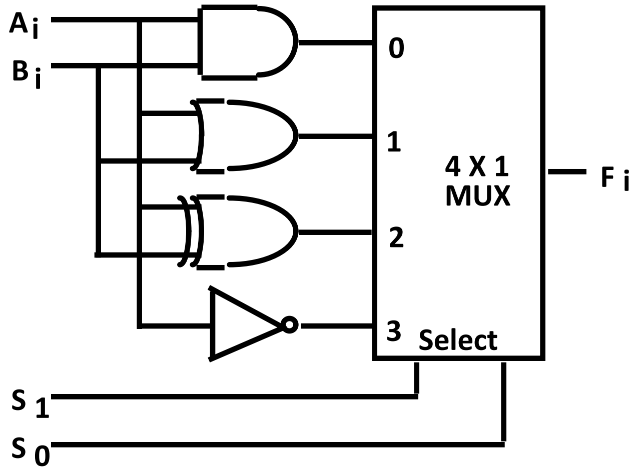
X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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 ∧ B	AND
0	1	F = A ∨ B	OR
1	0	F = A ⊕ 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 ← A · 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 ← A · 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

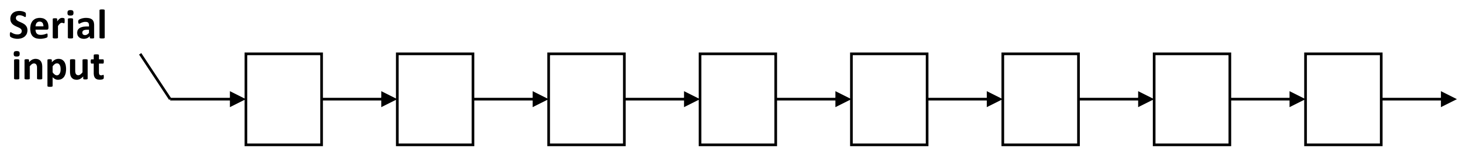
   - **A mask operation to clear the desired bit positions, followed by**
   - **An OR operation to introduce the new bits into the desired positions**
   - **Example**
     - **Suppose you wanted to introduce 1010 into the low order four bits of A:**
     - **1101 1000 1011 0001 A (Original)**
       1101 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)
       ```

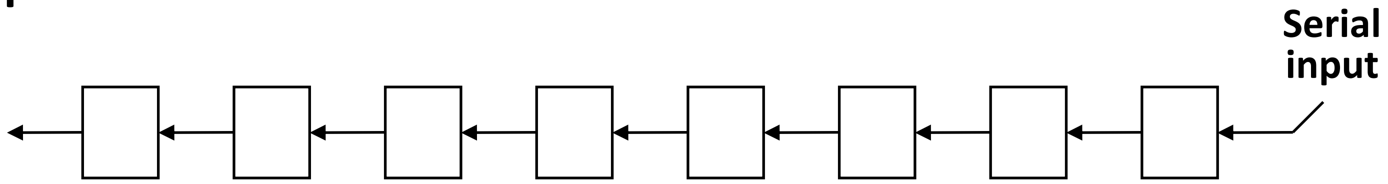
Shift Microoperations

- **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**

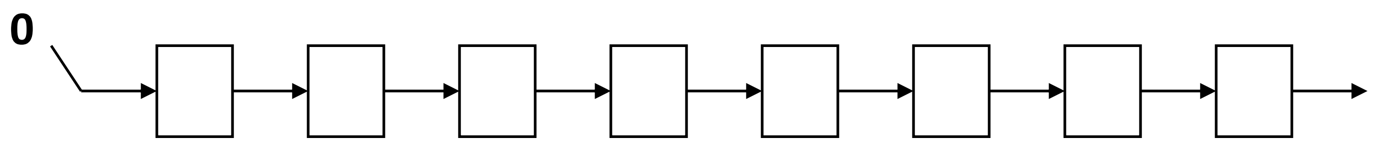


- **A left shift operation**

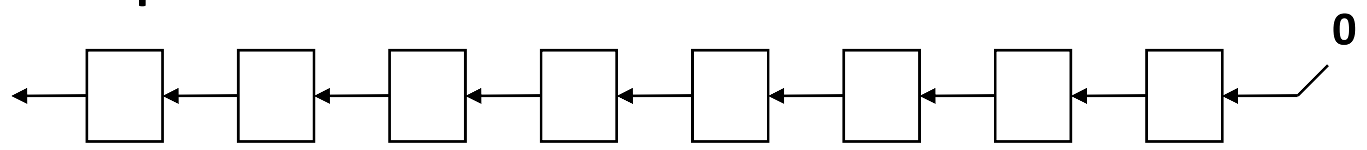


Logical Shift

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



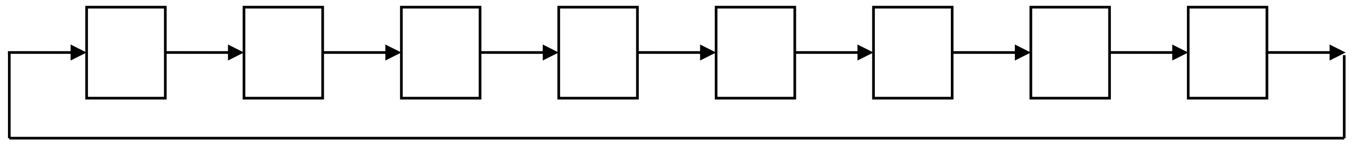
- **A left logical shift operation:**



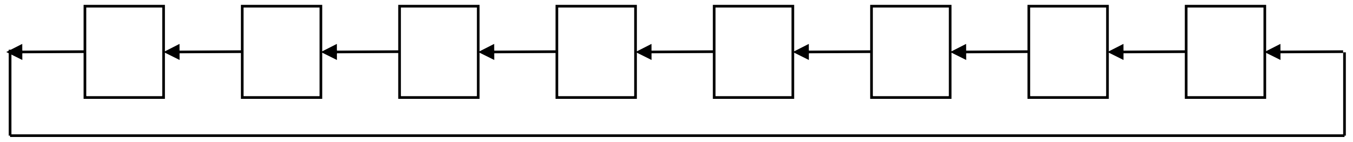
- **In a Register Transfer Language, the following notation is used**
  - ***shl*** **for a logical shift left**
  - ***shr*** **for a logical shift right**
  - **Examples:**
    - **R2 $\leftarrow$ *shr* R2**
    - **R3 $\leftarrow$ *shl* R3**

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:**

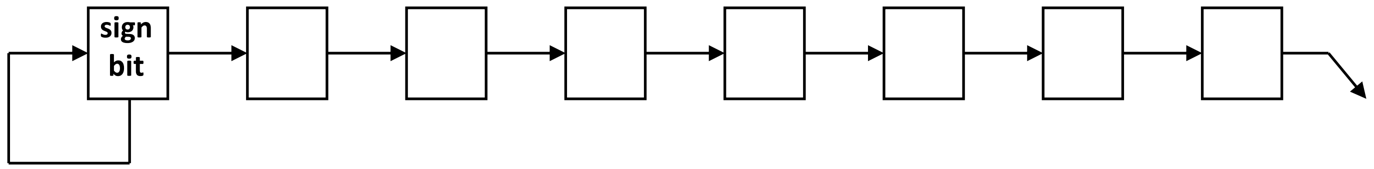


- **In a RTL, the following notation is used**
  - ***cil*** **for a circular shift left**
  - ***cir*** **for a circular shift right**
  - **Examples:**
    - **R2 ← *cir* R2**
    - **R3 ← *cil* R3**

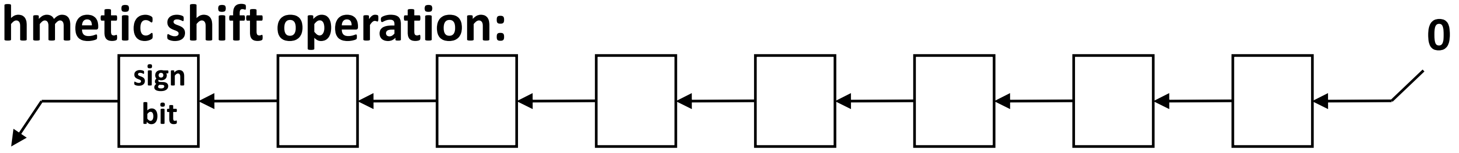
Arithmetic Shift

- **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:**

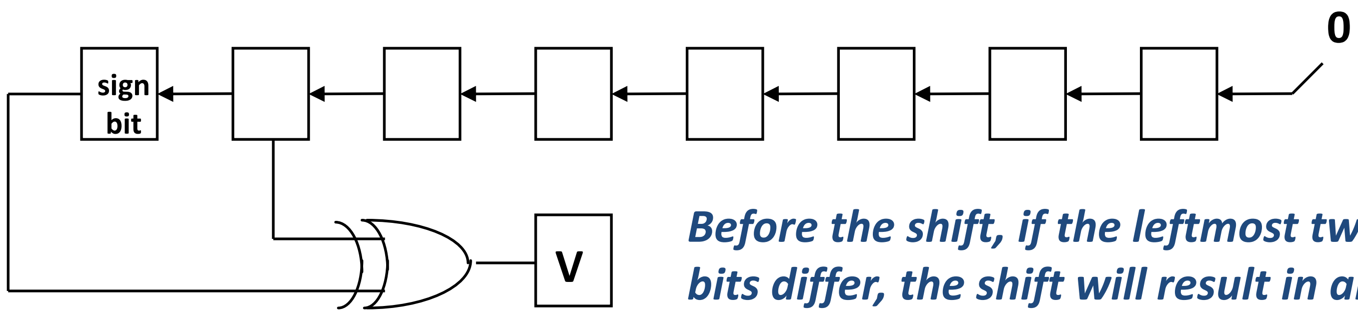


- **A left arithmetic shift operation:**



Arithmetic Shift

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

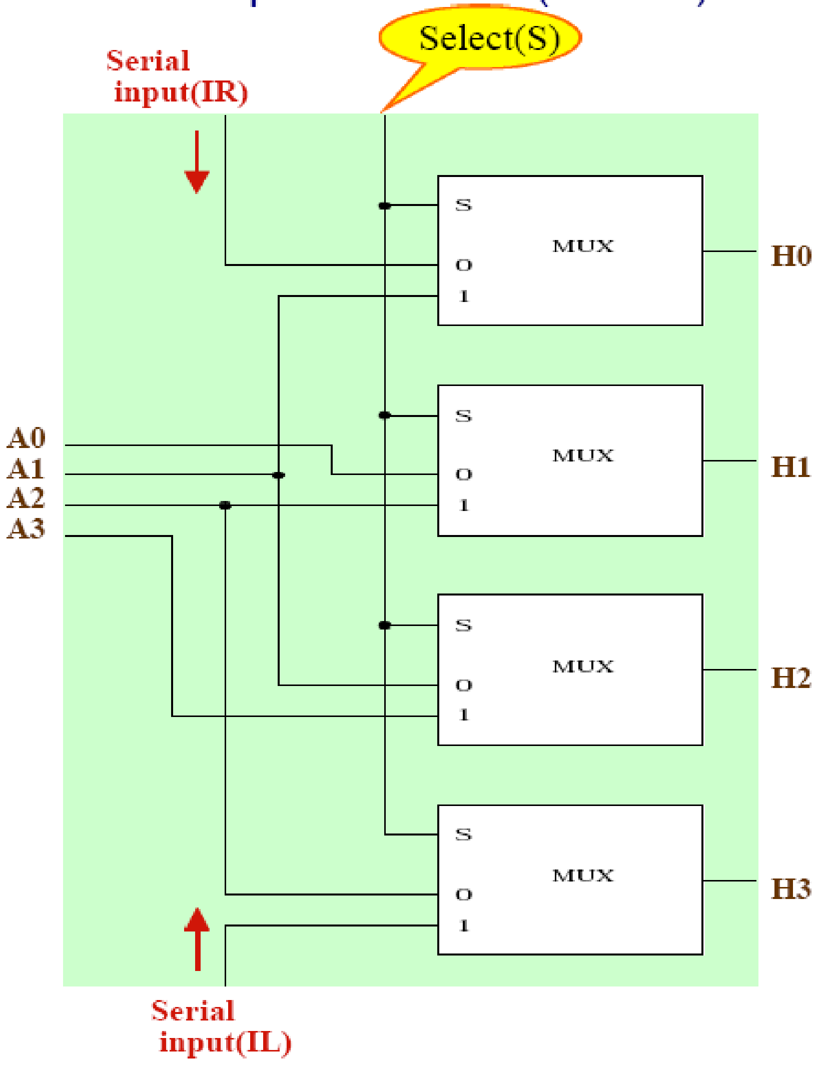


Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- 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

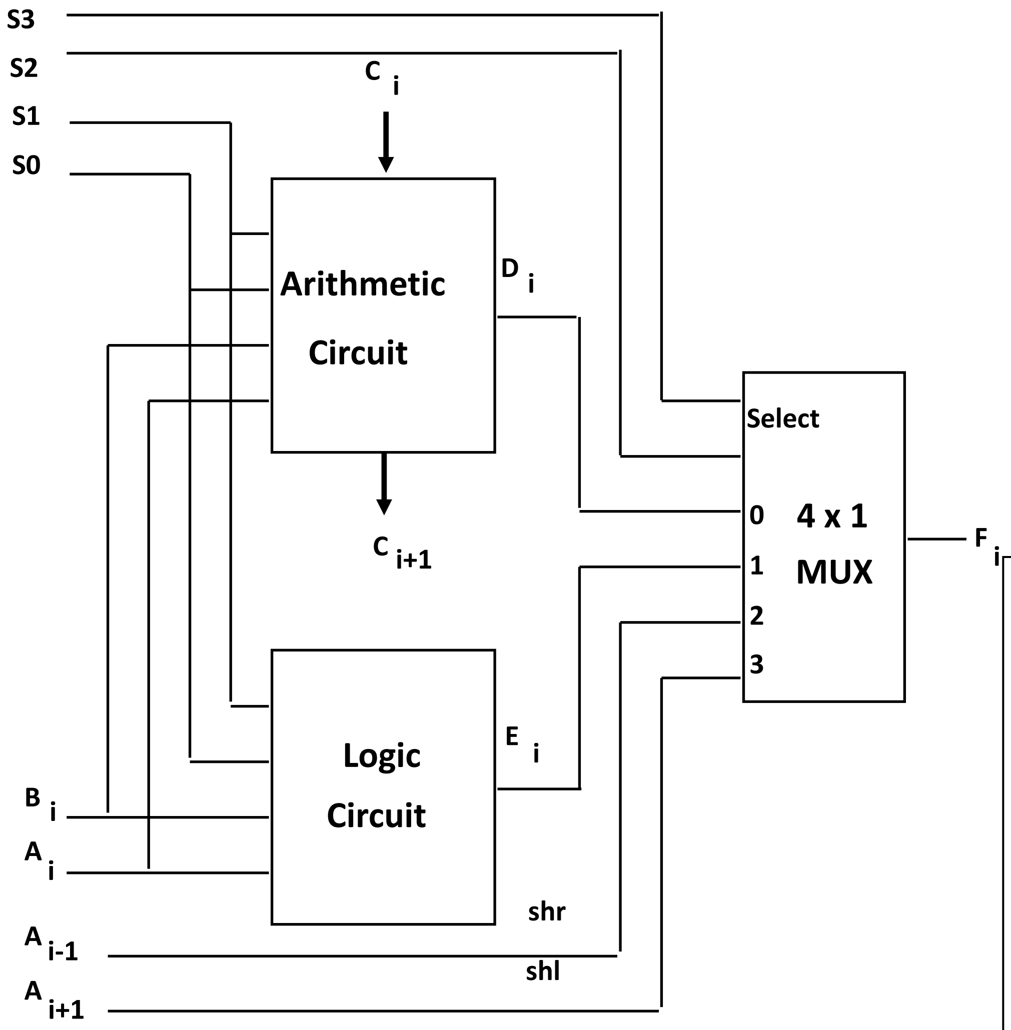
- Hardware Implementation(Shifter) :



Function Table

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

Arithmetic Logic and Shift Unit



S3	S2	S1	S0	Cin	Operation
0	0	0	0	0	F = A
0	0	0	0	1	F = A + 1
0	0	0	1	0	F = A + B
0	0	0	1	1	F = A + B + 1
0	0	1	0	0	F = A + B’
0	0	1	0	1	F = A + B’+ 1
0	0	1	1	0	F = A - 1
0	0	1	1	1	F = A
0	1	0	0	X	F = A $\wedge$ B
0	1	0	1	X	F = A $\vee$ B
0	1	1	0	X	F = A $\oplus$ B
0	1	1	1	X	F = A’
1	0	X	X	X	F = shr A
1	1	X	X	X	F = shl A