Computer system architecture Chapt 4. Register transfer & Microoperations

KyuheonKim@khu.ac.kr

Rm: 416

- **Register Transfer Language**

- **Register Transfer**

- **Bus and Memory Transfers**

- **Arithmetic Microoperations**

- **Logic Microoperations**

- **Shift Microoperations**

- **Arithmetic Logic Shift Unit**

❖ **Combinational and sequential circuits (learned in Chapters 1 and 2) 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 they perform.**
   ◆ **the control that initiates the sequence of microoperations**

❖ **Typically,**
   ◆ **What operations are performed on the data in the registers**
   ◆ **What information is passed between registers**

MICROOPERATIONS (1)

❖ **The operations on the data in registers are called microoperations.**

❖ **The functions built into registers are examples of microoperations**

   ◆ **Shift**

   ◆ **Load**

   ◆ **Clear**

   ◆ **Increment**

   ◆ **...**

MICROOPERATION (2)

An elementary operation performed (during one clock pulse), on the information stored in one or more registers

Registers (R)   ALU (f)   1 clock cycle
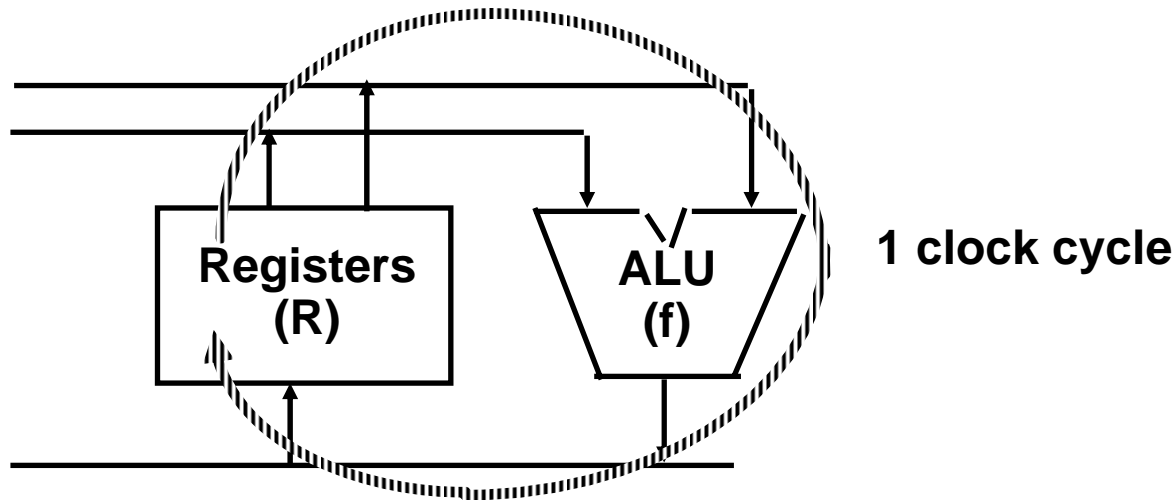
R ← f(R, R)

f: shift, load, clear, increment, add, subtract, complement, and, or, xor, …

REGISTER TRANSFER LANGUAGE

❖ **Rather than specifying a digital system in words, a specific notation is used,** *register transfer language*

❖ **For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations**

❖ **Register transfer language**

   ◆ **A symbolic language**

   ◆ **A convenient tool for describing the internal organization of digital computers**

   ◆ **Can also be used to facilitate the design process of digital systems.**

DESIGNATION OF REGISTERS

❖ **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***

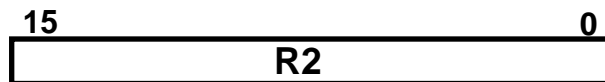◆ **A register can be viewed as a single entity:**

MAR

◆ **Registers may also be represented showing the bits of data they contain**

DESIGNATION OF REGISTERS

- **Designation of a register**
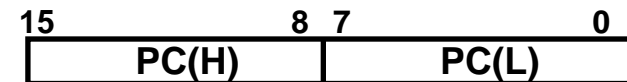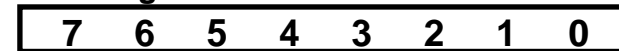 - **- a register**
  - **- portion of a register**
  - **- a bit of a register**

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

Register

R1

15   0
R2

Numbering of bits

Showing individual bits

7	6	5	4	3	2	1	0

15   8 7   0
PC(H)	PC(L)

Subfields

REGISTER TRANSFER

❖ **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 R2 are copied (loaded) from register R1**

◆ **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

❖ **A register transfer such as**

R3 ← 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
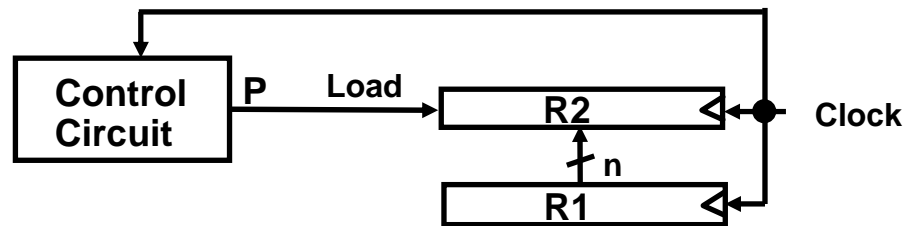
Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

HARDWARE IMPLEMENTATION OF CONTROLLED TRANSFERS

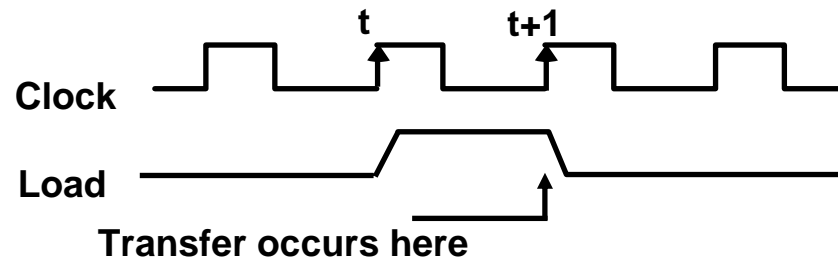**Implementation of controlled transfer**

$$P: \ R2 \leftarrow R1$$

Block diagram

Control Circuit — P — Load → R2 ← Clock

n

R1

Timing diagram

Clock

Load

t ... $t+1$

Transfer occurs here

- **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**

SIMULTANEOUS OPERATIONS

❖ **If two or more operations are to occur simultaneously, they are separated with commas**

$$P: R3 \leftarrow R5, MAR \leftarrow IR$$

❖ **Here, if the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR**

BASIC SYMBOLS FOR REGISTER TRANSFERS

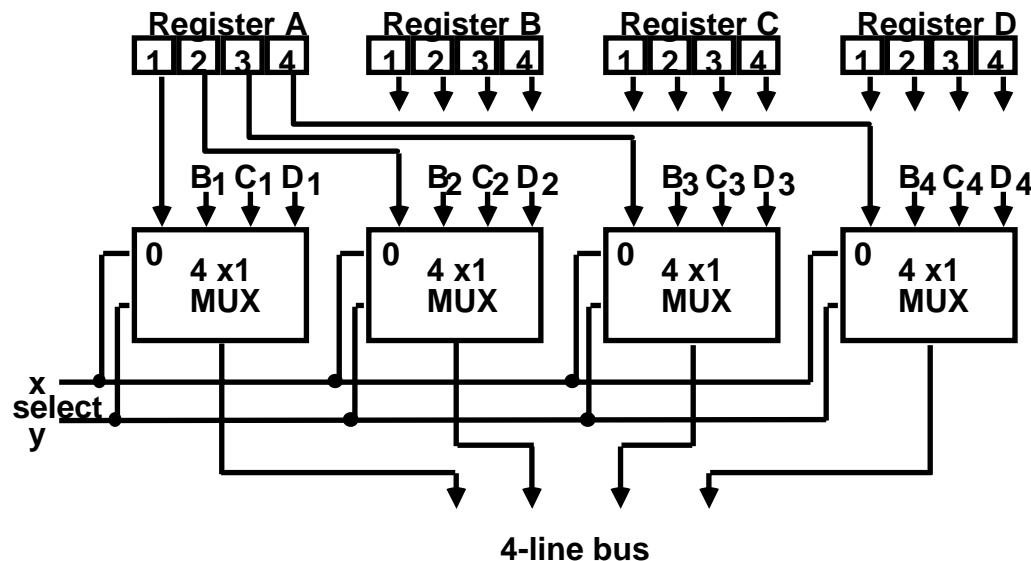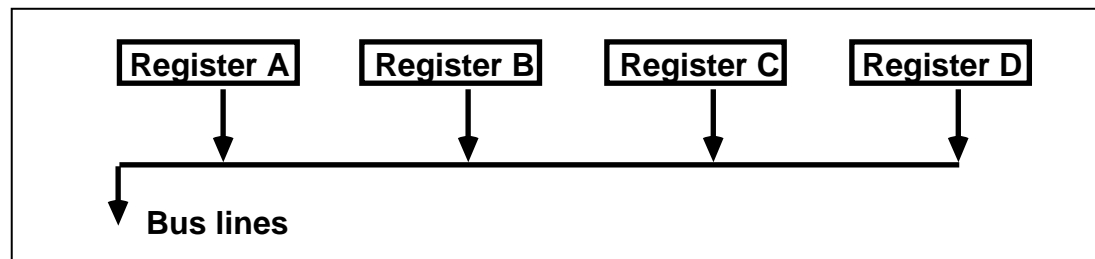| Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A ← B, B ← A

CONNECTING REGISTRS

❖ **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**

❖ **Have control circuits to select which register is the source, and which is the destination**
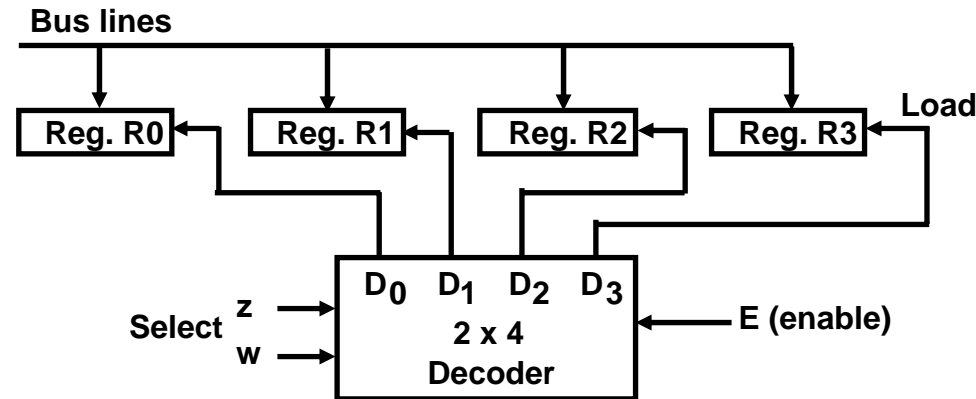
BUS AND BUS TRANSFER

Bus is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.
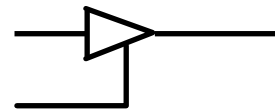
From a register to bus: BUS ← R

Register A   Register B   Register C   Register D

Bus lines

Register A   Register B   Register C   Register D
1 2 3 4   1 2 3 4   1 2 3 4   1 2 3 4

$B_1 C_1 D_1$ $B_2 C_2 D_2$ $B_3 C_3 D_3$ $B_4 C_4 D_4$

0 4 x1 MUX   0 4 x1 MUX   0 4 x1 MUX   0 4 x1 MUX

x
select
y

4-line bus

KyungHee Univ.
Media Lab.

TRANSFER FROM BUS TO A DESTINATION REGISTER

Bus lines

| Reg. R0 | Reg. R1 | Reg. R2 | Reg. R3 | **Load** |

D_0 D_1 D_2 D_3

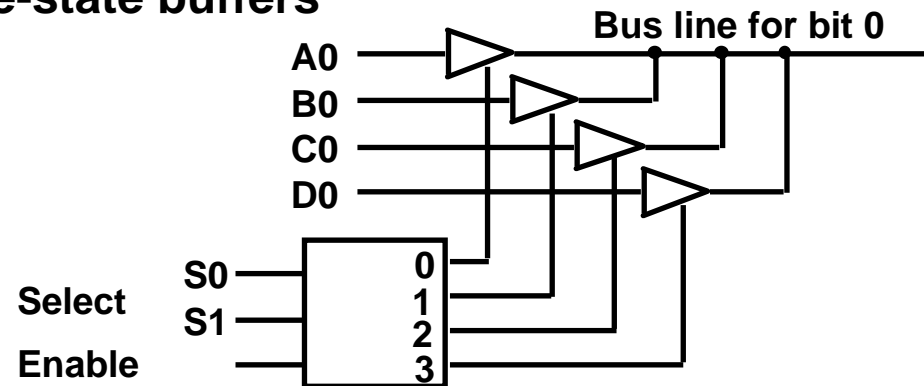**Select** **z** →
w →
2 x 4
Decoder

← **E (enable)**

Three-State Bus Buffers

Normal input A

Control input C

Output Y=A if C=1
High-impedence if C=0

Bus line with three-state buffers

Bus line for bit 0

A0
B0
C0
D0

Select S0
S1
Enable

0
1
2
3

BUS TRANSFER IN RTL

❖ **Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either**

$$R2 \leftarrow R1$$

or

$$BUS \leftarrow R1, R2 \leftarrow BUS$$

❖ **In the former case the bus is implicit, but in the latter, it is explicitly indicated**

KyungHee Univ.
Media Lab.

MEMORY (RAM)

❖ **Memory (RAM) can be thought as a sequential circuits containing some number of registers**

❖ **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 = 2k words. It needs the following**

 ◆ **n data input lines**

 ◆ **n data output lines**

 ◆ **k address lines**

 ◆ **A Read control line**

 ◆ **A Write control line**

data input lines

n

address lines

k

Read

Write

RAM unit

n

data output lines

KyungHee Univ.
Media Lab.
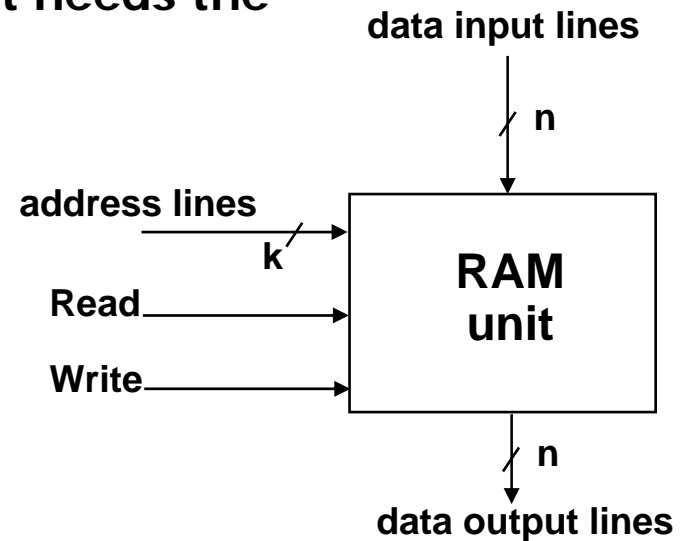
MEMORY TRANSFER

❖ **Collectively, the memory is viewed at the register level as a device, M.**

❖ **Since it contains multiple locations, we must specify which address in memory we will be using**

❖ **This is done by indexing memory references**

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

❖ **When memory is accessed, the contents of the MAR get sent to the memory unit's address lines**

M

AR → Memory unit ← Read
          ← Write

Data out   Data in
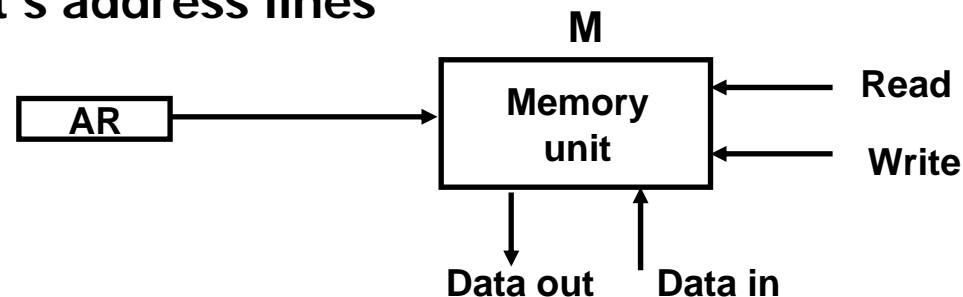
MEMORY READ

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

$$R1 \leftarrow M[MAR]$$

❖ **This causes the following to occur**

◆ **The contents of the MAR get sent to the memory address lines**

◆ **A Read (= 1) gets sent to the memory unit**

◆ **The contents of the specified address are put on the memory's output data lines**

◆ **These get sent over the bus to be loaded into register R1**

KyungHee Univ.
Media Lab.

MEMORY WRITE

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

$$M[MAR] \leftarrow R1$$

❖ **This causes the following to occur**

◆ **The contents of the MAR get sent to the memory address lines**

◆ **A Write (= 1) gets sent to the memory unit**

◆ **The values in register R1 get sent over the bus to the data input lines of the memory**

◆ **The values get loaded into the specified address in the memory**

KyungHee Univ.
Media Lab.

SUMMARY OF R. TRANSFER MICROOPERATIONS

A ← B	Transfer content of reg. B into reg. A
AR ← DR(AD)	Transfer content of AD portion of reg. DR into reg. AR
A ← constant	Transfer a binary constant into reg. A
ABUS ← R1,	Transfer content of R1 into bus A and, at the same time,
R2 ← ABUS	transfer content of bus A into R2
AR	Address register
DR	Data register
M[R]	Memory word specified by reg. R
M	Equivalent to M[AR]
DR ← M	Memory *read* operation: transfers content of memory word specified by AR into DR
M ← DR	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 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2'	Complement the contents of R2
R2 ← R2'+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2'+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

BINARY ADDER / SUBTRACTOR / INCREMENTER

Binary Adder
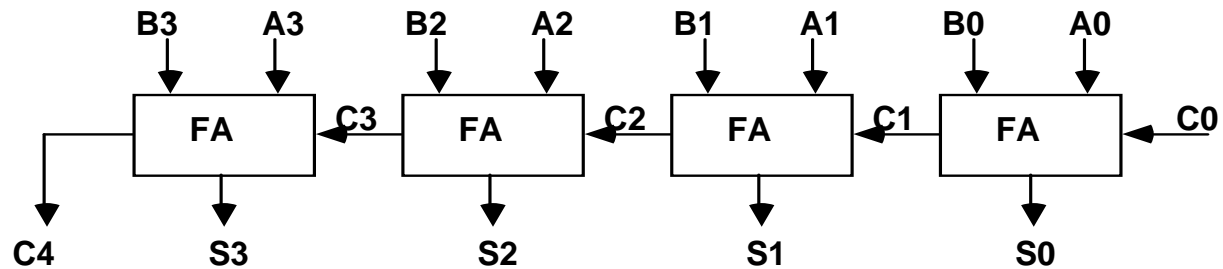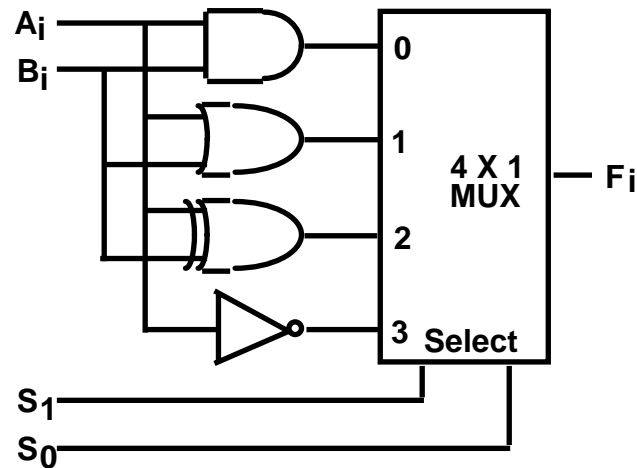

Binary Adder-Subtractor
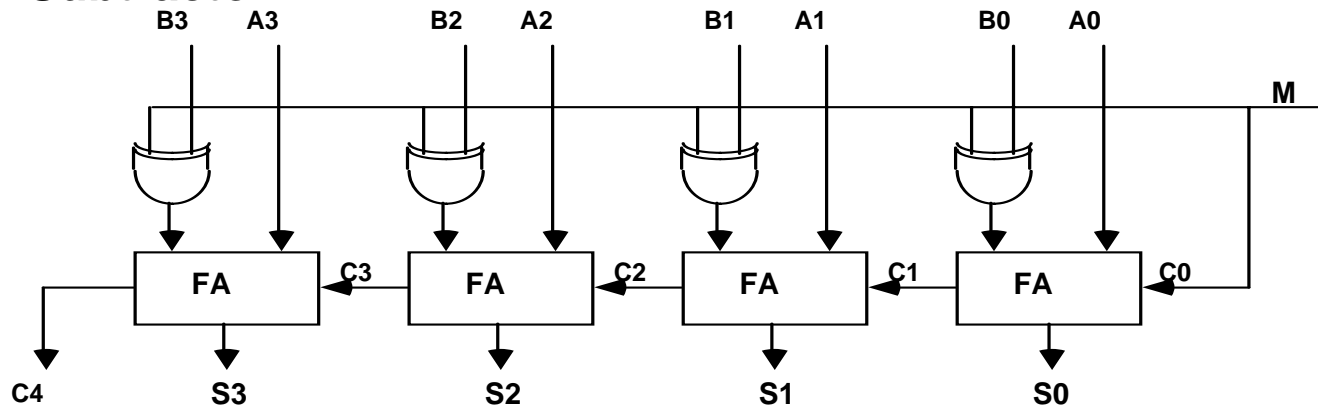
Binary Incrementer

KyungHee Univ.
Media Lab.
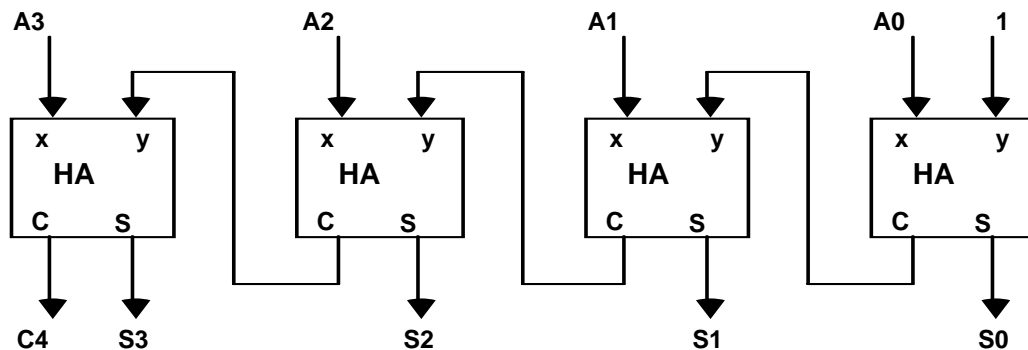
ARITHMETIC CIRCUIT
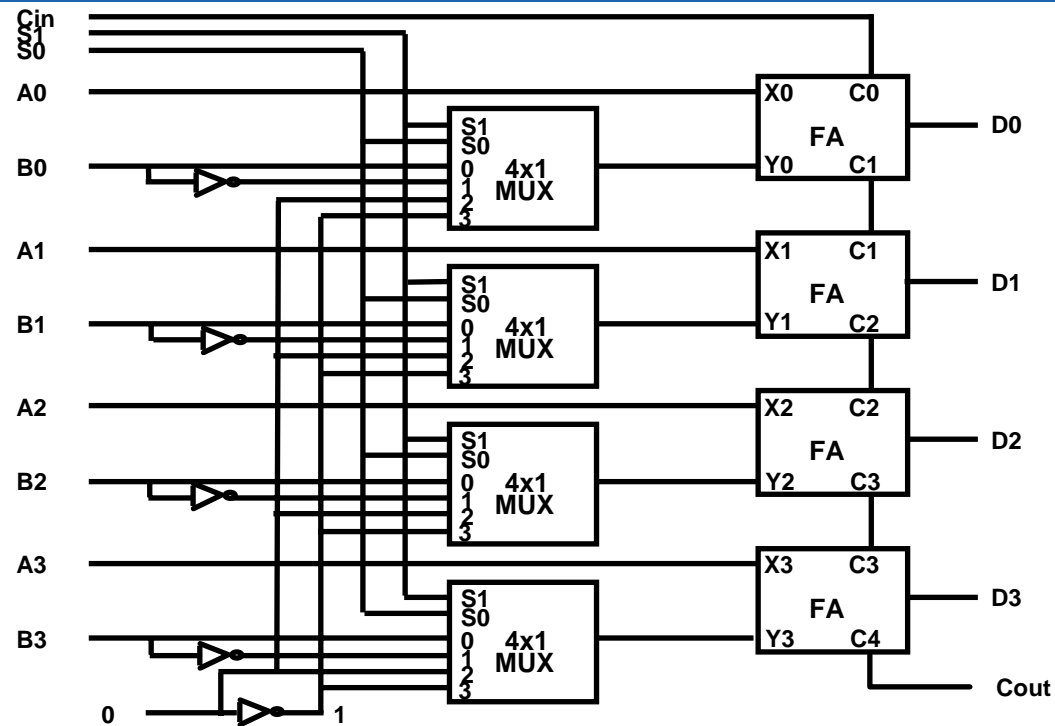
S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B'	D = A + B'	Subtract with borrow
0	1	1	B'	D = A + B'+ 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A - 1	Decrement A
1	1	1	1	D = A	Transfer A

LOGIC MICROOPERATIONS

Function table

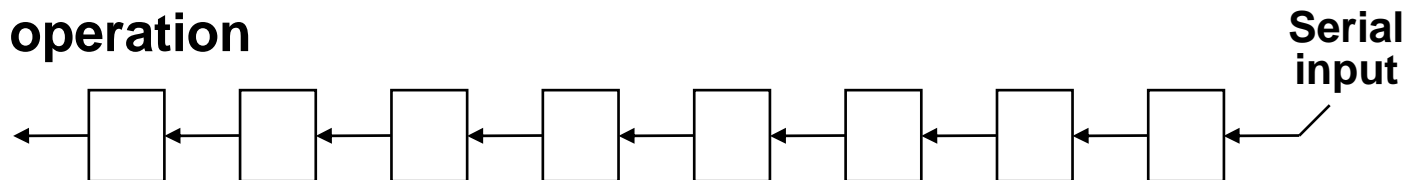| S_1 S_0	Output	μ-operation
0 0	$F = A \wedge B$	AND
0 1	$F = A \vee B$	OR
1 0	$F = A \oplus B$	XOR
1 1	$F = A'$	Complement

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**
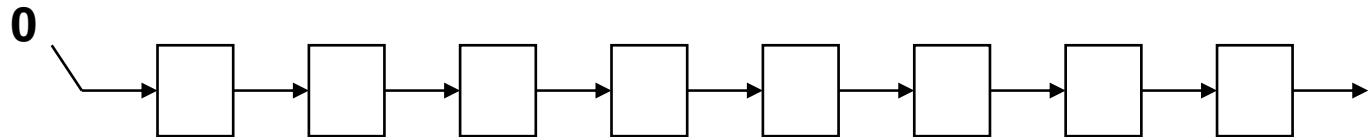
• A right shift operation
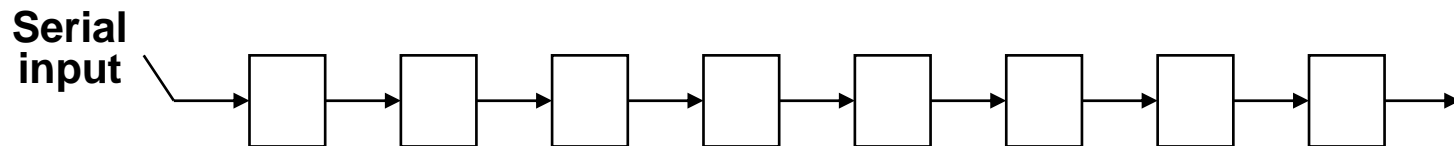
Serial input
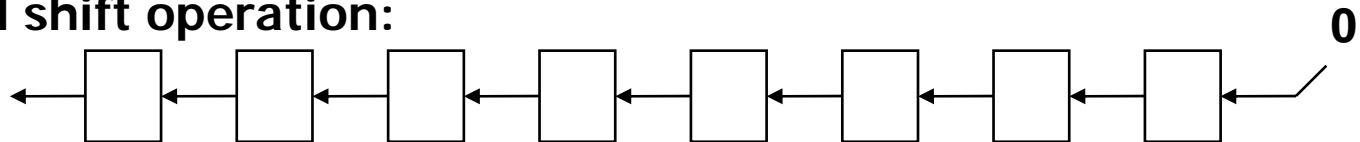
• A left shift operation

Serial input

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**

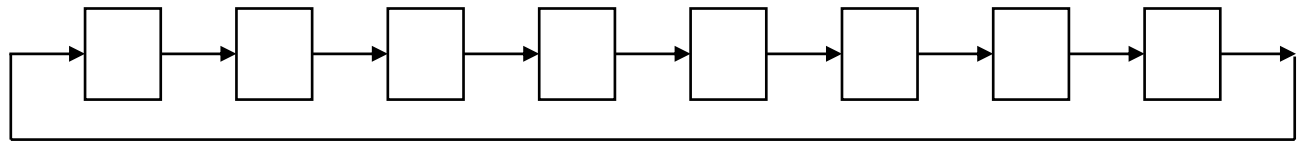◆ *shr*   **for a logical shift right**

◆ **Examples:**

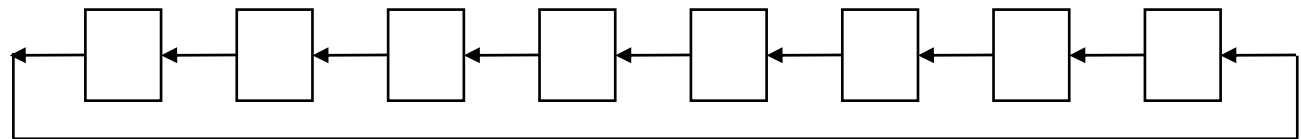 ✓ **R2 ← *shr* R2**

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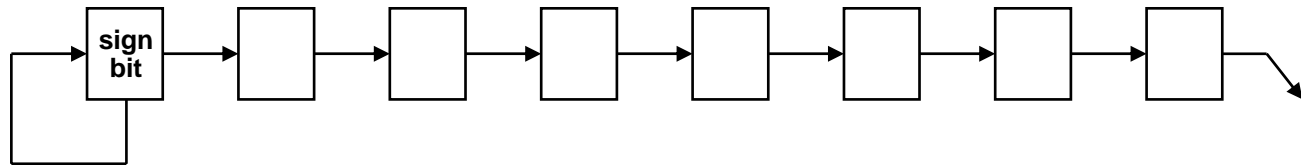
❖ **In a RTL, the following notation is used**
   ◆ *cil*   **for a circular shift left**
   ◆ *cir*   **for a circular shift right**
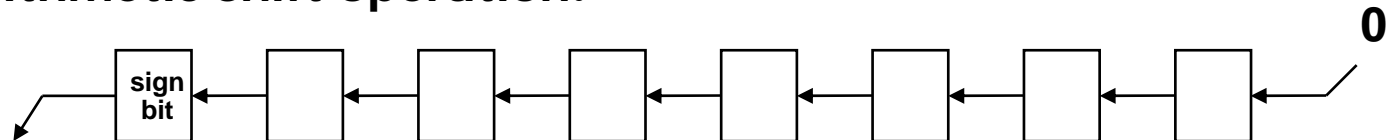   ◆ **Examples:**
      ✓ **R2 ← *cir* R2**
      ✓ **R3 ← *cil* R3**

KyungHee Univ.
Media Lab.

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**
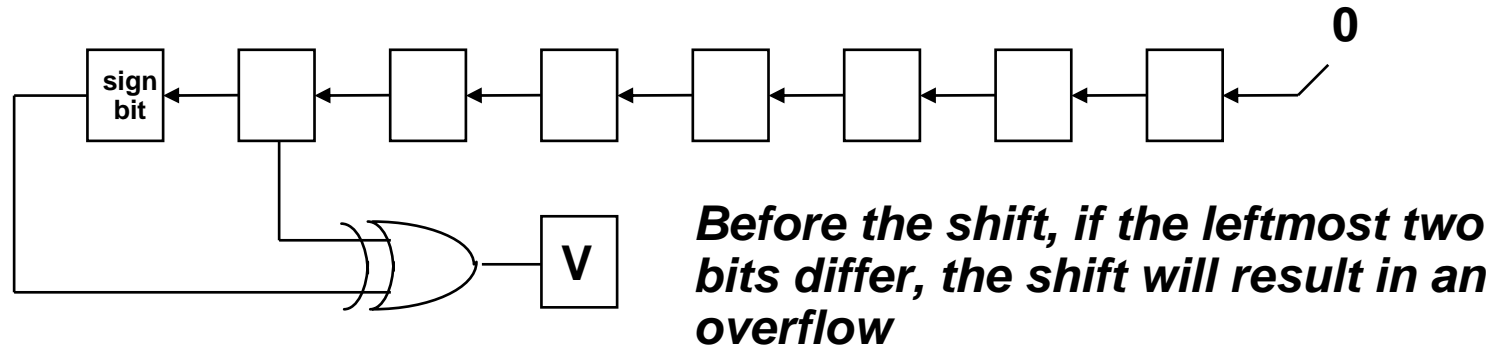
❖ **A right arithmetic shift operation:**

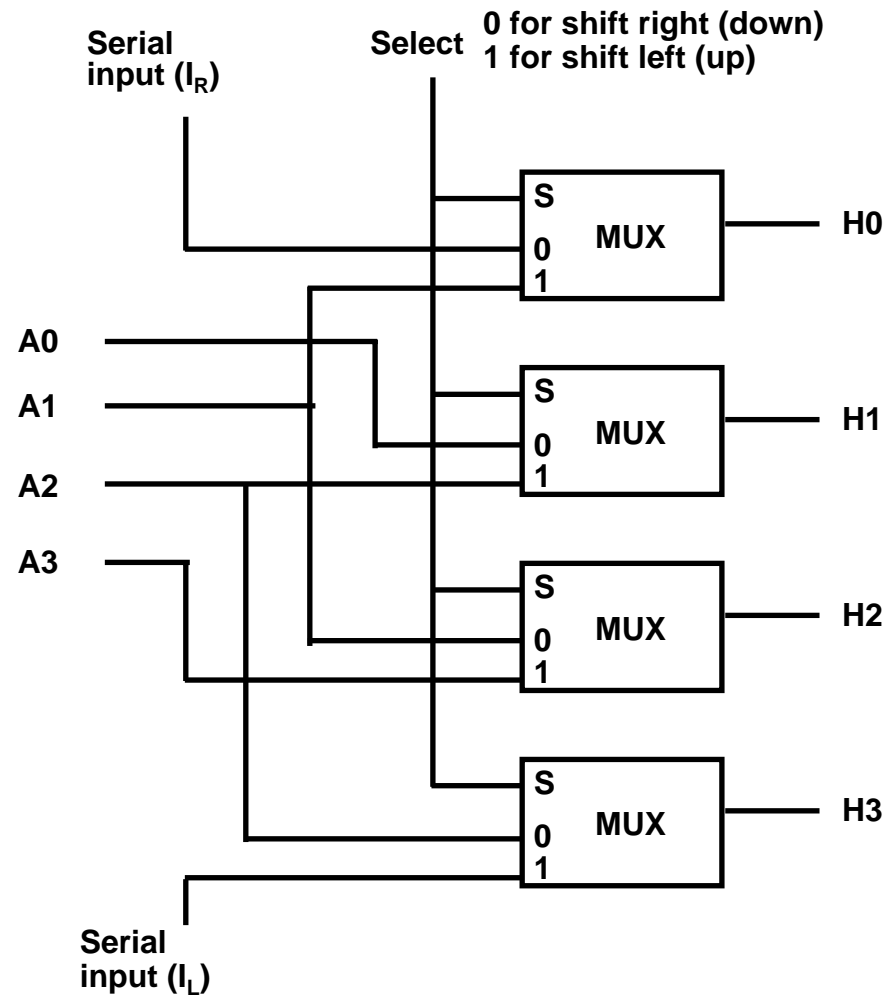❖ **A left arithmetic shift operation:**

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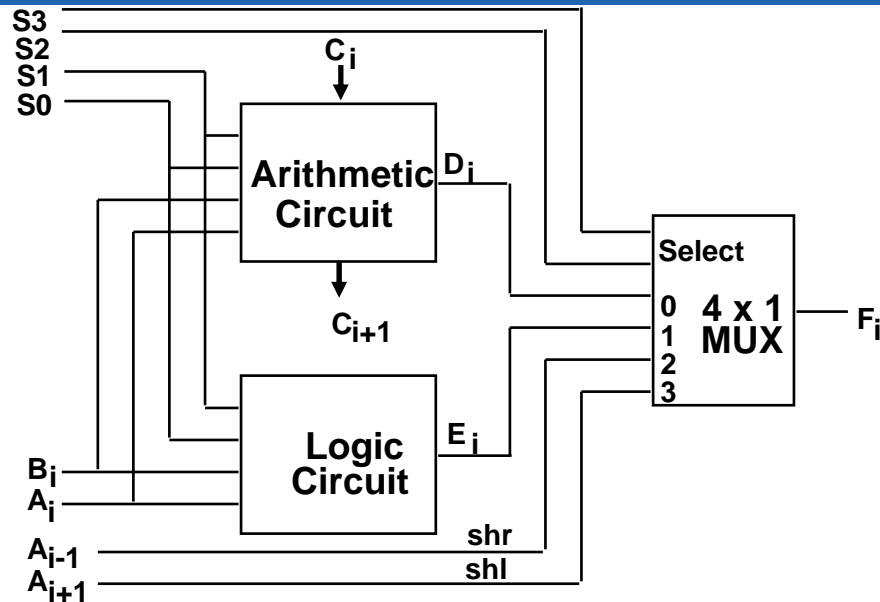
❖ **In a RTL, the following notation is used**
  ◆ *ashl*   for an arithmetic shift left
  ◆ *ashr*   for an arithmetic shift right
  ◆ Examples:
    ✓ R2 ← *ashr* R2
    ✓ R3 ← *ashl* R3

KyungHee Univ.
Media Lab.

HARDWARE IMPLEMENTATION OF SHIFT MICROOPERATIONS

KyungHee Univ.
Media Lab.

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