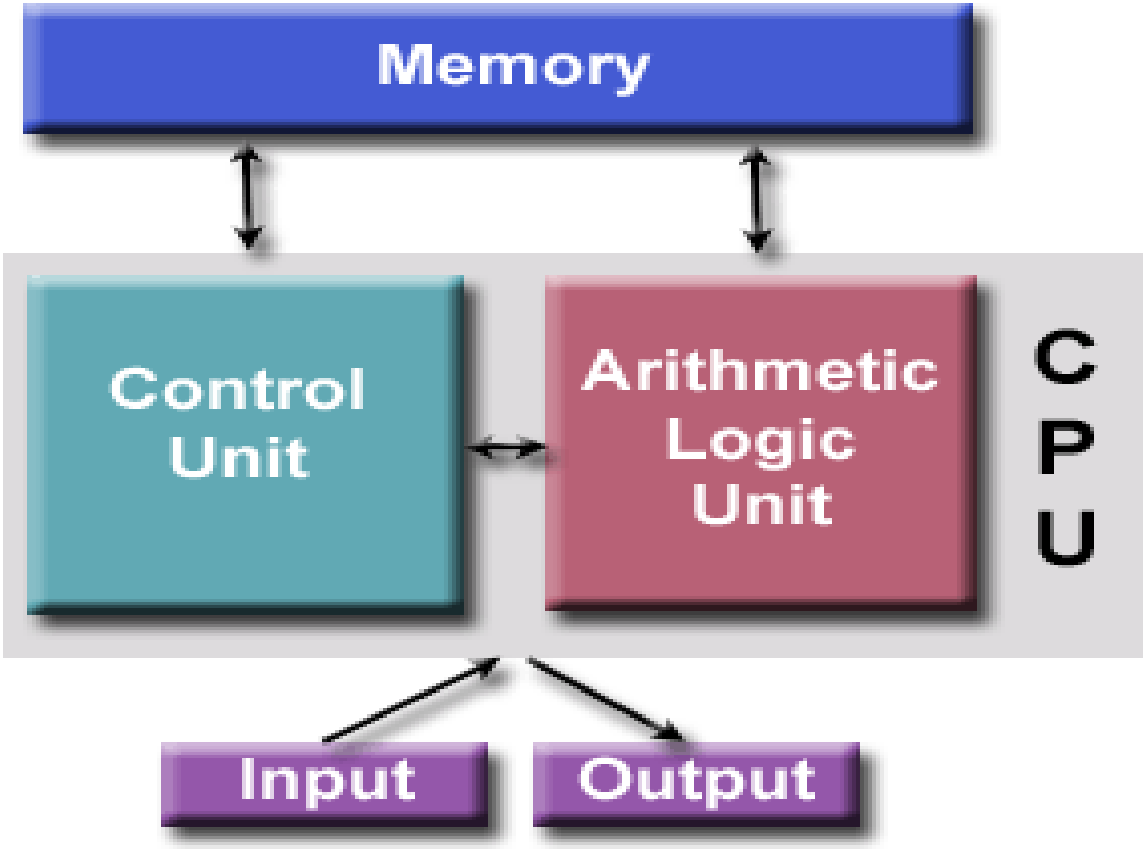
UNIT – I

REGISTER TRANSFERANDMICROOPERATIONS

Components of Computer System

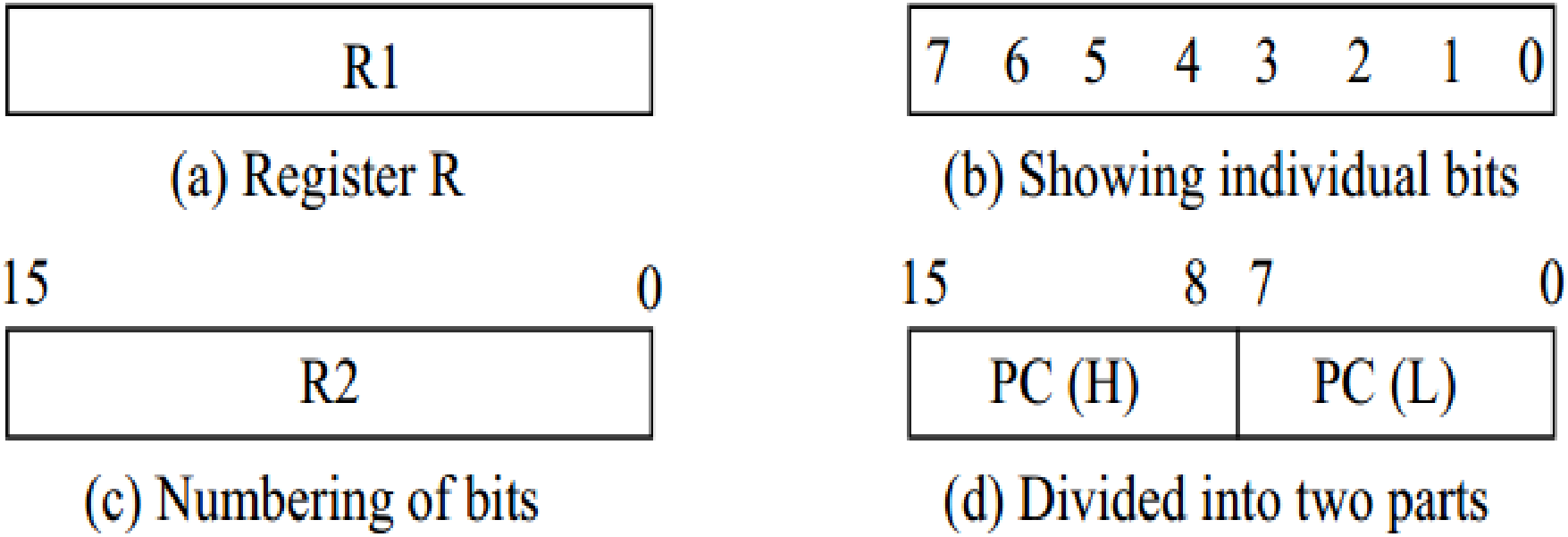


TOPICS

1. **Register Transfer Language**
2. **Register Transfer**
3. **Bus and Memory Transfers**
4. **Arithmetic Microoperations**
5. **Logic Microoperations**
6. **Shift Microoperations**
7. **Arithmetic Logic Shift Unit**

About Register Representation

- Register is a storage device which store one or more bits



(a) Register R

(b) Showing individual bits

(c) Numbering of bits

(d) Divided into two parts

Definition

- BUS: In computer architecture, a bus is a communication system that transfers data between components inside a computer, or between computers.
- Digital System: An interconnection of hardware modules that do a certain task on the information.
- Registers + Operations performed on the data stored in them = Digital Module
- Modules are interconnected with common data and control paths to form a digital computer system

1. Register Transfer Language

- Microoperations: operations executed on data stored in one or more registers.
- For any function of the computer, a sequence of microoperations is used to describe it
- The result of the operation may be:
  - replace the previous binary information of a register or
  - transferred to another register

1.Register Transfer Language cont.

- The internal hardware organization of a digital computer is defined by specifying:
  - The set of registers it contains and their function
  - The sequence of microoperations performed on the binary information stored in the registers
  - The control that initiates the sequence of microoperations
- Registers + Microoperations Hardware + Control Functions = Digital Computer

1 Register Transfer Language cont.

- Register Transfer Language (RTL) : a symbolic notation to describe the micro operation transfers among registers

So the next thing is

- Define symbols for various types of microoperations,
- Describe the hardware that implements these microoperations

2. Register Transfer

- Computer registers are designated by capital letters.
- Few specific registers are
  - R1: processor register
  - MAR: Memory Address Register (holds an address for a memory unit)
  - PC: Program Counter
  - IR: Instruction Register
  - SR: Status Register

2. Register Transfer cont.

- Information transfer from one register to another is described by a *replacement operator:*

  R2 ← R1

  - This statement denotes a transfer of the content of register R1 into register R2
- The transfer happens in one clock cycle
- We are assuming that the circuits are available from the outputs of the source register to the inputs of the destination register, and that the destination register has a parallel load capability

2. Register Transfer cont.

- Conditional transfer occurs only under a control condition.
- Representation of a (conditional) transfer

$$P: \quad R2 \leftarrow R1$$

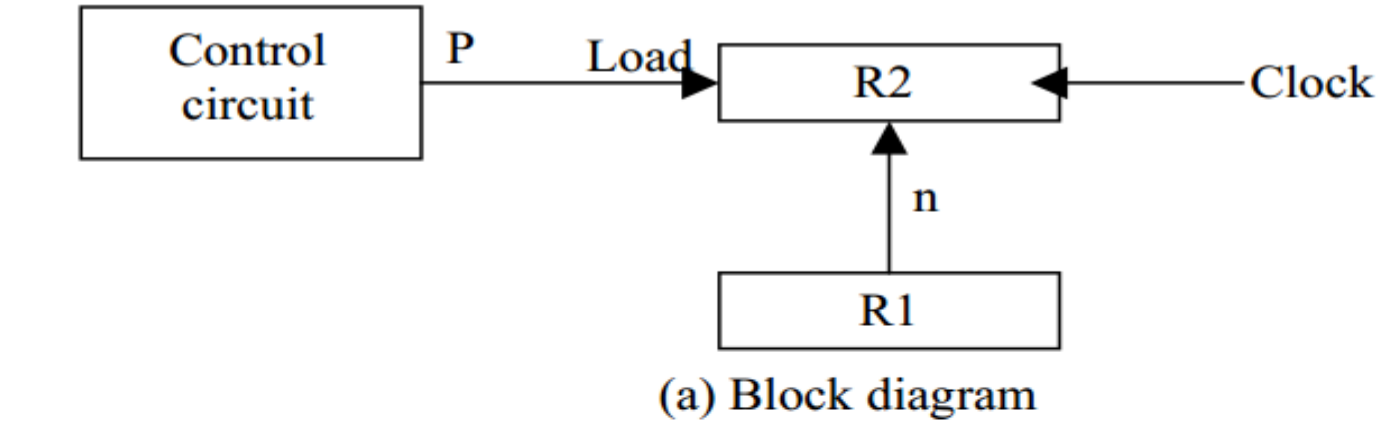
- The content of R1 is transferred into R2 only if P is 1

2 Register Transfer cont.

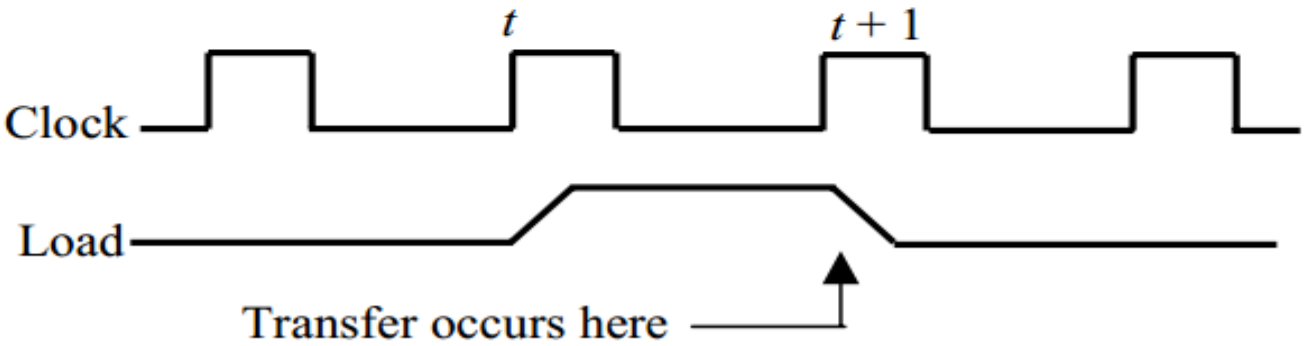
Basic Symbols for Register Transfers		
Symbol	Description	Examples
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two microoperations	R2 ← R1, R1 ← R2

2. Register Transfer cont.

Hardware implementation of a controlled transfer: P: R2 ← R1



(a) Block diagram



(b) Timing diagram

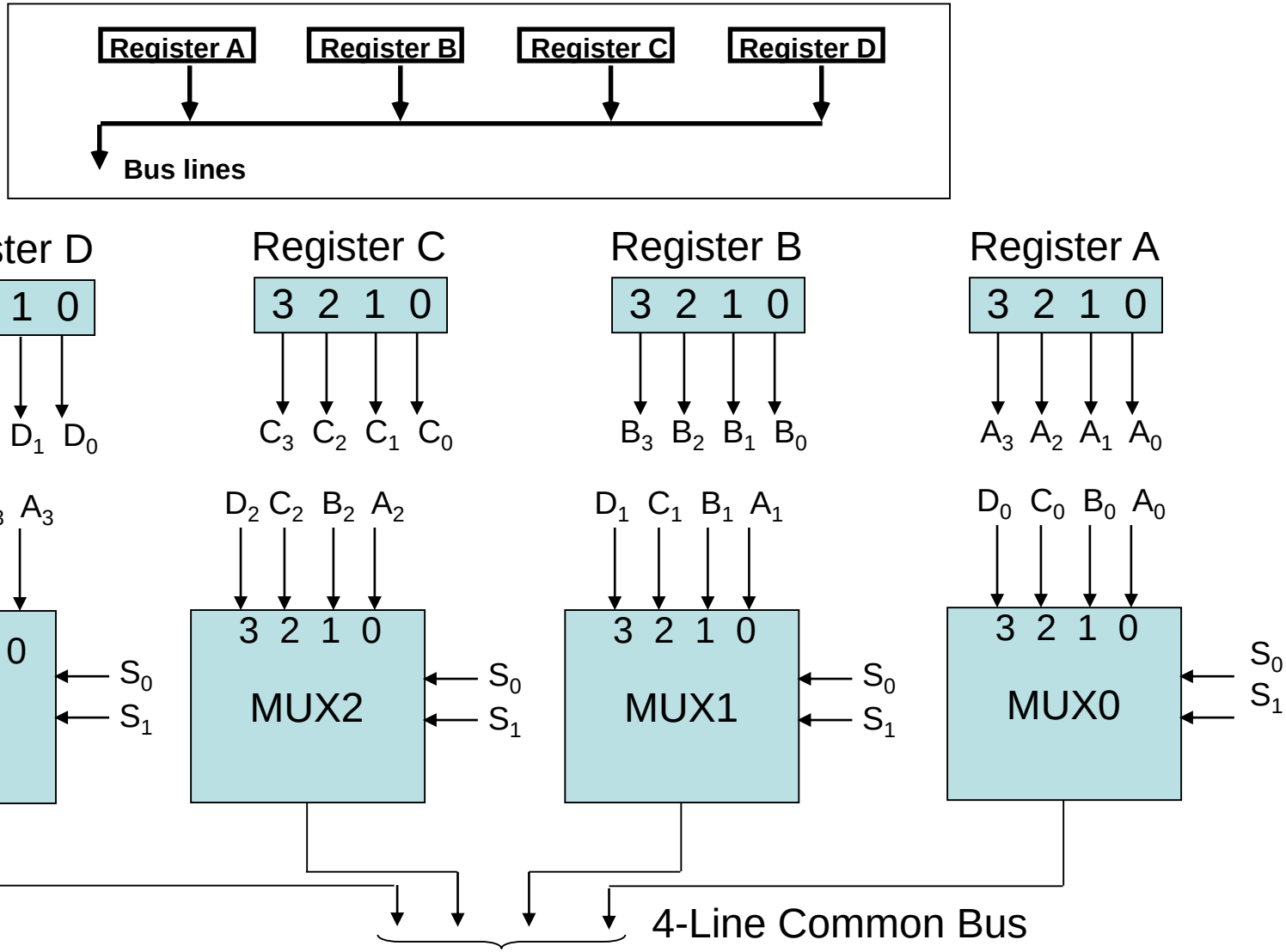
3 Bus and Memory Transfers

- A *Common Bus System* is a scheme for transferring information between registers in a multiple-register configuration

- A bus: set of common lines, one for each bit of a register, through which binary information is transferred one at a time

- Control signals determine which register is selected by the bus during each particular register transfer

3 Bus and Memory Transfers



3 Bus and Memory Transfers

Bus System for four registers

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

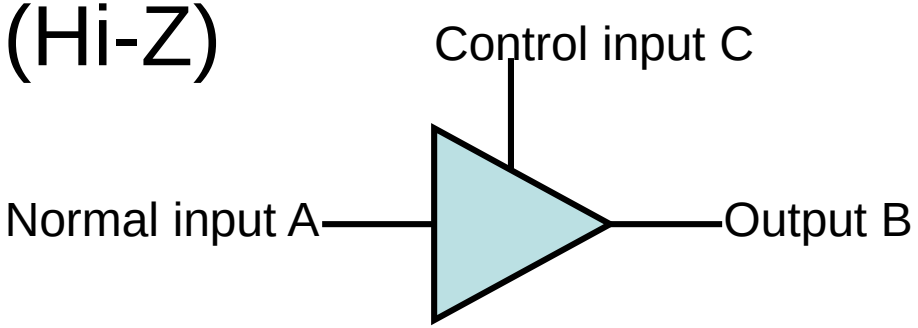
3. Bus and Memory Transfers

- We write: R2 ← C to symbolize that the content of register C is *loaded into* the register R2 using the common system bus
- It is equivalent to: BUS ←C, (select C)

  R2 ←BUS (Load R2)

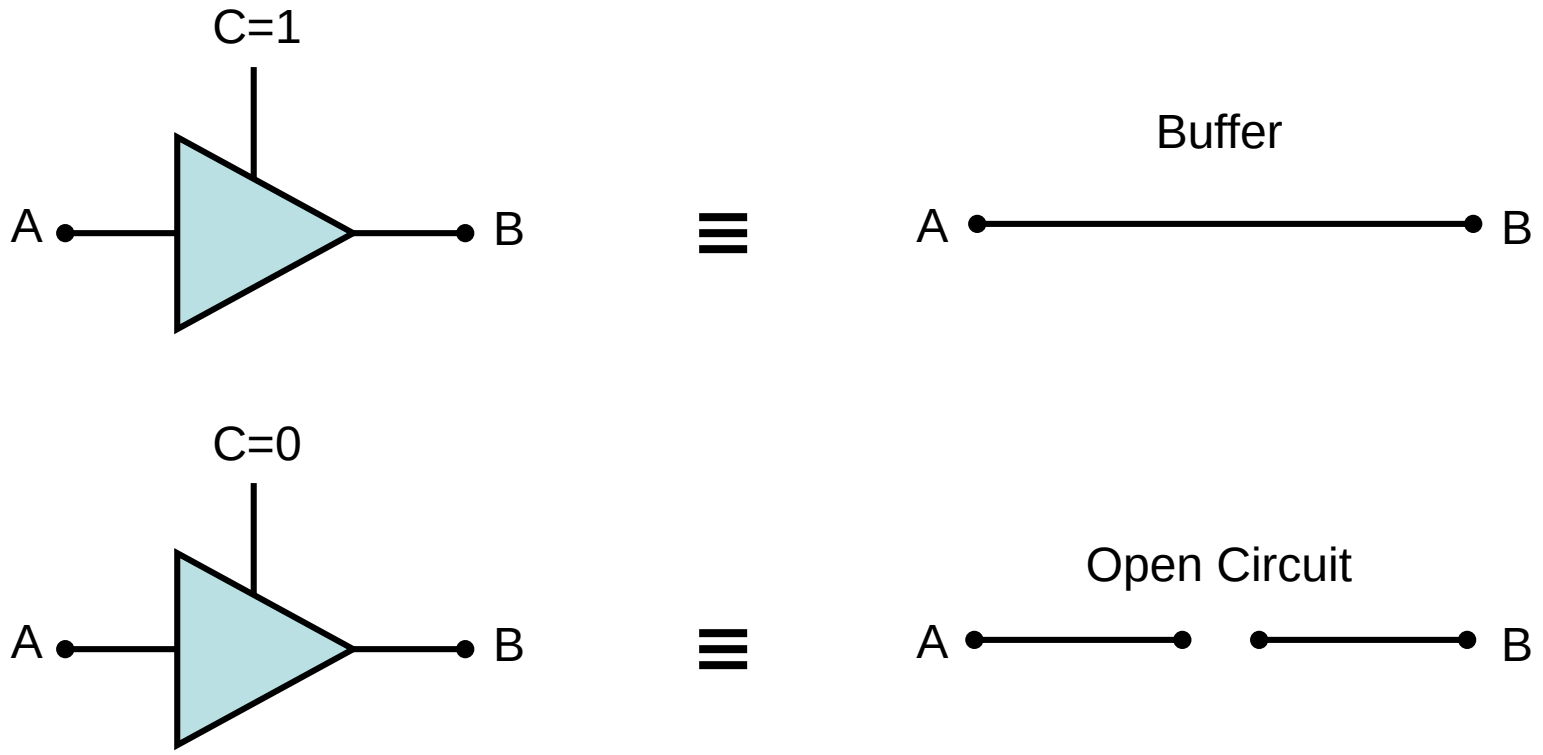
3. Bus and Memory Transfers: Three-State Bus Buffers

- A bus system can be constructed with three-state buffer gates instead of multiplexers
- A three-state buffer is a digital circuit that exhibits three states: logic-0, logic-1, and high-impedance (Hi-Z)

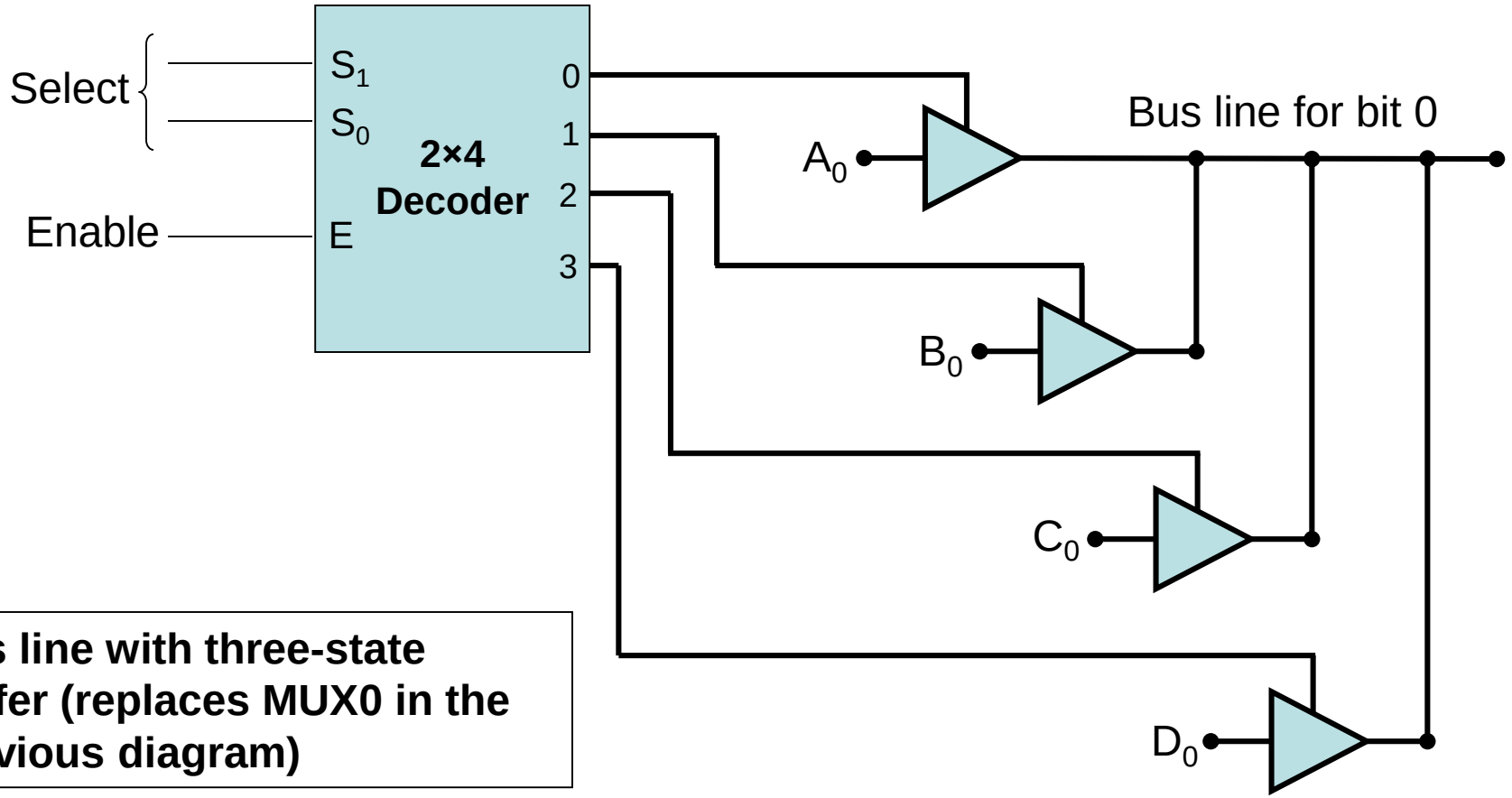


Three-State Buffer

3. Bus and Memory Transfers: Three-State Bus Buffers cont.



3. Bus and Memory Transfers: Three-State Bus Buffers cont.



Bus line with three-state buffer (replaces MUX0 in the previous diagram)

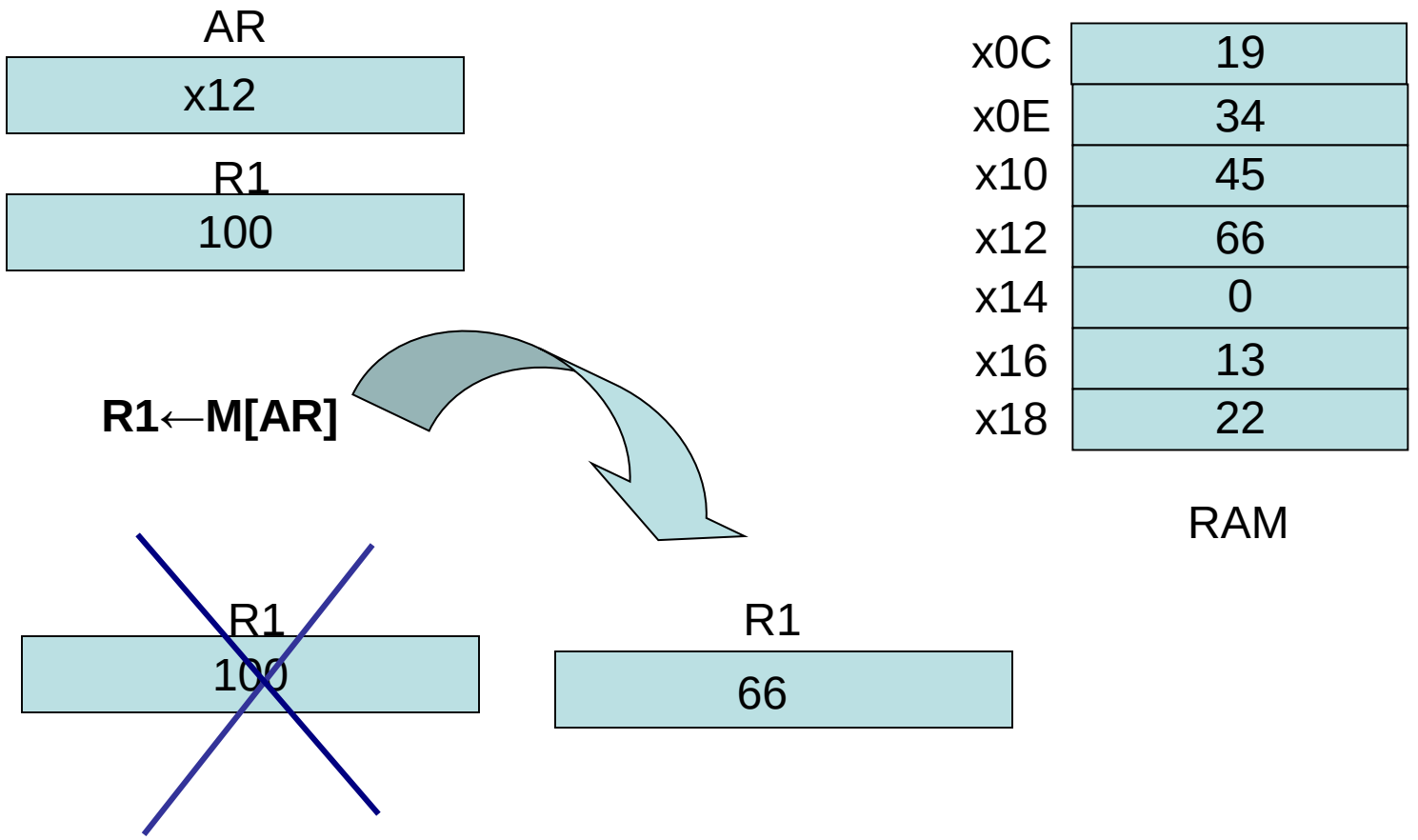
3. Bus and Memory Transfers: Memory Transfer

- Memory read : Transfer from memory
- Memory write : Transfer to memory
- Data being read or written is called a memory word
- It is necessary to specify the address of M when writing /reading memory
- This is done by enclosing the address in square brackets following the letter M
- Example: M[0016] : the memory contents at address 0x0016

3. Bus and Memory Transfers: Memory Transfer cont.

- Assume that the address of a memory unit is stored in a register called the Address Register AR
- Lets represent a Data Register with DR, then:
  - Read: DR ← M[AR]
  - Write: M[AR] ← DR

3. Bus and Memory Transfers: Memory Transfer cont.



Types of Microoperations

- The microoperations most often classified into four categories:
  1. **Register transfer microoperations**
  2. **Arithmetic microoperations (on numeric data stored in the registers)**
  3. **Logic microoperations (bit manipulations on non-numeric data)**
  4. **Shift microoperations**

4. Arithmetic Microoperations cont.

- The basic arithmetic microoperations are: addition, subtraction, increment, decrement, and shift
- Addition Microoperation:

$$R3 \leftarrow R1+R2$$

- Subtraction Microoperation:

$$R3 \leftarrow R1-R2 \text{ or :}$$

1's complement

$$R3 \leftarrow R1+\overline{R2}+1$$

4. Arithmetic Microoperations cont.

- One's Complement Microoperation:

$$R2 \leftarrow \overline{R2}$$

- Two's Complement Microoperation:

$$R2 \leftarrow R2+1$$

- Increment Microoperation:

$$R2 \leftarrow R2+1$$

- Decrement Microoperation:

$$R2 \leftarrow R2-1$$

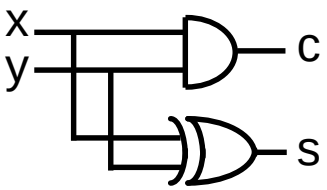
Half Adder/Full Adder

Half Adder

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

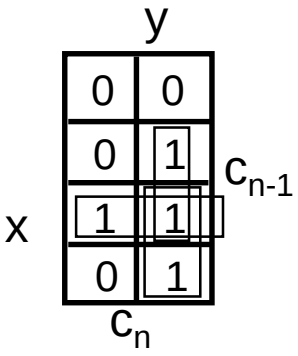
$c = xy$

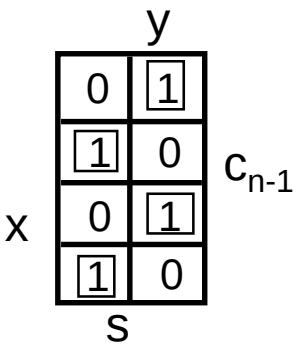
$s = xy' + x'y$
$= x \oplus y$



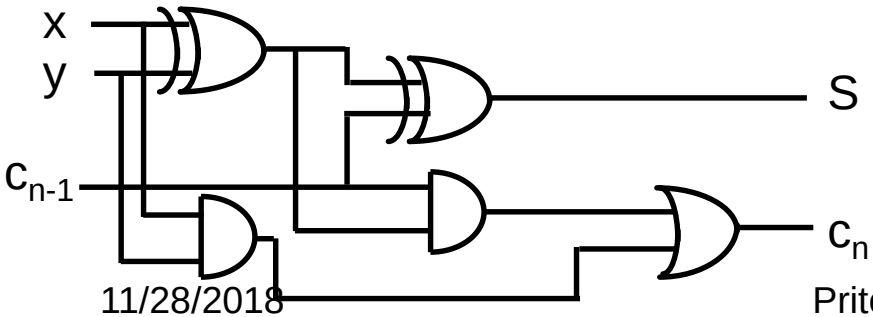
Full Adder

x	y	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



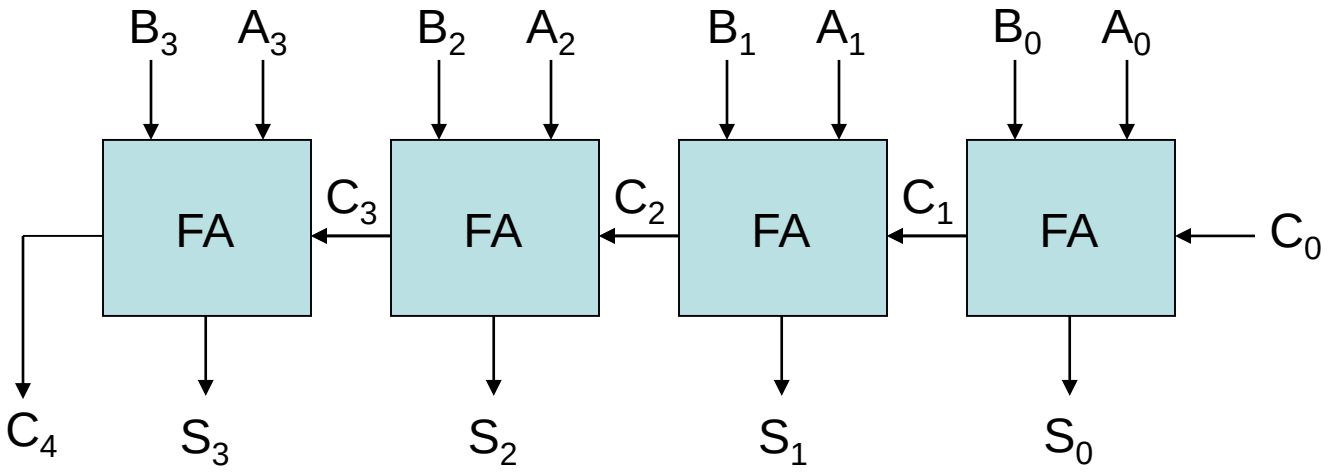


$c_n = xy + xc_{n-1} + yc_{n-1}$
$= xy + (x \oplus y)c_{n-1}$



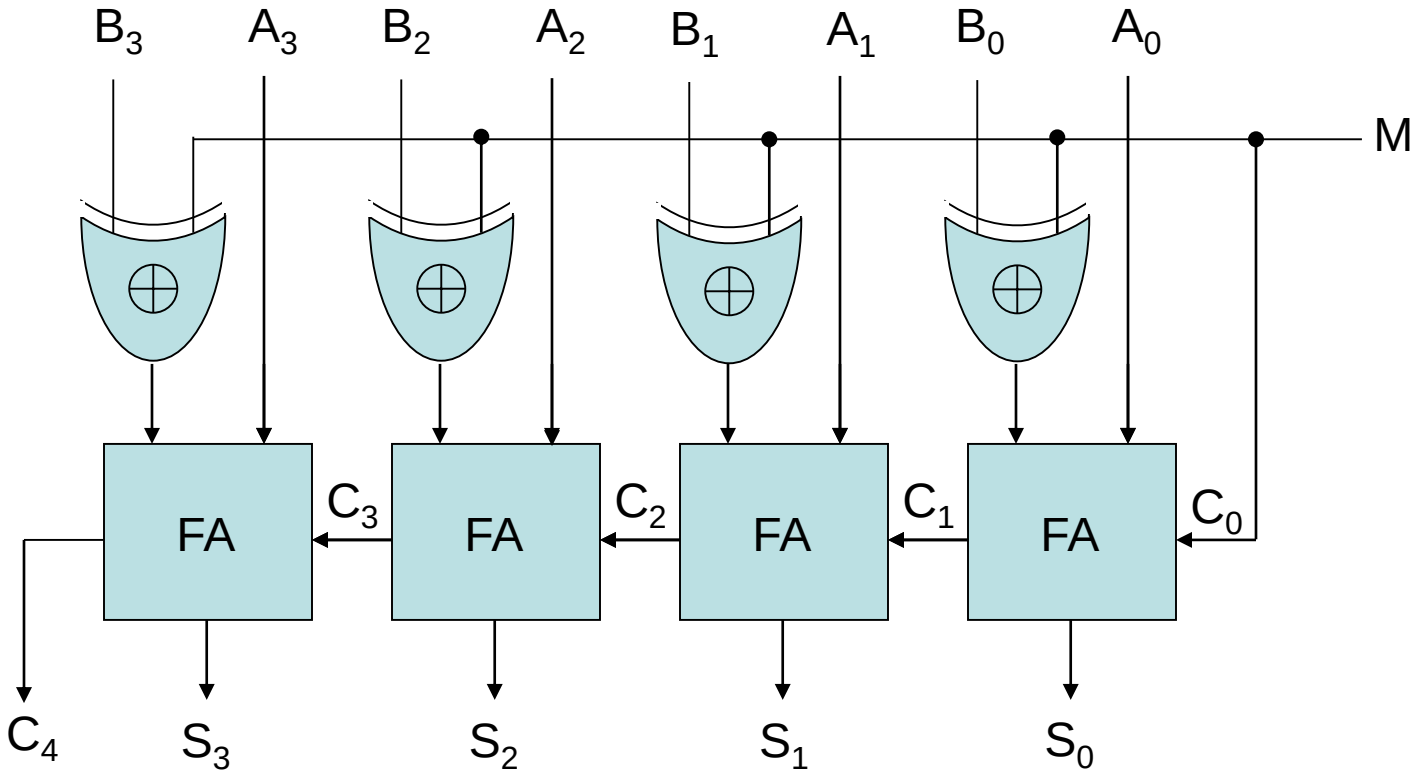
$s = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1}$
$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$

4. Arithmetic Microoperations
Binary Adder



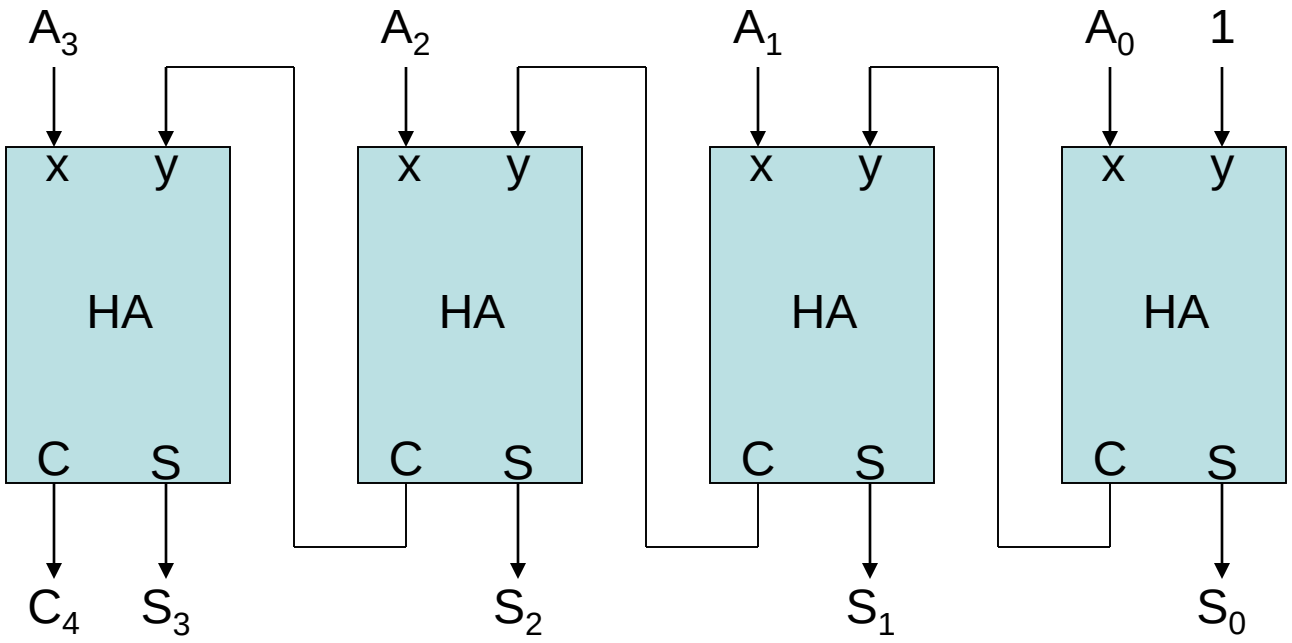
4-bit binary adder
(connection of FAs)

4. Arithmetic Microoperations
Binary Adder-Subtractor



4-bit adder-subtractor

4. Arithmetic Microoperations
Binary Incrementer



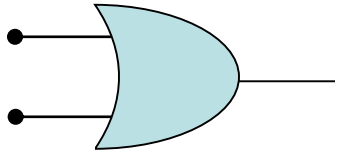
4-bit Binary Incrementer

4. Arithmetic Microoperations
Binary Incrementer

- Binary Incrementer can also be implemented using a counter
- A binary decrementer can be implemented by adding 1111 to the desired register each time!

5. Logic Microoperations
The four basic microoperations

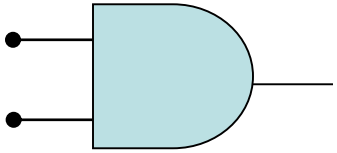
OR Microoperation

- Symbol: $\vee$, +
- Gate: 
- Example: $100110_2 \vee 1010110_2 = 1110110_2$

5. Logic Microoperations
The four basic microoperations
cont.

AND Microoperation

- Symbol: $\wedge$
- Gate:

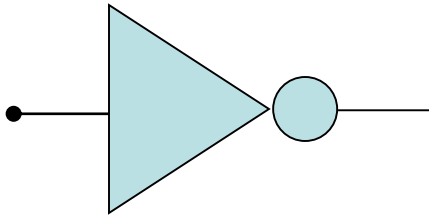


- Example: $100110_2 \wedge 1010110_2 = 0000110_2$

5. Logic Microoperations
The four basic microoperations
cont.

Complement (NOT) Microoperation

- Symbol: $\overline{}$
- Gate:

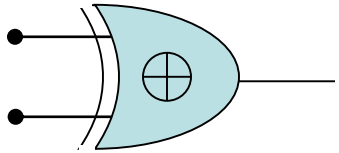


- Example: $\overline{1010110_2} = 0101001_2$

5. Logic Microoperations
The four basic microoperations
cont.

XOR (Exclusive-OR) Microoperation

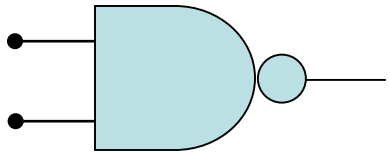
- Symbol: $\oplus$
- Gate:



- Example: $100110_2 \oplus 1010110_2 = 1110000_2$

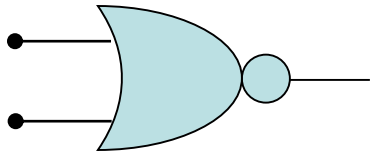
5. Logic Microoperations
Other Logic Microoperations
cont.

NAND Microoperation

- Symbols: $\wedge$ and $\overline{}$
- Gate: 
- Example: $100110_2 \wedge 1010110_2 = 1111001_2$

5. Logic Microoperations
Other Logic Microoperations
cont.

NOR Microoperation

- Symbols: $\vee$ and $\overline{}$
- Gate: 
- Example: $100110_2 \vee 1010110_2 = 0001001_2$

5. Logic Microoperations
Other Logic Microoperations
cont.

Set (Preset) Microoperation

- Force all bits into 1's by ORing them with a value in which all its bits are being assigned to logic-1
- Example: $100110_2 \vee 111111_2 = 111111_2$

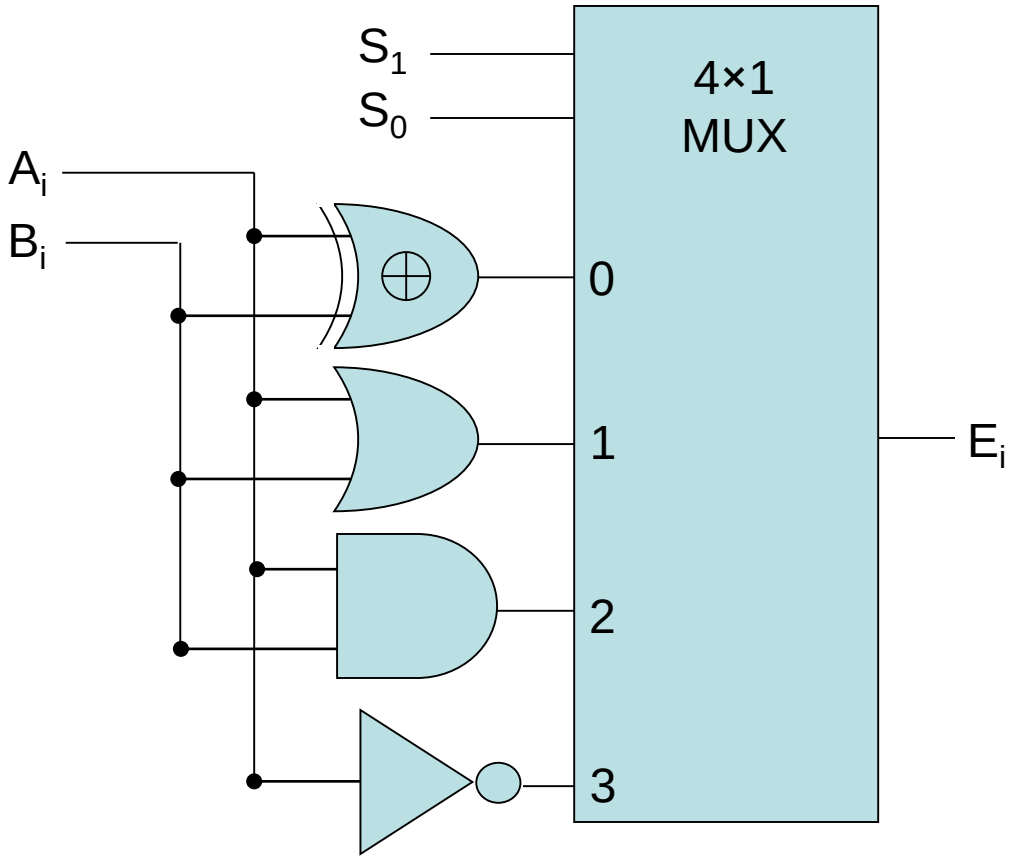
Clear (Reset) Microoperation

- Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0
- Example: $100110_2 \wedge 000000_2 = 000000_2$

5. 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.

5. Logic Microoperations Hardware Implementation cont.



S_1	S_0	Output	Operation
0	0	$E = A \oplus B$	XOR
0	1	$E = A \vee B$	OR
1	0	$E = A \wedge B$	AND
1	1	$E = \overline{A}$	Complement

This is for one bit i

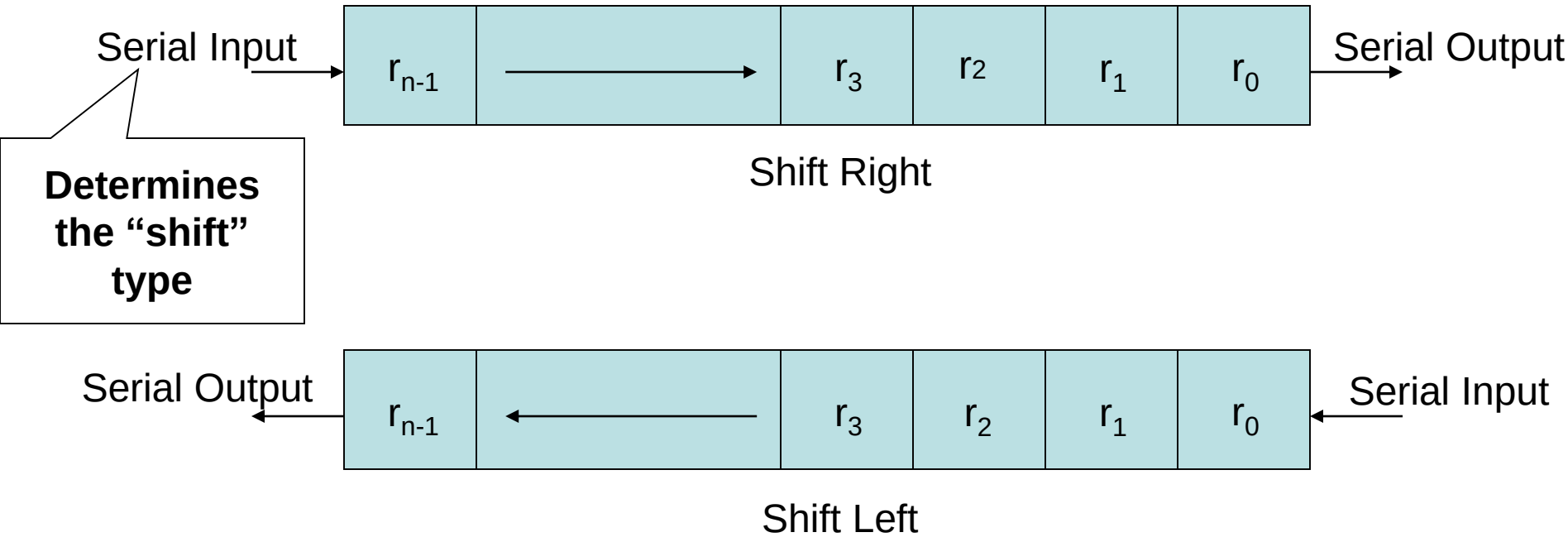
6. Shift Microoperations

- Used for serial transfer of data
- The contents of the register can be shifted to the left or to the right
- As being shifted, the first flip-flop receives its binary information from the serial input
- Three types of shift: Logical, Circular, and Arithmetic

6. Shift Microoperations cont.

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

6. Shift Microoperations cont.



**Note that the bit r_i is the bit at position (i) of the register

6. Shift Microoperations: Logical Shifts

- Transfers 0 through the serial input
- Logical Shift Right: R1←shr R1

The same

- Logical Shift Left: R2←shl R2

The same

Logical Shift Left

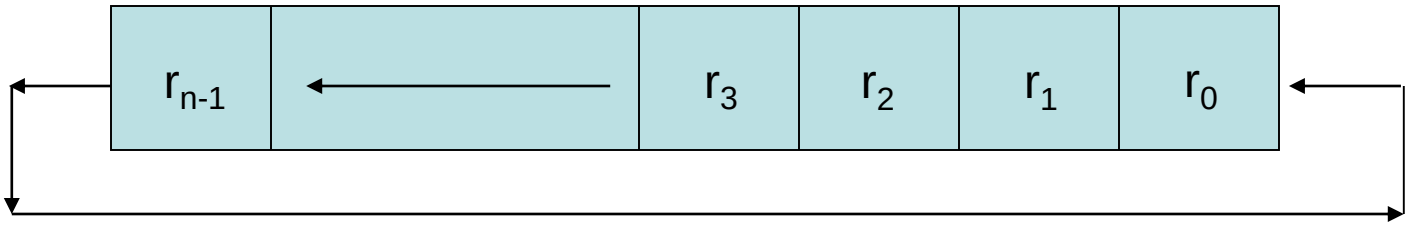
6. Shift Microoperations:
Circular Shifts (Rotate Operation)

- Circulates the bits of the register around the two ends without loss of information
- Circular Shift Right: R1 ← cir R1

The same

- Circular Shift Left: R2 ← cil R2

The same



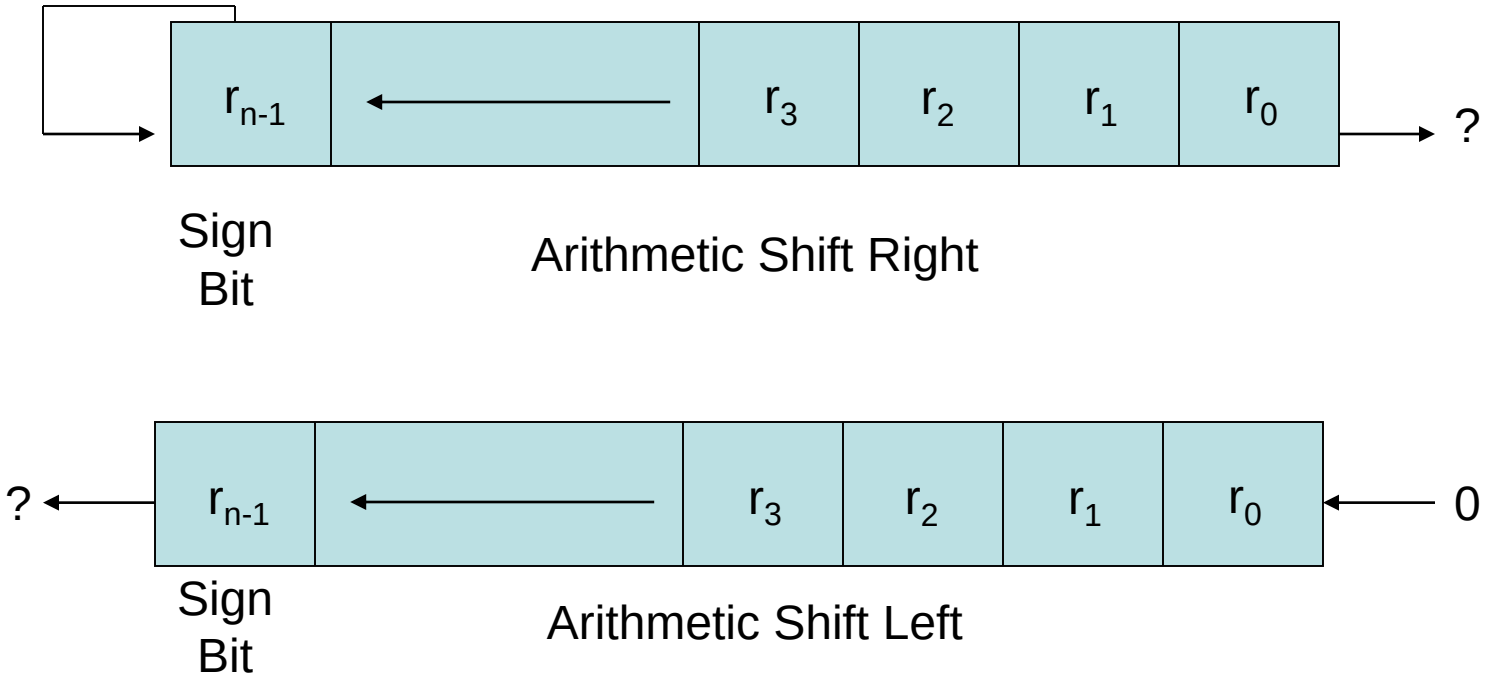
Circular Shift Left

6. Shift Microoperations
Arithmetic Shifts

- Shifts a signed binary number to the left or right
- An arithmetic shift-left multiplies a signed binary number by 2: ashl (00100): 01000
- An arithmetic shift-right divides the number by 2

  ashr (00100) : 00010

6. Shift Microoperations
Arithmetic Shifts cont.



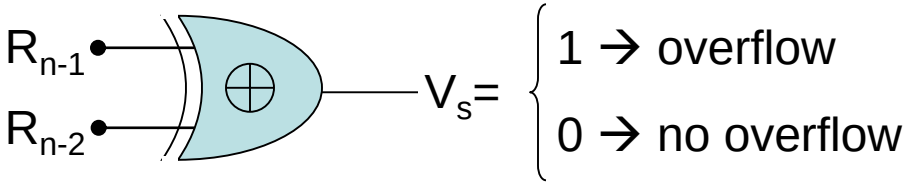
6. Shift Microoperations Arithmetic Shifts cont.

- Arithmetic Shift Right operation divides the number by 2.
- Arithmetic Shift left operation multiplies the number by 2.
- Overflow means whether we required extra bit or not
- Overflow may occur in Arithmetic Left Shift operation if the sign bit has changed.

6. Shift Microoperations
Arithmetic Shifts cont.

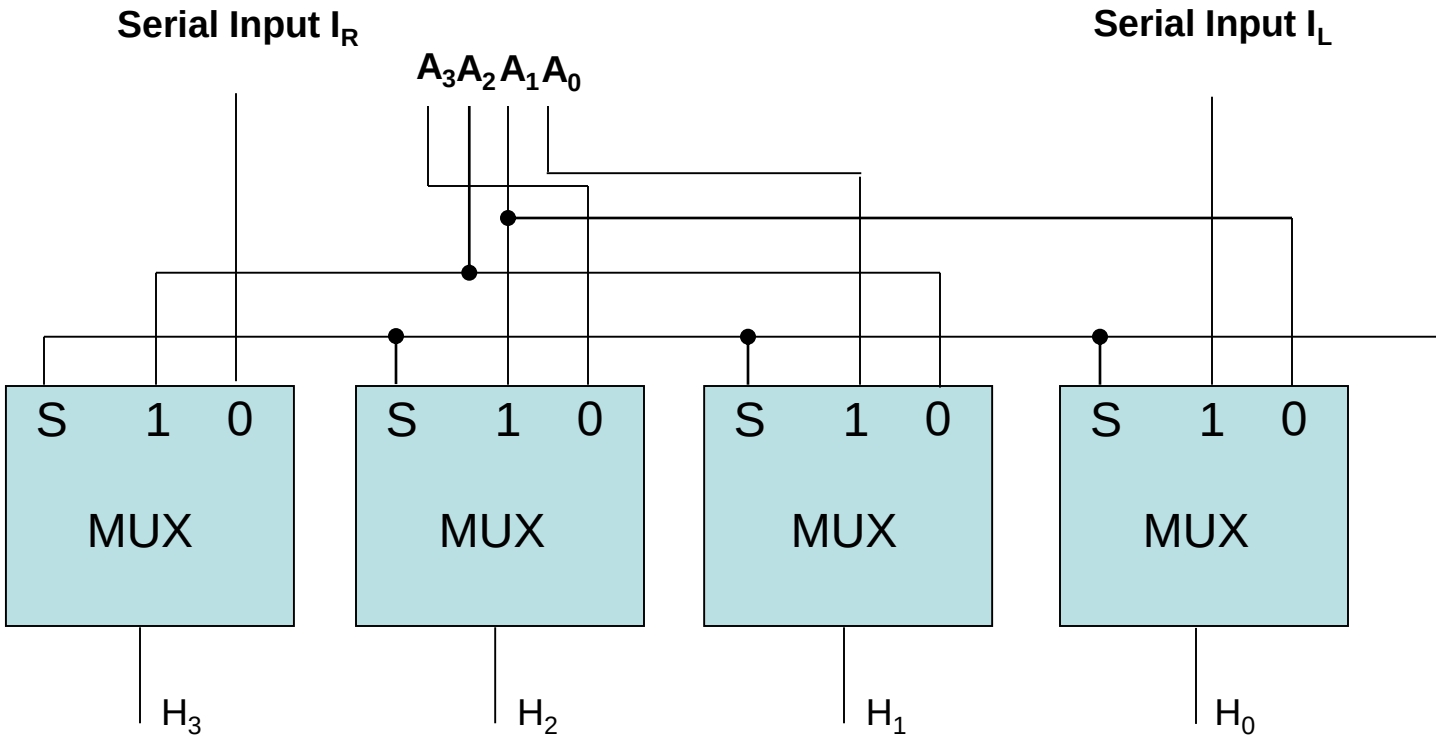
- An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow

$$V_s = R_{n-1} \oplus R_{n-2}$$



6. Shift Microoperations Hardware Implementation cont.

- Implement of shift operation with a combinational circuit



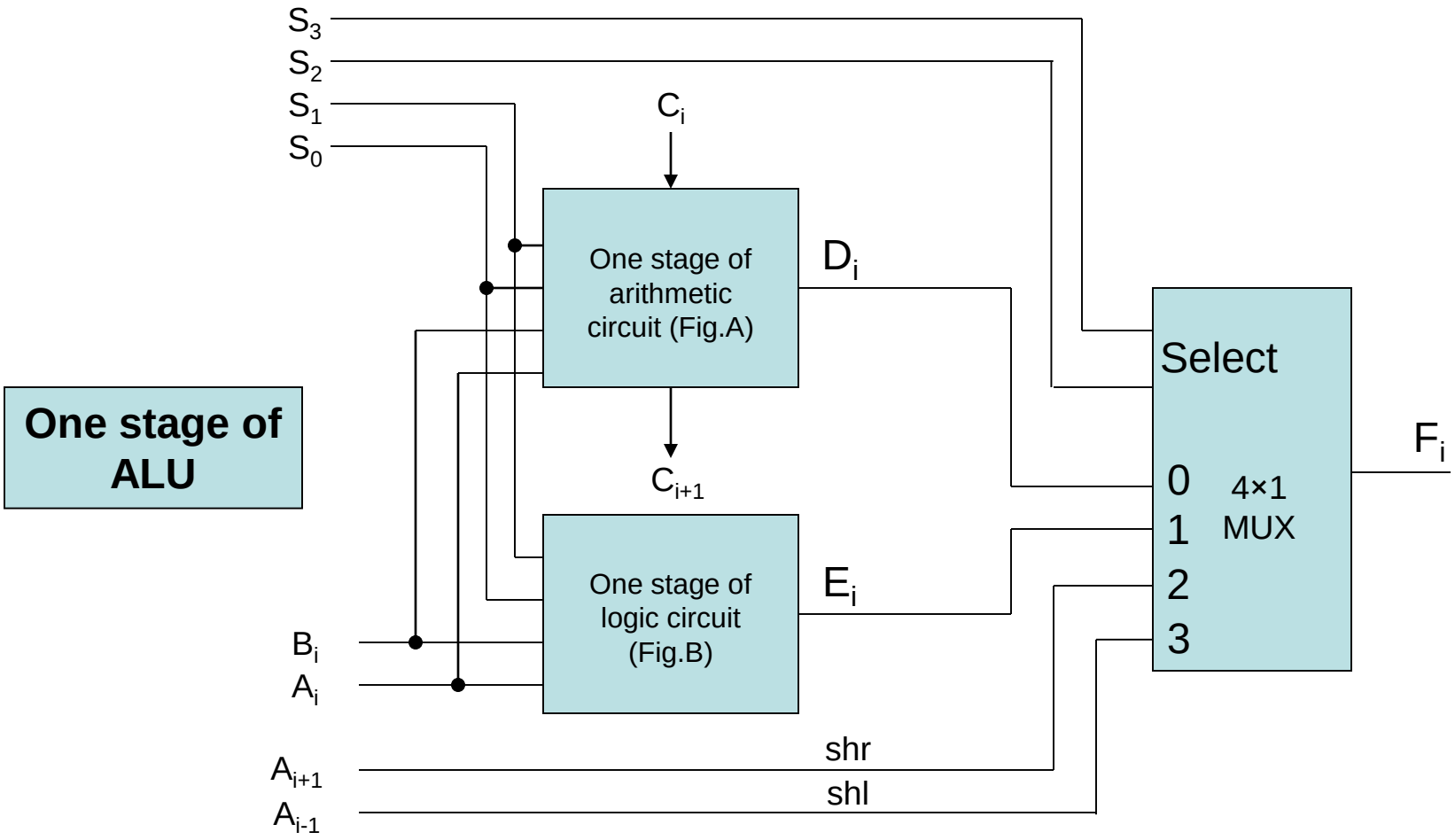
4-bit Combinational Circuit Shifter

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

7. Arithmetic Logic Shift Unit

- Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an Arithmetic Logic Unit (**ALU**)

7. Arithmetic Logic Shift Unit cont.



7. Arithmetic Logic Shift Unit cont.

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

Assignment - 1.

1. What is the range of unsigned numbers that can be represented in 4 bits?
2. What is the range of signed numbers that can be represented in 4 bits?
3. Example: Assume R1=11001110,

then:

Arithmetic shift right once ,Arithmetic shift right twice, Arithmetic shift left once, Arithmetic shift left twice, Logical shift right once, Logical shift left once, Circular shift right once, Circular shift left once

Assignment - 1

- Example: Assume R1=11001110, then:
  - Arithmetic shift right once : R1 = 11100111
  - Arithmetic shift right twice : R1 = 11110011
  - Arithmetic shift left once : R1 = 10011100
  - Arithmetic shift left twice : R1 = 00111000
  - Logical shift right once : R1 = 01100111
  - Logical shift left once : R1 = 10011100
  - Circular shift right once : R1 = 01100111
  - Circular shift left once : R1 = 10011101