

Register Transfer and Micro operations

contents

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

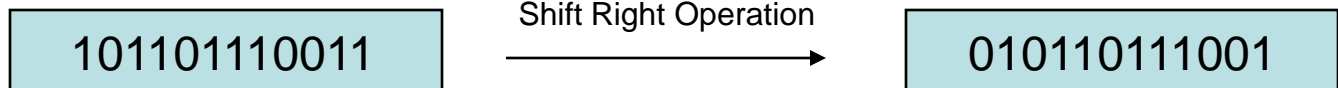
4-1 Register Transfer Language (RTL)

- 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

4-1 Register Transfer Language

cont.

- 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



4-1 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

4-1 Register Transfer Language

cont.

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

Next steps:

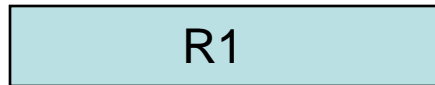
- Define symbols for various types of microoperations,
- Describe the hardware that implements these microoperations

4-2 Register Transfer (our first microoperation)

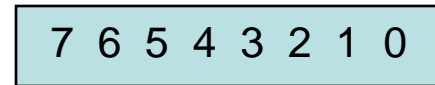
- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register
 - R1: processor register
 - MAR: Memory Address Register (holds an address for a memory unit)
 - PC: Program Counter
 - IR: Instruction Register
 - SR: Status Register

4-2 Register Transfer ^{cont.}

- The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1 (from the right position toward the left position)



Register R1

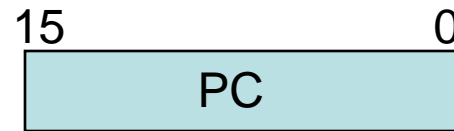


Showing individual bits

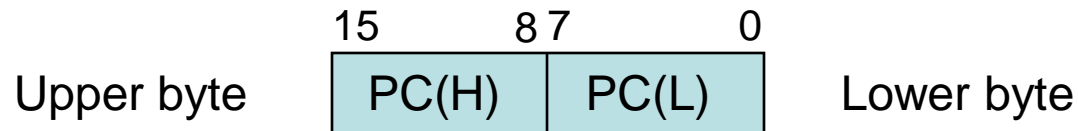
A block diagram of a register

4-2 Register Transfer ^{cont.}

Other ways of drawing the block diagram of a register:



Numbering of bits



Partitioned into two parts

4-2 Register Transfer cont.

- Information transfer from one register to another is described by a *replacement operator*: **$R2 \leftarrow R1$**
 - This statement denotes a transfer of the content of register R1 into register R2
- The transfer happens in one clock cycle
- The content of the R1 (source) does not change
- The content of the R2 (destination) will be lost and replaced by the new data transferred from R1
- 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

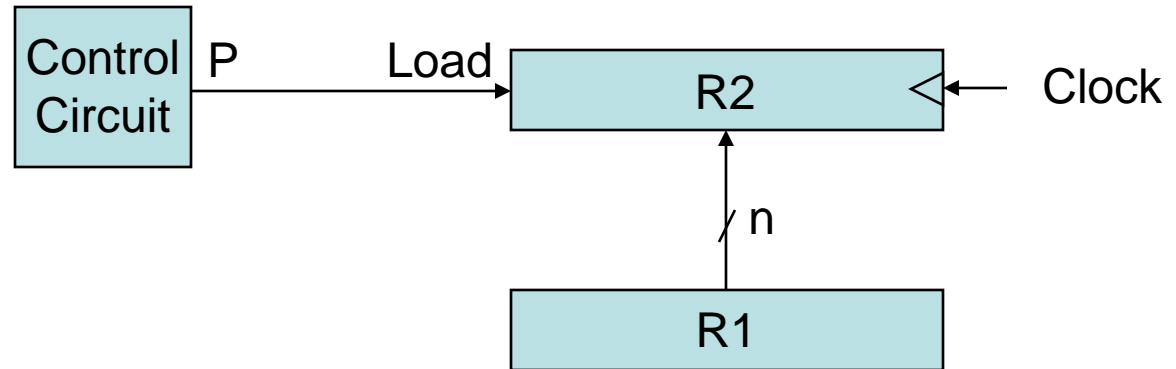
4-2 Register Transfer ^{cont.}

- Conditional transfer occurs only under a control condition
- Representation of a (conditional) transfer
P: $R2 \leftarrow R1$
- A binary condition (P equals to 0 or 1) determines when the transfer occurs
- The content of R1 is transferred into R2 only if P is 1

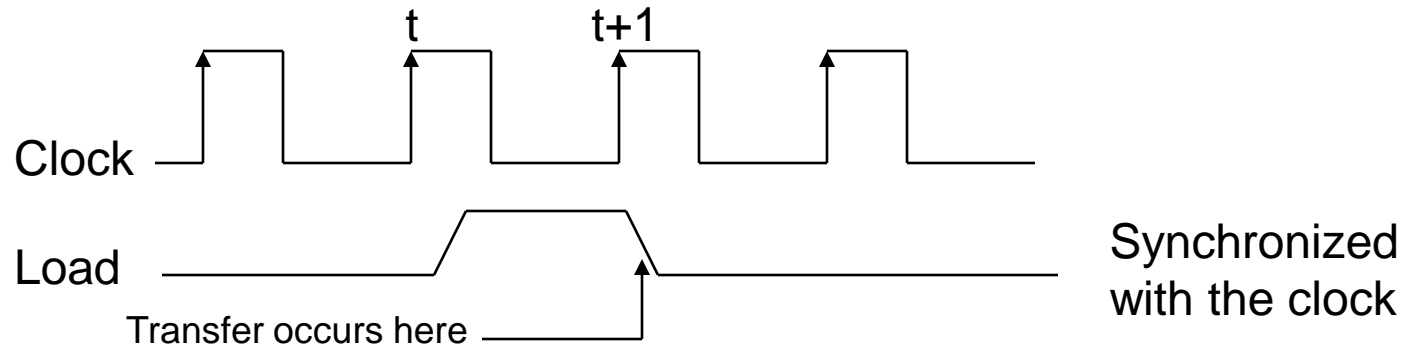
4-2 Register Transfer cont.

Hardware implementation of a controlled transfer: $P: R2 \leftarrow R1$

Block diagram:



Timing diagram



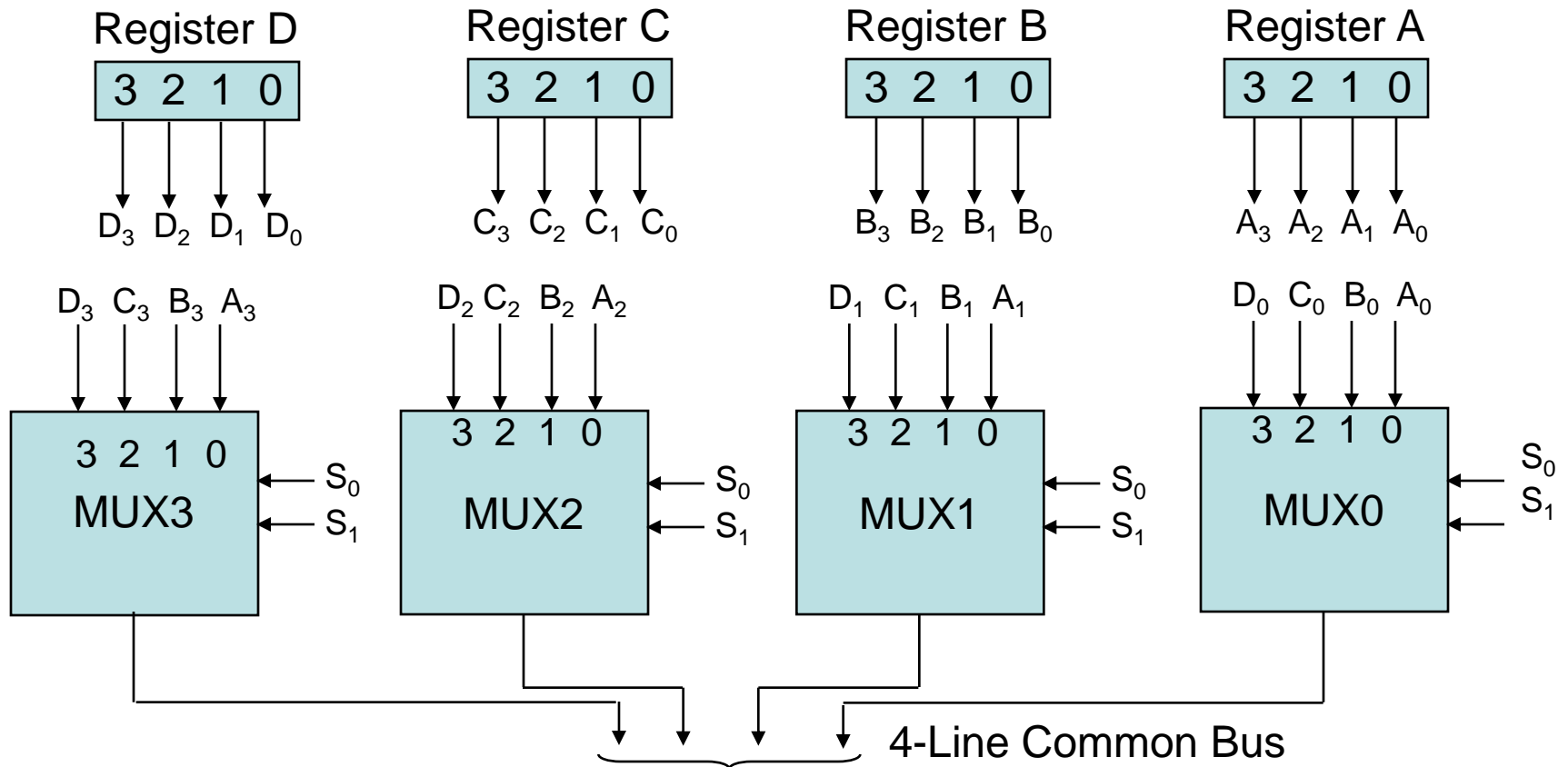
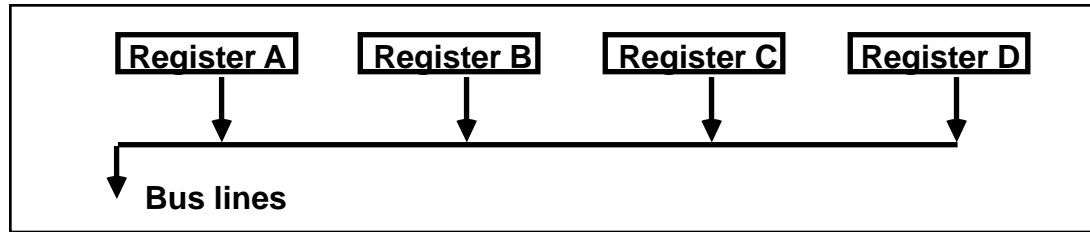
4-2 Register Transfer cont.

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

4-3 Bus and Memory Transfers

- Paths must be provided to transfer information from one register to another
- 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

4-3 Bus and Memory Transfers

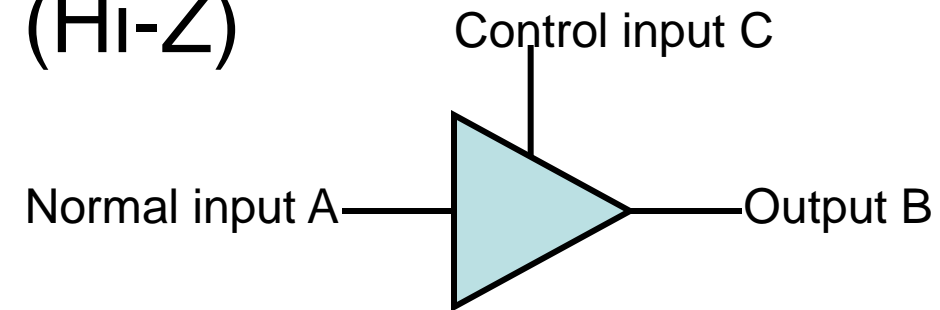


4-3 Bus and Memory Transfers

- The transfer of information from a bus into one of many destination registers is done:
 - By connecting the bus lines to the inputs of all destination registers and then:
 - activating the load control of the particular destination register selected
- We write: $R2 \leftarrow C$ to symbolize that the content of register C is *loaded into* the register $R2$ using the common system bus
- It is equivalent to: $BUS \leftarrow C$, (select C)
 $R2 \leftarrow BUS$ (Load $R2$)

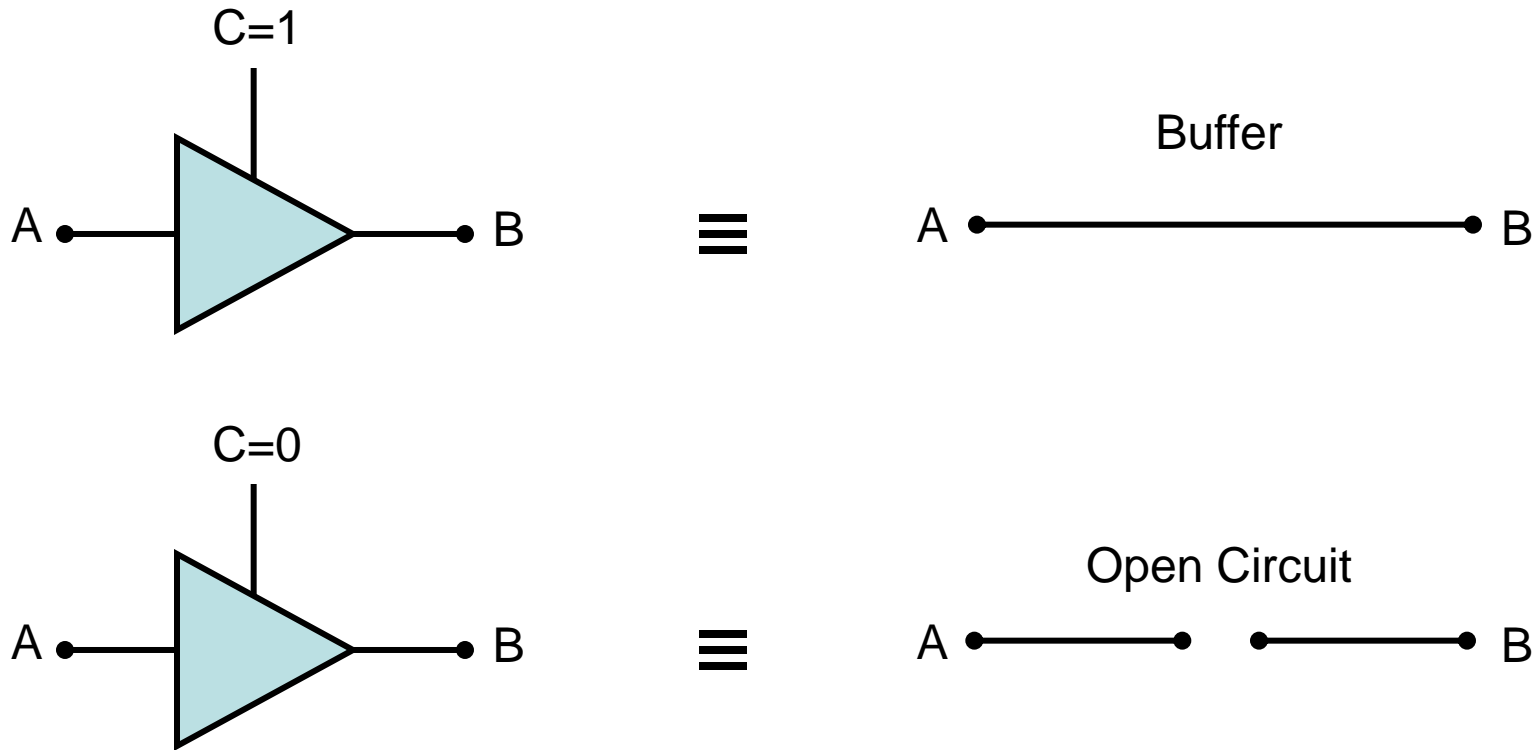
4-3 Bus and Memory Transfers: Three-State Bus Buffers

- 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)

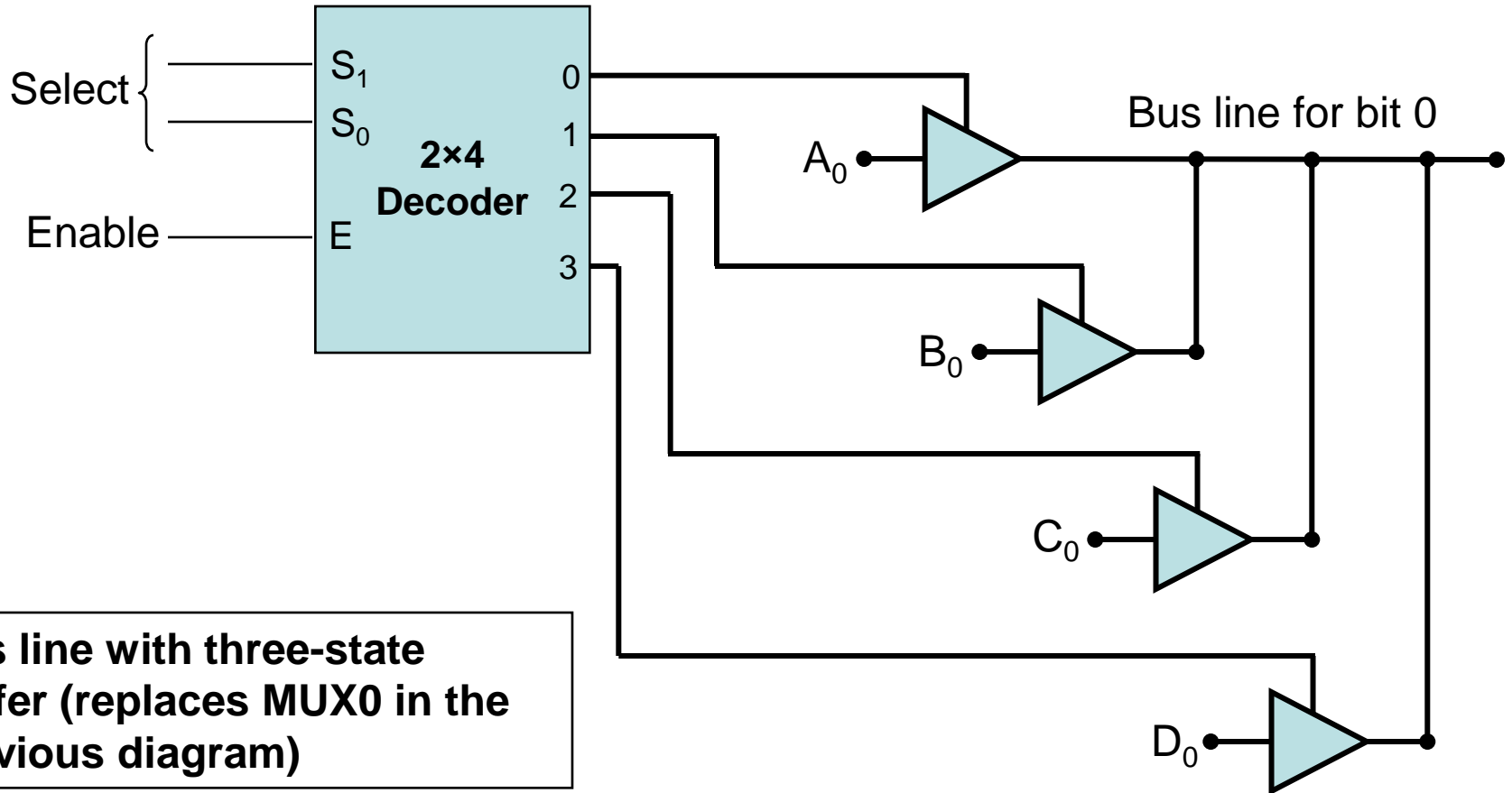


Three-State Buffer

4-3 Bus and Memory Transfers: Three-State Bus Buffers ^{cont.}



4-3 Bus and Memory Transfers: Three-State Bus Buffers cont.



4-3 Bus and Memory Transfers:

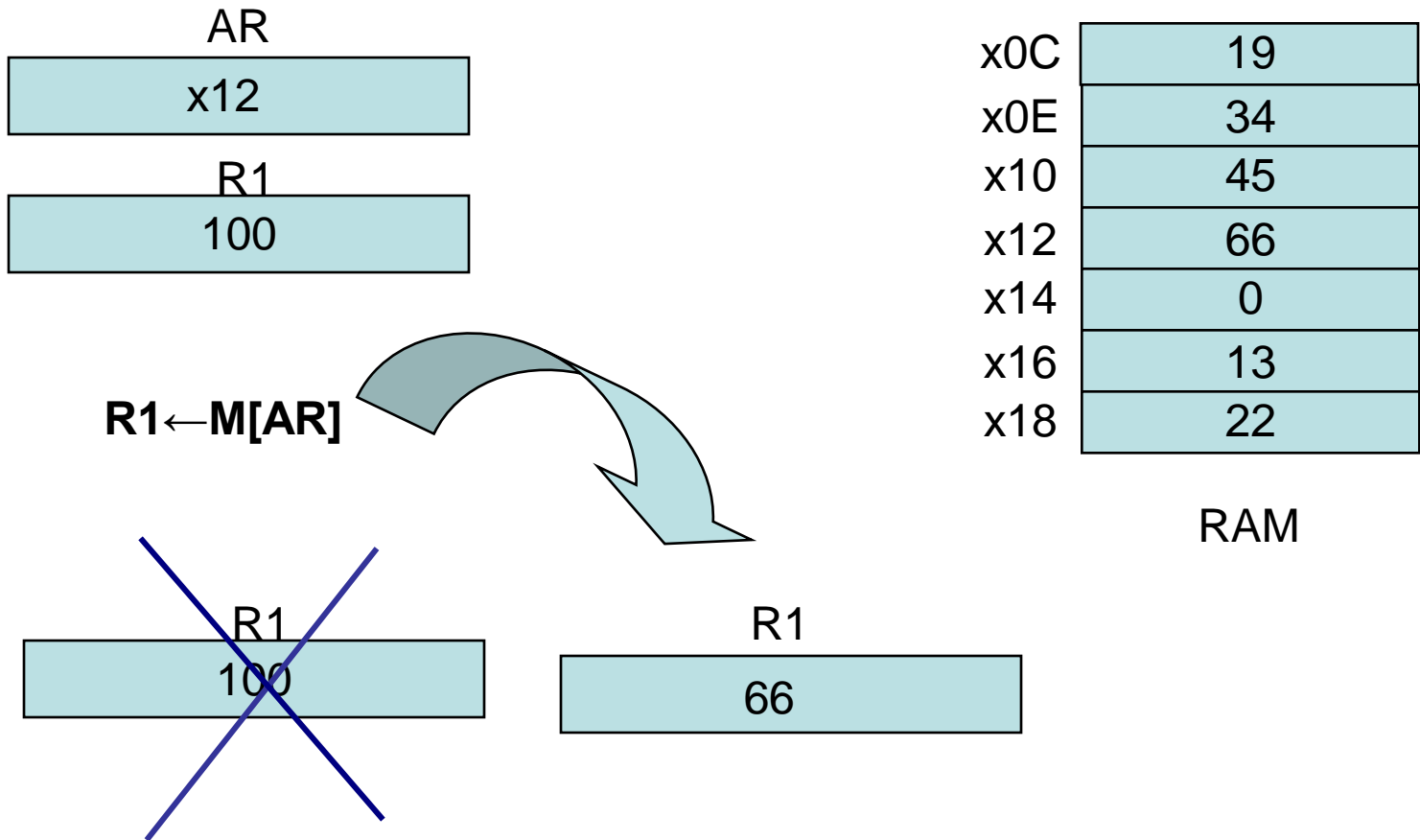
Memory Transfer

- Memory read : Transfer from memory
- Memory write : Transfer to memory
- Data being read or wrote is called a memory word (called M)- (refer to section 2-7)
- 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

4-3 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: $DR \leftarrow M[AR]$
 - Write: $M[AR] \leftarrow DR$

4-3 Bus and Memory Transfers: Memory Transfer cont.



4-4 Arithmetic Microoperations

- The microoperations most often encountered in digital computers are classified into four categories:
 - Register transfer microoperations
 - Arithmetic microoperations (on numeric data stored in the registers)
 - Logic microoperations (bit manipulations on non-numeric data)
 - Shift microoperations

4-4 Arithmetic Microoperations ^{cont.}

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

$$\mathbf{R3 \leftarrow R1 + R2}$$

- Subtraction Microoperation:

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

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

1's complement

4-4 Arithmetic Microoperations ^{cont.}

- One's Complement Microoperation:

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

- Two's Complement Microoperation:

$$\mathbf{R2 \leftarrow \overline{R2} + 1}$$

- Increment Microoperation:

$$\mathbf{R2 \leftarrow R2 + 1}$$

- Decrement Microoperation:

$$\mathbf{R2 \leftarrow R2 - 1}$$

Half Adder/Full Adder

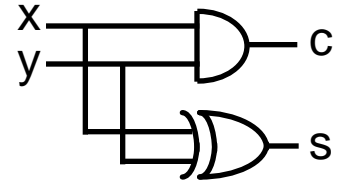
Half Adder

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

$$c = xy$$

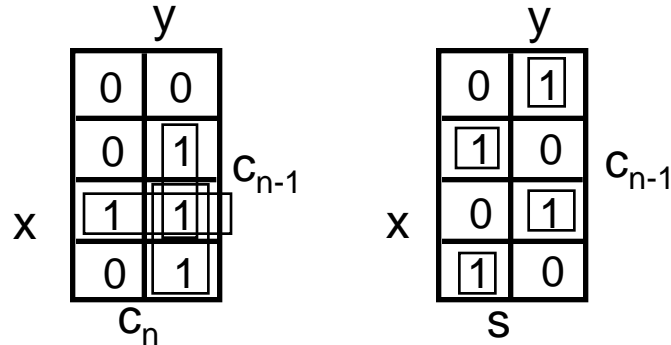
$$s = xy' + x'y$$

$$= x \oplus y$$



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

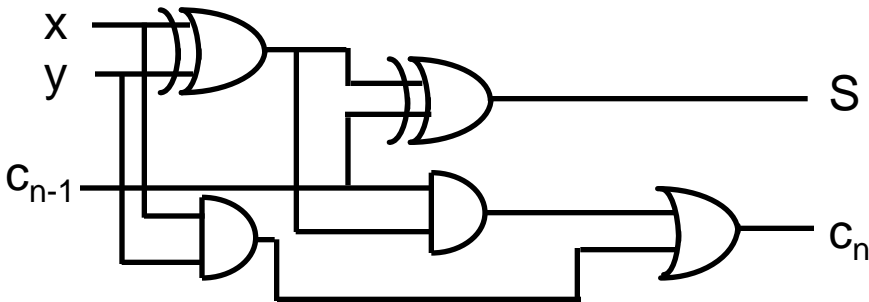


$$c_n = xy + xc_{n-1} + yc_{n-1}$$

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

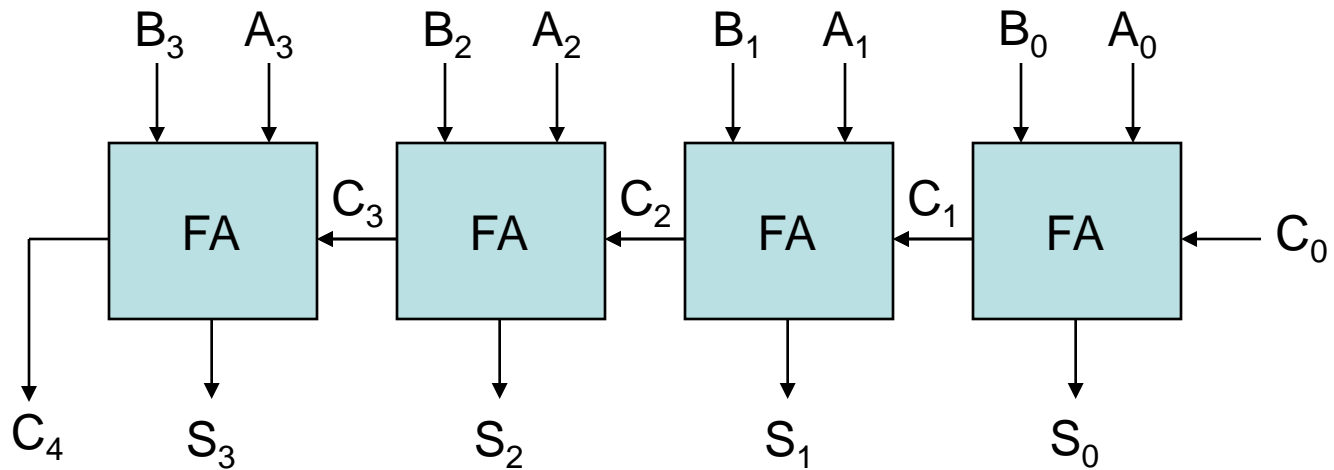
$$s = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1}$$

$$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$$



4-4 Arithmetic Microoperations

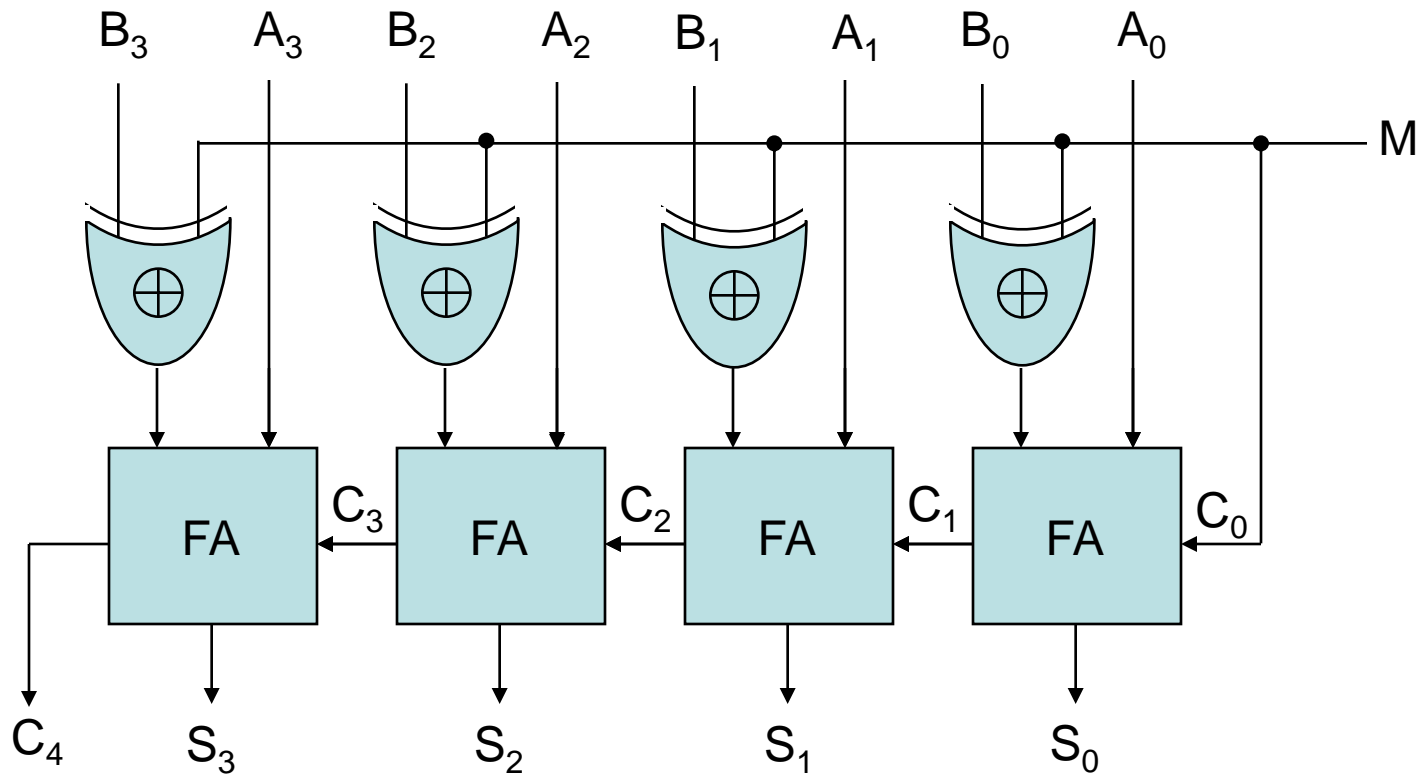
Binary Adder



**4-bit binary adder
(connection of FAs)**

4-4 Arithmetic Microoperations

Binary Adder-Subtractor



4-bit adder-subtractor

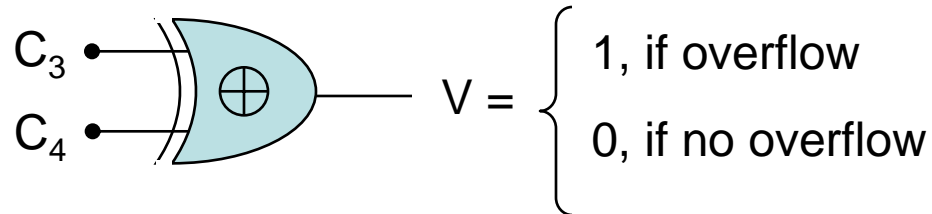
4-4 Arithmetic Microoperations

Binary Adder-Subtractor

- For unsigned numbers, this gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$
(example: $3 - 5 = -2 = 1110$)
- For signed numbers, the result is $A - B$ provided that there is no overflow. (example : $-3 - 5 = -8$)

1101
1011 +

1000



Overflow detector for signed numbers

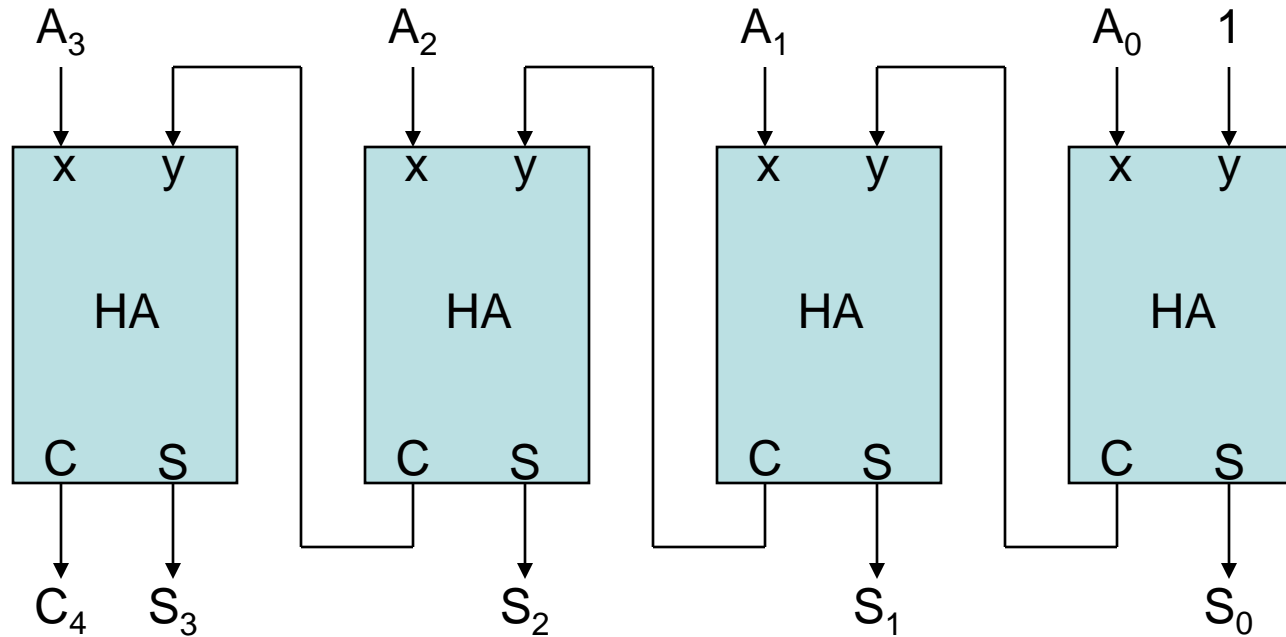
4-4 Arithmetic Microoperations

Binary Adder-Subtractor ^{cont.}

- What is the range of unsigned numbers that can be represented in 4 bits?
- What is the range of signed numbers that can be represented in 4 bits?
- Repeat for n-bit?!

4-4 Arithmetic Microoperations

Binary Incrementer



4-bit Binary Incrementer

4-4 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!

4-4 Arithmetic Microoperations

Arithmetic Circuit

- This circuit performs seven distinct arithmetic operations and the basic component of it is the parallel adder
- The output of the binary adder is calculated from the following arithmetic sum:
 - $D = A + Y + C_{in}$

4-4 Arithmetic Microoperations

Arithmetic Circuit cont.

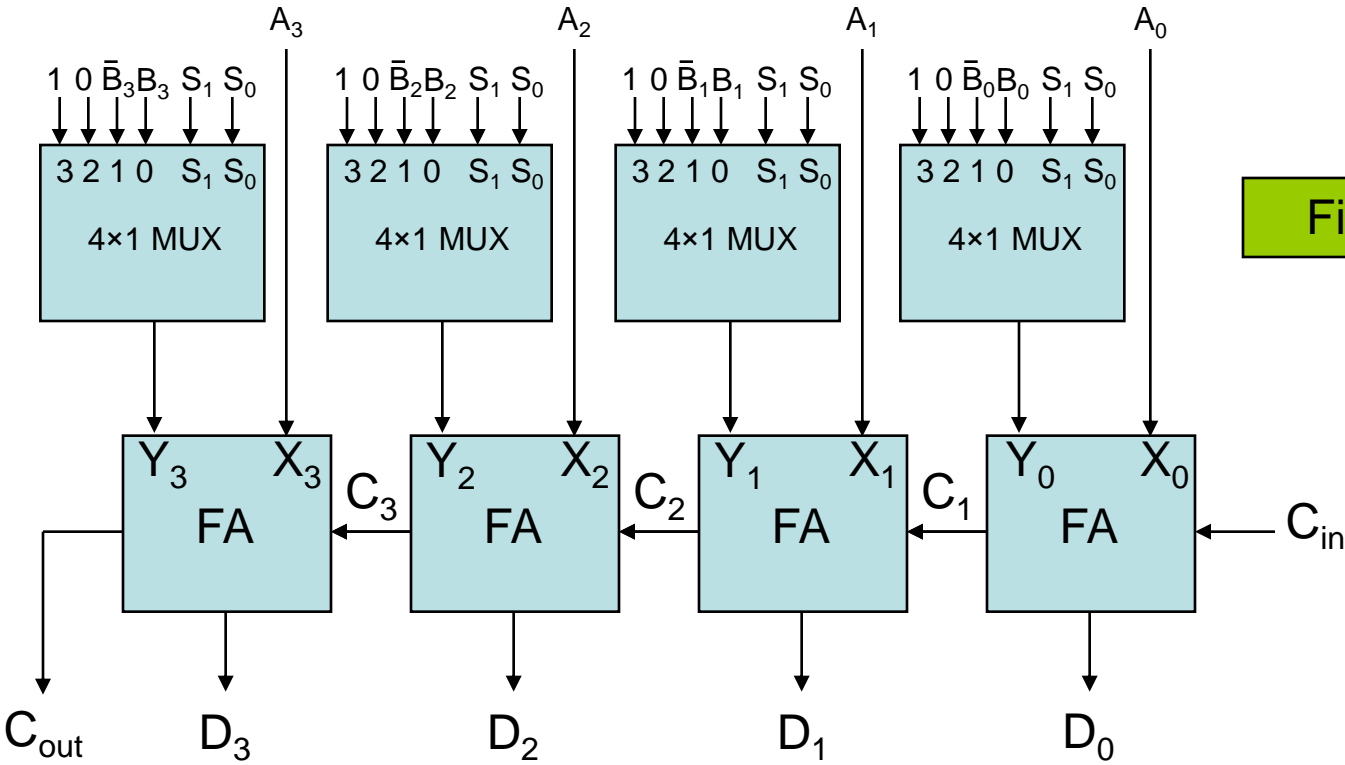


Figure A

4-bit Arithmetic Circuit

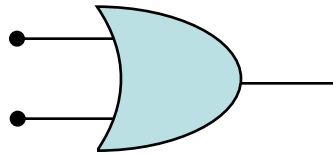
4-5 Logic Microoperations

The four basic microoperations

OR Microoperation

- Symbol: \vee , +

- Gate:



- Example: $100110_2 \vee 1010110_2 = 1110110_2$



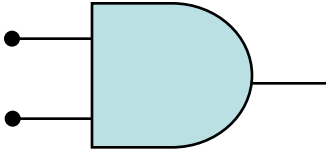
4-5 Logic Microoperations

The four basic microoperations

cont.

AND Microoperation

- Symbol: \wedge

- Gate: 

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

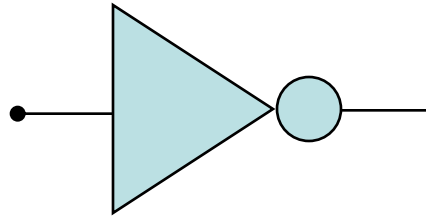
4-5 Logic Microoperations

The four basic microoperations

cont.

Complement (NOT) Microoperation

- Symbol: $\bar{\quad}$



- Gate:

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

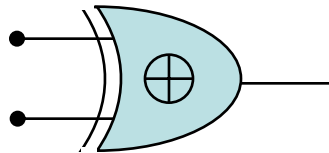
4-5 Logic Microoperations

The four basic microoperations

cont.

XOR (Exclusive-OR) Microoperation

- Symbol: \oplus



- Gate:

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

4-5 Logic Microoperations

Other Logic Microoperations

Selective-set Operation


- Used to force selected bits of a register into logic-1 by using the OR operation

- Example: $0100_2 \vee 1000_2 = 1100_2$

In a processor register



Loaded into a register from memory to perform the selective-set operation



4-5 Logic Microoperations

Other Logic Microoperations ^{cont.}

Selective-complement (toggling) Operation

- Used to force selected bits of a register to be complemented by using the XOR operation

- Example: $0001_2 \oplus 1000_2 = 1001_2$

In a processor register

Loaded into a register from memory to perform the selective-complement operation

4-5 Logic Microoperations

Other Logic Microoperations ^{cont.}

Insert Operation

- Step1: mask the desired bits
- Step2: OR them with the desired value
- Example: suppose $R1 = 0110\ 1010$, and we desire to replace the leftmost 4 bits (0110) with 1001 then:
 - Step1: $0110\ 1010 \wedge 0000\ 1111$
 - Step2: $0000\ 1010 \vee 1001\ 0000$
- $\rightarrow R1 = 1001\ 1010$

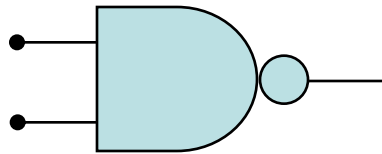
4-5 Logic Microoperations

Other Logic Microoperations cont.

NAND Microoperation

- Symbols: \wedge and $\bar{\quad}$

- Gate:



- Example: $100110_2 \wedge 1010110_2 = 1111001_2$

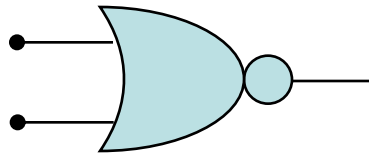
4-5 Logic Microoperations

Other Logic Microoperations cont.

NOR Microoperation

- Symbols: \vee and $\bar{}$

- Gate:



- Example: $100110_2 \vee 1010110_2 = 0001001_2$

4-5 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

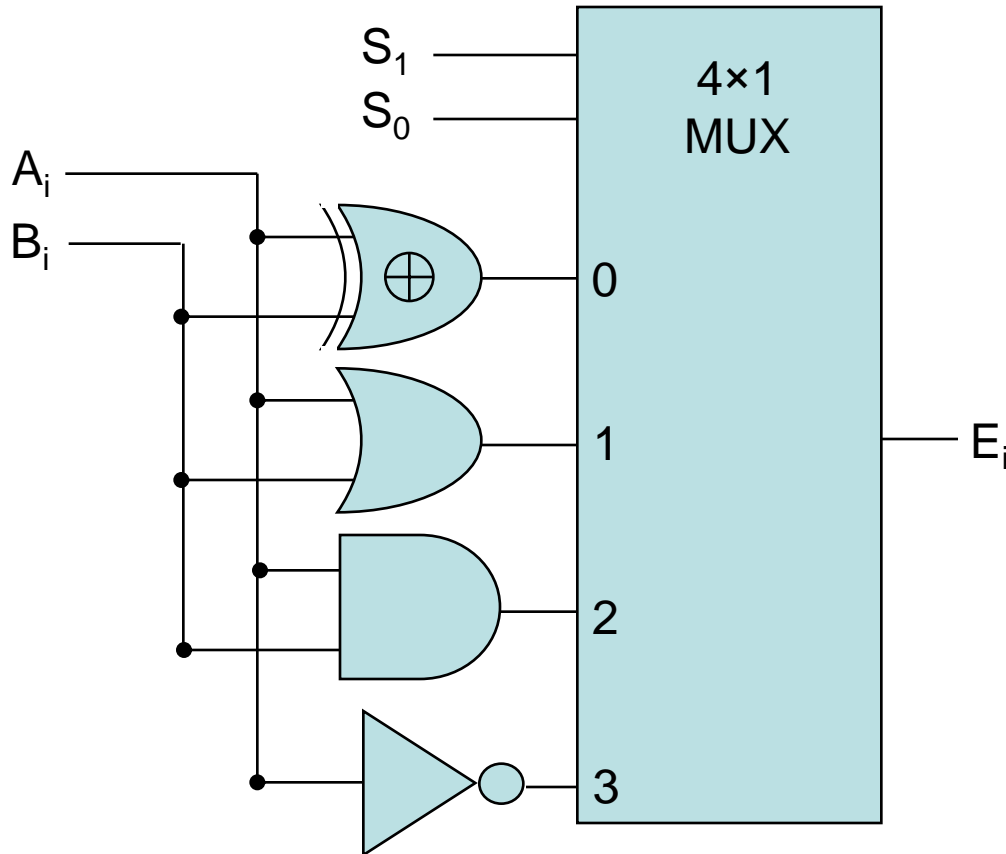
- Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0
- Example: $100110_2 \wedge 000000_2 = 000000_2$

4-5 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.

4-5 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	$E = A$	Complement

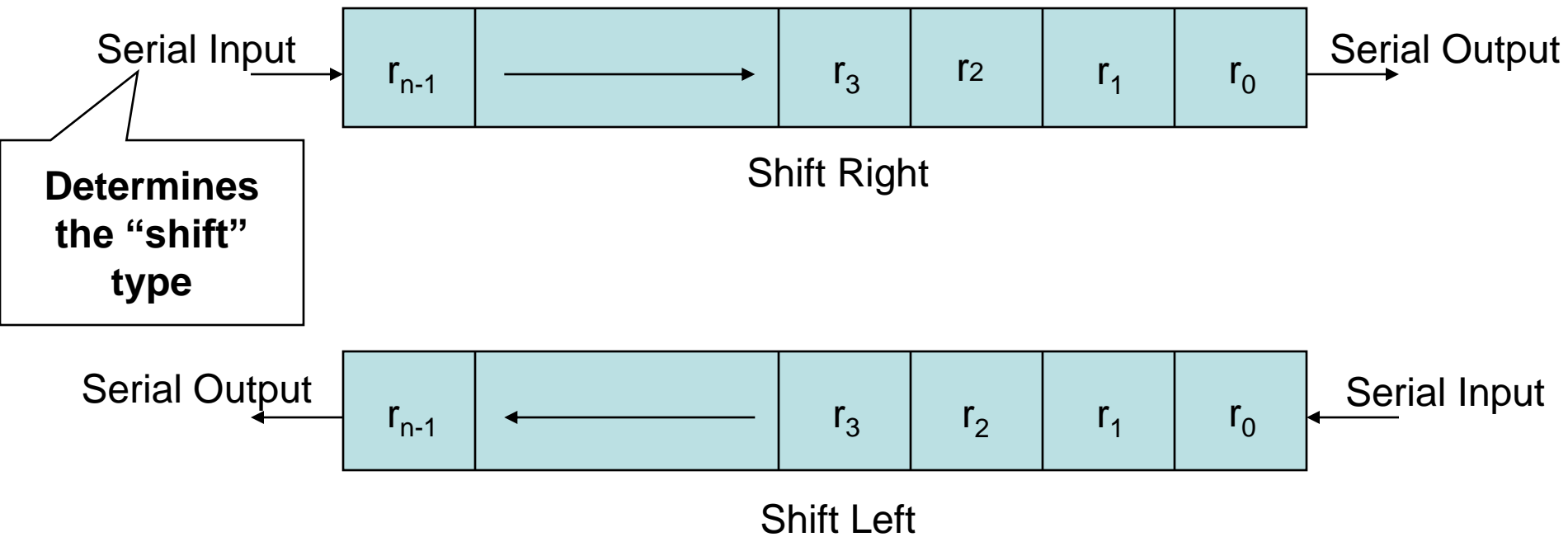
This is for one bit i

Figure B

4-6 Shift Microoperations

- Used for serial transfer of data
- Also used in conjunction with arithmetic, logic, and other data-processing operations
- 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

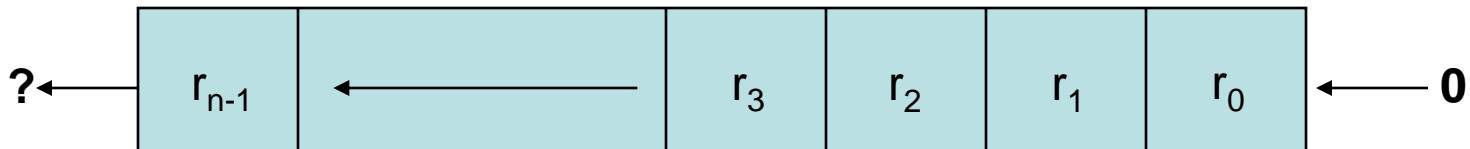
4-6 Shift Microoperations ^{cont.}



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

4-6 Shift Microoperations: Logical Shifts

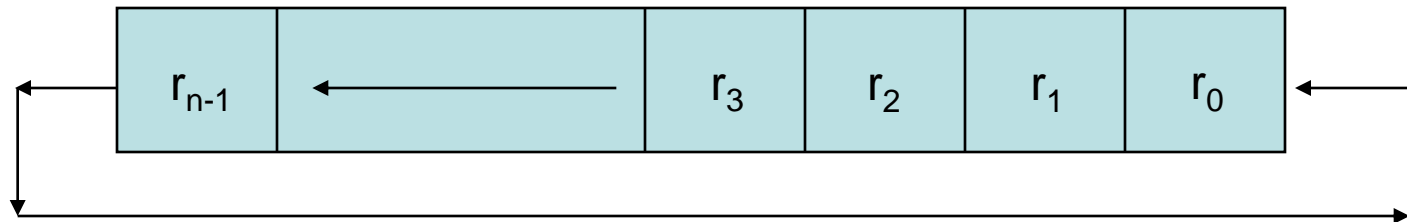
- Transfers 0 through the serial input
- Logical Shift Right: $R1 \leftarrow \text{shr } R1$
The same
- Logical Shift Left: $R2 \leftarrow \text{shl } R2$
The same



Logical Shift Left

4-6 Shift Microoperations: Circular Shifts (Rotate Operation)

- Circulates the bits of the register around the two ends without loss of information
- Circular Shift Right: $R1 \leftarrow \text{cir } R1$
The same
- Circular Shift Left: $R2 \leftarrow \text{cil } R2$
The same



Circular Shift Left

4-6 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`
- An overflow may occur in arithmetic shift-left, and occurs when the sign bit is changed (sign reversal)

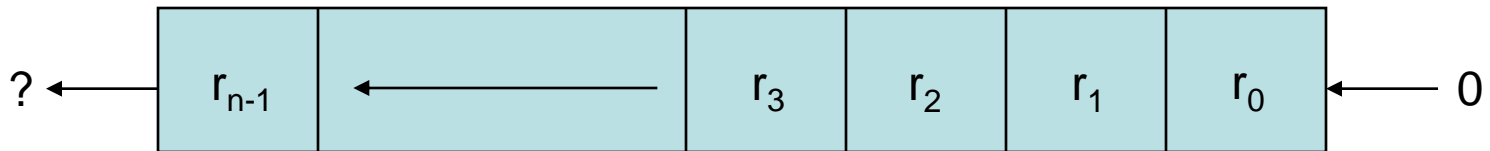
4-6 Shift Microoperations

Arithmetic Shifts ^{cont.}



Sign
Bit

Arithmetic Shift Right



Sign
Bit

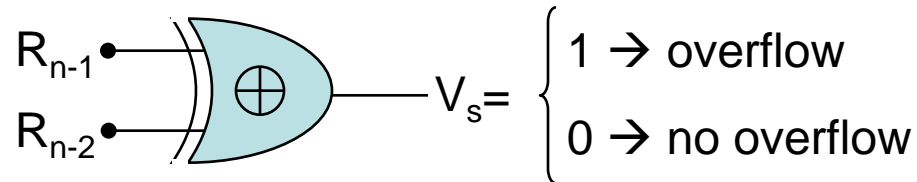
Arithmetic Shift Left

4-6 Shift Microoperations

Arithmetic Shifts ^{cont.}

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

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



4-6 Shift Microoperations ^{cont.}

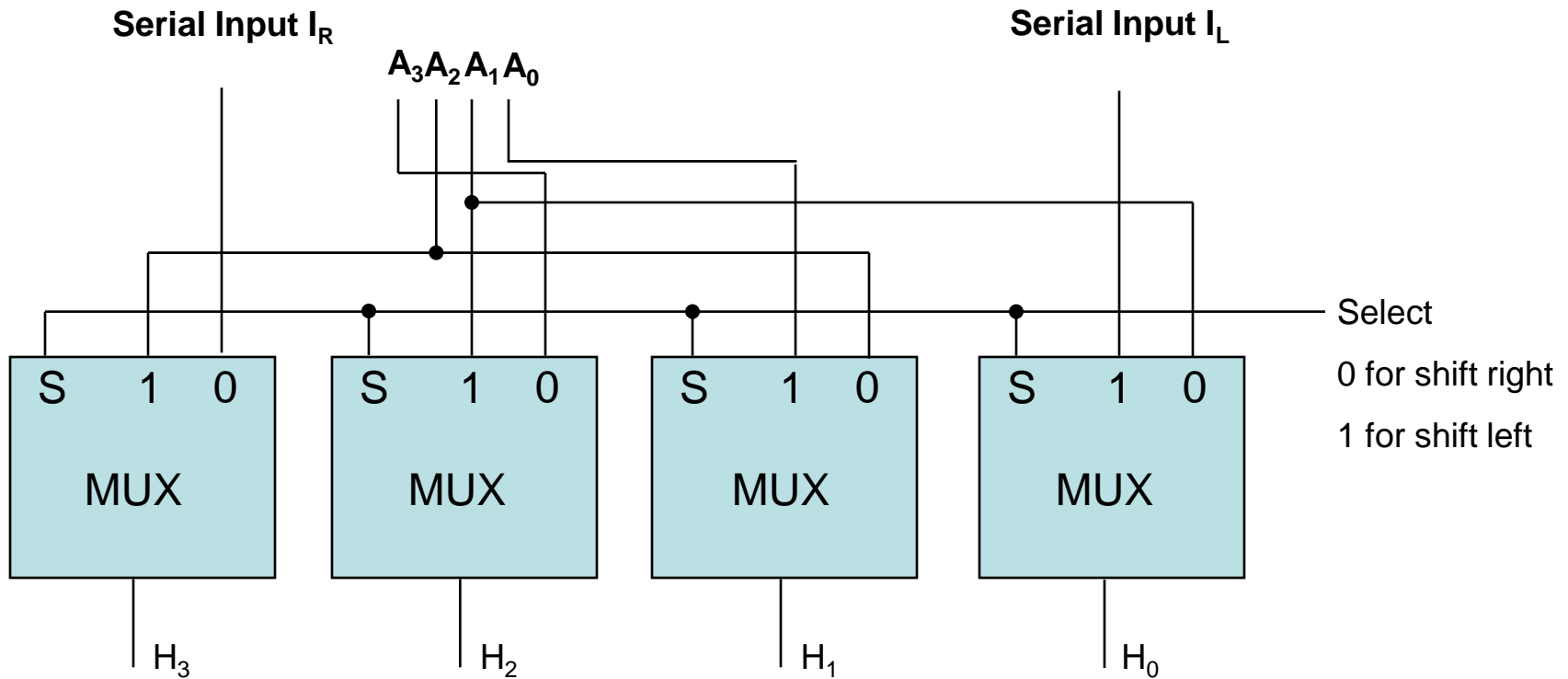
- Example: Assume $R1=11001110$, then:
 - Arithmetic shift right once : $R1 = 11100111$
 - Arithmetic shift right twice : $R1 = 11110011$
 - Arithmetic shift left once : $R1 = 10011100$
 - Arithmetic shift left twice : $R1 = 00111000$
 - Logical shift right once : $R1 = 01100111$
 - Logical shift left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

4-6 Shift Microoperations

Hardware Implementation ^{cont.}

- A possible choice for a shift unit would be a bidirectional shift register with parallel load (refer to Fig 2-9). Has drawbacks:
 - Needs two pulses (the clock and the shift signal pulse)
 - Not efficient in a processor unit where multiple number of registers share a common bus
- It is more efficient to implement the shift operation with a combinational circuit

4-6 Shift Microoperations Hardware Implementation cont.



4-bit Combinational Circuit Shifter

4-7 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**)

4-7 Arithmetic Logic Shift Unit cont.

