جامعة بيرزيت
BIRZEIT UNIVERSITY

ENCS238: Computer Organization

Mano - Chapter 4

Register Transfer and Microoperations

Contents

What is Register?

- **A special, high-speed storage area within the CPU.**
- **Registers are normally measured by the number of bits they can hold,**
  - for example, an "8-bit register" or a "32-bit register".
- **All data must be represented in a register before it can be processed.**
  - For example, if two numbers are to be multiplied, both numbers must be in registers, and the result is also placed in a register.
- **The register can contain the address of a memory location where data is stored rather than the actual data itself**

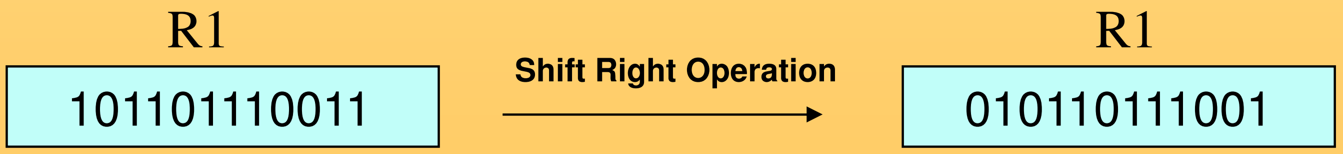
4-1 Register Transfer Language (RTL)

- **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 $^{\text{cont.}}$

- **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.

- **RTL:** a symbolic notation to describe the microoperation transfers among registers

Next steps:

- Define symbols (Codes) for various types of microoperations
- Describe the hardware that implements these microoperations

4-2 Register Transfer

- **Computer registers are designated by Capital Letters (sometimes followed by Numerals) to denote the function of the register**

  R1: processor register

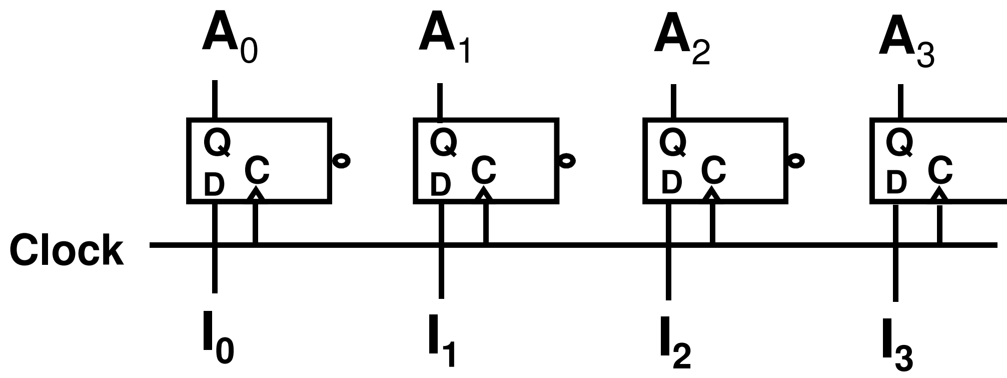
  MAR: Memory Address Register (holds an address for a memory unit)

  PC: Program Counter

  IR: Instruction Register

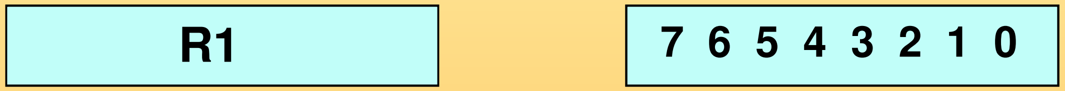
  SR: Status Register

Digital Review: 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)



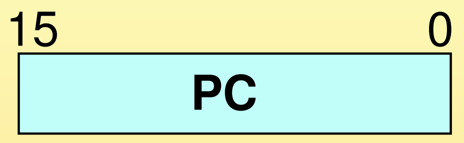
Register R1

Showing individual bits

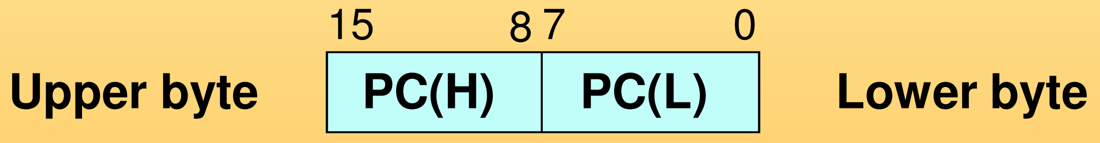
A block diagram of a register

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 $^{\text{cont.}}$

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

  P: $R2 \leftarrow R1$

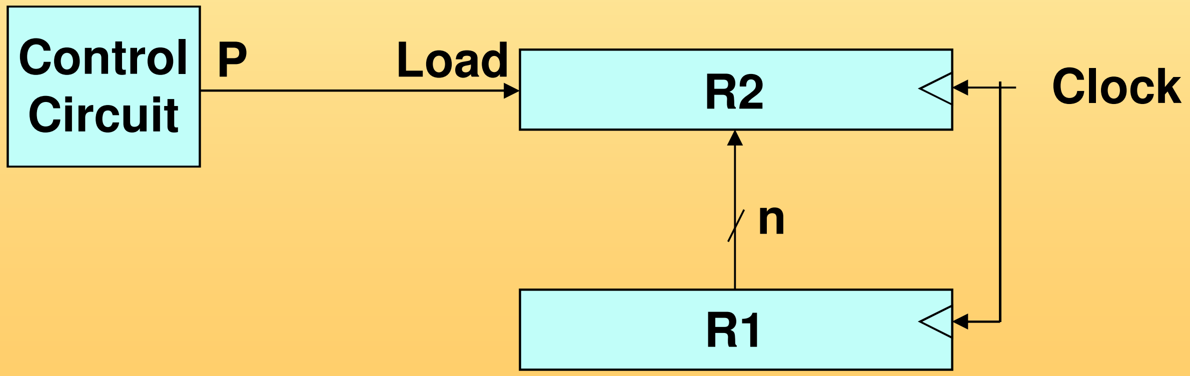
- A binary condition (P equals to 0 or 1) determines when the transfer occurs
- The content of R1 is transferred into R2 only **if P is 1**

4-2 Register Transfer cont.

Hardware implementation of a controlled transfer:

P: R2 ← R1

Block diagram:



4-2 Register Transfer $^{\text{cont.}}$

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 $\leftarrow$	Denotes transfer of information	R2 $\leftarrow$ R1
Comma ,	Separates two microoperations	R2 $\leftarrow$ R1 , R1 $\leftarrow$ R2

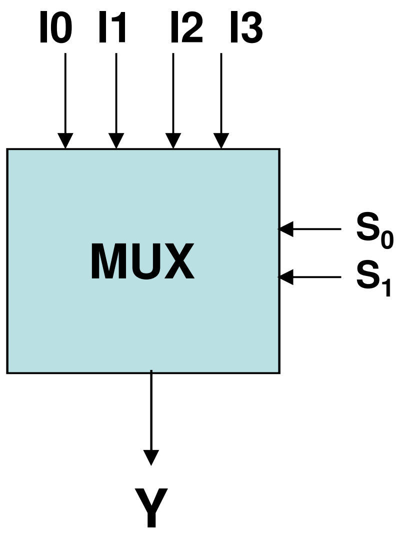
4-3 Bus and Memory Transfers

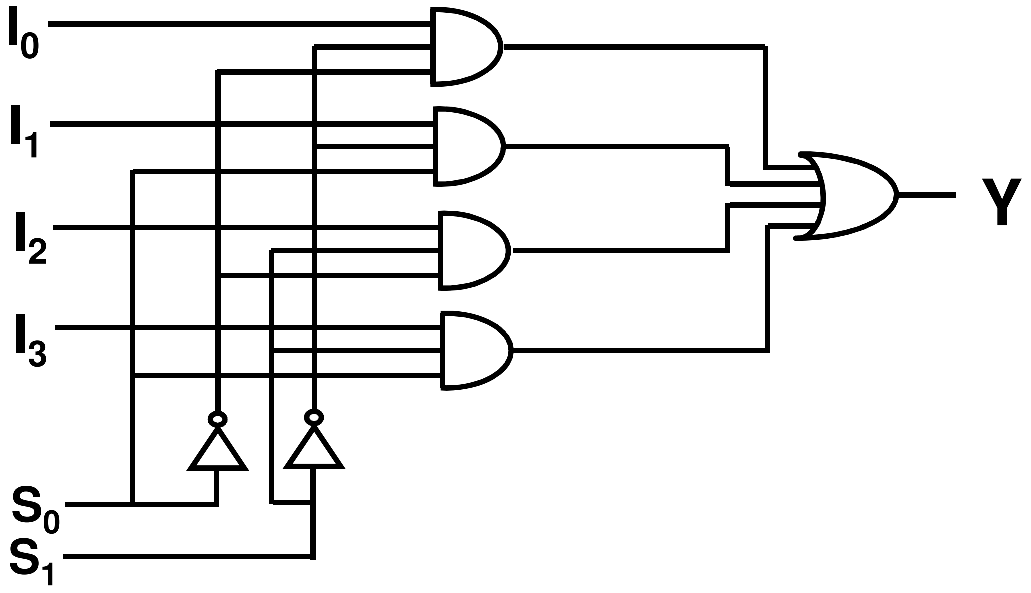
- **Paths** must be provided to transfer information from one register to another
- A **Common Bus System** is a scheme for transferring information between registers in a multiple-register configuration
- **A bus**: set of common lines, one for each bit of a register, through which binary information is transferred one at a time
- **Control signals** determine which register is selected by the bus during each particular register transfer

Digital Review: MULTIPLEXER

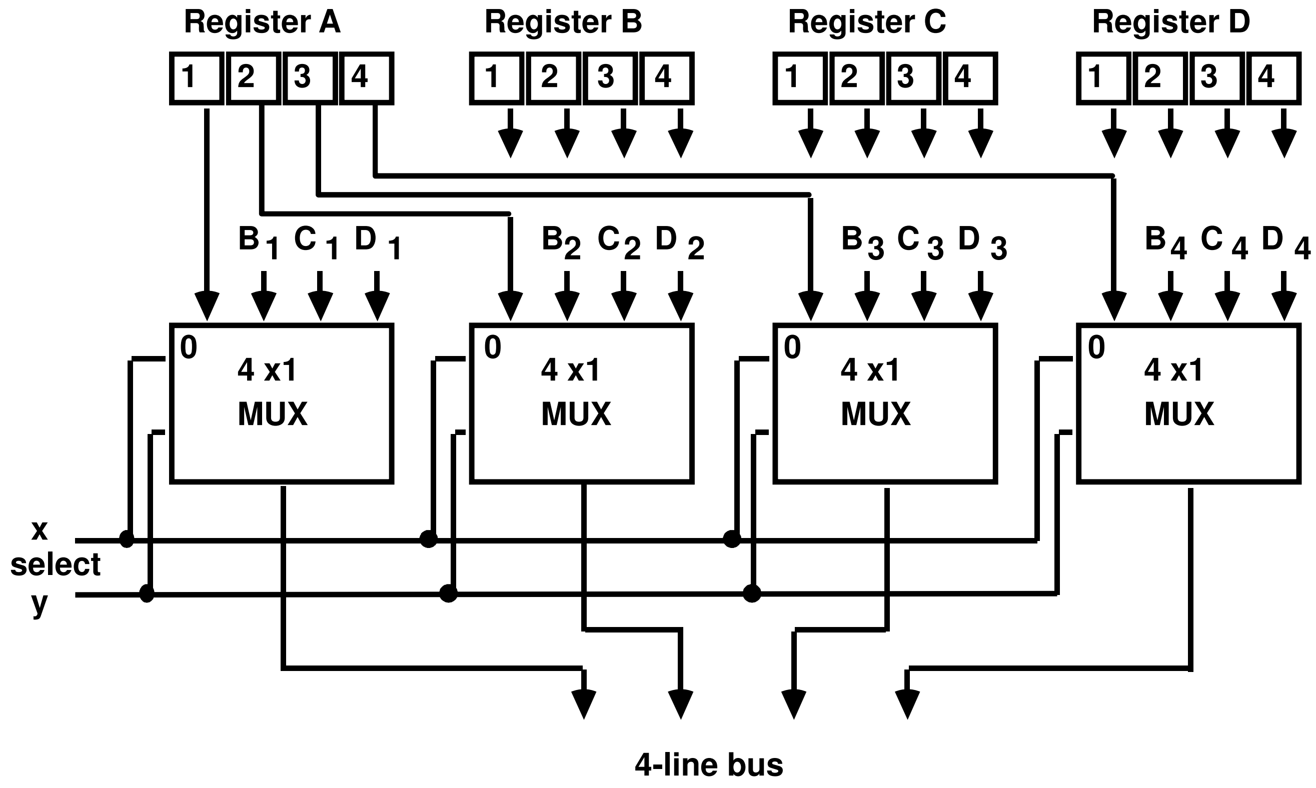
4-to-1 Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3





4-3 Bus and Memory Transfers



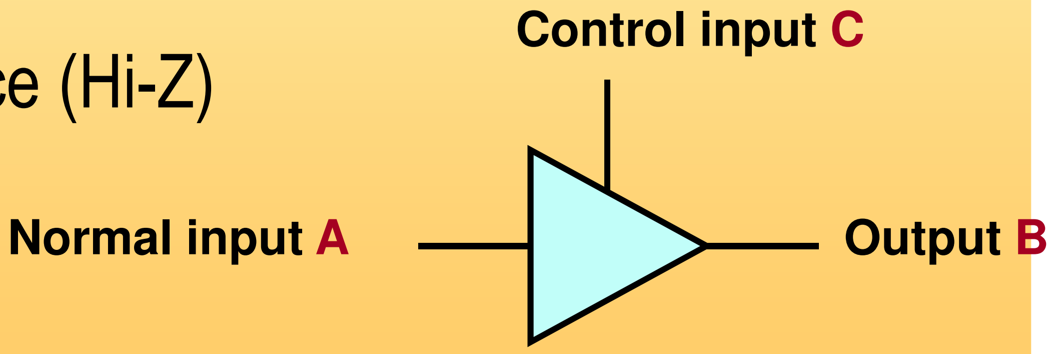
4-3 Bus and Memory Transfers

- **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)

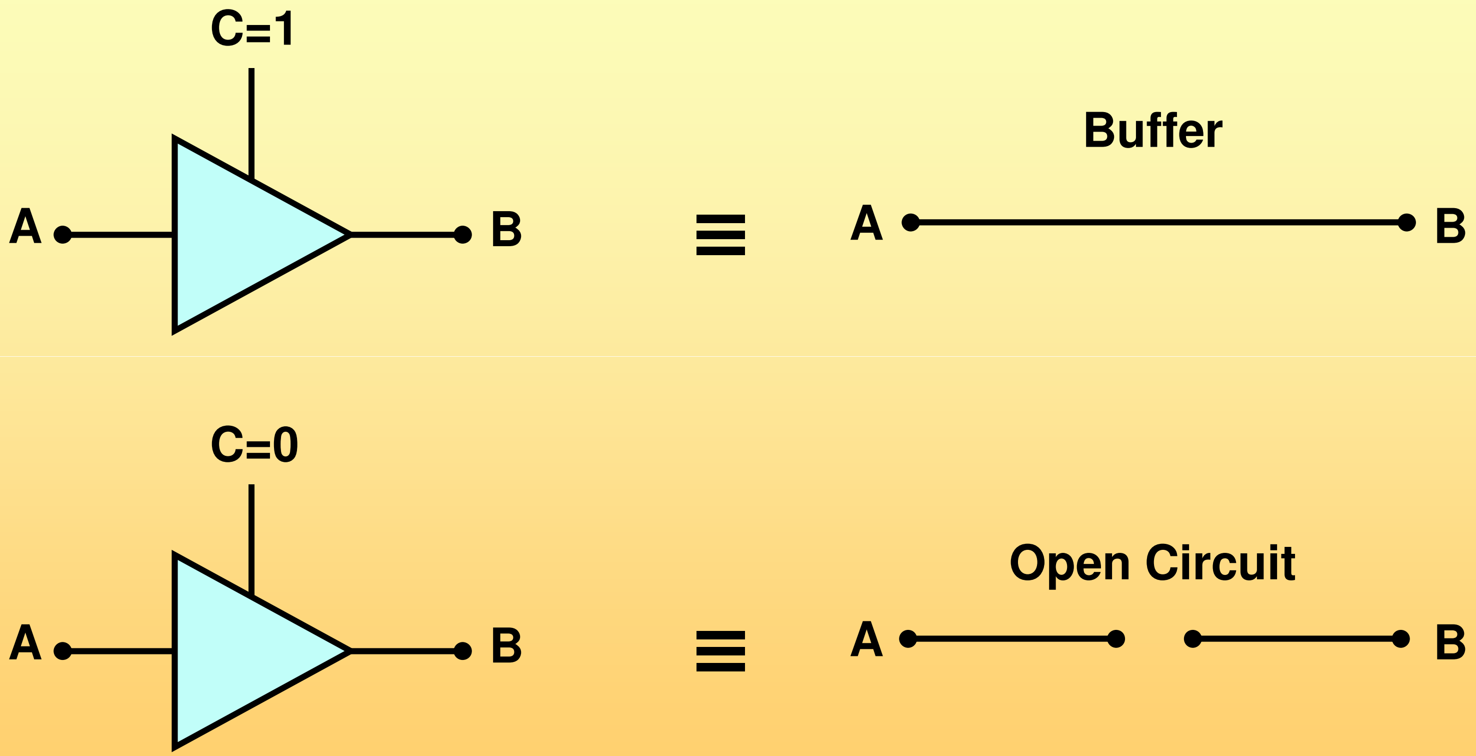
Three-State Bus Buffers

- 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

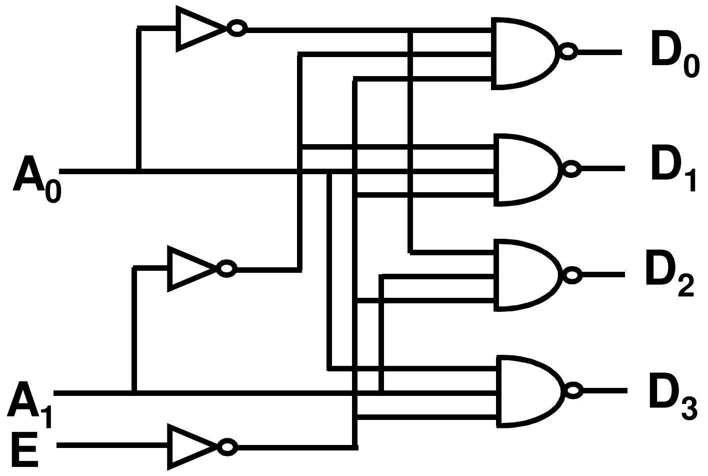
Three-State Bus Buffers cont.



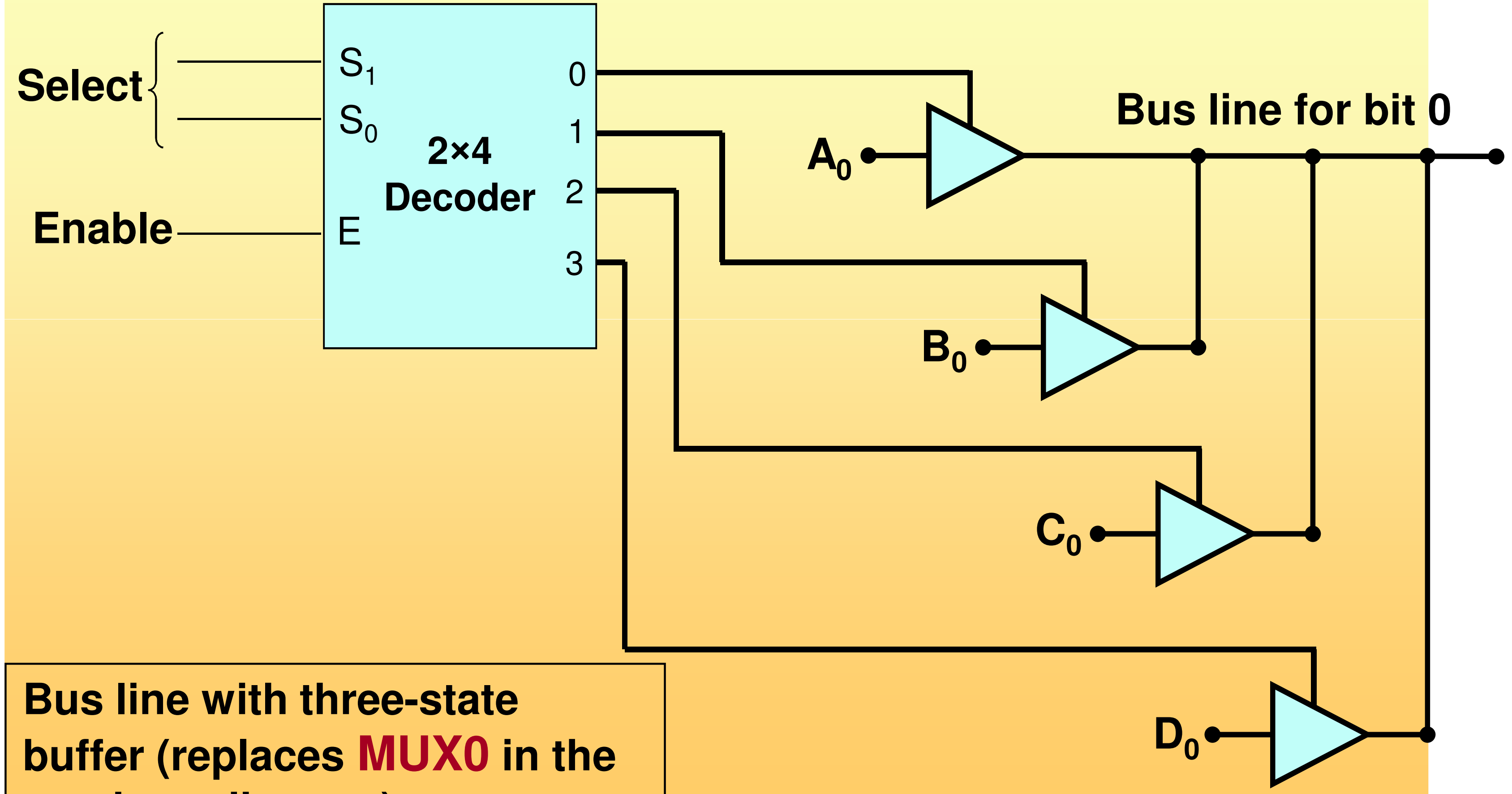
Digital Review **DECODER**

2-to-4 Decoder

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	**0**	1	1	1
0	0	1	1	**0**	1	1
0	1	0	1	1	**0**	1
0	1	1	1	1	1	**0**
1	X	x	1	1	1	1



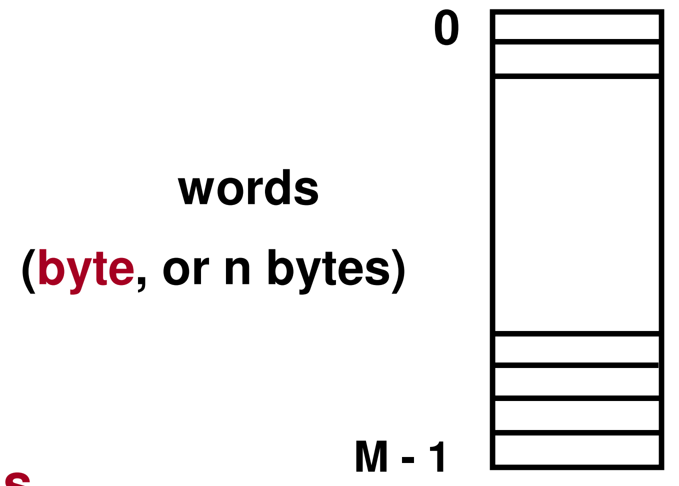
Three-State Bus Buffers cont.



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

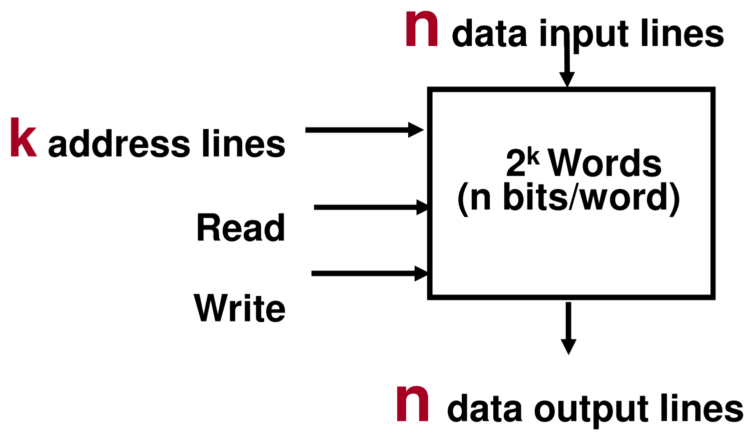
Digital Review: **MEMORY COMPONENTS**

Logical Organization



Random Access Memory

- Each word has a unique address
- Access to a word requires the same time independent of the location of the word



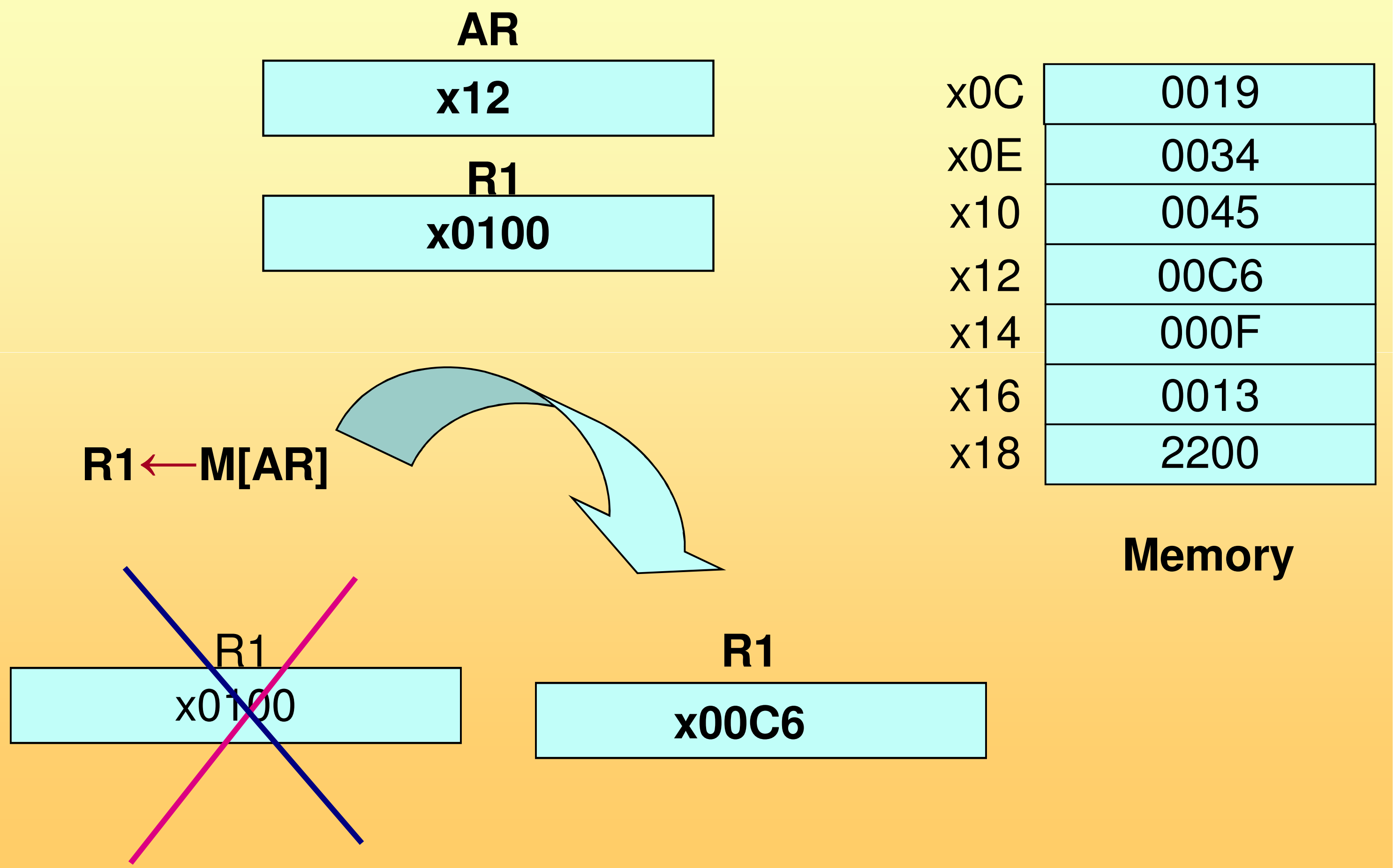
Memory Transfer

- **Memory read** : Transfer **from** memory
- **Memory write** : Transfer **to** memory
- Data being read or wrote is called a **Memory Word** (called **M**)
- 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 **0X**0016

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:
  - **R**ead: **$DR \leftarrow M[AR]$**
  - **W**rite: **$M[AR] \leftarrow DR$**

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:**

$$R3 \leftarrow R1 + R2$$

- **Subtraction Microoperation:**

$$R3 \leftarrow R1 - R2$$

1's Complement

or :

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

4-4 Arithmetic Microoperations $^{\text{cont.}}$

- **One's Complement Microoperation:**

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

- **Two's Complement Microoperation:**

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

- **Increment Microoperation:**

$$R2 \leftarrow R2+1$$

- **Decrement Microoperation:**

$$R2 \leftarrow R2-1$$

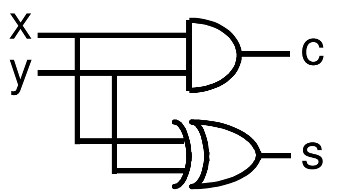
Digital Review **Half Adder/Full Adder**

Half Adder

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

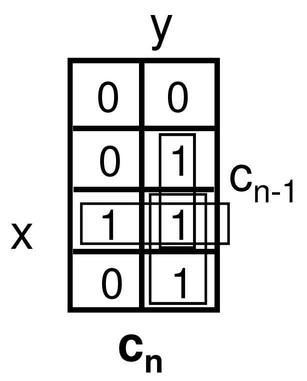
$\mathbf{c} = xy$

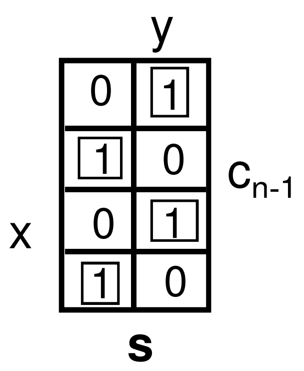
$\mathbf{s} = xy' + x'y$
$= x \oplus y$



Full Adder

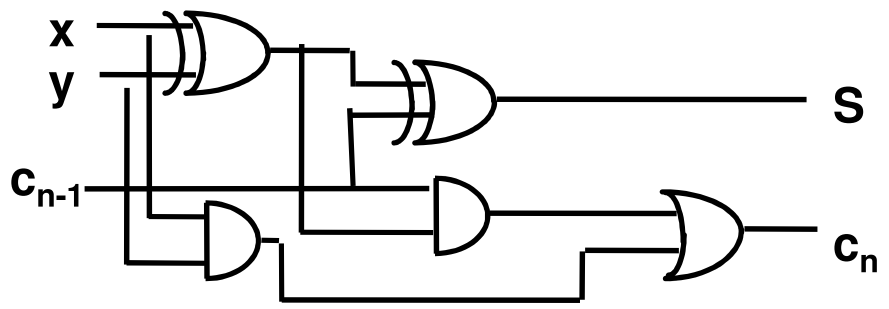
x	y	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



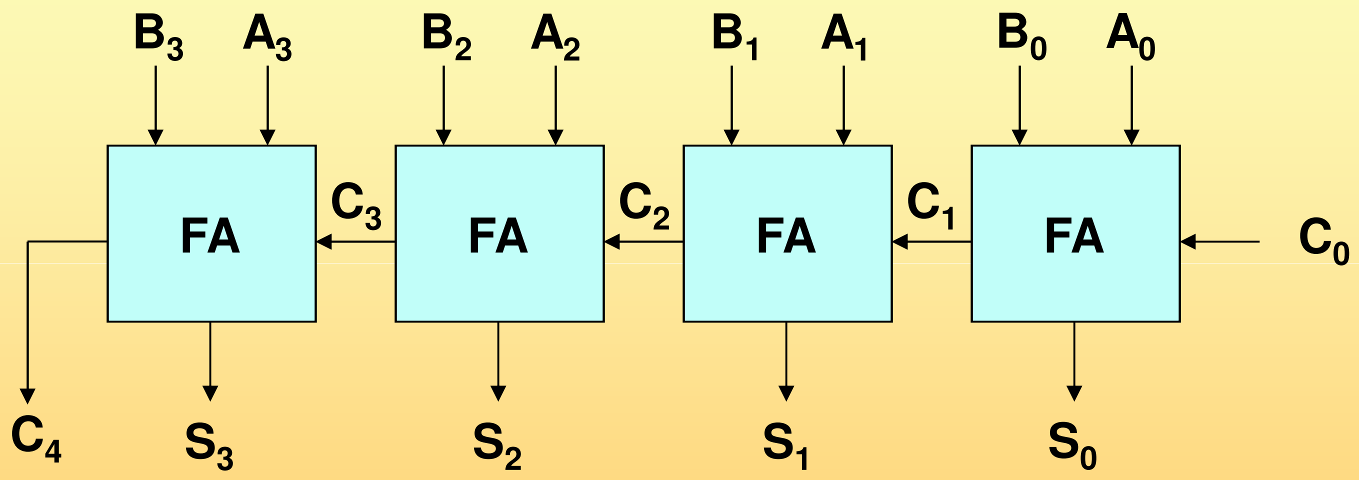


$\mathbf{c_n} = xy + xc_{n-1} + yc_{n-1}$
$= xy + (x \oplus y)c_{n-1}$

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

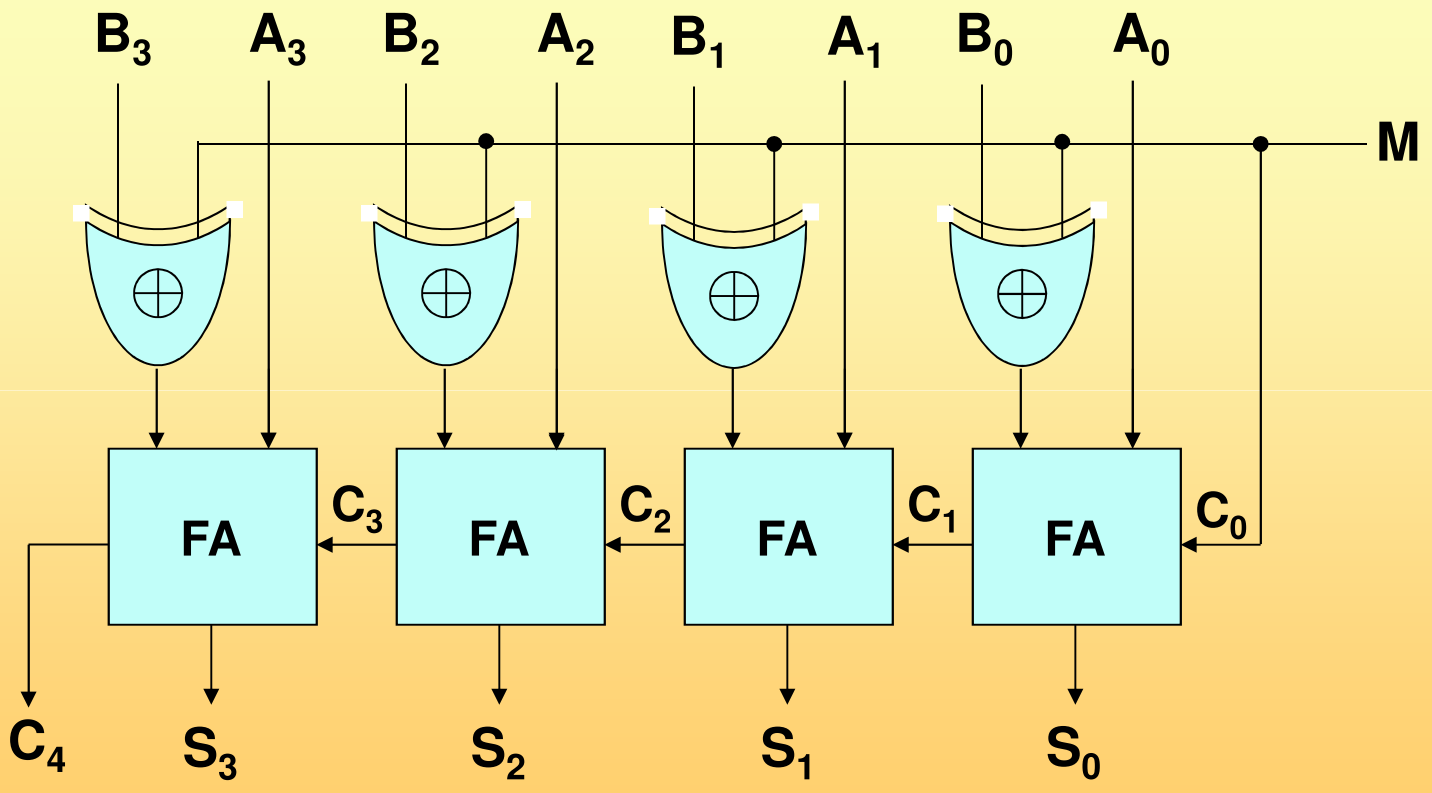


Binary Adder



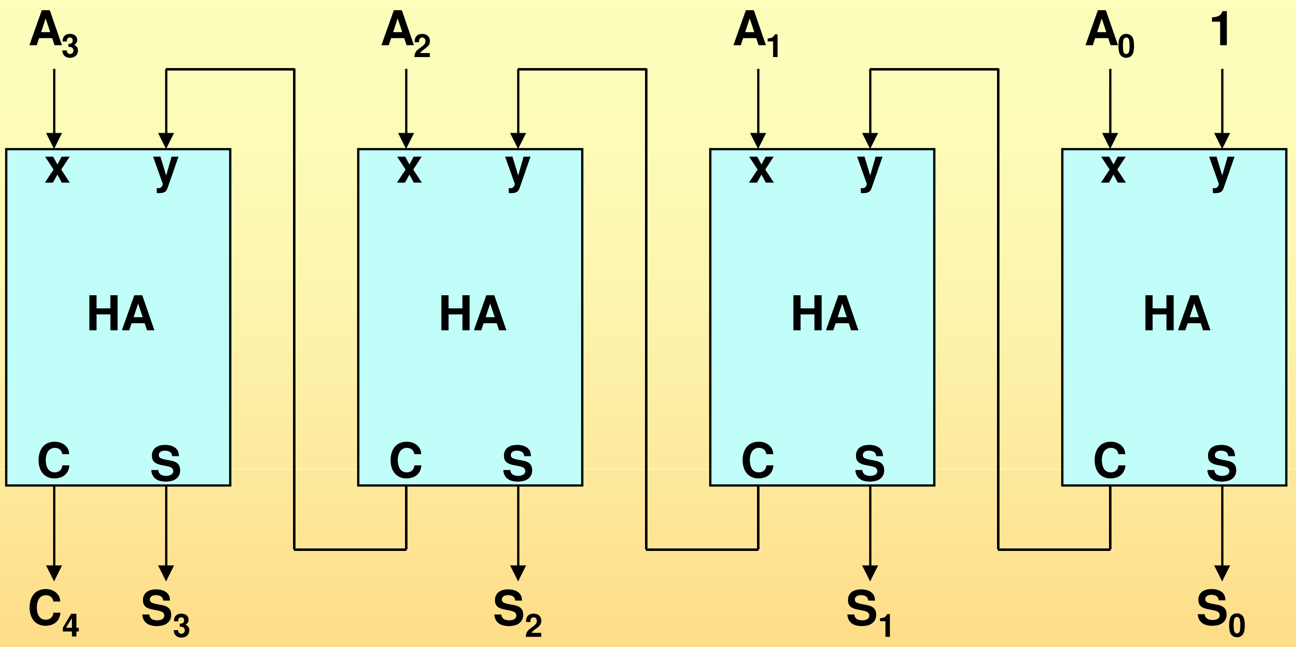
4-bit binary adder
(connection of FAs)

Binary Adder-Subtractor



4-bit adder-subtractor

Binary Incrementer



4-bit Binary Incrementer

- Binary Incrementer can also be implemented using a **Counter**
- A binary decrementer can be implemented by adding **1's** to the desired register each time!

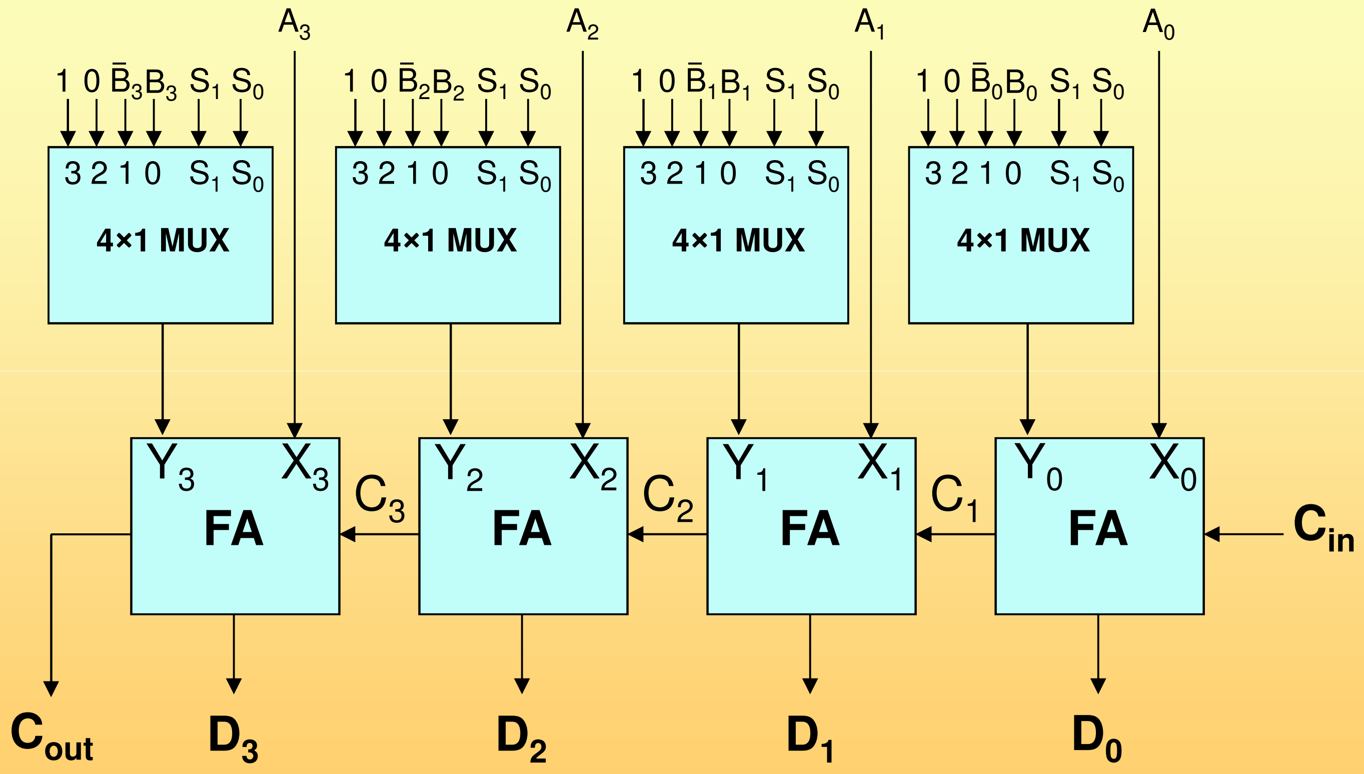
Arithmetic Circuit

- 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:

$$D = A + Y + C_{in}$$

Arithmetic Circuit cont.

Figure A



4-bit Arithmetic Circuit

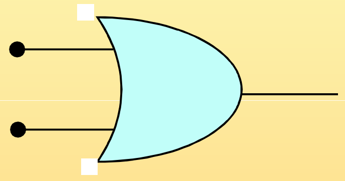
Arithmetic Circuit $^{\text{cont.}}$

	Select			Input **Y**	Output **D = A + Y + Cin**	Microoperations
	S1	**S0**	**Cin**			
1	0	0	0	B	D = A + B	Add
2	0	0	1	B	D = A + B + 1	Add with carry
3	0	1	0	$\bar{B}$	D = A + $\bar{B}$	Sub. With borrow
4	0	1	1	$\bar{B}$	D = A + $\bar{B}$ + 1	Sub
5	1	0	0	0	D = A	Transfer A
6	1	0	1	0	D = A + 1	Increment
7	1	1	0	1	D = A - 1	Decrement
8	1	1	1	1	D = A	Transfer A

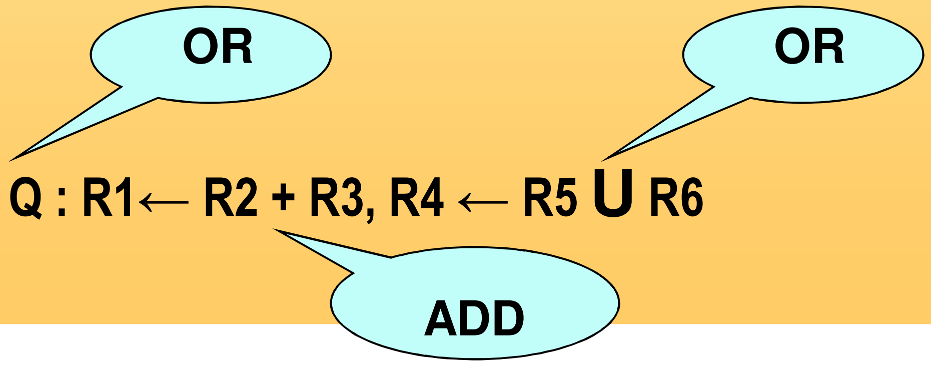
The 4 basic Logic microoperations

OR Microoperation

- Symbol: $\cup$, $+$
- Gate:



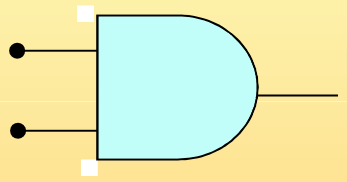
- Example: $0100110_2 \cup 1010110_2 = 1110110_2$



$P + Q : R1 \leftarrow R2 + R3,\ R4 \leftarrow R5 \cup R6$

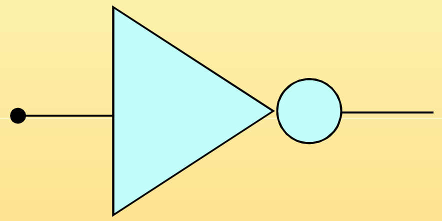
The 4 basic logic microoperations cont.

AND Microoperation

- Symbol: ∩ , .
- Gate: 
- Example: $100110_2 \cap 1010110_2 = 0000110_2$

The 4 basic logic microoperations $^{\text{cont.}}$

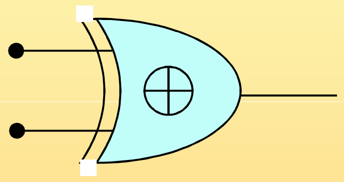
Complement (NOT) Microoperation

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

The 4 basic logic microoperations cont.

XOR (Exclusive-OR) Microoperation

- Symbol: $\oplus$
- Gate:



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

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 **U** $1000_2 = 1100_2$

Selective-complement (toggling) Operation

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

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

Other Logic Microoperations cont.

Insert Operation

- Step1: mask the desired bits (**AND**)
- Step2: **OR** them with the desired value

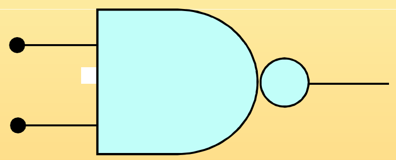
- Example: suppose **R1 = 0110 1010**, and we desire to replace the leftmost **4** bits (**0110**) with **1001** then:
  - Step1: **0110** 1010 ∩ **0000** 1111
  - Step2: **0000** 1010 U **1001** 0000

  → **R1 = 1001 1010**

4-5 Logic Microoperations
Other Logic Microoperations cont.

NAND Microoperation

- Symbols: $\cap$ and $\overline{\quad}$
- Gate:

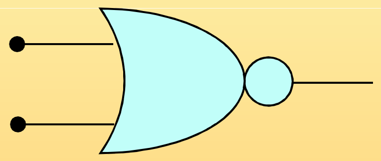


- Example: $\overline{100110_2 \cap 1010110_2} = 1111001_2$

4-5 Logic Microoperations
Other Logic Microoperations $^{\text{cont.}}$

NOR Microoperation

- Symbols: $\cup$ and $\overline{}$
- Gate:



- Example: $\overline{100110_2 \cup 1010110_2} = 0001001_2$

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: $100110_2 \cup 111111_2 = 111111_2$

Clear (Reset) Microoperation

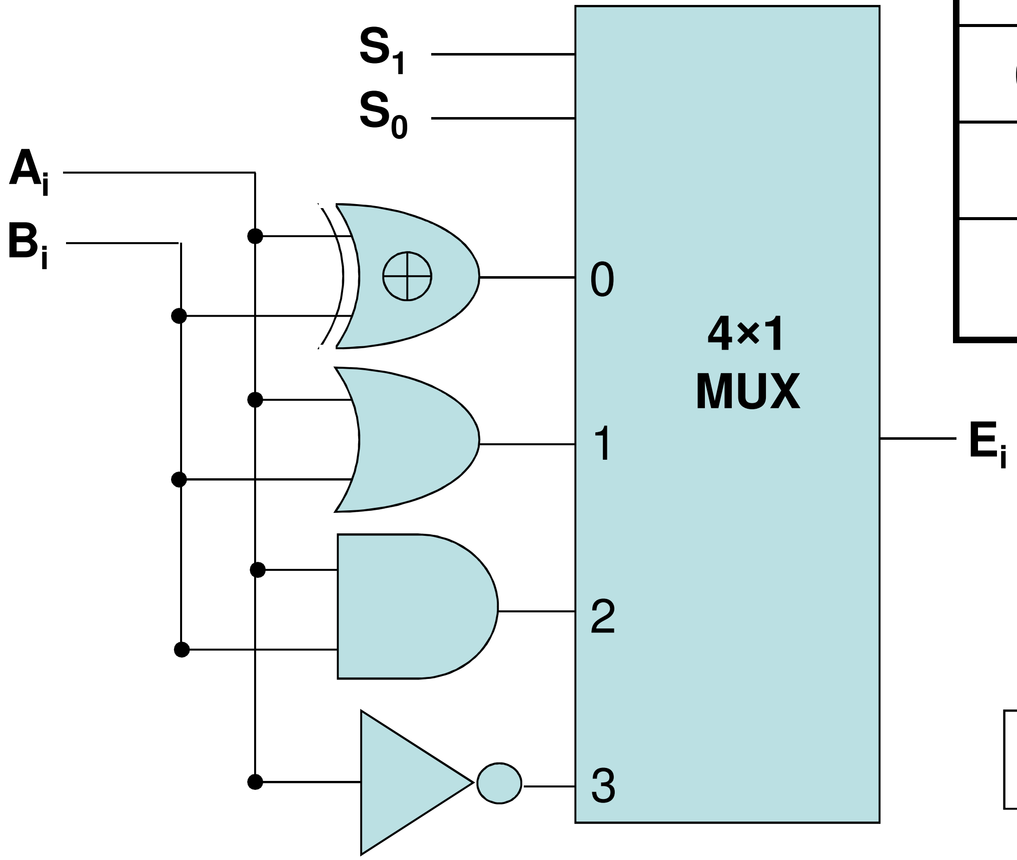
- Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0
- Example: $100110_2 \cap 000000_2 = 000000_2$

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 $^{\text{cont.}}$

Figure B



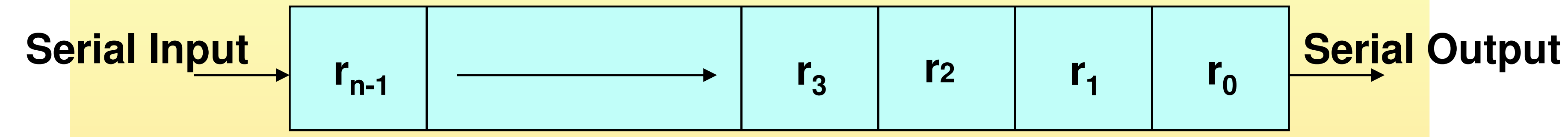
S_1	S_0	Output	Operation
0	**0**	$E = A \oplus B$	**XOR**
0	**1**	$E = A \cup B$	**OR**
1	**0**	$E = A \cap B$	**AND**
1	**1**	$E = \overline{A}$	**Complement**

This is for one bit i

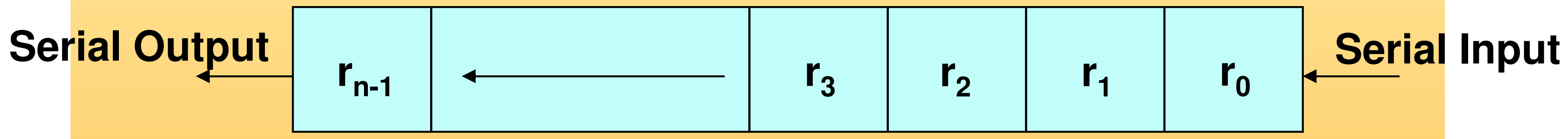
4-6 Shift Microoperations

- 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.



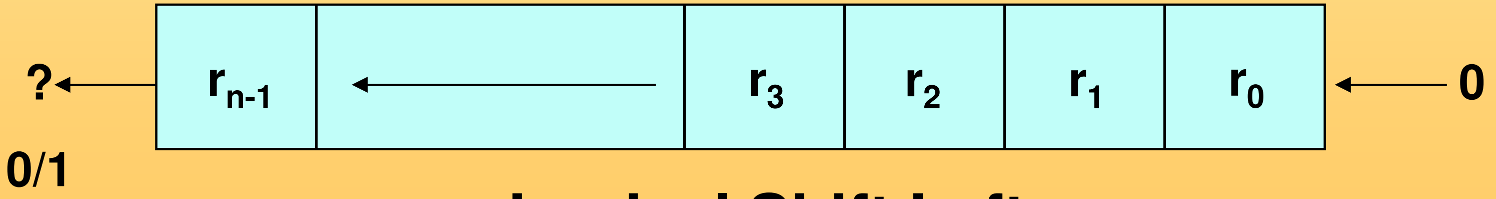
Shift Right



Shift Left

Logical Shifts

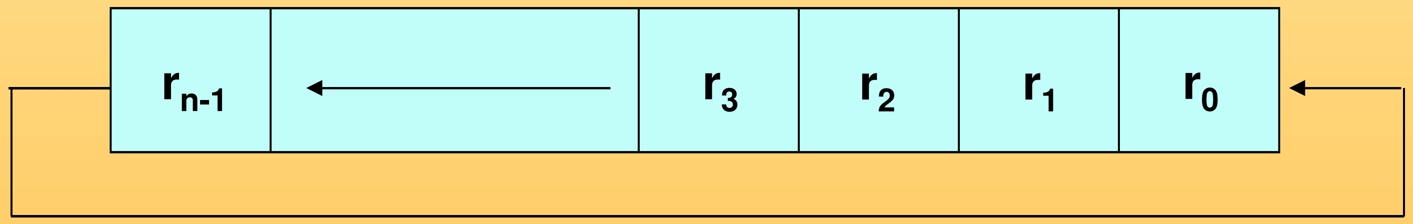
- Transfers 0 through the serial input
- Logical Shift Right: R1 $\leftarrow$ shr R1
- Logical Shift Left: R2 $\leftarrow$ shl R2



Logical Shift Left

Circular Shifts (Rotate Operation)

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



Circular Shift Left

Arithmetic Shifts

- Shifts a signed binary number to the left or right
- An arithmetic shift-left Multiplies a signed binary number by 2

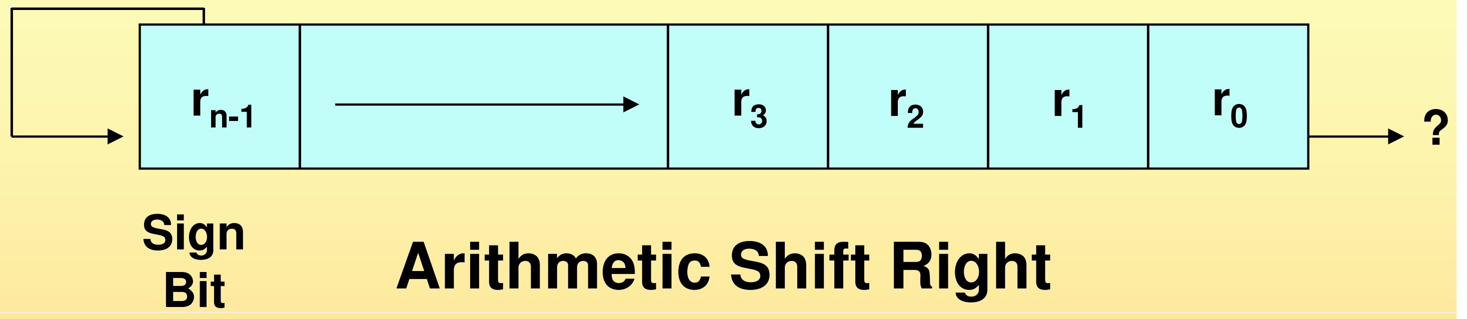
  ashl (00100): 01000

- An arithmetic shift-right Divides the number by 2

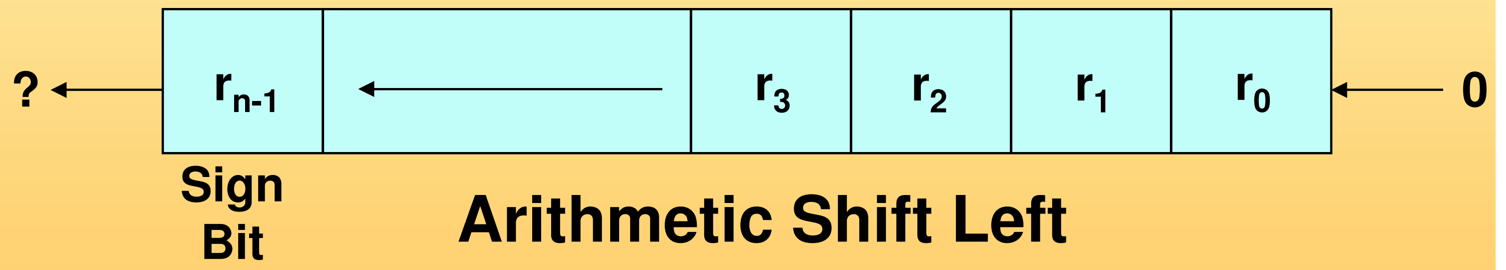
  ashr (00100) : 00010

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

Arithmetic Shifts cont.



Arithmetic Shift Right

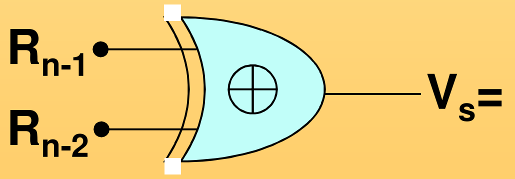


Arithmetic Shift Left

Arithmetic Shifts $^{\text{cont.}}$

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

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

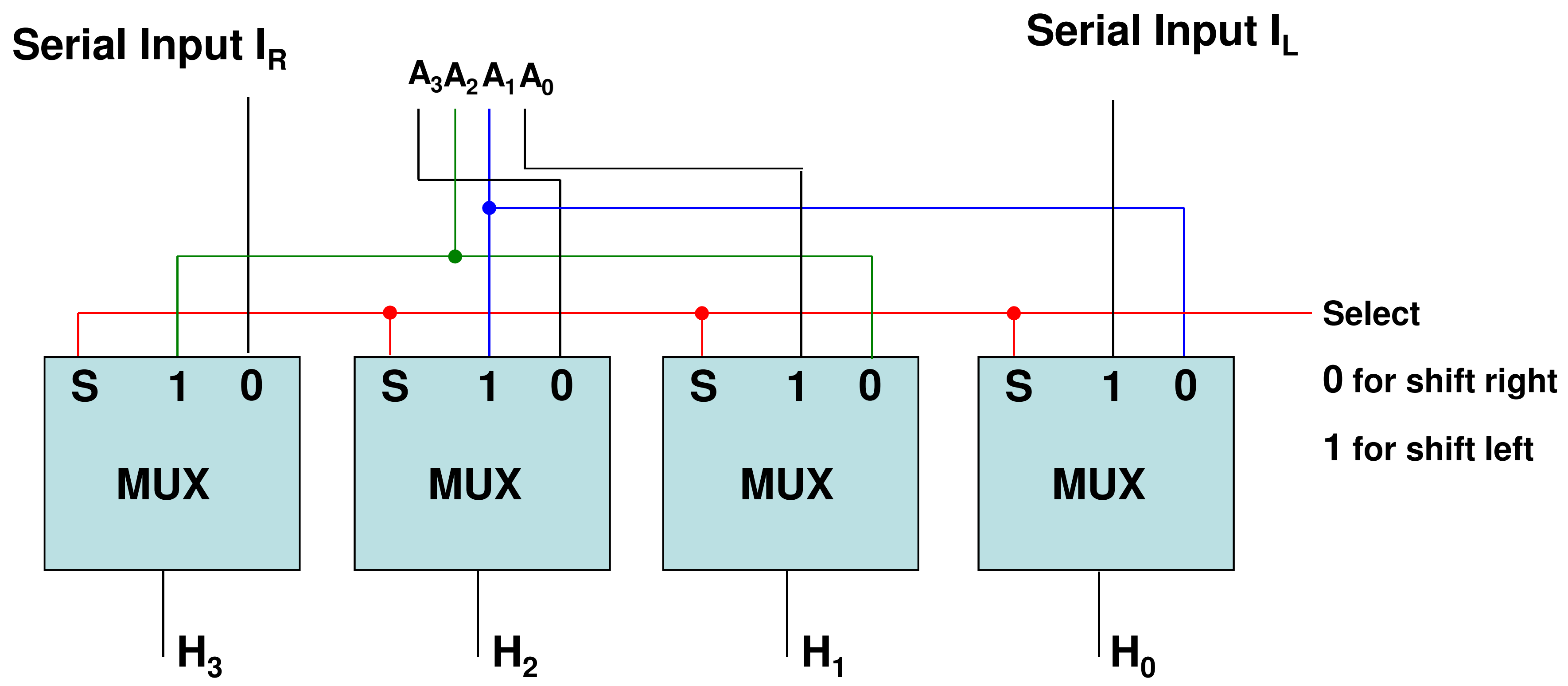


$$\begin{cases} 1 \rightarrow \text{overflow} \\ 0 \rightarrow \text{no overflow} \end{cases}$$

Hardware Implementation cont.

- **A possible choice for a shift unit would be a bidirectional shift register with parallel load**
- **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**

Hardware Implementation cont.

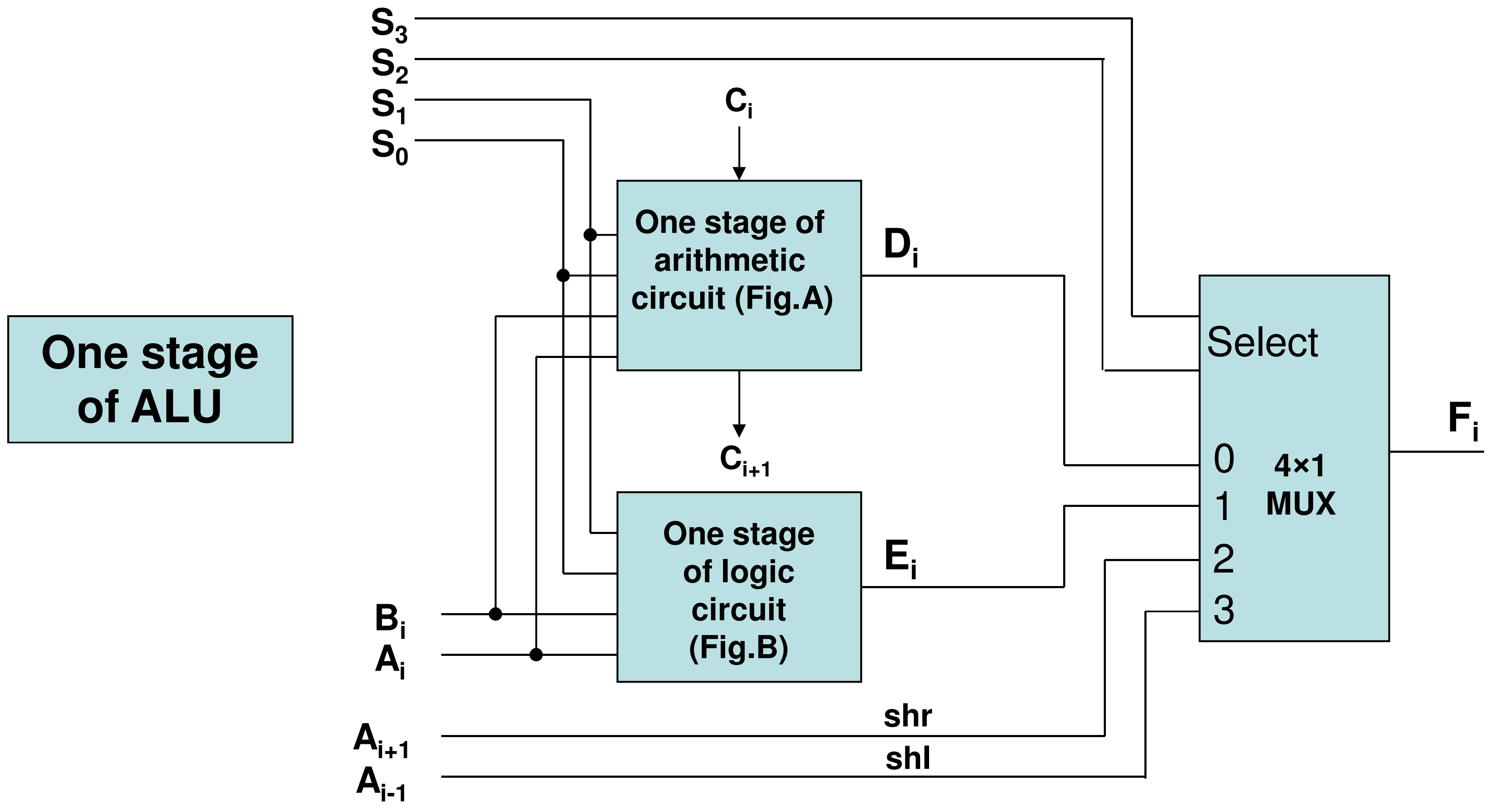


4-bit Combinational Circuit Shifter

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.



4-7 Arithmetic Logic Shift Unit cont.

Operation Selection					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A + B$	Addition
0	0	0	0	1	$F = A + B + 1$	Addition with carry
0	0	0	1	0	$F = A + \overline{B}$	Sub. With borrow
0	0	0	1	1	$F = A + \overline{B} + 1$	Sub
0	0	1	0	0	$F = A$	Transfer A
0	0	1	0	1	$F = A + 1$	Increment
0	0	1	1	0	$F = A - 1$	Decrement
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	X	$F = A \oplus B$	XOR
0	1	0	1	X	$F = A \cup B$	OR
0	1	1	0	X	$F = A \cap B$	AND
0	1	1	1	X	$F = \overline{A}$	Complement
1	0	X	X	X	F= shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left into F