OF THE
DEPARTMENT OF
INFORMATION ENGINEERING

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Digital Systems

Registers and Microoperations

Marta Bagatin, marta.bagatin@unipd.it

Degree Course in Information Engineering

Academic Year 2023-2024

Purpose of the Lesson

- So far, we have seen combinational circuits and sequential circuits, now we will bring the two ideas together
  - In Chapter 4 we studied sequential systems with a limited number of flip-flops. These were easily represented by a state diagram
- Now we will study more complex structures, partitioned into different modules, some of which consist of a number of similar or identical blocks repeated several times, in particular we will
  - Describe the structure of a register and how to load data
  - Define the datapath and control unit blocks and understand how they interact with each other in digital systems
  - Study the main types of data transfer operations between registers (microoperations) and how to express them in Register Transfer Language (RTL)

Registers and Counters: Definitions

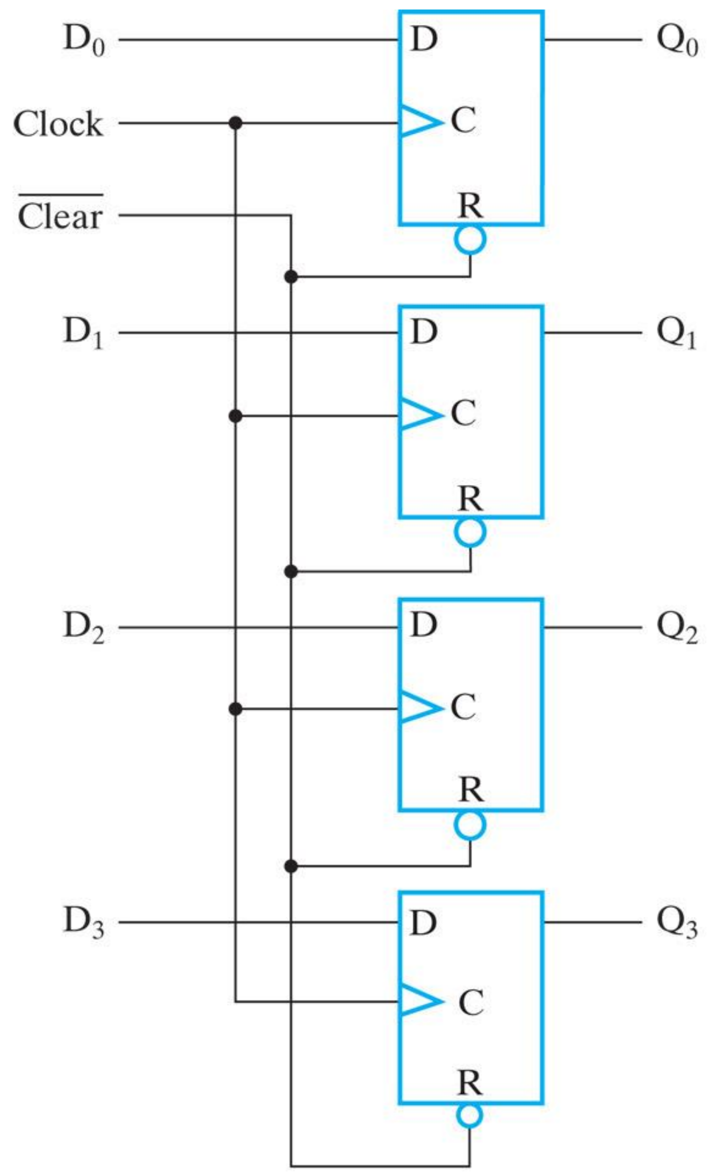
- Registers and counters are extensively used in digital systems
  - Registers are used to store information during the processing of data
  - Counters assist in sequencing the process

- An **n-bit register** consists of n flip-flops and a set of logic gates
  - Flip-flops store data, logic gates transform data to be transferred to flip-flops

- A **counter** is a particular type of register that passes through a predefined sequence of states

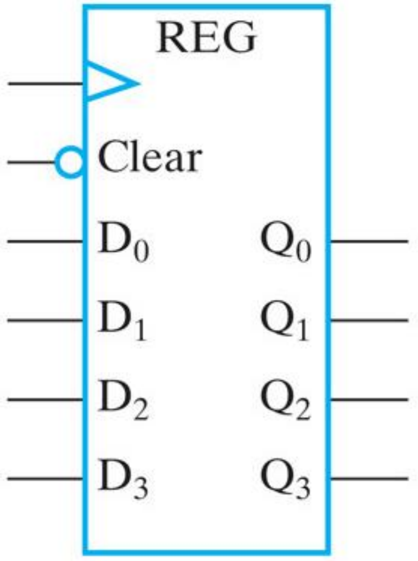
Registers

4-bit Register

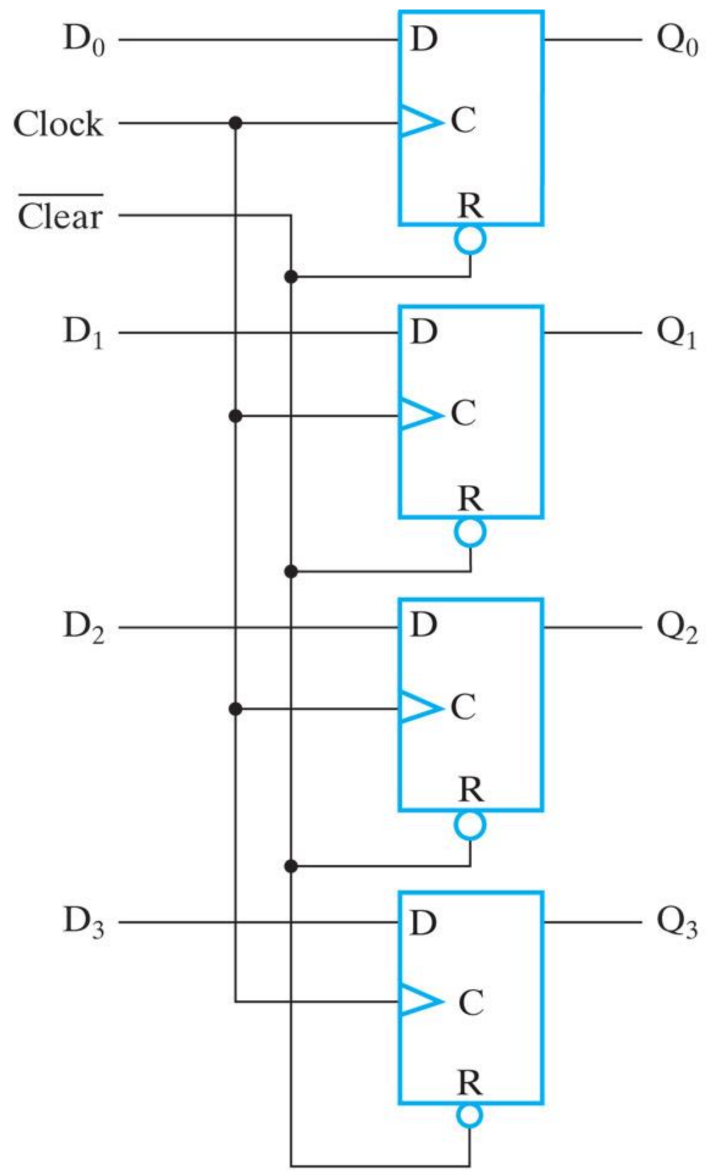


- The simplest register **consists of a set of flip-flops** (taken as a single entity) and does not contain any logic gates
  - Each of the 4 D flip-flops stores one bit

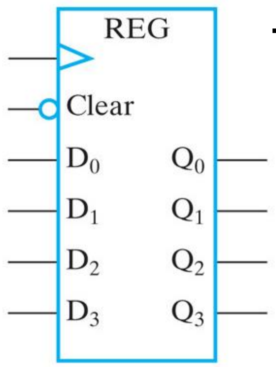
Logic symbol, with inputs on the left and outputs on the right



4-bit Register



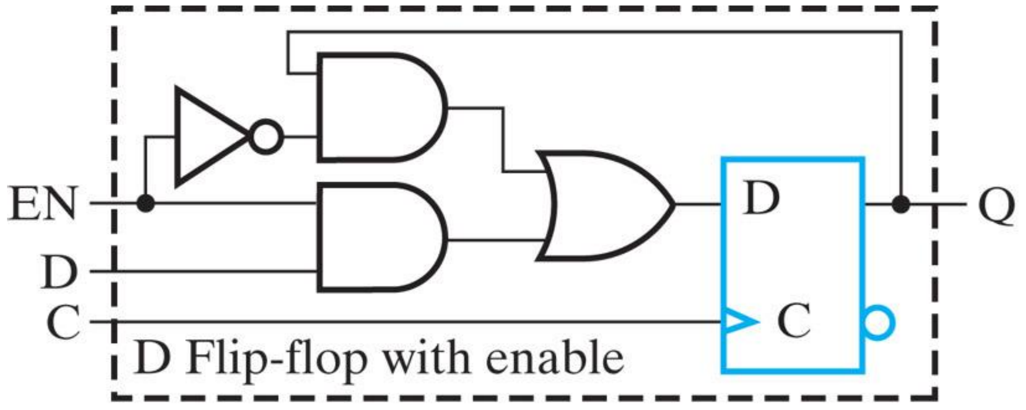
- The **clock is shared** between the 4 flip-flops: at the clock rising edge, data are transferred from the input to the output of the FFs
- The **transfer of new data** to the register is referred as **loading** the register. In this case loading is performed in parallel
- An optional signal of **common RESET** ($\overline{clear}$) allows us to clear all outputs Q to 0 **asynchronously**



  – Being complemented (bubble), $\overline{clear}$ is 1 during normal operation. When $\overline{clear}$ is 0, all the bits in the register are reset ($\overline{clear}$ is an active-low signal)

Register with Load Control: Load Enable

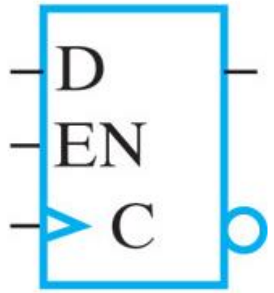
- Load **control** is needed if we want to update the content of the flip-flops only at specific times, and not at each clock cycle
- In particular, the loading of data into a register can be controlled by an **enable signal** (**Load EN**)
- The circuit consists of a 2-to-1 MUX and a D-FF: the EN signal selects the value to be transmitted to the FF input (either D or the present FF output)



  - **if EN = 1: D is selected**

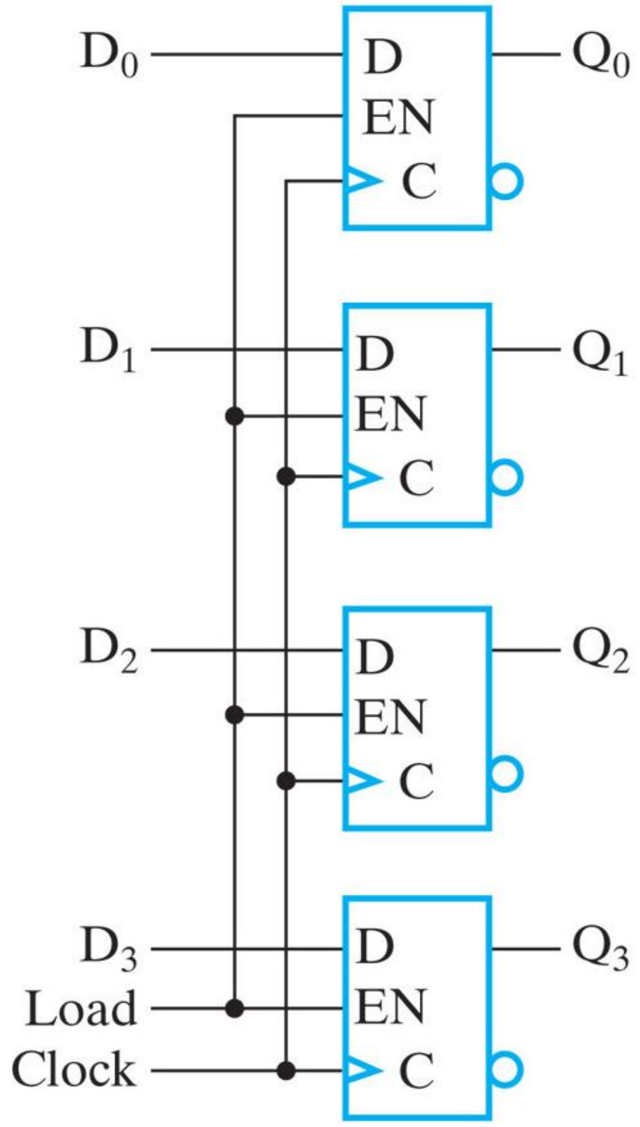
  → the register is loaded with data D at the next active edge of the clock
  - **if EN = 0: Q is selected**

  → the register is loaded with its output Q, i.e. it is not updated and keeps the previous value



Symbol of **D Flip-flop with enable (EN)**

Register with Load Enable



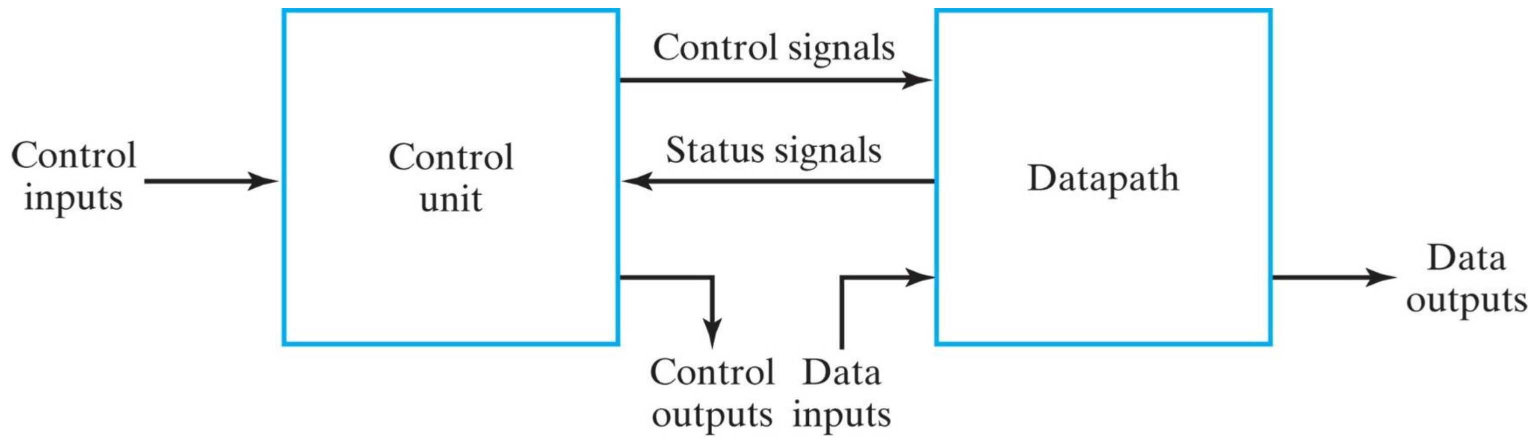
- The register is made with **4 D flip-flops with EN in parallel**, controlled by the same clock signal
- The clock is always applied to the C inputs of the flip-flops. It is the Load signal which determines whether new values will be loaded into the register at the next rising edge of the clock or whether the previous value will be kept

Partitioning a Digital System into Datapath and Control Unit

Datapath and Control Unit

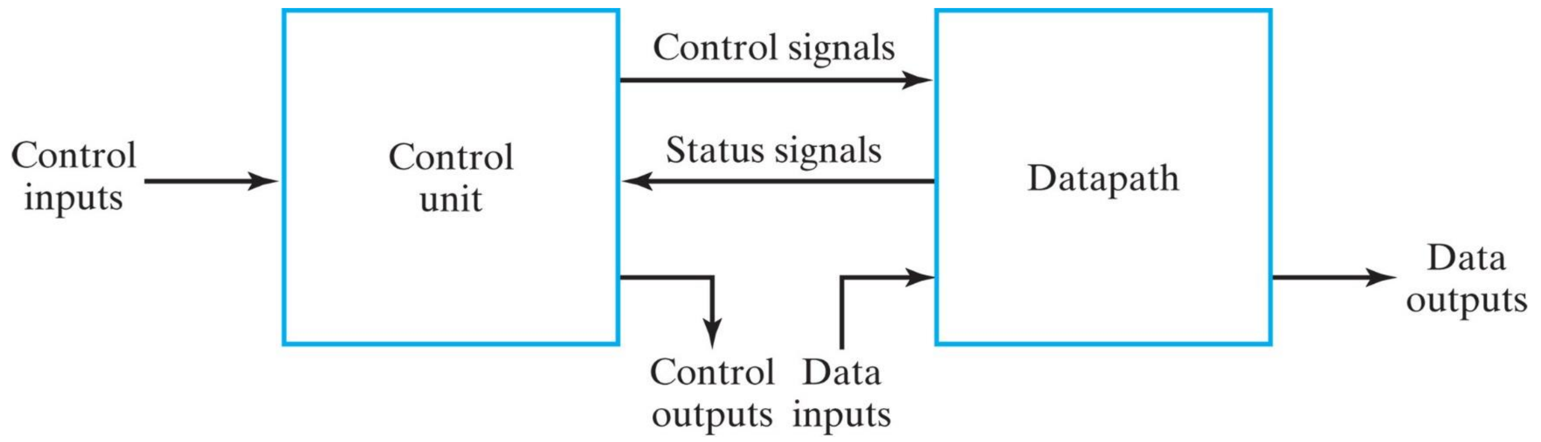
- We have seen that a sequential system can be represented with a state diagram (or state table)
- In systems with a large number of states, specifying the behavior of the circuit with a state diagram/table can be very difficult, if not impossible
- It is convenient to use a modular, **hierarchical approach**, **partitioning the system into sub-blocks**, each with its own function

Datapath and Control Unit



- Complex digital systems can be usually partitioned into two types of modules: a **datapath performs data-processing operations** (arithmetic, logic, …), and a **control unit determines the sequence of those operations**
  - Control unit sends to datapath control signals, which determine the data processing operations
  - Datapath, in turn, sends to control unit status signals, which describe the datapath status and on the basis of which the control unit determines the sequence of operations to be activated
- Datapath and control unit can also communicate with other system blocks, such as memory and I/O

Datapath and Control Unit



- The **control unit** can be thought of as a **synchronous sequential machine** that, based on the control inputs and the status signals, determines the operations to be performed by the datapath, generating appropriate control signals
- The **datapath** is constituted by **registers and blocks performing operations on the data stored in the registers**. The datapath receives the control signals produced by the control unit as inputs, in addition to data inputs. The outputs of the datapath are data outputs and status signals (that inform the control unit on the status/outcome of operations)
- Data transfer operations between registers and elementary data operations that occur in the datapath are referrred to as **microoperations**

Programmable and Non-programmable Systems

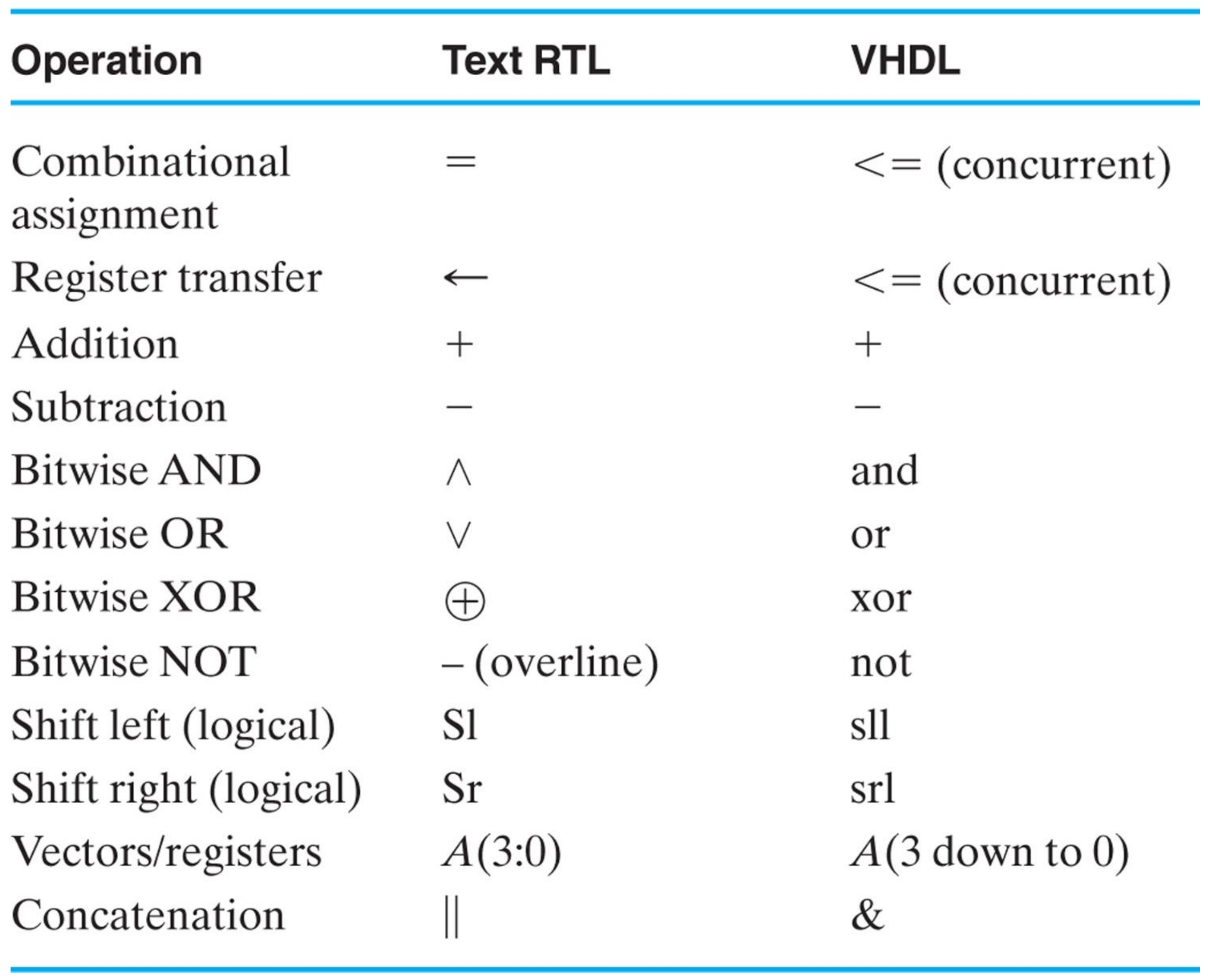
- In **programmable systems** the inputs of the control unit contain an instruction flow (program)
  - Each instruction activates an appropriate sequence of microoperations to be given to the datapath. The instructions specify the microoperations to be performed, with which data, and where the results are to be stored
  - The program is typically stored in a memory (RAM or ROM). The control unit contains a register, program counter (PC), which specifies the address of the next instruction (to be taken from the memory at the next clock cycle)
  - Example: CPU
- In **non-programmable systems**, the control unit does not execute instructions contained in a program, but determines the microoperations only on the basis of the input signals and the status signals of the datapath
  - The system is designed to perform a single task
  - Example: state machine in which the state update and the calculation of the outputs is carried out using a fixed logic (for example the chips used in digital clocks)

Microoperations

Microoperations and Register Transfer Language

- An **elementary operation performed on data stored in a register** is called a microoperation
  - Example: incrementing the content of a register, loading the content of a register into another register, adding the content of two registers
- The result of the microoperation can either replace the value present in the register itself or be transferred to another register, leaving the content of the first unchanged
- The task of the **control unit is to provide the control signals to direct the sequence of microoperations performed by the datapath**
- **Register Transfer Language (RTL)** is the language to represent registers and specify the operations on their content (i.e. the language to describe microoperations)
  - Expression set resembling the statements used in HDLs

Register Transfer Language (RTL)

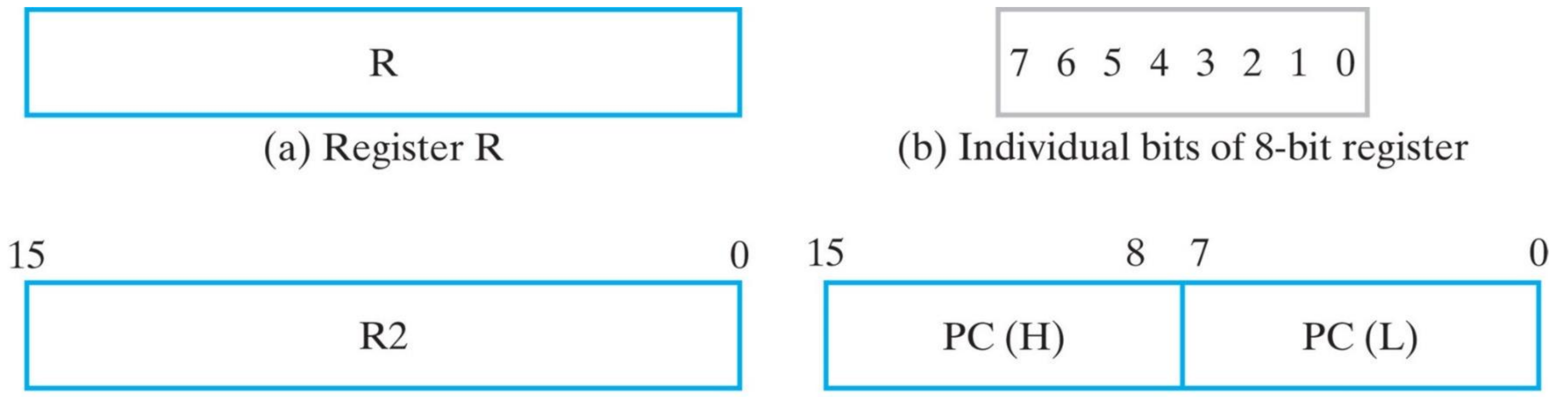


Operation	Text RTL	VHDL
Combinational assignment	=	<= (concurrent)
Register transfer	←	<= (concurrent)
Addition	+	+
Subtraction	−	−
Bitwise AND	∧	and
Bitwise OR	∨	or
Bitwise XOR	⊕	xor
Bitwise NOT	– (overline)	not
Shift left (logical)	Sl	sll
Shift right (logical)	Sr	srl
Vectors/registers	*A*(3:0)	*A*(3 down to 0)
Concatenation	||	&

The RTL expressions are similar but not identical to VHDL statements

Register Notations

- **Registers** are indicated with capital letters, sometimes followed by numbers (for example R2: Register 2, Address Register: AR, Program Counter: PC, etc.)
- The individual bits can be indicated in brackets after the name of the register
  - Sometimes they are divided into most significant (H) and least significant (L) bytes



(a) Register R (b) Individual bits of 8-bit register

(c) Numbering of 16-bit register (d) Two-part 16-bit register

Types of Microoperations

- Microoperations are the elementary **operations between data stored in the registers**
- The most frequent microoperations in a digital system can be classified into
  - **a)** **Data transfer** microoperations: transfer binary data between registers
  - **b)** **Arithmetic** microoperations: perform arithmetic operations on data contained in registers
  - **c)** **Logic** microoperations: perform bit manipulations on data contained in registers
  - **d)** **Shift** microoperations: shift data in registers
- A microoperation can be of more than one type (e.g. 1s complement is both a logic and an arithmetic microoperation)

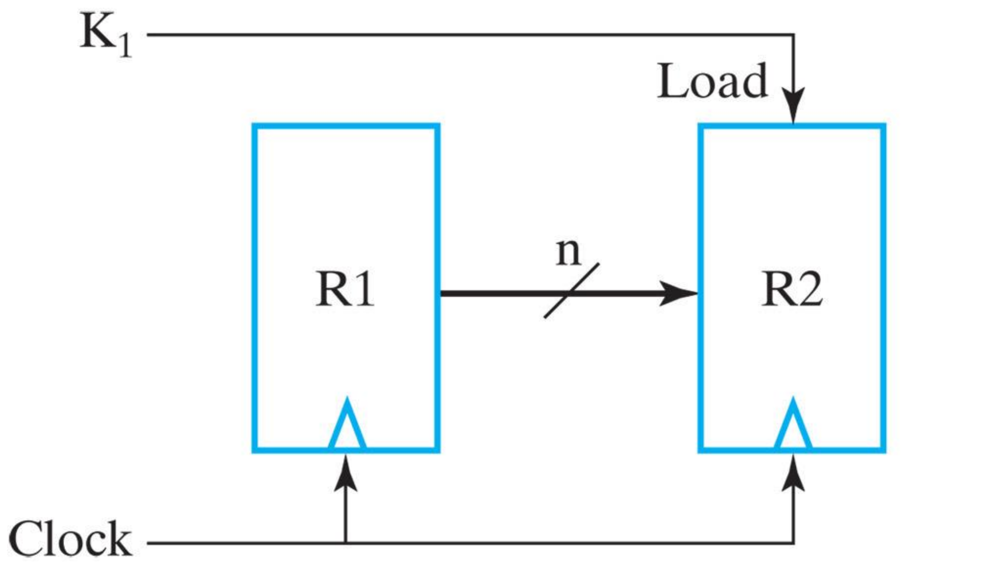
Data Transfer between Registers

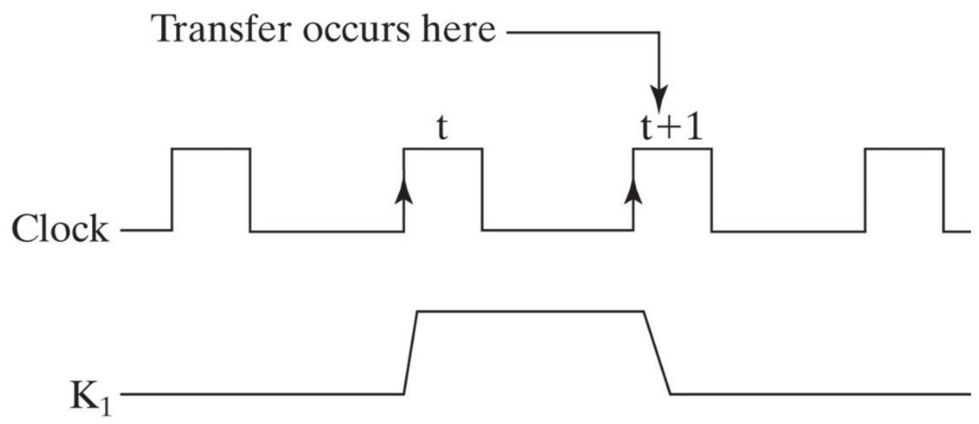
- The transfer of data contained in registers is indicated with the **transfer operator:** $\leftarrow$
  - Ex: $\mathbf{R2 \leftarrow R1}$

    Indicates the transfer of the content of R1 (source) to R2 (destination): in this type of operation, the content of the destination register changes, but the content of the source register is not changed
- A transfer between registers can be **conditional**, that is **based on the value of one or more control signals** generated in the control unit
  - Example: $\mathbf{if\ \ (K_1 = 1)\ \ then\ \ (R2 \leftarrow R1)}$
  - Compact notation: $\mathrm{K_1{:}\ R2 \leftarrow R1}$

    Indicates the transfer of the contents of R1 into R2 only at the condition (expressed before ":") that the control signal K_1 is equal to '1'

Data Transfer between Registers



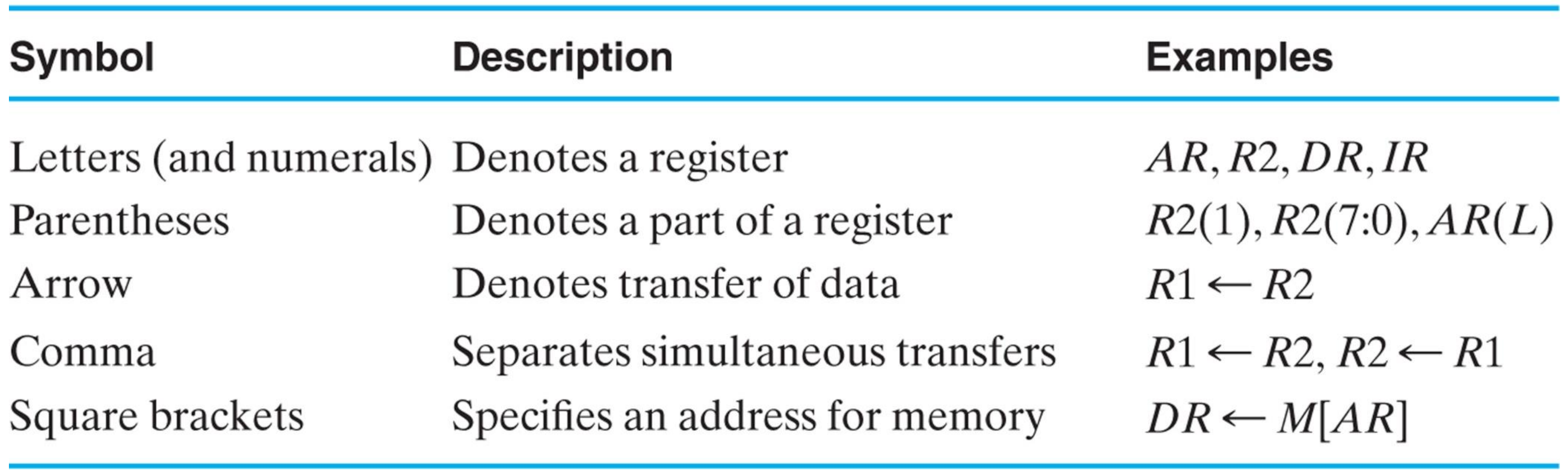


if $(K_1 = 1)$ then $(R2 \leftarrow R1)$

- n output bits of R1 are connected to R2 input
- K_1 (synchronized with the clock) is connected to the Load input of R2. **K_1 is set to '1' at the clock rising edge and it activates the transfer of the R1 contents into R2, at the next clock rising edge**

NOTE: The clock does not appear among the inputs in the RTL statement: transfers are assumed to occur in response to a clock transition

Register Transfer: Basic Symbols



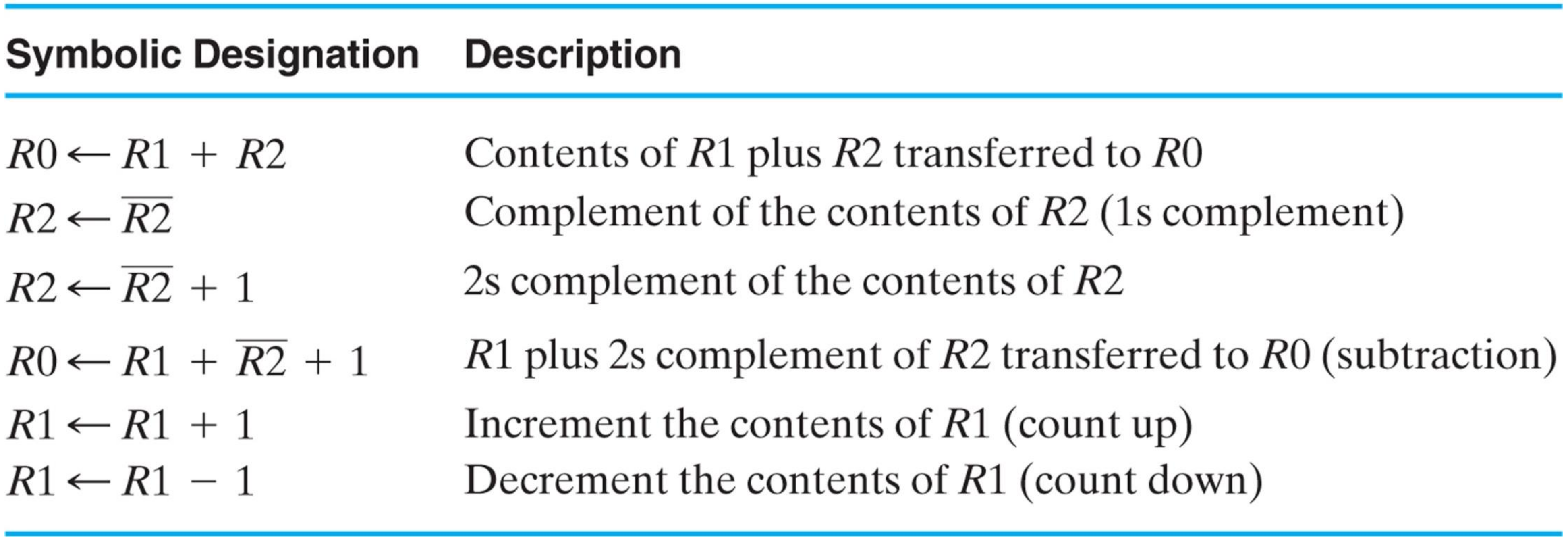
Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7:0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2, R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

- Simultaneous data transfers between registers are separated by commas
  - Example: $\mathbf{K_3}$: $\mathbf{R2} \leftarrow \mathbf{R1},\ \mathbf{R1} \leftarrow \mathbf{R2}$

    Indicates the exchange between the contents of R1 and R2 at the clock rising edge at which the control signal K_3 is equal to 1

Arithmetic Microoperations

- The **basic arithmetic operations** are addition (+), subtraction (-), increment (+1), decrement (-1), and complement ($\overline{x}$)
- Multiplication (*) and division (/) are not included in the set of basic operations, since they can be realized through a sequence of basic microoperations



Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1s complement)
$R2 \leftarrow \overline{R2} + 1$	2s complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2s complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

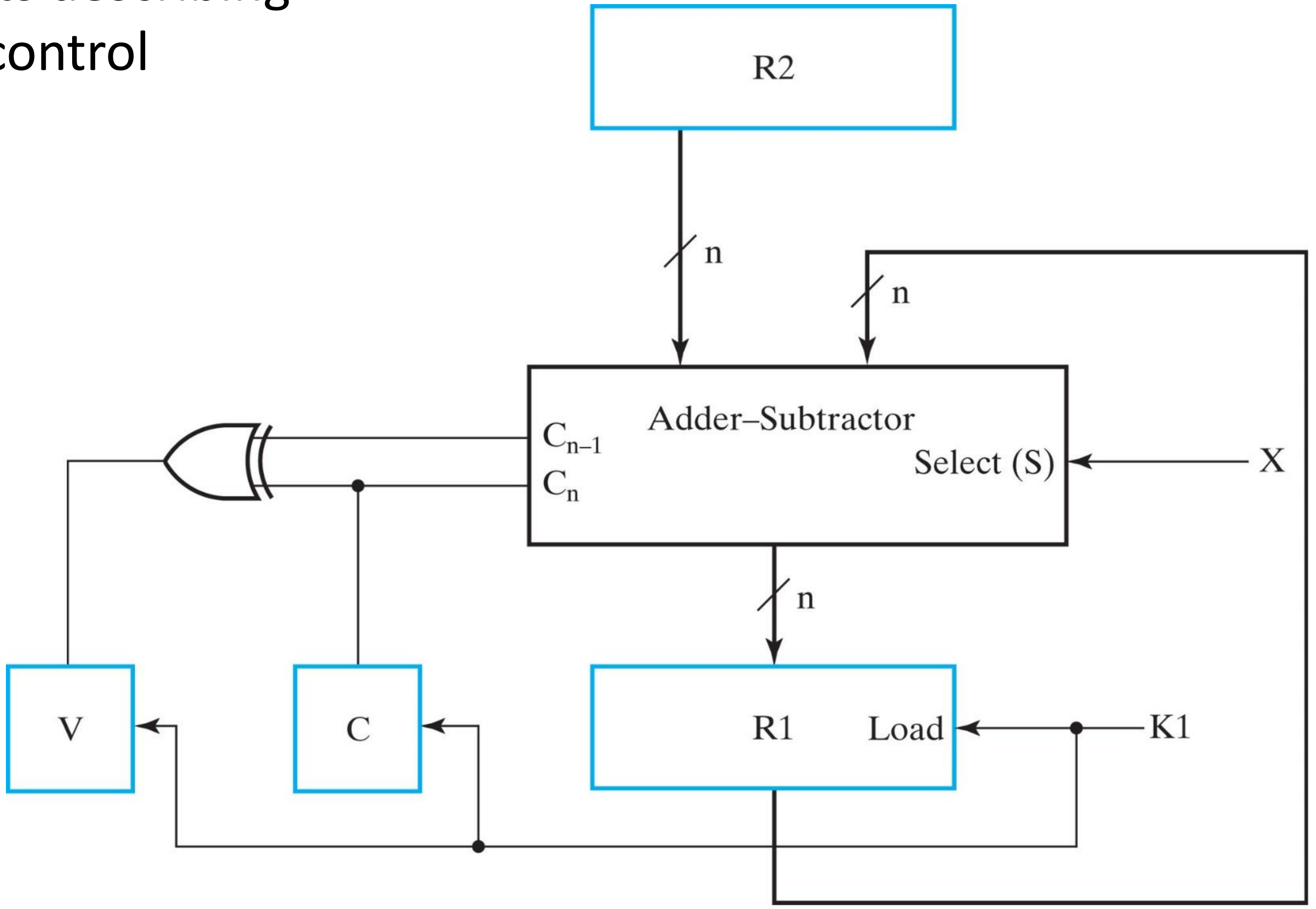
Arithmetic Microoperations: Example

Let’s consider the RTL statements describing a binary adder-subtractor with control signals X and K_1 :

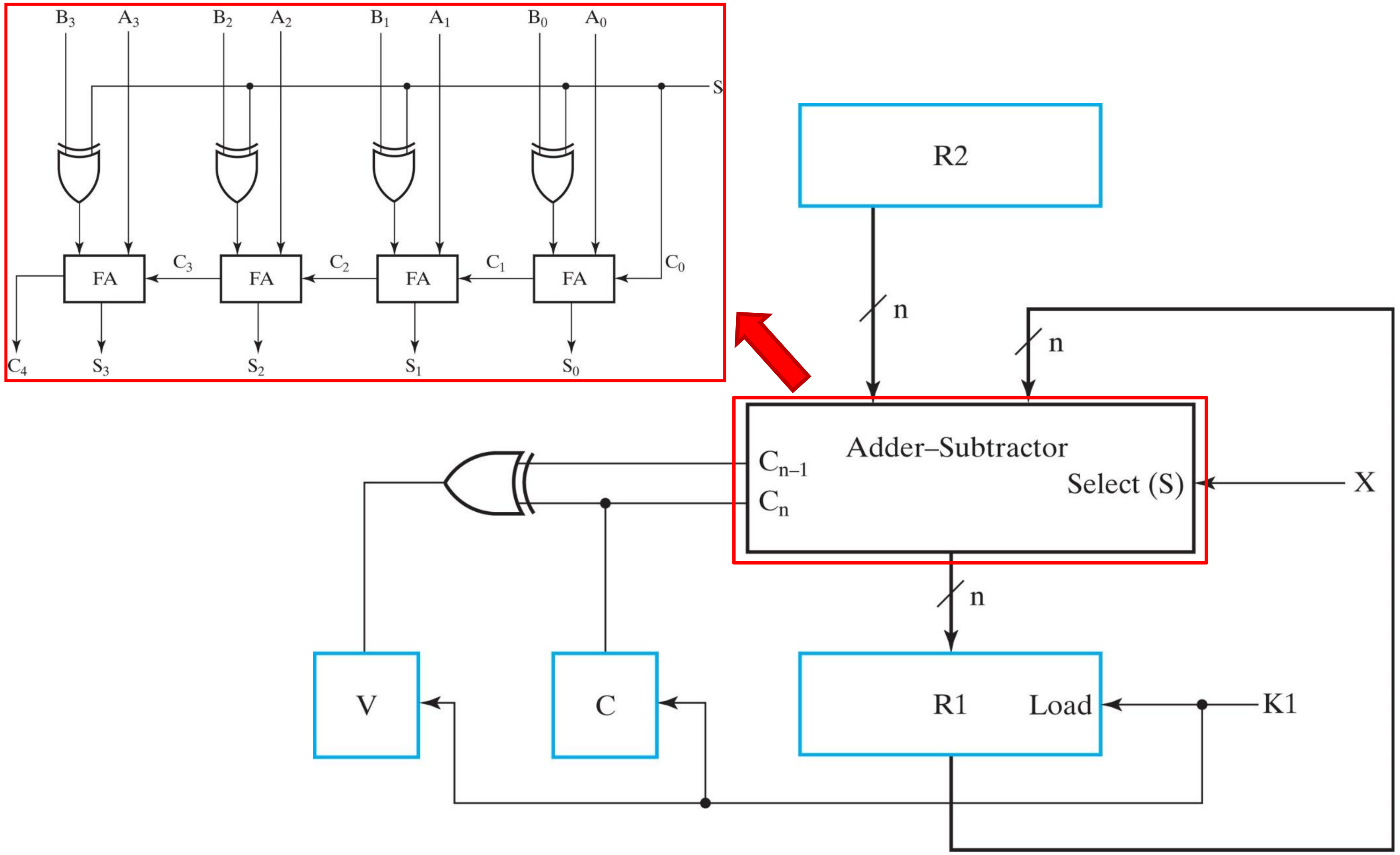
$$\bar{X}K_1\text{: } R1 \leftarrow R1 + R2$$

$$XK_1\text{: } R1 \leftarrow R1 + \overline{R2} + 1$$

Hardware performing the addition or subtraction between the data stored in registers R1 and R2



Review: n bit Adder-Subtractor



Arithmetic Microoperations: Example

$\mathbf{\overline{X}K_1}$: $\mathbf{R1 \leftarrow R1 + R2}$

$\mathbf{XK_1}$: $\mathbf{R1 \leftarrow R1 + \overline{R2} + 1}$

K_1: control signal that starts the execution of the operation

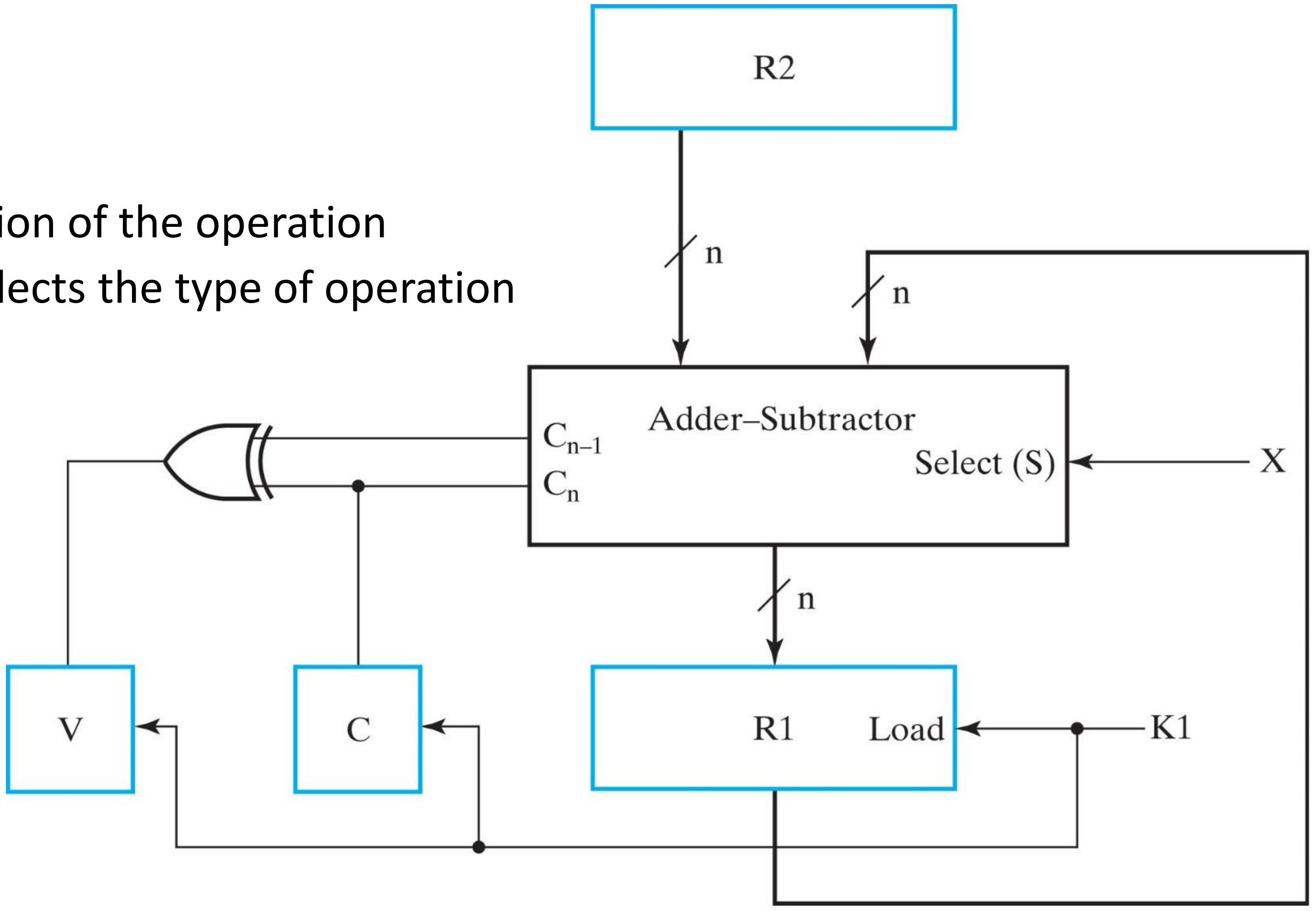
X: S input of the adder/subtractor. X selects the type of operation

X = 0: sum

X = 1: subtraction

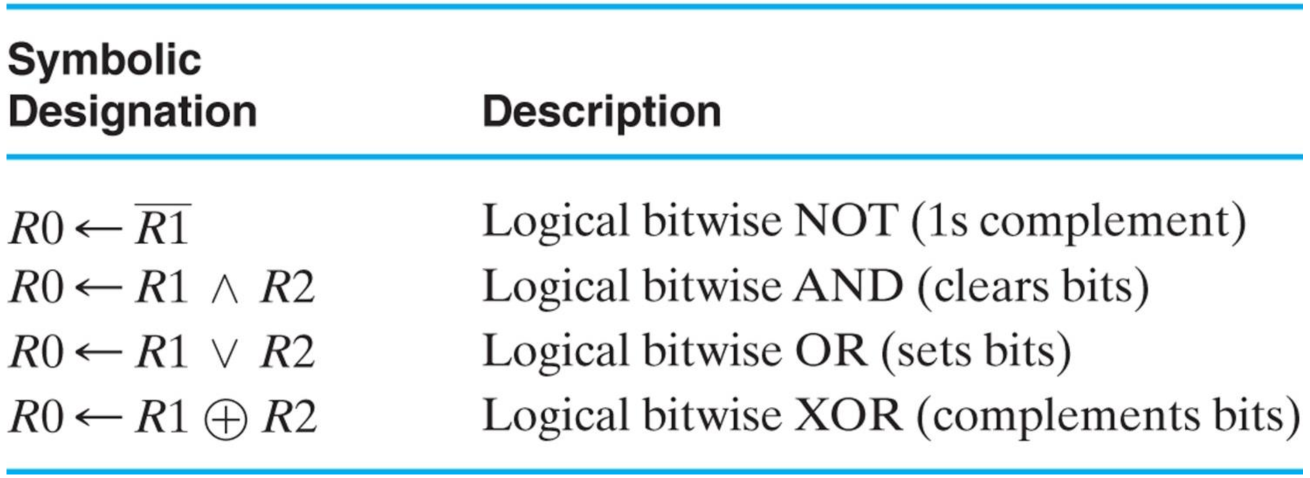
At each rising edge of the clock in which K_1 is 1, the sum or difference between R1 and R2 is loaded into R1

The carry output bit and the overflow bit are transferred to the corresponding flip-flops when $K_1 = 1$



Logic Microoperations

- Logic microoperations are used to manipulate the bits stored in the registers. They perform **bitwise** operations, i.e. they consider each bit of the register separately as a single binary variable
- **Basic logic operations:** NOT ($\overline{x}$), AND ($\wedge$), OR ($\vee$), XOR ($\oplus$)



Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1s complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

Note: the + symbol can indicate either addition or logic OR, it must be deduced from the context

Example $K_1 + K_2\text{: } R1 \leftarrow R2 + R3,\ R4 \leftarrow R2 \vee R3$

- ➢ If it is inside a Boolean function, + denotes logic OR
- ➢ Otherwise, + stands for sum

Logic Microoperations

- Logic microoperations can also be used to change the value of some bits, clear, or remove some of the bits stored in a register
  - **AND**: can be used to set to 0 (**clear**) a group of bits in a register
  - **OR**: can be used to set to 1 (**set**) a group of bits in a register
  - **XOR**: can be used to **complement** a group of bits in a register

- A group of bits used to perform a logic operation that changes one or more bits stored in a register is referred to as **mask**. The operation is called masking out the bits

Logic Microoperations: Clear

- **Clear**: **AND** between the content of a register and a mask having '0' at the bit positions that we want to reset and '1' at the bit positions that we want to leave unchanged

$$X \land 0 = 0, \quad X \land 1 = X$$

Example:

10101101 10101011	$R1$ (data)
00000000 11111111	$R2$ (mask)
00000000 10101011	$R1 \leftarrow R1 \land R2$

Logic Microoperations: Set

- **Set**: **OR** between the content of a register and a mask having '1' at the bit positions that we want to set and '0' at the bit positions that we want to leave unchanged

$$X \vee 1 = 1, \quad X \vee 0 = X$$

Example:

10101101 10101011	$R1$ (data)
11111111 00000000	$R2$ (mask)
11111111 10101011	$R1 \leftarrow R1 \vee R2$

Logic Microoperations: Complement

- **Complement**: **XOR** between the content of a register and a mask having '1' at the bit positions that we want to complement and '0' at the bit positions that we want to leave unchanged

$$\mathrm{X} \oplus 1 = \overline{\mathrm{X}}, \quad \mathrm{X} \oplus 0 = \mathrm{X}$$

Example:

10101101 10101011	$\mathrm{R1}$ (data)
11111111 00000000	$\mathrm{R2}$ (mask)
01010010 10101011	$\mathrm{R1} \leftarrow \mathrm{R1} \oplus R2$

Logic Microoperations: Shift

- Shift operations are used for **lateral movement of data**
  - **Left Shift (LS)**: shifts the contents of a register towards the MSB. The rightmost bit of the destination register is called the incoming bit, the leftmost bit of the source register is called the outgoing bit
  - **Right Shift (RS)**: shifts the contents of a register towards the LSB. The leftmost bit of the destination register is called the incoming bit, the rightmost bit of the source register is called the outgoing bit
  - Note: with this type of shifts, the incoming bit is '0' and the outgoing bit is discarded (this is not the case for other types of shift, such as barrel shift)

		Eight-Bit Examples	
Type	**Symbolic Designation**	**Source $R2$**	**After Shift: Destination $R1$**
Shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
Shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

Summary

- We introduced two blocks that are widely used in digital systems: **registers** and **counters**
- In registers, the control of parallel data loading can be achieved with different techniques, such as **load enable**
- A complex digital system can be partitioned into two modules, **control unit** and **datapath**: the datapath performs the operations, the control unit coordinates the sequence of operations
- The elementary operations on the bits stored in a register are called **microoperations**, expressed in **Register Transfer Language (RTL)**. Microoperations can be of 4 types:
  - **Transfer** microoperations
  - **Arithmetic** microoperations
  - **Logic** microoperations
  - **Shift** microoperations

Disclaimer