İstanbul Bilgi Üniversitesi

CMPE321 - Computer Architecture

Lecture 1
Register Transfer Language

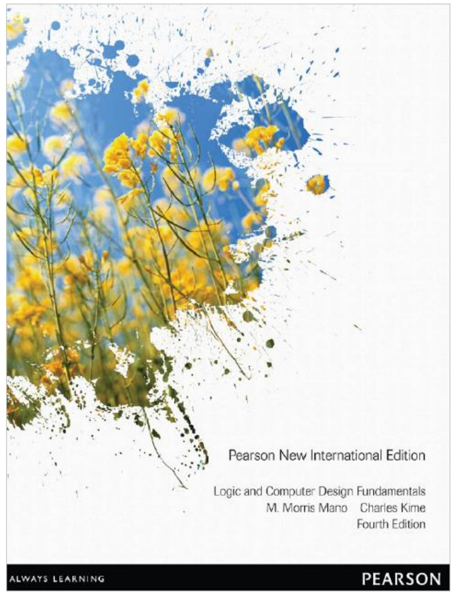
Hakan Ayral, PhD.

Administrivia

- Textbook : Logic and Computer Design Fundamentals, 4th ed., M.Morris Mano, Charles Kime. Pearson Publishing
  - Starting from chapter 7

- Lab participation is mandatory
  - There will be quizzes

- Grading
  - Quizzes 10%
  - 1 Midterm 30%
  - 1 Final 60%


Pearson New International Edition
Logic and Computer Design Fundamentals
M. Morris Mano Charles Kime
Fourth Edition
ALWAYS LEARNING
PEARSON

Prerequisite Knowledge

- Chapter 2 – Combinational Logic Circuits
  - Binary Logic
  - Digital Logic Gates
  - Boolean Algebra
  - Circuit Optimization
    - K-Map
    - Don’t care conditions
  - High Impedence
    - Tri-state buffer
- Chapter 3 – Combinational Logic Design
  - Technology Mapping
  - Multi-bit Functions
  - Enabling
    - Decoder
    - Decoder-based Combinational Circuits
    - Encoder
  - Selecting
    - Multiplexer
    - Multiplexer-based Combinational Circuits
- Chapter 4 – Arithmetic Functions
  - Half adder/Full adder
  - Binary Ripple Carry Adder
  - Subtraction
    - Complements, 2’s Complement
  - Signed numbers
    - Signed addition/subtraction
  - Overflow
  - Increment/Decrement
  - Multiplication/Division with a constant
- Chapter 5 – Sequential Circuits
  - Latches
  - Flip-flops
    - Edge triggering
  - Sequential Circuit Analysis
  - Input Equations
  - State Table/State Diagram
  - Design Procedure
  - State Machine

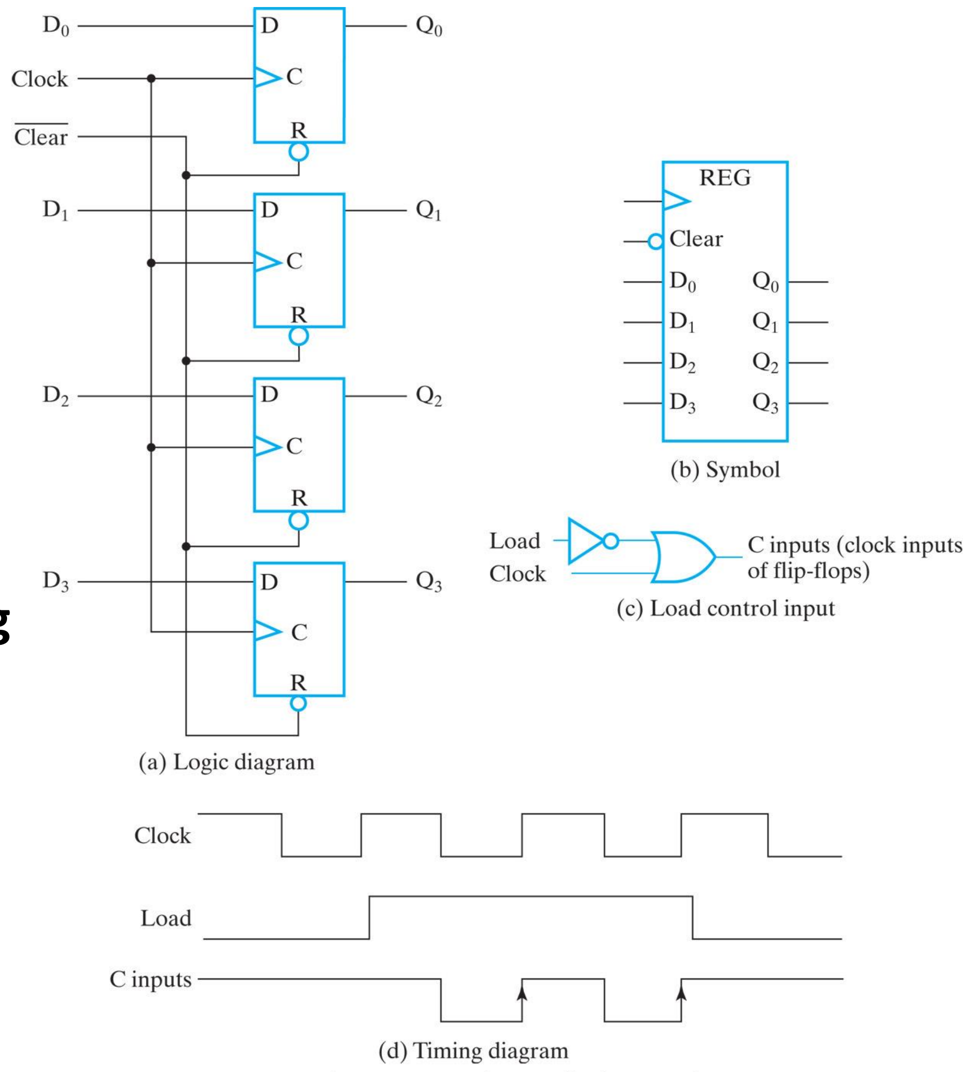
Registers

- A ***register*** consists of a set of flip-flops, together with gates that implement their state transitions.
  - Registers are useful for storing and manipulating information

- A ***counter*** is a register that goes through a predetermined sequence of states upon the application of clock pulses.
  - Although counters are a special type of registers, it is common to differentiate them from registers.
  - Counters are employed in circuits that sequence and control operations in a digital system.

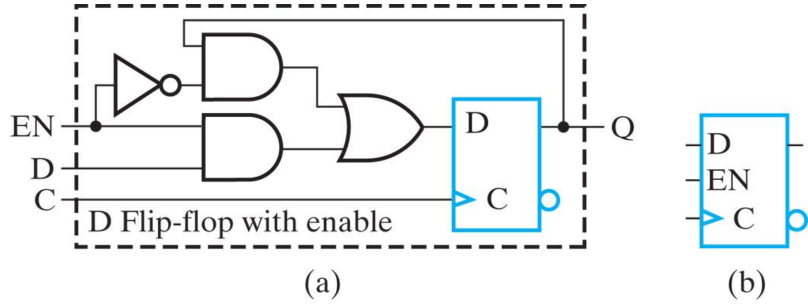
Registers

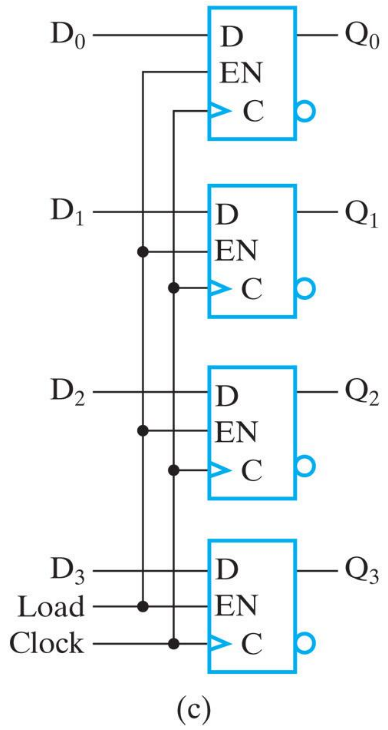
- The simplest register consists of only flip-flops without external gates.
- The *Clock* input triggers all flip-flops on the rising edge of each pulse; the binary information available at the four *D* inputs is transferred into the register.
- The four *Q* outputs can be sampled to obtain the binary information stored in the register.
- The transfer of new information into a register is called **loading** the register.
  - If all the bits of the register are loaded simultaneously with a common clock pulse, we say that the **loading** is done in **parallel**.



Load Enable

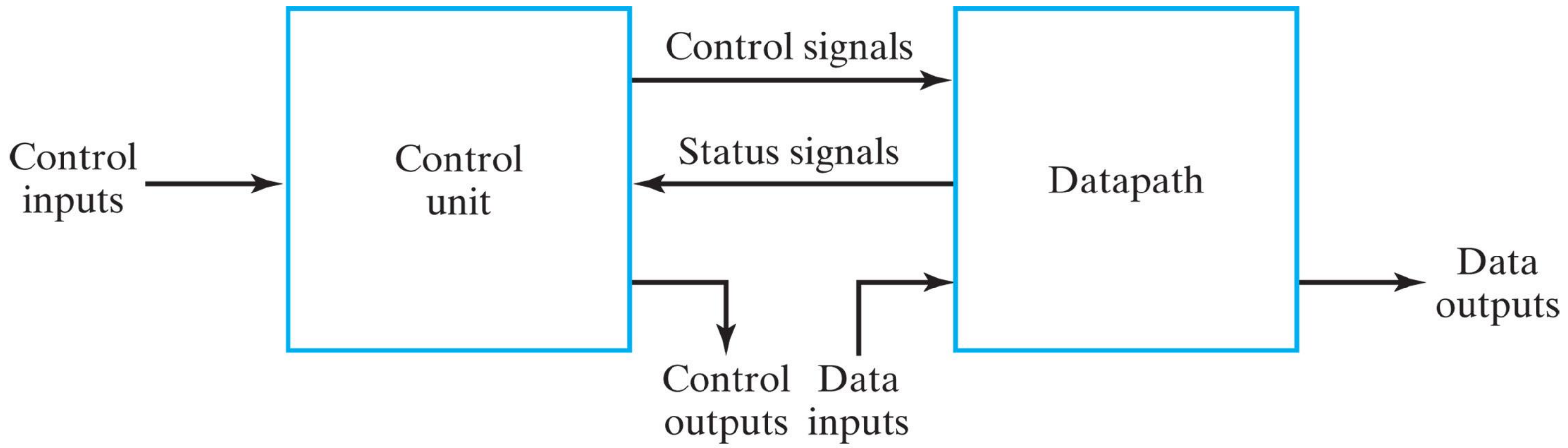
- You can modulate the clock signal with another signal.
- If clock signal does not arrive to register the content does not change.
- C inputs = $\overline{\text{Load}}$ + Clock ← when Load is 1 C inputs = Clock
- Clock is turned on and off, this is called **clock gating.**
- Having gates on clock line introduce different delays
  - Clock signal must ideally arrive to all registers at same time
  - It's better to have an enable signal which doesn't involve the clock line.
- Flip-flop with load enable without involving clock line is implemented as shown on figure (a); let's call it **D flip-flop with enable**.
  - An 2-to-1 MUX selects between D and Q for input based on EN.





Control Unit and Datapath

- In digital system designs, we partition the system into two types of modules:
  - a **datapath**, which performs data-processing operations,
  - and a **control unit**, which determines the sequence of those operations
- ***Control signals*** are binary signals that activate the various data-processing operations.
- Control unit receives **status bits** from the datapath.
  - status bits describe aspects of the state of the datapath.
- Datapaths are defined by their registers and the operations performed on binary data stored in the registers.

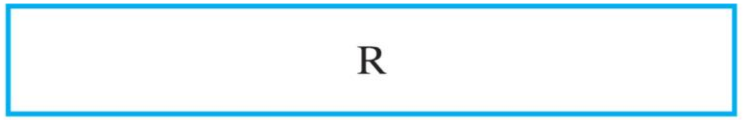


Register Transfer Operations

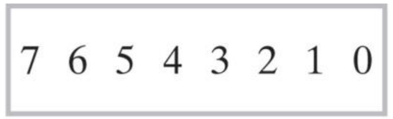
- The registers are assumed to be basic components of the digital system.
- The movement of the data stored in registers and the processing performed on the data are referred to as ***register transfer operations***.
- The register transfer operations of digital systems are specified by the following three basic components:
    1. the set of registers in the system,
    2. the operations that are performed on the data stored in the registers, and
    3. the control that supervises the sequence of operations in the system.
- A register has the capability to perform one or more ***elementary operations*** such as load, count, add, subtract, and shift.
- An elementary operation performed on data stored in registers is called a ***microoperation****.*
    - Examples of microoperations are loading the contents of one register into another, adding the contents of two registers, and incrementing the contents of a register
- The control unit provides signals that sequence the microoperations in a prescribed manner.

Registers

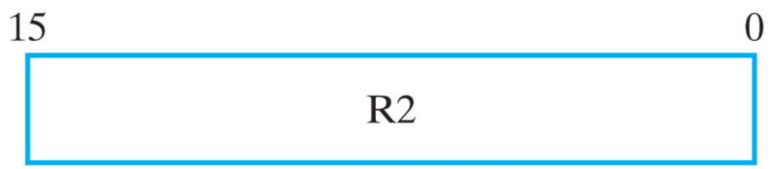
- We denote the registers in a digital system by uppercase letters that indicate the function of the register.
  - For example, a register that holds an address for the memory unit is usually called an **address register** and can be designated by the name ***AR***.
  - Other designations for registers are ***PC*** for **program counter**, ***IR*** for **instruction register**, and *R*2 for register 2.
- Flip-flops in an *n*-bit register are numbered in sequence from 0 to *n* – 1, starting with 0 in the least significant position and increasing toward the most significant position.



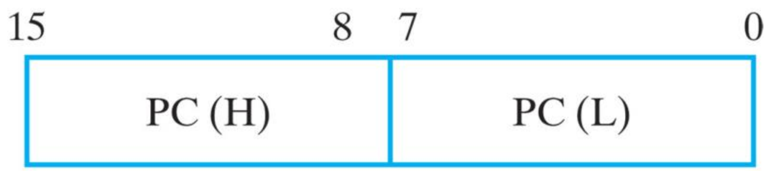
(a) Register R



(b) Individual bits of 8-bit register



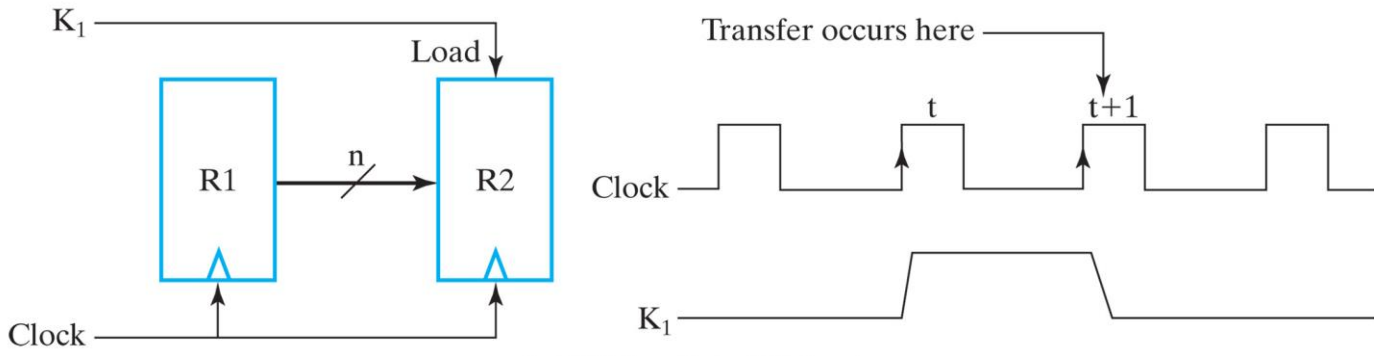
(c) Numbering of 16-bit register



(d) Two-part 16-bit register

Register Transfer

- Data transfer from one register to another is designated with $\leftarrow$ symbol.
- The statement R2$\leftarrow$R1 denotes copying the contents of register *R*1 into register *R*2.
  - *R*1 is the ***source*** and the register *R*2 is the ***destination***.
  - By definition, the contents of the source register do not change as a result of the transfer; only the contents of the destination register, *R*2, change.
- Normally, we don't want a transfer to occur every clock pulse, but only for specific values of the control signals.
  - This can be specified by a ***conditional statement***, with the *if-then* form like: if (K1=1) then (R2$\leftarrow$R1)
    - Here K1 is a control signal generated in the control unit.
    - A more concise way to write the if-then form is: K1: R2$\leftarrow$R1



Register Transfer

- Every statement written in register-transfer notation presupposes a hardware construct for implementing the transfer.
- Figure 5 shows a block diagram that depicts the transfer from *R*1 to *R*2.
    - The *n* outputs of register *R*1 are connected to the *n* inputs of register *R*2.
        - The letter *n* is used to indicate the number of bits in the register transfer path from *R*1 to *R*2.
        - When the width of the path is known, *n* is replaced by an actual number.
- Register *R*2 has a load control input that is activated by the control signal *K1*.
- It is assumed that the signal is synchronized with the same clock as the one applied to the register.
- The flip-flops are assumed to be positive-edge triggered by this clock.
- Timing diagram shows that, *K1 i*s set to 1 on the rising edge of a clock pulse at time *t*.
- The next positive transition of the clock at time *t*+1 finds *K*1=1, and the inputs of *R*2 are loaded into the register in parallel.
- In this case, *K1 r*eturns to 0 on the positive clock transition at time *t*+1, so that only a single transfer from *R*1 to *R*2 occurs.

Register Transfer

- Note that the clock is not included as a variable in the register-transfer statements.
  - It is assumed that all transfers occur in response to a clock transition.
- Even though the control condition *K*1 becomes active at time *t*, the actual transfer does not occur until the register is triggered by the next positive transition of the clock, at time *t*+1.

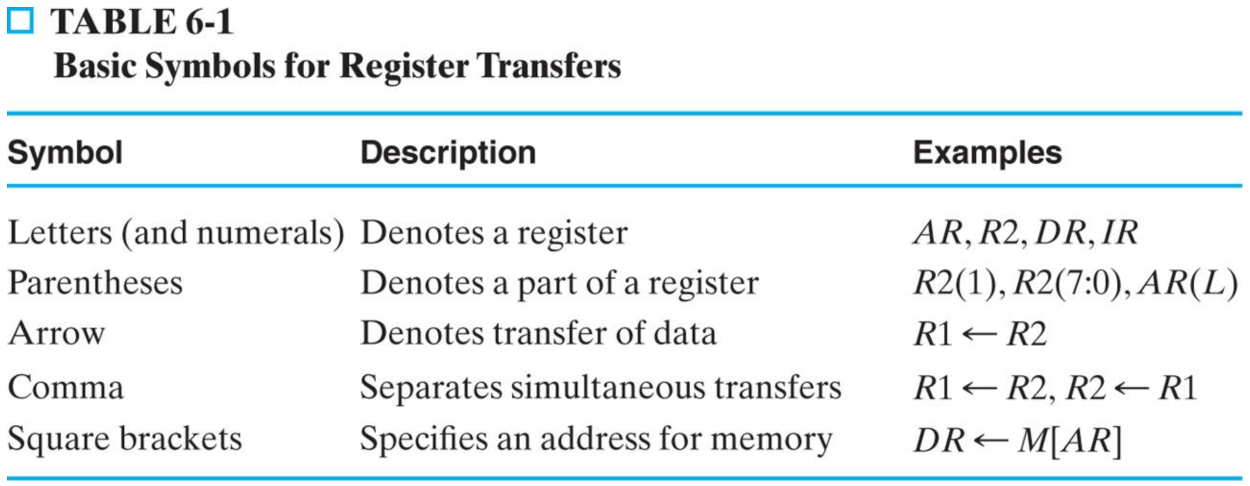


TABLE 6-1
Basic Symbols for Register Transfers

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	$DR \leftarrow M[AR]$

Microoperations

- A microoperation is an elementary operation performed on data stored in registers or in memory.
- The microoperations most often encountered in digital systems are of four types:
  1. **Transfer microoperations**, which transfer binary data from one register to another.
  2. **Arithmetic microoperations**, which perform arithmetic on data in registers.
  3. **Logic microoperations**, which perform bit manipulation on data in registers.
  4. **Shift microoperations**, which shift data in registers.
- A given microoperation may be of more than one type. For example, a 1s complement operation is both an arithmetic microoperation and a logic microoperation.
- Transfer microoperations do not change the binary data they just move it.
- Arithmetic, logic and shift microoperations produce new binary data.
- Basic sets of operations are used to form sequences that implement more complicated operations.

Arithmetic Microoperations

- Basic arithmetic microoperations are add, subtract, increment, decrement, and complement.
- The statement $R0 \leftarrow R1 + R2$ specify an addition microoperation
- To implement this statement with hardware, we need three registers and a combinational component that performs the addition, such as a parallel adder.
- Subtraction is most often implemented through addition with 2's complement, we can specify it by the statement $R0 \leftarrow R1 + \overline{R2} + 1$

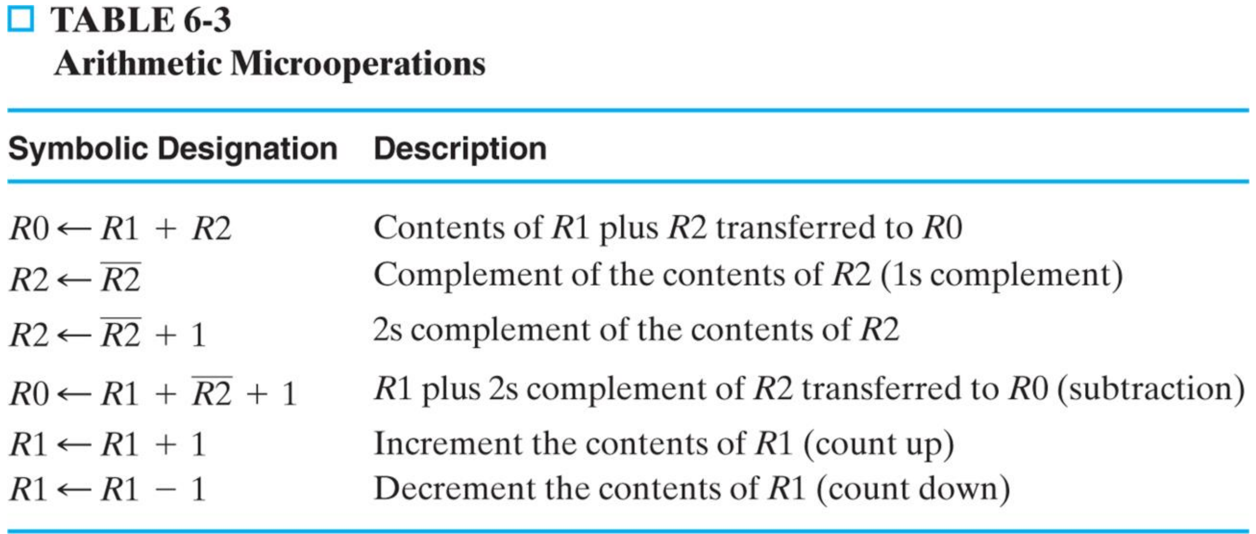


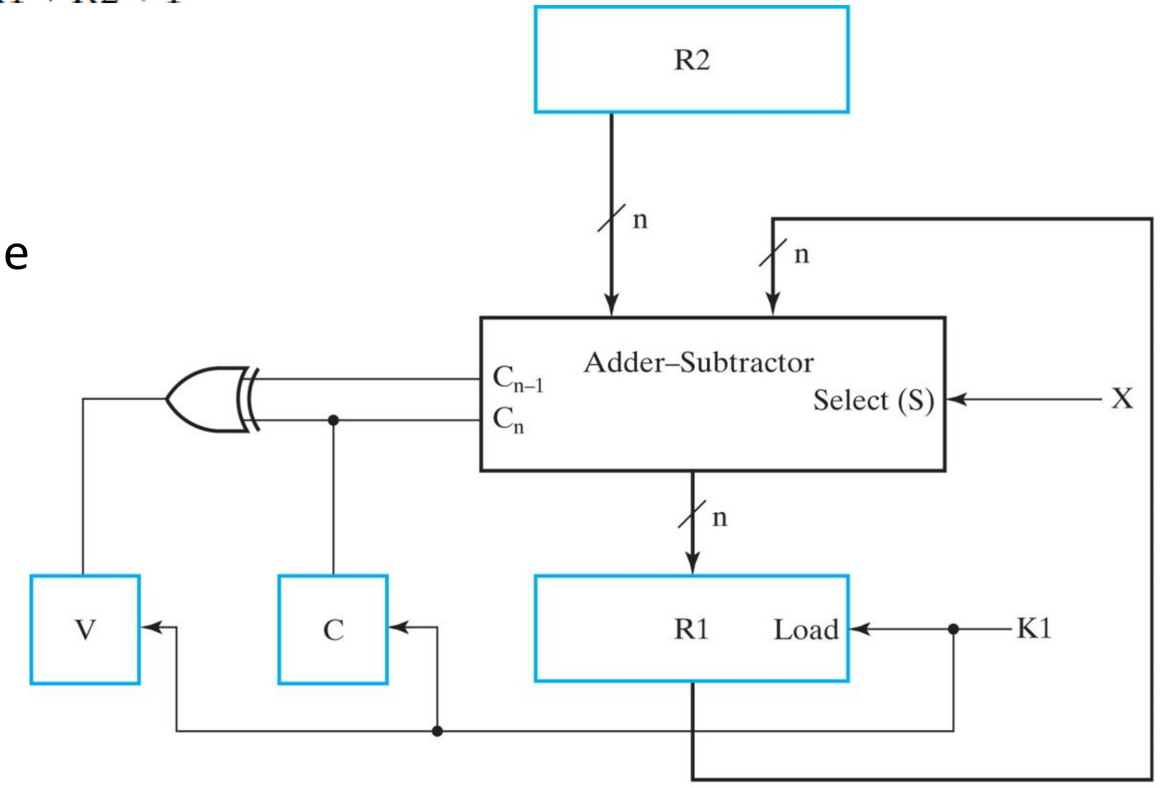
TABLE 6-3
Arithmetic Microoperations

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

- Consider the following two statements $\overline{X}K_1\!: \quad R1 \leftarrow R1 + R2$
  $XK_1\!: \quad R1 \leftarrow R1 + \overline{R2} + 1$

- Control variable *K*1 activates an operation to add or subtract.
- If control variable *X* is equal to 0, then $\overline{\text{X}}$K1 = 1 and the value of *R*1 is updated as the sum of R1 and R2.
- If *X* is equal to 1, then *XK*1 = 1, and the value of *R*2 is subtracted from the value of *R*1.


R2
n
n
Adder–Subtractor
C_{n-1}
C_n
Select (S)
X
n
V
C
R1
Load
K1