Register Transfer Logic

Lesson-4:

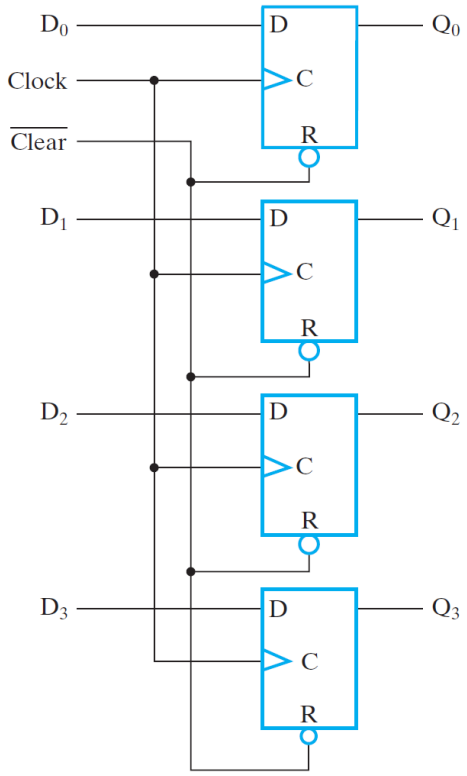
Topic	Lesson Learning Outcomes	Teaching-Learning Methodology	Assessment Method
Introduction to Register transfer Logic, Inter-register transfer	• To understand the modular approach and Operation of Digital system • To get overview of Register transfer operations • To use the register transfer language or HDL to represent digital systems and vice versa.	Class Lecture, Question and answer	Test, exams, quiz, etc

Introduction

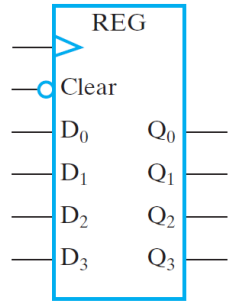
- Modular approach
  - Design Large digital systems for example processor unit of computers
    - Difficult to use state table
      - Large number of states
    - Use modular approach to simplify the design
      - The system is partitioned into modular subsystem
      - Each subsystem performs specific tasks
    - The subsystems are interconnected to form a complete digital system.

Introduction

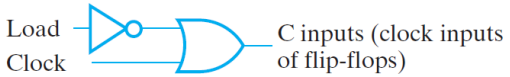
- A 4-bit Register



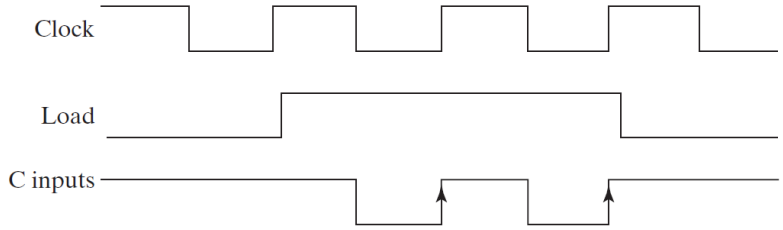
(a) Logic diagram



(b) Symbol



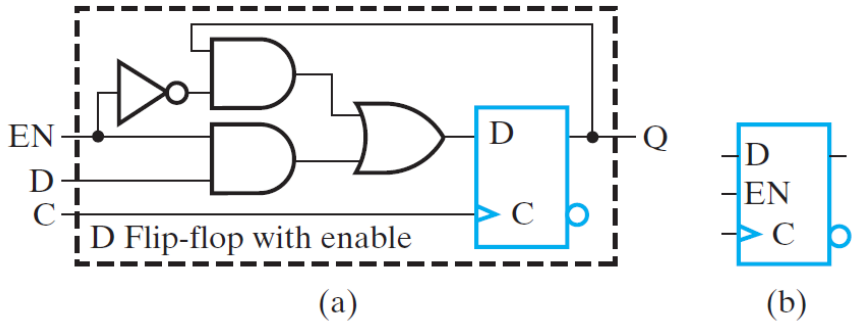
(c) Load control input

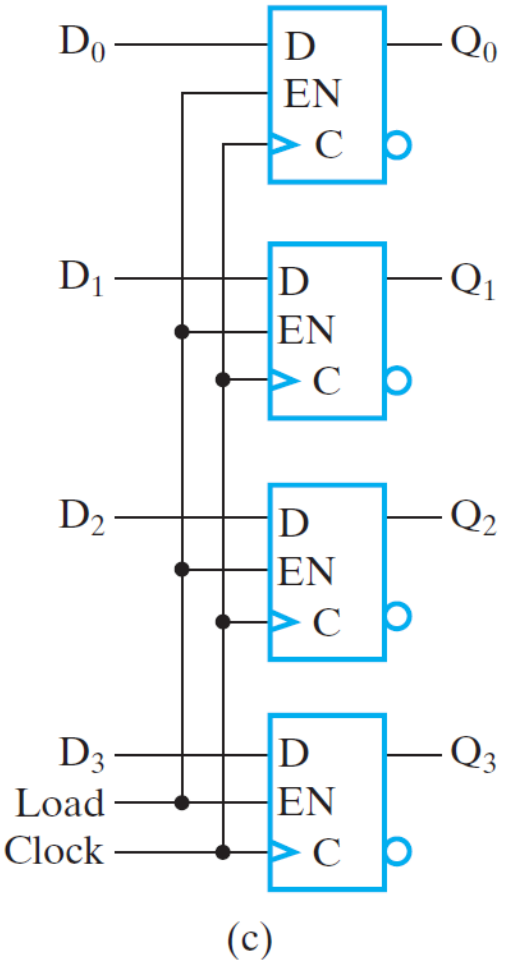


(d) Timing diagram

Introduction

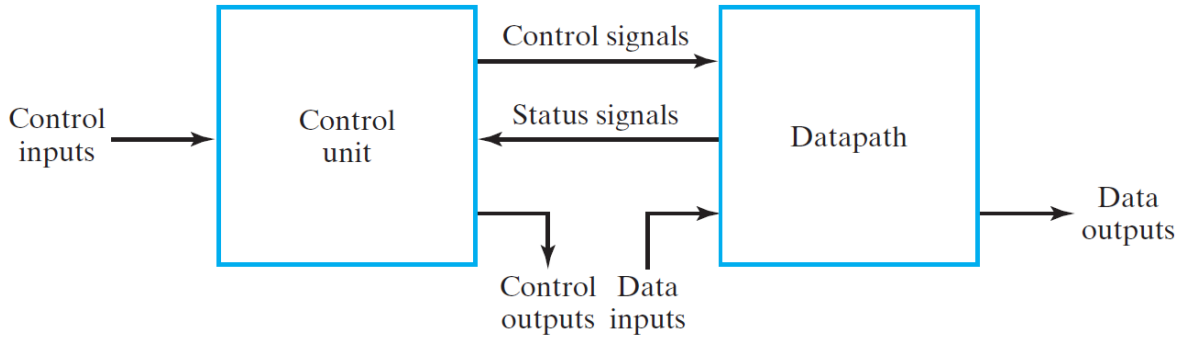
- A 4-bit Register with Parallel Load





Introduction

- 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
  3. the control that supervises the sequence of operations in the system.



Introduction

- Register Transfer Logic
  - Describe a digital system is concise and precise manner
    - Registers are primitive components of digital system
    - Use a set of expressions and statements
      - Similar to a programming language
    - The subsystems are interconnected to form a complete digital system.

Introduction

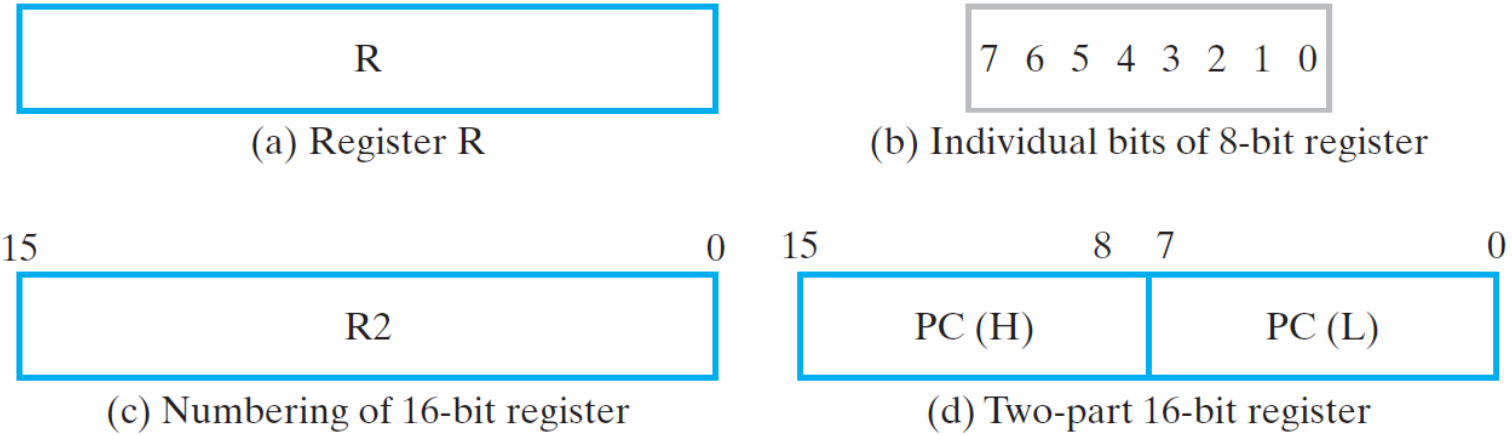
- **Basic operations of a digital system**
  1. The set of registers and their operations
     - Registers, counters, memory units, etc.
  2. Binary information stored in the registers
     - Numbers, characters, etc.
  3. Operations performed on the information
     - Arithmetic, logic, shift, etc.
  4. Control function that initiates the operations
     - Binary variables that initiates a operation condition on its state.

Introduction

- **Register Transfer Language (RTL) or Hardware Description Language (HDL)**
  - Symbolic notations to represents the basic operations
  - Contains statements
    - Control functions
      - Control conditions and timing sequences
    - List of micro-operations

Register Transfer

- **Most common ways to represent a register**



(a) Register R

(b) Individual bits of 8-bit register

(c) Numbering of 16-bit register

(d) Two-part 16-bit register

Register Transfer

- **The replacement operator**

$$\mathbf{A \leftarrow B}$$

  - Contains of register **B** transfer to register **A**
  - The contains of **B** do not change after transfer
  - The transfer operation performed in every clock pulse
  - But we do not want to perform the operation in every clock pulse
    - A predetermined condition is applied.

Register Transfer

- **Control Function**
  - A Boolean function with value 0 or 1.
  - Control function is terminated with a colon and represented as

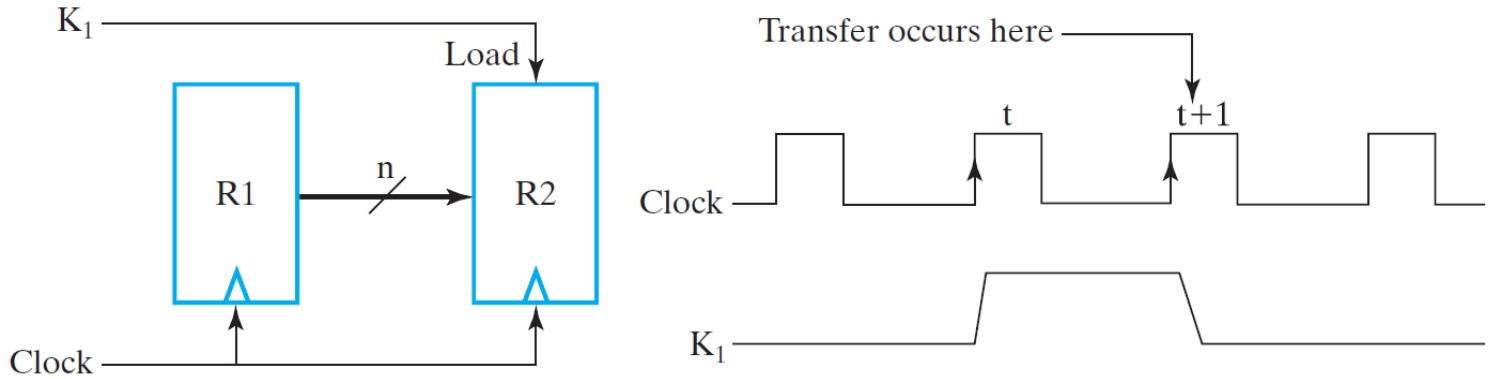
$$x'T_1: \quad \mathbf{A} \leftarrow \mathbf{B}$$

- The transfer operation is executed by the hardware only when the control Boolean function is 1.

Register Transfer

- **Control Function**
  - Hardware implementation of K1: R2 ← R1

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



Register Transfer

- **Basic symbols of register transfer operations**

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	$DR \leftarrow M[AR]$

Register Transfer

- **Textbook RTL, VHDL, and Verilog Symbols for Register Transfers**

Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
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)	*A*[3:0]
Concatenation	||	&	{ , }

Lesson-5:

Topic	Lesson Learning Outcomes	Teaching-Learning Methodology	Assessment Method
Arithmetic, Logic and Shift microoperation, conditional control statement, Representation of binary data.	• To implement the Arithmetic, Logic and Shift microoperation with H/W and HDL • To implement conditional control statement in H/W and HDL • To review the representation of binary data.	Class Lecture PBL (Solving some problems in class)	Test, exams, quiz, etc

Register Transfer

- **Most Commonly Used Micro-operations**
  - Inter-register transfer micro-operations
  - Arithmetic micro-operations
  - Logic micro-operations
  - Shift micro-operations

Arithmetic Micro-operations

❑ List of operations

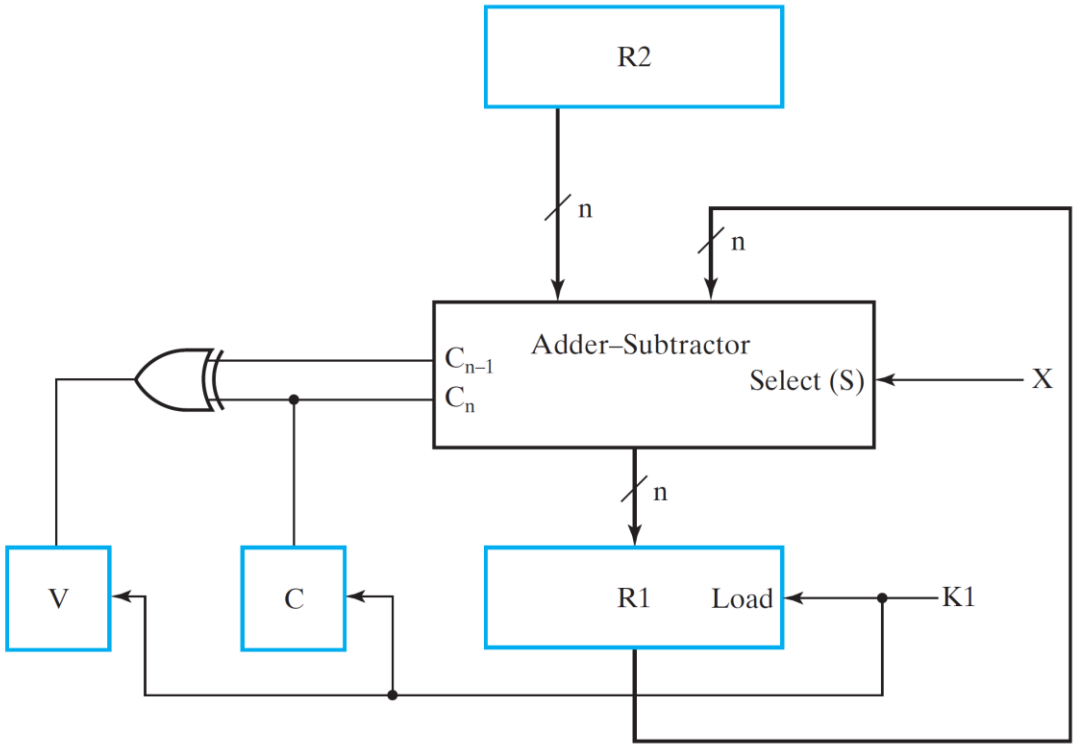
Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1s complement)
$R2 \leftarrow \overline{R2} + 1$	2s complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ 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)

Note that, arithmetic operations multiply (*) and divide (/) are not listed here

Arithmetic Micro-operations

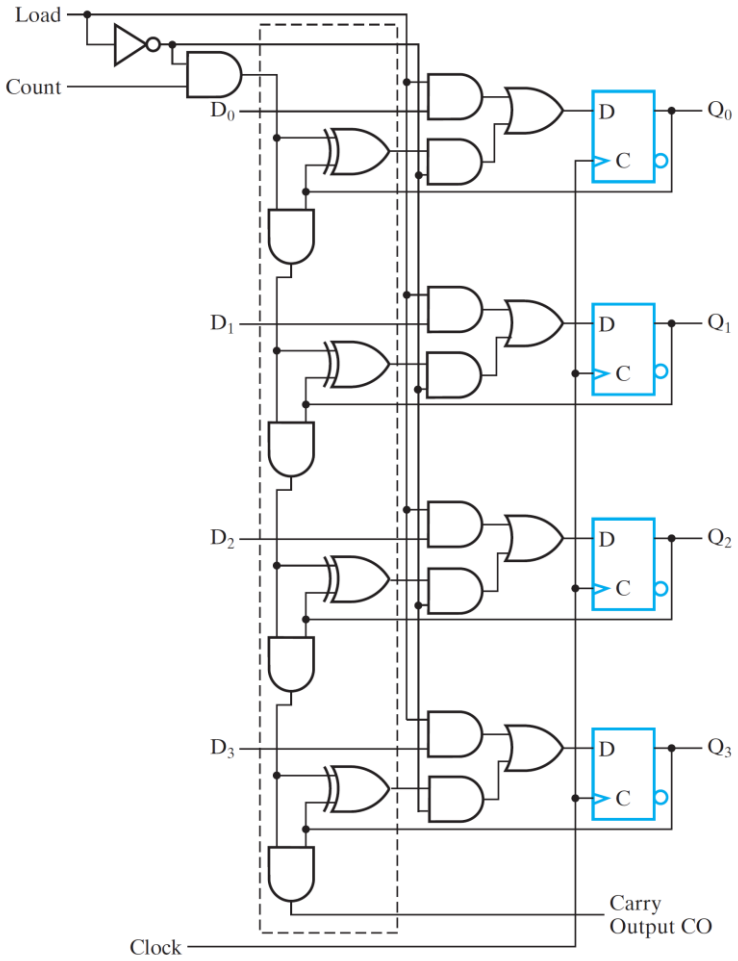
- **Implement the following micro-operations**

$$\overline{X}K_1: R1 \leftarrow R1 + R2$$
$$XK_1: R1 \leftarrow R1 + \overline{R2} + 1$$



Arithmetic Micro-operations

- 4-Bit Binary Counter with Parallel Load

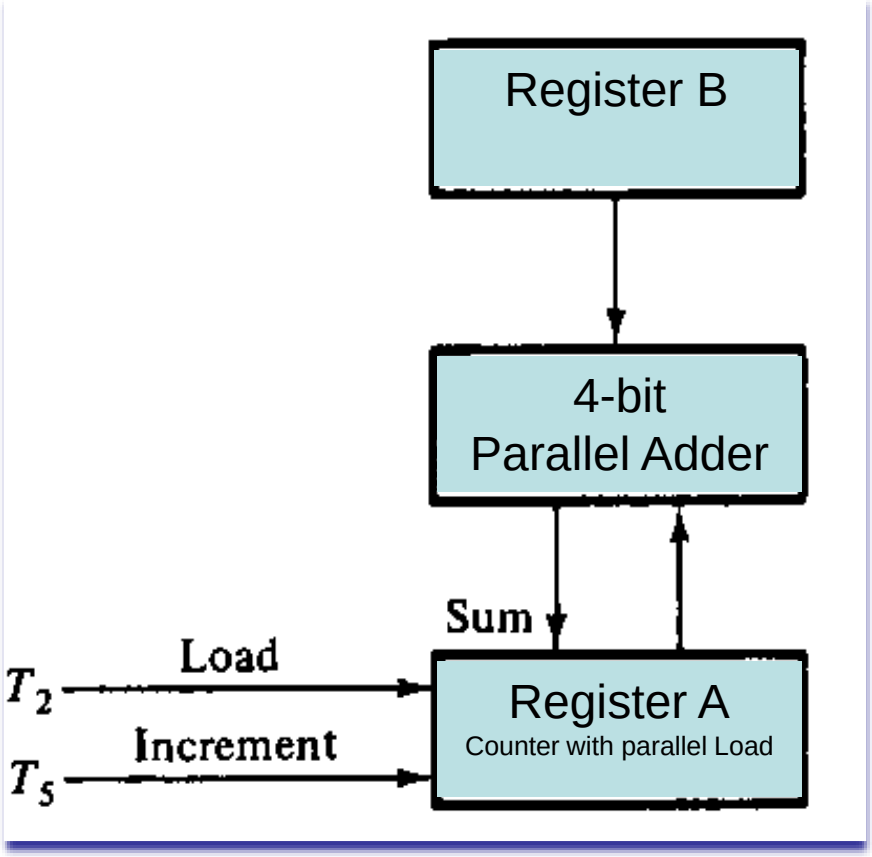


Arithmetic Micro-operations

- **Implement the following micro-operations**

T_2: $A \leftarrow A + B$

T_5: $A \leftarrow A + 1$



Logic Micro-operations

- **List of operations**

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)

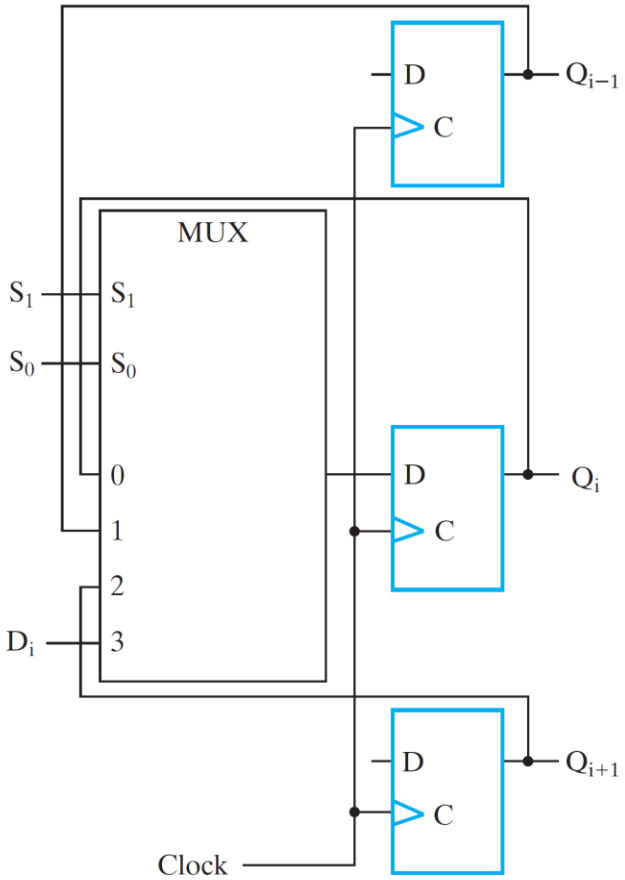
Shift Micro-operations

- **List of operations**

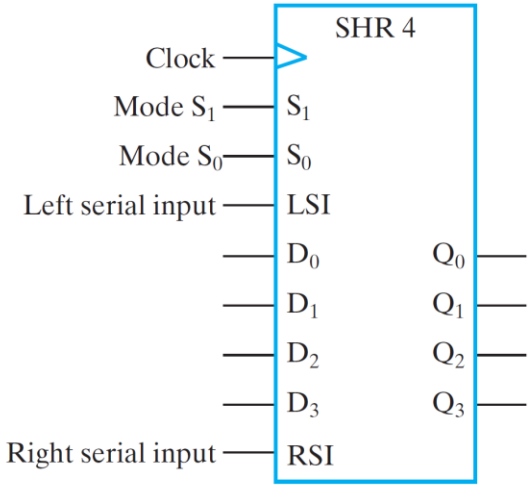
Type	Symbolic Designation	Eight-Bit Examples	
		Source *R2*	After Shift: Destination *R1*
Shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
Shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

Shift Micro-operations

❑ Bidirectional Shift Register with Parallel Load



(a) Logic diagram of one typical stage



(b) Symbol

$\overline{S}_1 \cdot S_0$: $Q \leftarrow \text{sl}\, Q$

$S_1 \cdot \overline{S}_0$: $Q \leftarrow \text{sr}\, Q$

$S_1 \cdot S_0$: $Q \leftarrow D$

Implement

$R0 \leftarrow \text{sr}\ R0,\ \ R1 \leftarrow \text{sl}\ R2$

Conditional Control Statements

- **Syntax**

  P: *if* **(condition)** *than* **[micro-operations]**

  else **[micro-operations**

- **Example**

$$T_2: \quad \text{If } (C = 0) \text{ then } (F \leftarrow 1) \text{ else } (F \leftarrow 0)$$

  - Equivalent Statements (if C is one bit)

$$C'T_2: \quad F \leftarrow 1$$

$$CT_2: \quad F \leftarrow 0$$

  - Equivalent Statements (if C is not one bit)

$$x = C_1'C_2'C_3'C_4' = (C_1 + C_2 + C_3 + C_4)'$$

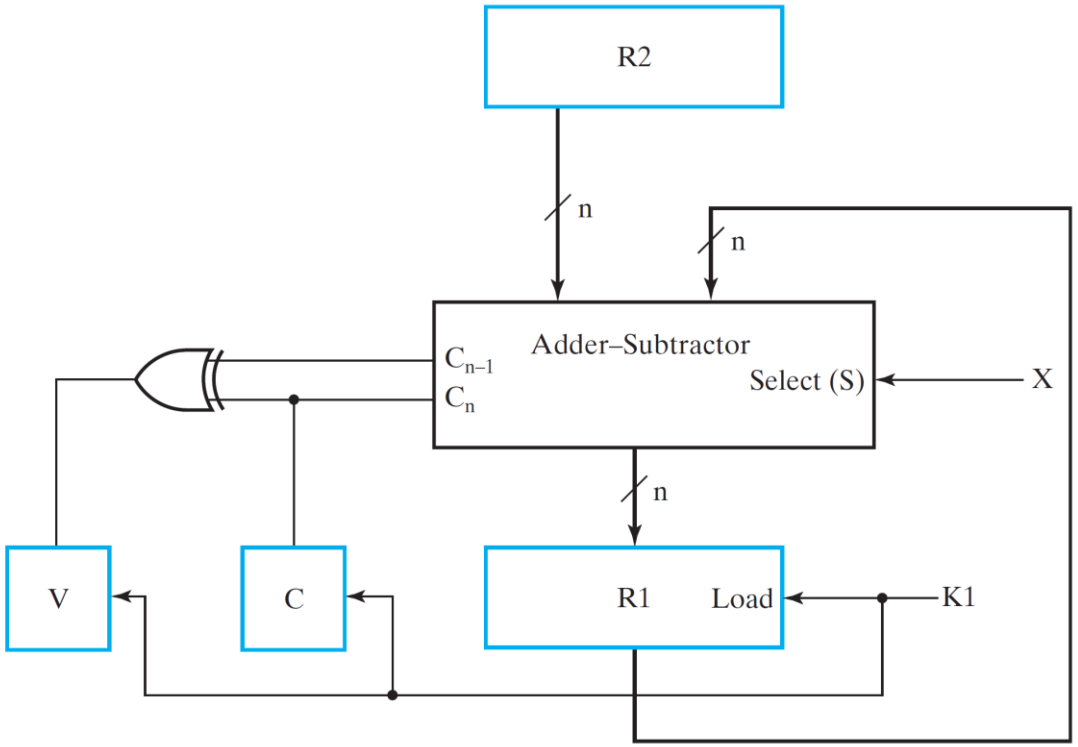
$$xT_2: \quad F \leftarrow 1$$

$$x'T_2: \quad F \leftarrow 0$$

Conditional Control Statements

- **Example**

$$\overline{X}K_1: R1 \leftarrow R1 + R2$$
$$XK_1: R1 \leftarrow R1 + \overline{R2} + 1$$



Register Cell Design

❑ Example

- A register *A* is to implement the following register transfers:

$$\begin{aligned}&\text{AND:} && A \leftarrow A \wedge B\\ &\text{EXOR:} && A \leftarrow A \oplus B\\ &\text{OR:} && A \leftarrow A \vee B\end{aligned}$$

- Unless specified otherwise, we assume that
  1. Only one of AND, EXOR, and OR is equal to 1, and
  2. For all of AND, EXOR, and OR equal to 0, the content of A remains unchanged.

  – A simple design approach for a register cell with conditions 1 and 2 uses a register with parallel load constructed from *D* flip-flops with Enable (*EN* = *LOAD*)

$$\text{LOAD} = \text{AND} + \text{EXOR} + \text{OR}$$
$$D_i = A(t+1)_i = \text{AND} \cdot A_i B_i + \text{EXOR} \cdot (A_i \overline{B}_i + \overline{A}_i B_i) + \text{OR} \cdot (A_i + B_i)$$

Register Cell Design

❑ Example (cont…)

- A more complex approach is to design directly for *D* flip-flops using a sequential circuit design approach
- The State Table and Flip-Flop Inputs

Present State A	Next State A(t + 1)						
	(AND = 0) (EXOR = 0) (OR = 0)	(OR = 1) (B = 0)	(OR = 1) (B = 1)	(EXOR = 1) (B = 0)	(EXOR = 1) (B = 1)	(AND = 1) (B = 0)	(AND = 1) (B = 1)
0	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1

- The corresponding flip-flop input equation is written as

$$D_i = A(t+1)_i = \text{AND} \cdot A_i \cdot B_i + \text{EXOR} \cdot (A_i\overline{B}_i + \overline{A}_iB_i) + \text{OR} \cdot (A_i + B_i) + \overline{AND} \cdot \overline{EXOR} \cdot \overline{OR} \cdot A_i$$

Register Cell Design

Example (Cont…)

– Rewrite D_i in terms of minterms of variables A_i and B_i:

$$\begin{aligned} D_i &= (AND + OR + \overline{AND} \cdot \overline{EXOR} \cdot \overline{OR})(A_iB_i) + (EXOR + OR \\ &\quad + \overline{AND} \cdot \overline{EXOR} \cdot \overline{OR})(A_i\overline{B}_i) + (\mathrm{EXOR} + \mathrm{OR})(\overline{A}_iB_i) \\ &= (AND + OR + \overline{EXOR})(A_iB_i) + (EXOR + OR \\ &\quad + \overline{AND})(A_i\overline{B}_i) + (\mathrm{EXOR} + \mathrm{OR})(\overline{A}_iB_i) \end{aligned}$$

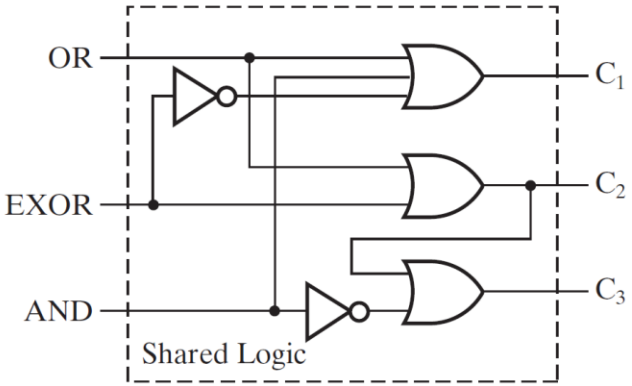
– Using C1, C2, and C3 as intermediate variables, we get

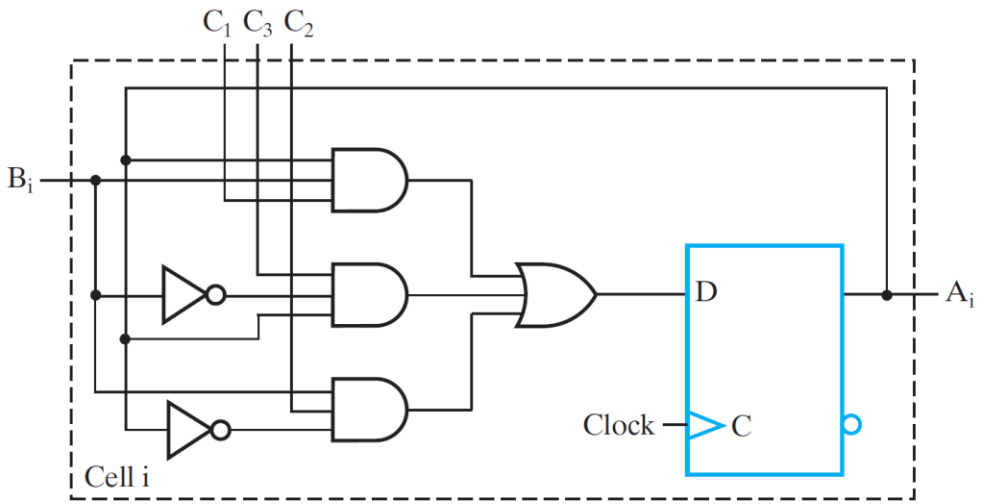
$$\begin{aligned} C_1 &= OR + AND + \overline{EXOR} \\ C_2 &= OR + EXOR \\ C_3 &= C_2 + \overline{AND} \\ D_i &= C_1A_iB_i + C_3A_i\overline{B}_i + C_2\overline{A}_iB_i \end{aligned}$$

Register Cell Design

- **Example (Cont…)**
  - Design of Cell-i

$$C_1 = OR + AND + \overline{EXOR}$$
$$C_2 = OR + EXOR$$
$$C_3 = C_2 + \overline{AND}$$
$$D_i = C_1 A_i B_i + C_3 A_i \overline{B}_i + C_2 \overline{A}_i B_i$$





Register Cell Design

❑ Example

- A register *A* is to implement the following register transfers:

$$\begin{aligned} &\text{SHL:} && A \leftarrow \text{sl } A \\ &\text{EXOR:} && A \leftarrow A \oplus B \\ &\text{ADD:} && A \leftarrow A + B \end{aligned}$$

- Unless specified otherwise, we assume that
  1. Only one of SHL, EXOR, and ADD is equal to 1, and
  2. For all of SHL, EXOR, and ADD equal to 0, the content of A remains unchanged.

Homework

Self Study

- **Text book 2 Digital Logic and Computer Design (chapter 8)**
  - Section 8.5 :Fixed Point Binary Data
  - Section 8.6 : Overflow
  - Section 8.7 : Arithmetic Shift
  - Section 8.8 : Decimal Data
  - Section 8.9 :Floating Point Data
  - Section 8.10 : Non-numeric Data

Lesson-6:

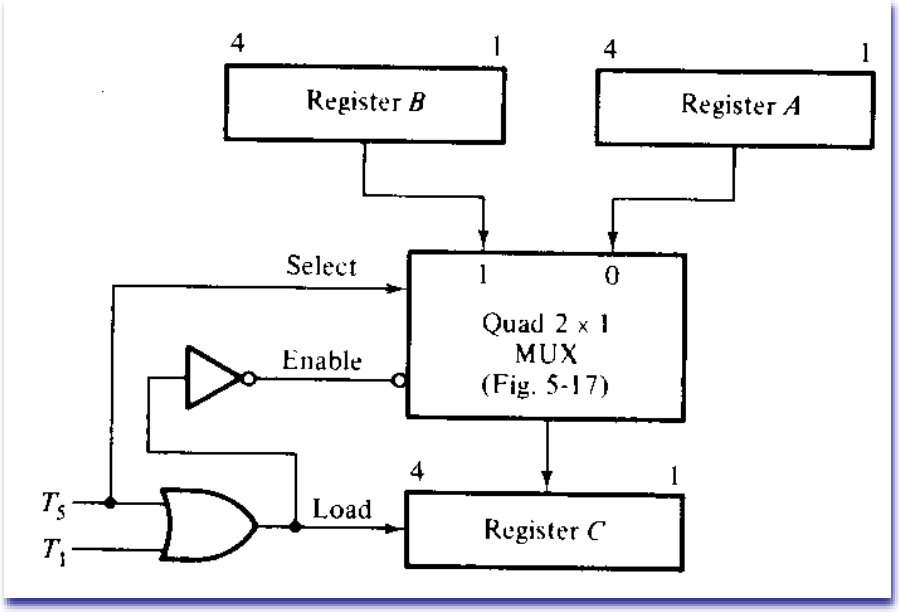
Topic	Lesson Learning Outcomes	Teaching-Learning Methodology	Assessment Method
Inter-register transfer	• To design busses with multiplexers and tristate buffers and use bus transfer operation • To design memory transfer model	Class Lecture, Question and answer	Test, exams, quiz, etc

Multiplexer-Based Transfers

- **Problem**
  - Draw the block diagram to implement the following statements

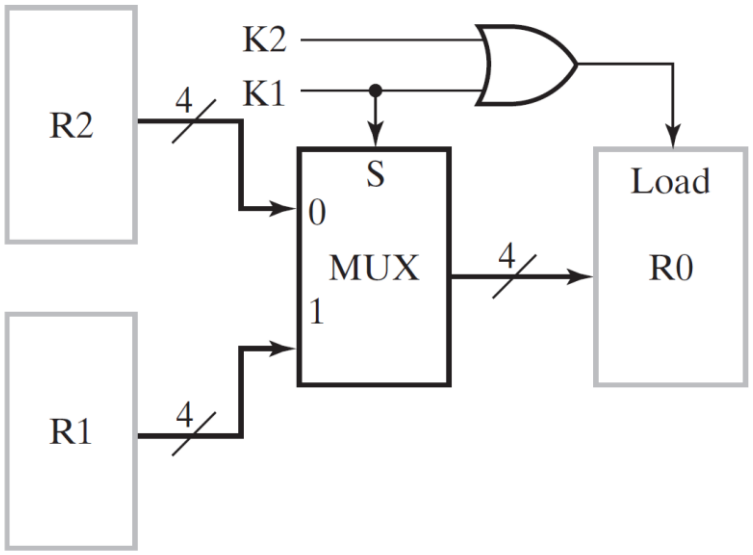
$$T_1: \quad \mathbf{C} \leftarrow \mathbf{A}$$

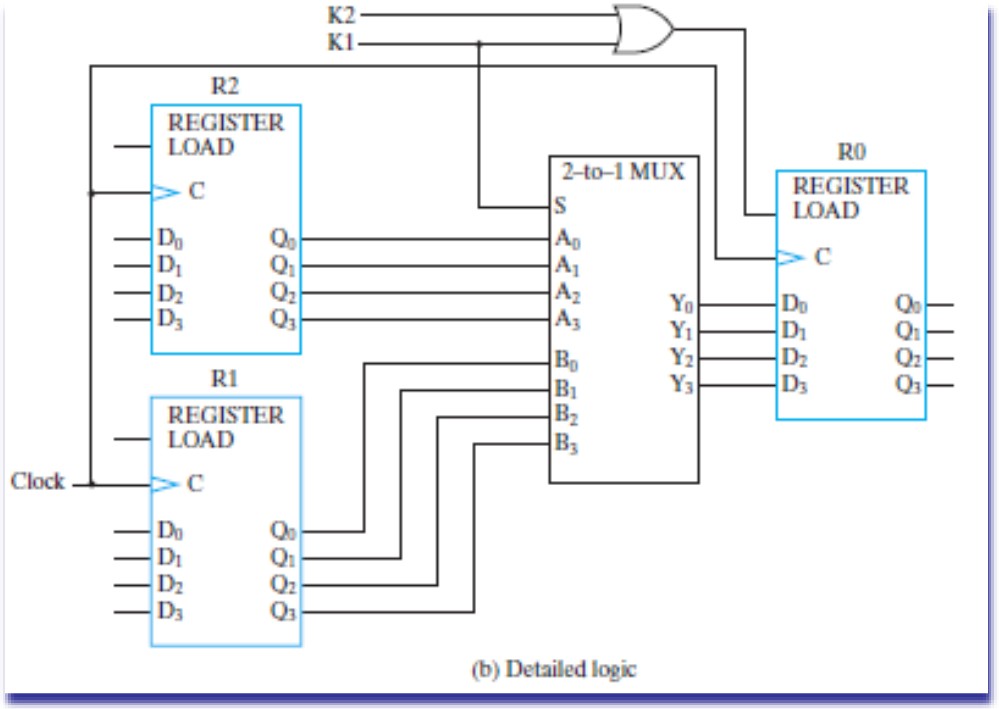
$$T_5: \quad \mathbf{C} \leftarrow \mathbf{B}$$



Multiplexer-Based Transfers

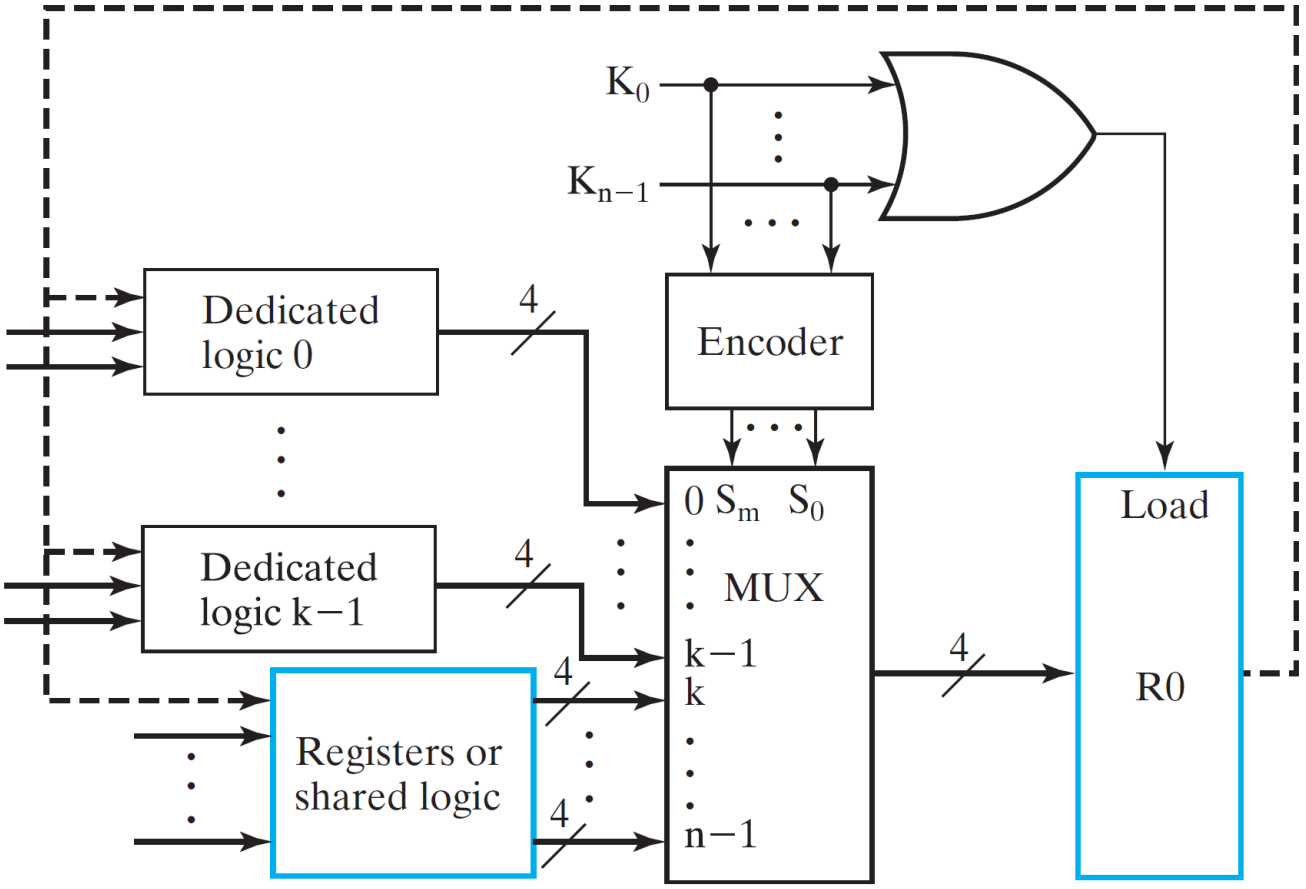
- Use of Multiplexers to Select Between Two Registers
  - if (K1 = 1) then (R0→R1) else if (K2 = 1) then (R0 → R2)
  - K1: R0 → R1, K1' K2: R0 → R2





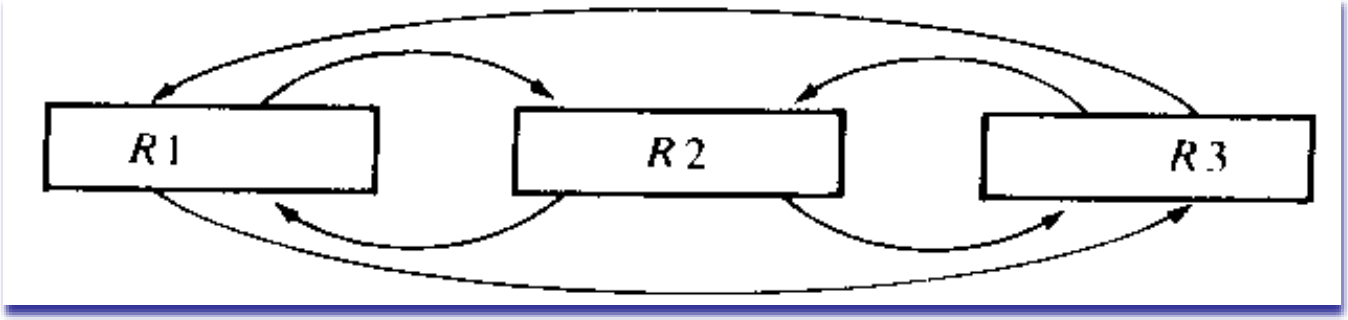
Multiplexer-Based Transfers

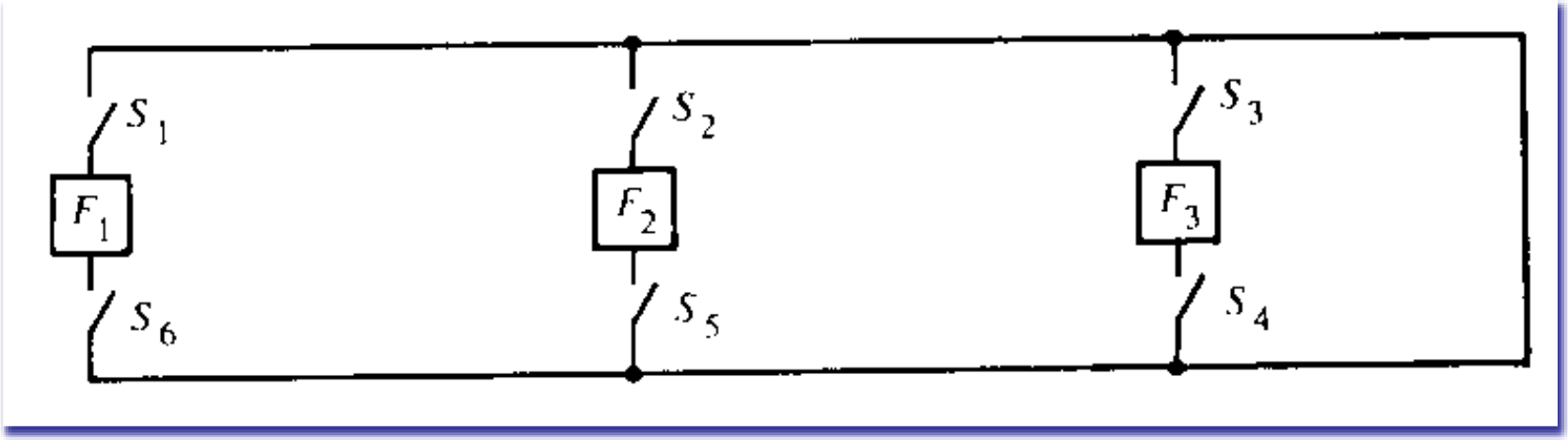
- Generalization of Multiplexer Selection for *n* Sources



Bus Transfer

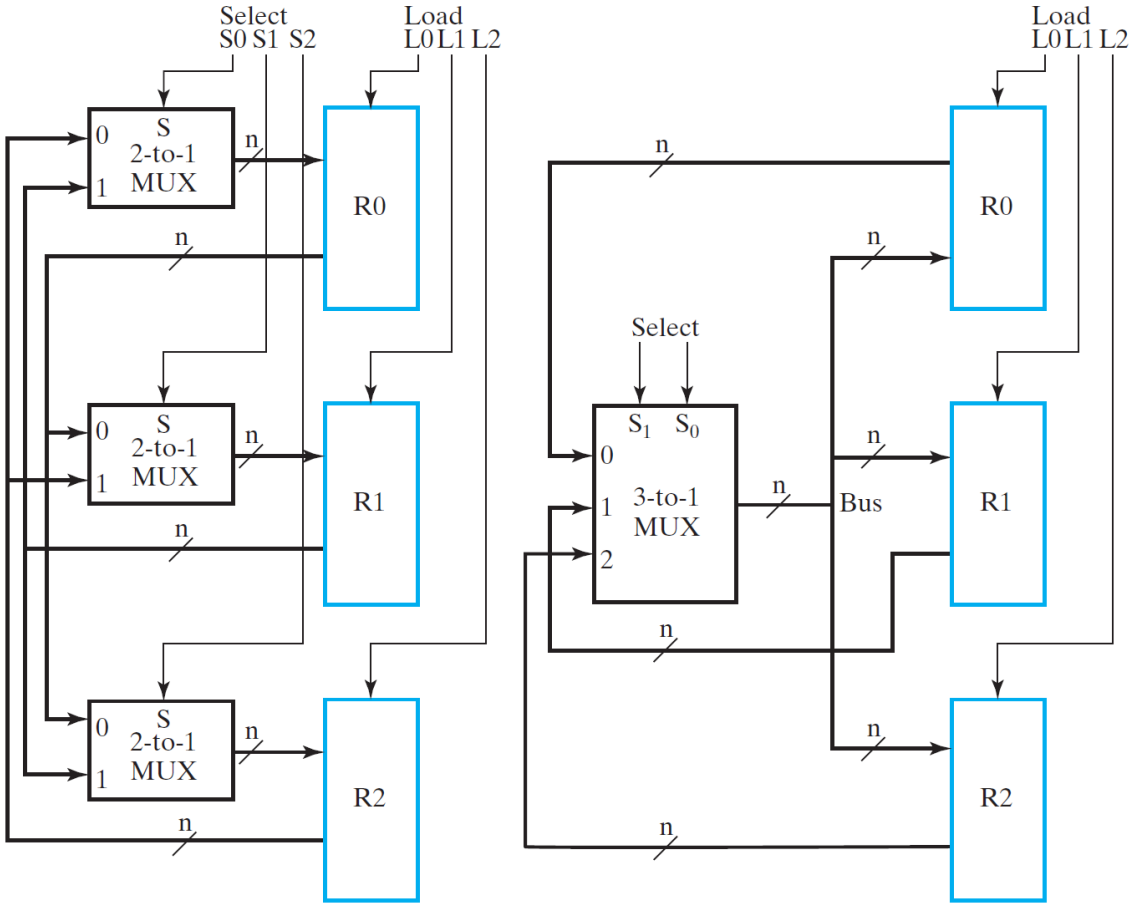
- **The Bus Transfer Concept**





Bus Transfer

- Single Bus versus Dedicated Multiplexers



(a) Dedicated multiplexers

(b) Single bus

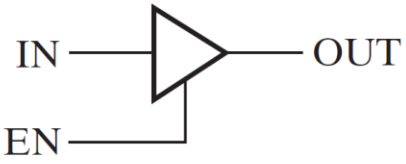
Example of Bus Transfer

	Select		Load		
Register Transfer	**S1**	**S0**	**L2**	**L1**	**L0**
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1, R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1, R1 \leftarrow R0$			Impossible		

Bus Transfer

❑ High-Impedance Outputs

- Another method for constructing a bus involves a type of gate called a *three-state buffer*
  - In adition to 0 and 1, it provides a third output value referred to as the *high-impedance state*
  - Denoted by Hi-Z or just plain Z or z
  - The Hi-Z value behaves as an open circuit, the output appears to be disconnected internally



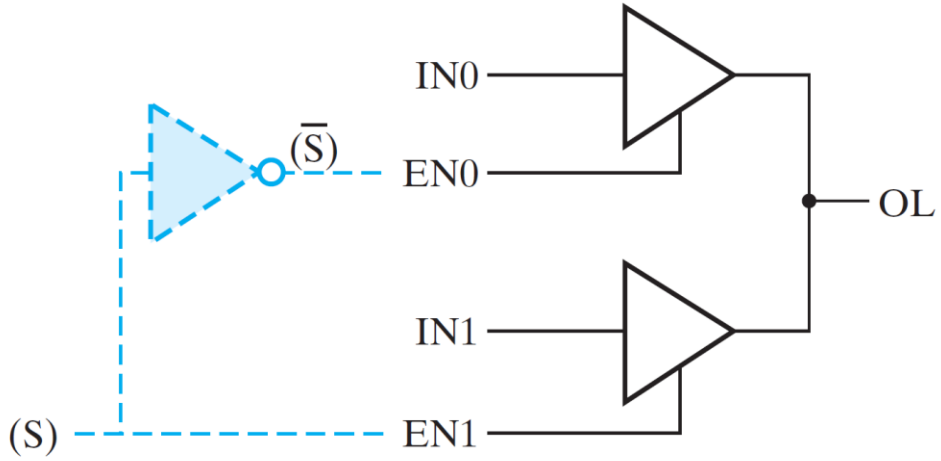
(a) Logic symbol

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

(b) Truth table

Bus Transfer

- Three-State Buffers Forming a Multiplexed Line OL



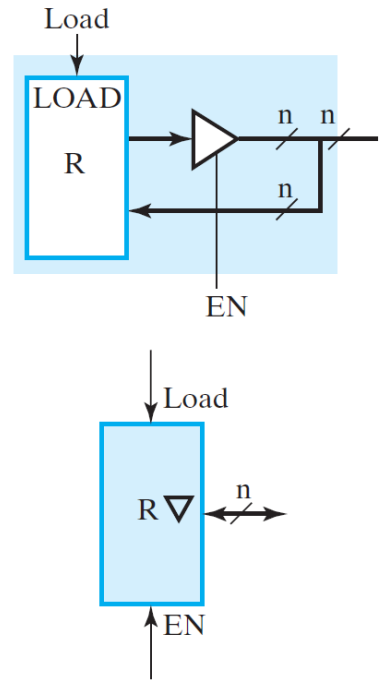
(a) Logic Diagram

EN1	EN0	IN1	IN0	OL
0	0	X	X	Hi-Z
(S) 0	$\overline{(S)}$ 1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	1	1	1
1	1	0	1	
1	1	1	0	

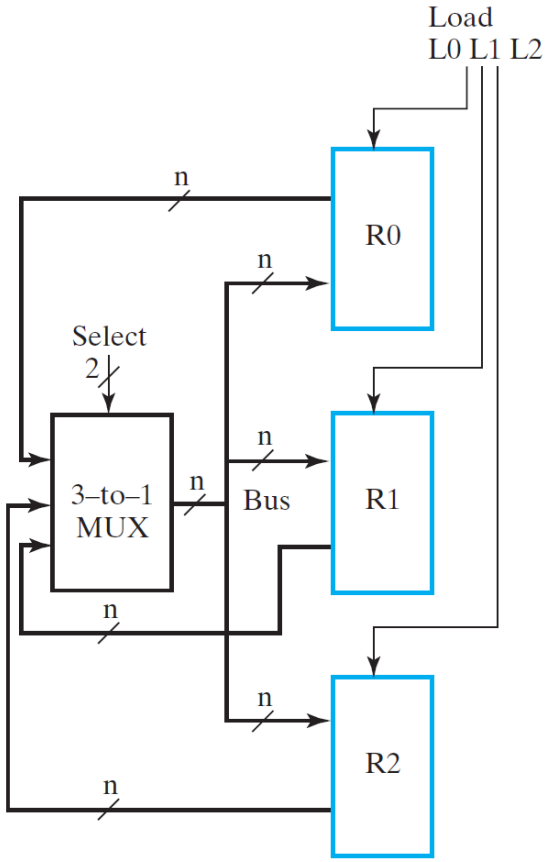
(b) Truth table

Bus Transfer

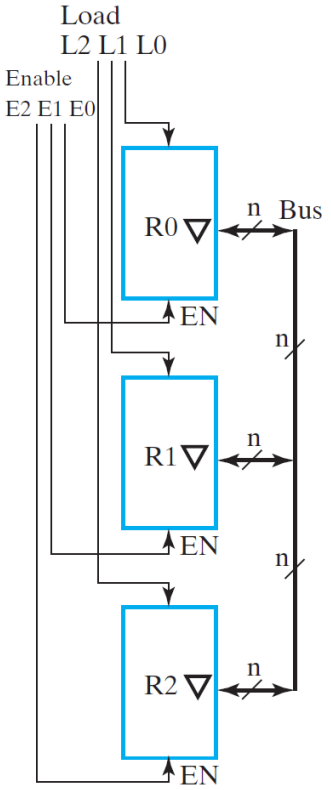
- Three-State Bus versus Multiplexer Bus



(a) Register with bidirectional input–output lines and symbol



(b) Multiplexer bus



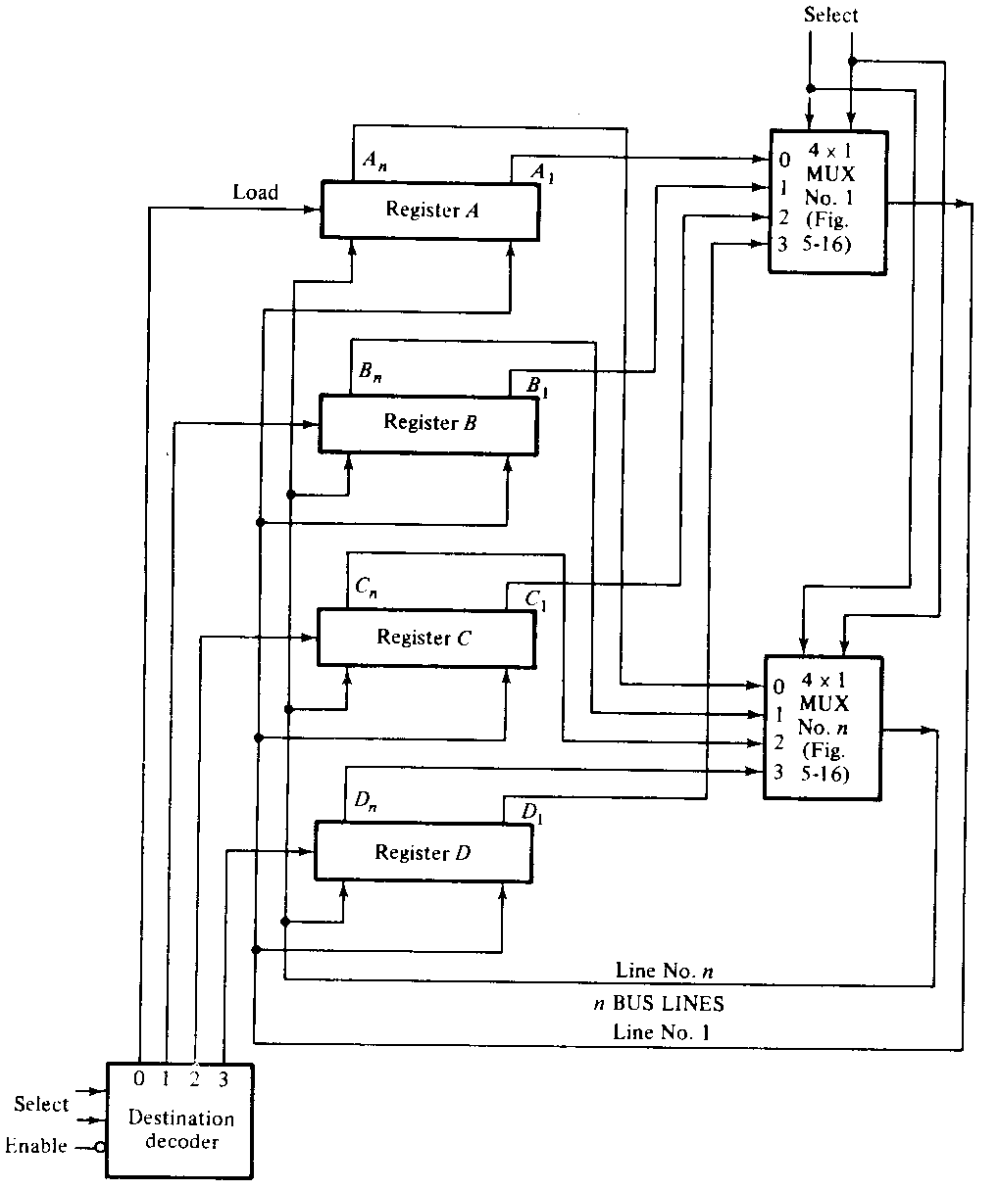
(c) Three-state bus using registers with bidirectional lines

Register Transfer

- **Implementation of Bus Transfer**
  - *Implement the statement.*

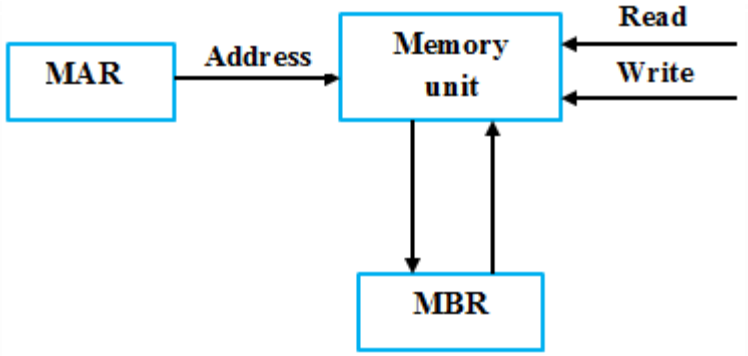
$$x'T_1: \quad \mathbf{A} \leftarrow \mathbf{B}$$

- Select source – 01
- Select destination – 00
- Decoder enable – 0



Register Transfer

- **Memory Transfer**
  - Memory unit with two external register



- **Read operation**

$$R: \quad \mathbf{MBR} \leftarrow \mathbf{M}$$

- **Write operation**

$$W: \quad \mathbf{M} \leftarrow \mathbf{MBR}$$

Register Transfer

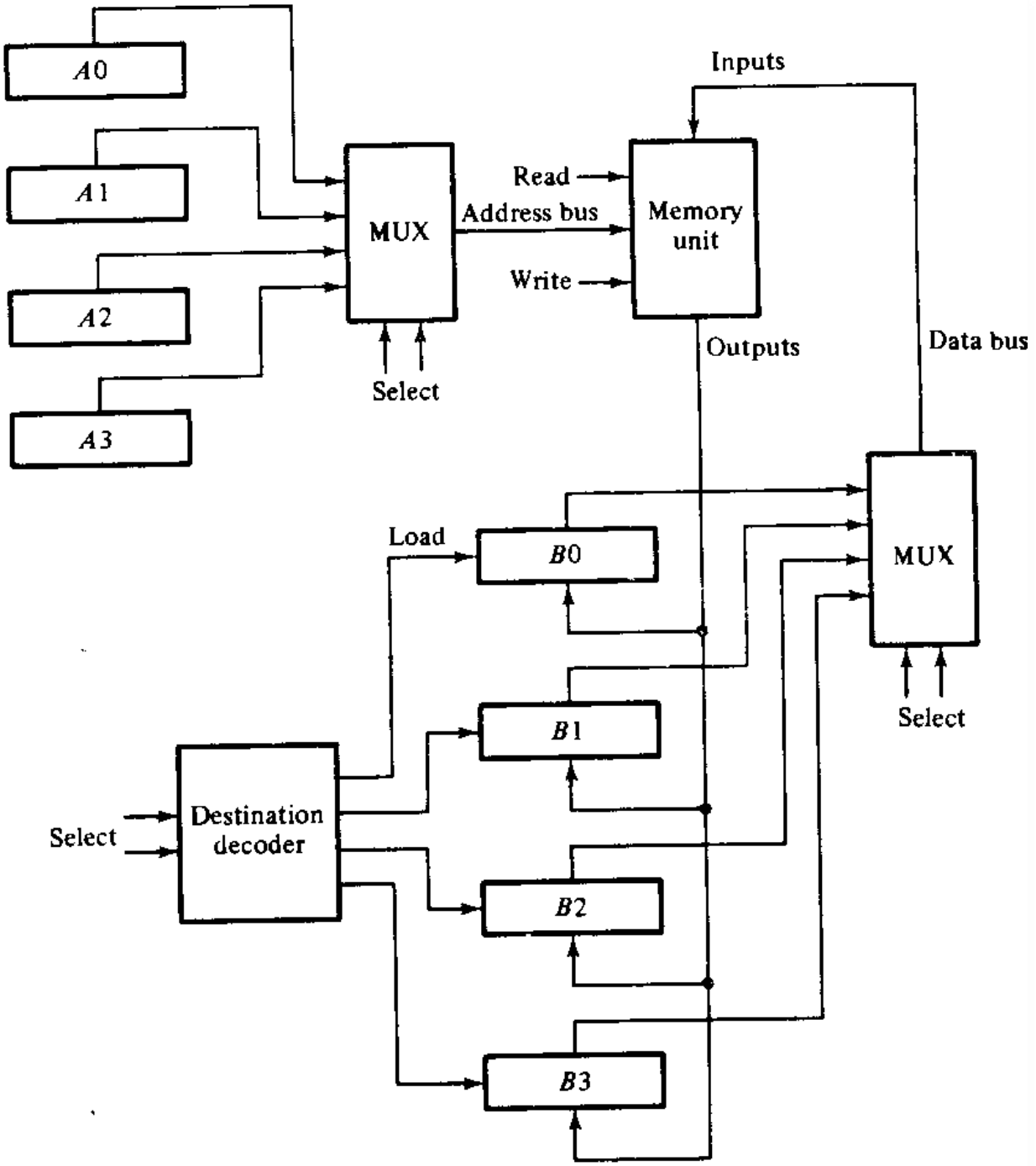
- **Memory Transfer**
  - Memory unit with multiple external registers
    - Multiple MAR form a address bus
    - Multiple MBR form a data bus
  - Multiplexers are used to form a bus.
  - Example read and write operations

$$R: \quad \mathbf{B0} \leftarrow \mathbf{M[A3]}$$

$$W: \quad \mathbf{M[A1]} \leftarrow \mathbf{B2}$$

Register Transfer

Memory transfer



Lesson-7:

Topic	Lesson Learning Outcomes	Teaching-Learning Methodology	Assessment Method
Instruction Code	• To distinguish between general purpose and special purpose digital system. • To understand the instruction code format • To know how Instruction codes are stored in memory.	Class Lecture Question and answer	Test, exams, quiz, etc

Instruction Codes

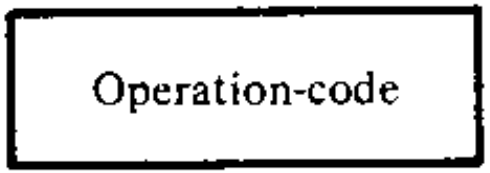
- **Special purpose digital system**
  - Performs a specific task
    - Sequence of micro-operations is fixed
    - Performs the same task over and over again.
- **General purpose digital system**
  - Example, Digital computers.
    - Capable of executing various operations
    - Users can control the operations by using *program*
    - It has the ability to store and execute instructions
      - Called the stored program concept

Instruction Codes

- **Instruction Code**
  - An instruction code is a group of bits that tell the computer to perform a specific operation.
  - Usually divided into several parts
  - The most basic part is operation part known as operation code (opcode).
    - A group of bits that define an operation such as
      - Add, subtract. multiply, shift, complement, etc.
    - *n* bit operation code is required for a given 2^n distinct operations.
    - Also contains the Address of memeory.

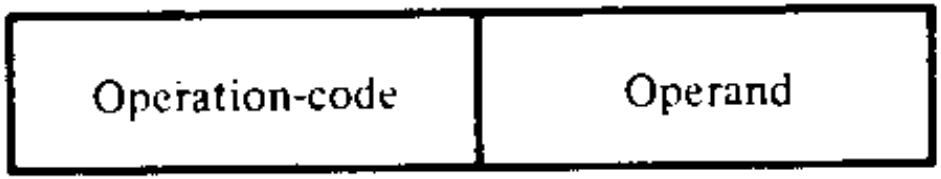
Instruction Codes

❑ Instruction Code format



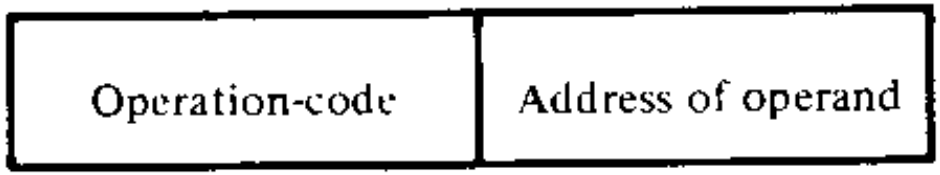
(a) Implied

EX: Clear, complement, transfer the contents of a register, etc



(b) Immediate operand

EX: Add, transfer the operand to a register, etc

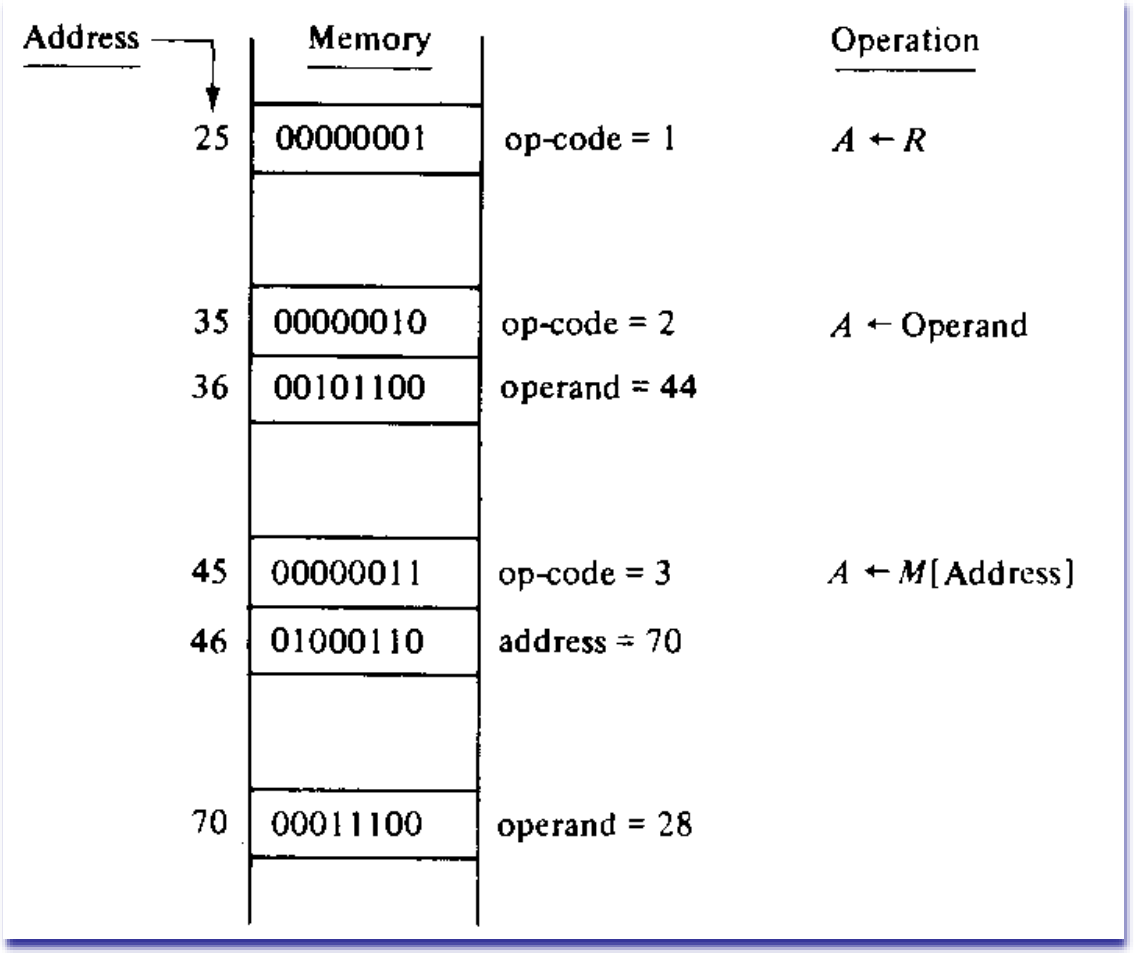


(c) Direct address

EX: same as (b) except memory with address is used instead of register.

Instruction Codes

- **Memory representation of instructions**



Instruction Codes

- **Micro-operation versus Macro-operations**
  - If the hardware requires only one control function then the operation is said to be micro-operation.
  - If the hardware requires more than one control function then the operation is said to be macro-operation.
  - To determine a register transfer statement is micro-operation of a macro-operation, we need to know the exact hardware configuration.

Instruction Codes

- **Micro-operation versus Macro-operations**
  - Consider the statement of example (fig. 8.13)
  - $A \leftarrow \text{operand}$ micro- or macro-operation?
  - It requires sequence of micro-operations
    1. Read the operation code from address 35
    2. Transfer the operation code to a transfer register
    3. The control decode the operation code and recognize it as an immediate operand instruction, so it read the operand from address 36.
    4. The operand read from memory is transferred into register A

Register Transfer

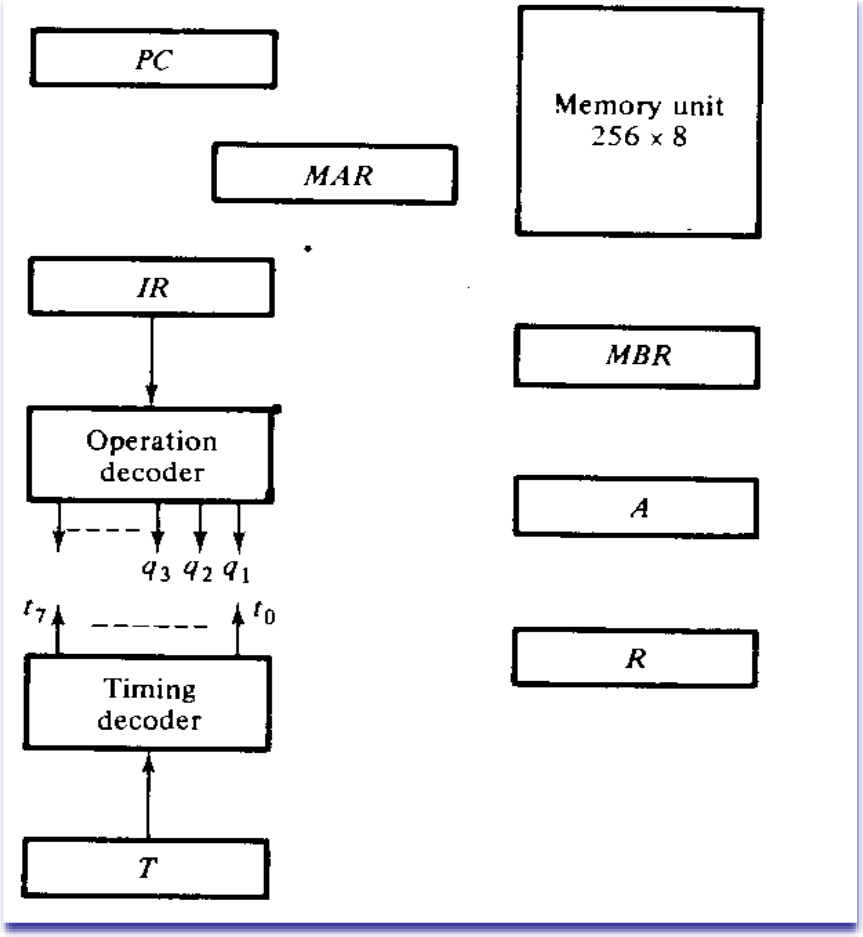
- **Summary**
  - Register transfer method can be used for the following tasks
    1. Define computer instructions concisely by macro-operation statements.
    2. Describe operations without specific hardware implementation.
    3. Define internal organization of digital systems by using control function and micro-operations.
    4. Design a digital system by specifying the hardware and their interconnections.

Lesson-8 and 9:

Topic	Lesson Learning Outcomes	Teaching-Learning Methodology	Assessment Method
Design of a simple Computer	• To learn the step by step procedure of designing a computer with simple examples	Class Lecture Homework	Test, exams, quiz, Assignment, etc

Design of a Simple Computer

- **Basic blocks of a simple computer**



Design of a Simple Computer

- **List of registers for a simple computer**

Symbol	Number of bits	Name of register	Function
MAR	8	Memory address register	Holds address for memory
MBR	8	Memory buffer register	Holds contents of memory word
A	8	*A* register	Processor register
R	8	*R* register	Processor register
PC	8	Program counter	Holds address of instruction
IR	8	Instruction register	Holds current operation code
T	3	Timing counter	Sequence generator

Design of a Simple Computer

- **Three instructions of a simple computer**

Operation code	Mnemonic		Description	Function
00000001	MOV	R	Move R to A	$A \leftarrow R$
00000010	LDI	OPRD	Load OPRD into A	$A \leftarrow$ OPRD
00000011	LDA	ADRS	Load operand specified by ADRS into A	$A \leftarrow M$[ADRS]

Design of a Simple Computer

- **Instruction fetch cycle**
  - PC initializes the first address of the program stored in memory
  - Start switch follows basic pattern of sequences
    - An operation code (opcode) whose address is in PC is read from memory to MBR.
    - PC is incremented by 1
    - Opcode is transferred from BBR to IR to decode.
    - Timing variables are used as control function

Design of a Simple Computer

- **Instruction fetch cycle**
  - Register transfer statements of instruction fetch cycle can be written as

t_0:	$MAR \leftarrow PC$	transfer op-code address
t_1:	$MBR \leftarrow M, PC \leftarrow PC + 1$	read op-code, increment PC
t_2:	$IR \leftarrow MBR$	transfer op-code to IR

  - Assume T starts with 000.
  - T is incremented by 1 in every clock
  - Instruction fetch cycle is same for all instructions

Design of a Simple Computer

- **Execution of instructions**
  - At t_3, the opcode is in IR and the output of the decoder is q_1, q_2 or q_3.
  - If q_1=*1* then the computer execute the instruction **MOV R** as,

$$q_1 t_3: \quad A \leftarrow R, T \leftarrow 0$$

Design of a Simple Computer

- **Execution of instructions**
  - The instruction **LDI OPND** is executed if $q_2=1$ as

q_2t_3:	$MAR \leftarrow PC$	transfer operand address
q_2t_4:	$MBR \leftarrow M, PC \leftarrow PC + 1$	read operand, increment PC
q_2t_5:	$A \leftarrow MBR, T \leftarrow 0$	transfer operand, go to fetch cycle.

Design of a Simple Computer

- **Execution of instructions**
  - The instruction **LDA ADRS** is executed if $q_3=1$ as

q_3t_3:	$MAR \leftarrow PC$	transfer next instruction address
q_3t_4:	$MBR \leftarrow M$, $PC \leftarrow PC + 1$	read ADRS, increment PC
q_3t_5:	$MAR \leftarrow MBR$	transfer operand address
q_3t_6:	$MBR \leftarrow M$	read operand
q_3t_7:	$A \leftarrow MBR$, $T \leftarrow 0$	transfer operand to A, go to fetch cycle

Design of a Simple Computer

- **The register transfer statements**

FETCH	t_0:	$MAR \leftarrow PC$
	t_1:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	t_2:	$IR \leftarrow MBR$
MOV	$q_1 t_3$:	$A \leftarrow R, T \leftarrow 0$
LDI	$q_2 t_3$:	$MAR \leftarrow PC$
	$q_2 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	$q_2 t_5$:	$A \leftarrow MBR, T \leftarrow 0$
LDA	$q_3 t_3$:	$MAR \leftarrow PC$
	$q_3 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	$q_3 t_5$:	$MAR \leftarrow MBR$
	$q_3 t_6$:	$MBR \leftarrow M$
	$q_3 t_7$:	$A \leftarrow MBR, T \leftarrow 0$

Design of a Simple Computer

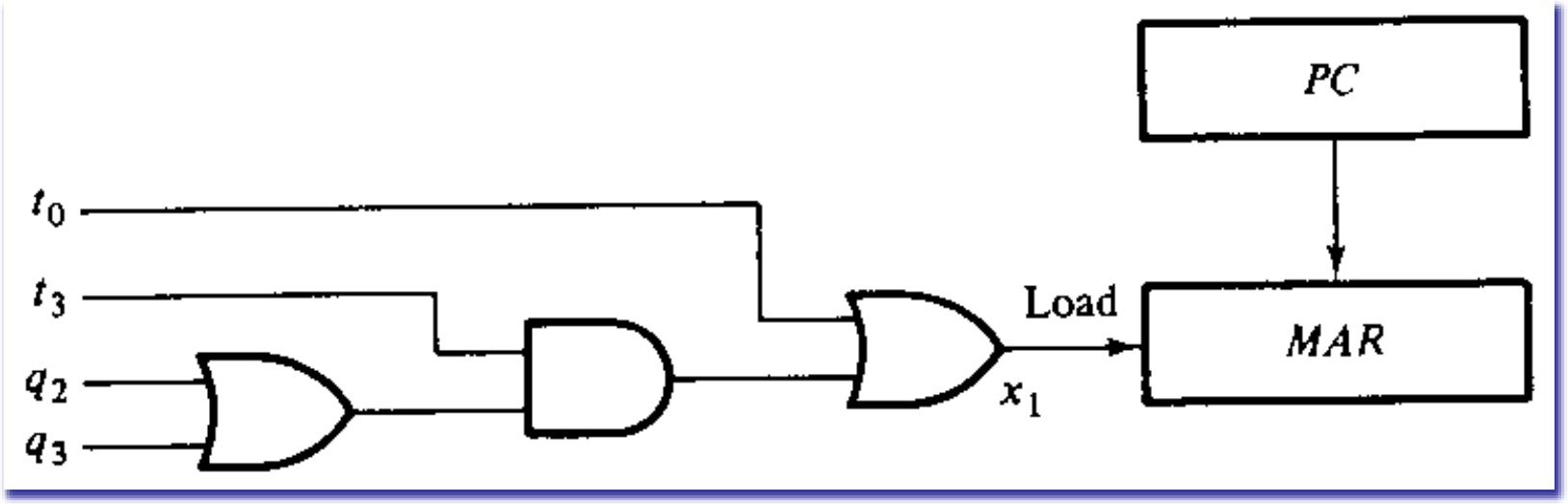
- **Design of computer**
  - The first step is scan the register transfer operations listed in table
    - Retrieve all the statements that perform the same micro operation.
  - For example **MAR** ← **PC** appear 3 times
  - Combine them into one statement as

$$t_0 + q_2t_3 + q_3t_3: \quad MAR \leftarrow PC$$

  - Note:
    - Control functions are Boolean function hence, **+** in control statement is **OR** operation.

Design of a Simple Computer

- **Design of computer**
  - Implementation of **MAR ← PC**



Here,

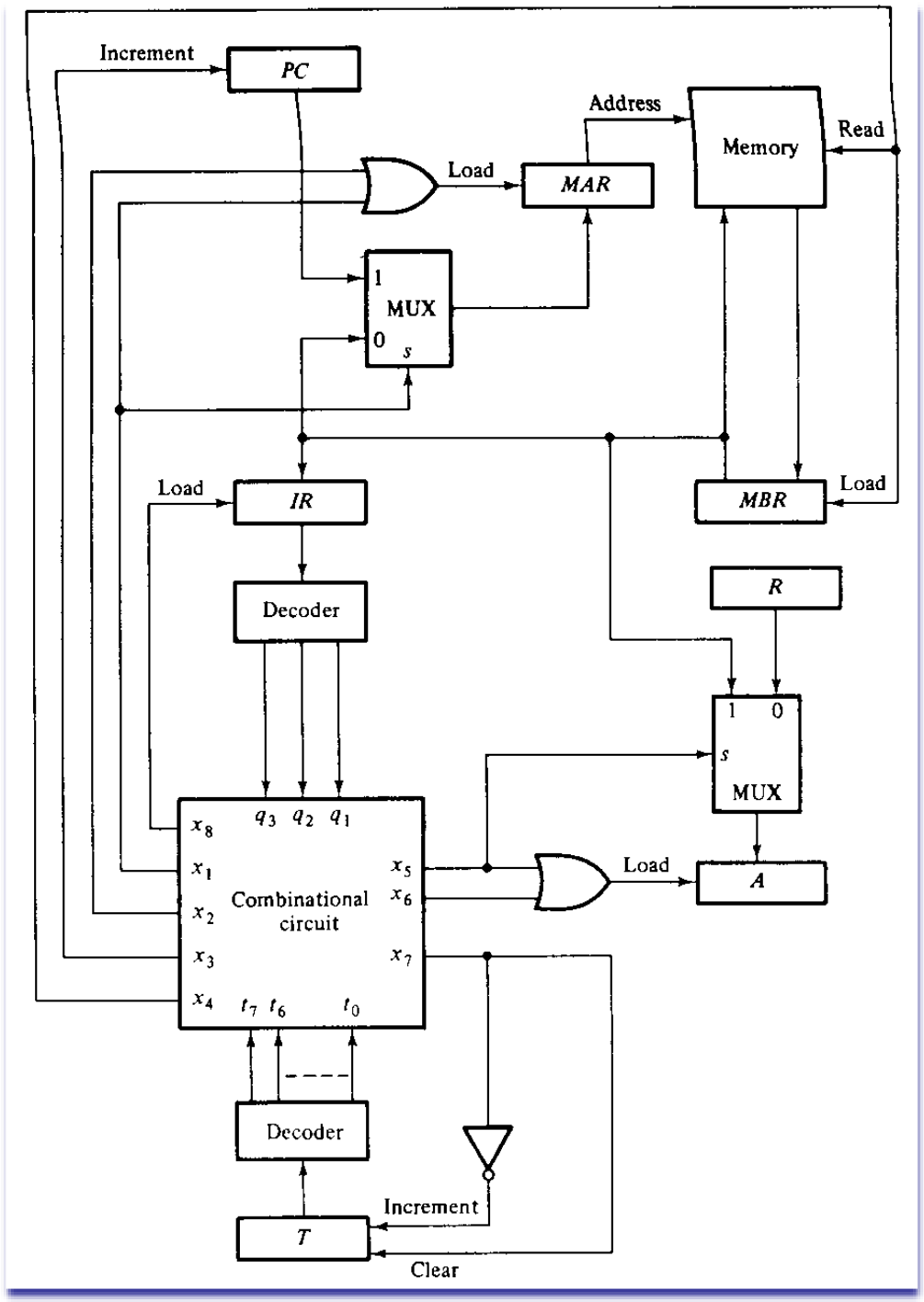
$$x_1 = t_0 + q_2t_3 + q_3t_3 = t_0 + (q_2 + q_3)t_3$$

Design of a Simple Computer

- **Design of computer**
  - There are 8 distinct micro operation can can be listed with control function as

$x_1 = t_0 + q_2t_3 + q_3t_3$:	$MAR \leftarrow PC$
$x_2 = q_3t_5$:	$MAR \leftarrow MBR$
$x_3 = t_1 + q_2t_4 + q_3t_4$:	$PC \leftarrow PC + 1$
$x_4 = x_3 + q_3t_6$:	$MBR \leftarrow M$
$x_5 = q_2t_5 + q_3t_7$:	$A \leftarrow MBR$
$x_6 = q_1t_3$:	$A \leftarrow R$
$x_7 = x_5 + x_6$:	$T \leftarrow 0$
$x_8 = t_2$:	$IR \leftarrow MBR$

  - Finally the computer block diagram is



$x_1 = t_0 + q_2t_3 + q_3t_3$:	$MAR \leftarrow PC$
$x_2 = q_3t_5$:	$MAR \leftarrow MBR$
$x_3 = t_1 + q_2t_4 + q_3t_4$:	$PC \leftarrow PC + 1$
$x_4 = x_3 + q_3t_6$:	$MBR \leftarrow M$
$x_5 = q_2t_5 + q_3t_7$:	$A \leftarrow MBR$
$x_6 = q_1t_3$:	$A \leftarrow R$
$x_7 = x_5 + x_6$:	$T \leftarrow 0$
$x_8 = t_2$:	$IR \leftarrow MBR$