ساختار و زبان کامپیوتر

فصل دو

مبانی طراحی کامپیوترها

Computer
Input
Datapath
Output
Processor
Memory

The specification for a computer consists of a description of its appearance to a programmer at the lowest level, its *instruction set architecture* (*ISA*).

From the ISA, a high-level description of the hardware to implement the computer, called the *computer architecture*, is formulated.

This architecture, for a simple computer, is typically divided into a *datapath* and a *control*.

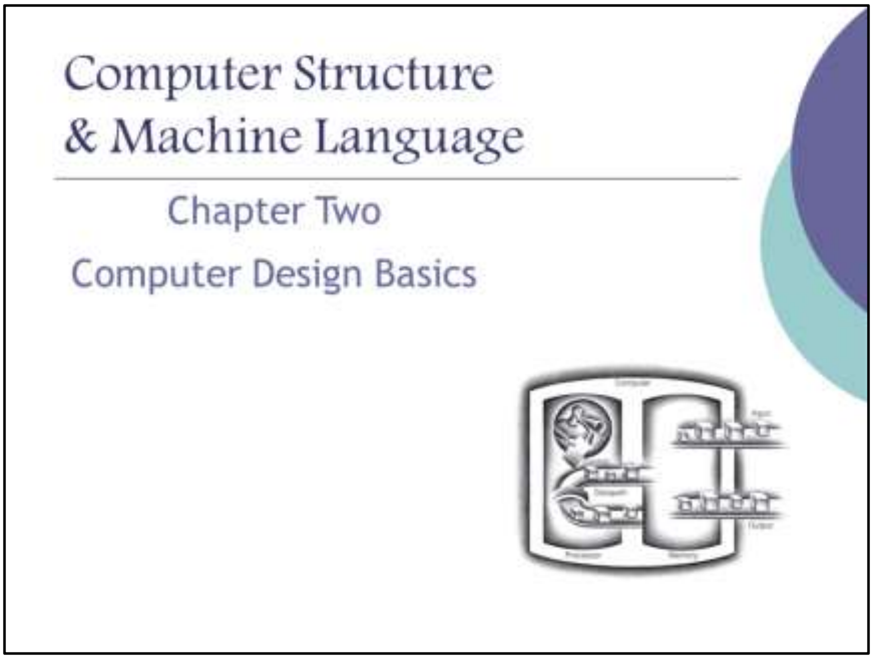
The datapath is defined by three basic components:

1. a set of registers,

2. the microoperations performed on data stored in the registers, and

3. the control interface.

The control unit provides signals that control the microoperations performed in the datapath and in other components of the system, such as memories.



Computer Structure & Machine Language

Chapter Two

Computer Design Basics

The specification for a computer consists of a description of its appearance to a programmer at the lowest level, its *instruction set architecture* (*ISA*).

From the ISA, a high-level description of the hardware to implement the computer, called the *computer architecture*, is formulated.

This architecture, for a simple computer, is typically divided into a *datapath* and a *control*.

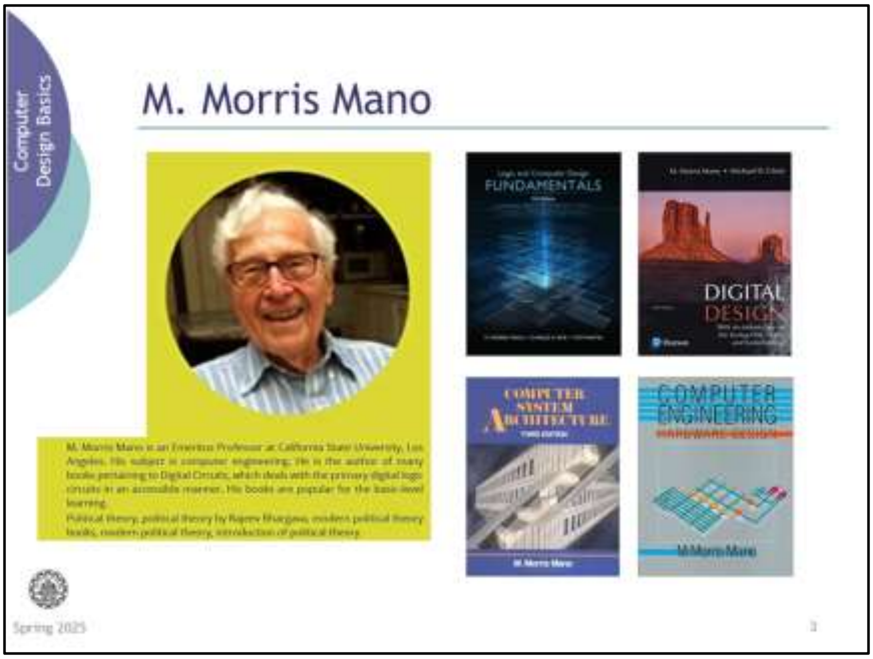
The datapath is defined by three basic components:

1. a set of registers,

2. the microoperations performed on data stored in the registers, and

3. the control interface.

The control unit provides signals that control the microoperations performed in the datapath and in other components of the system, such as memories.

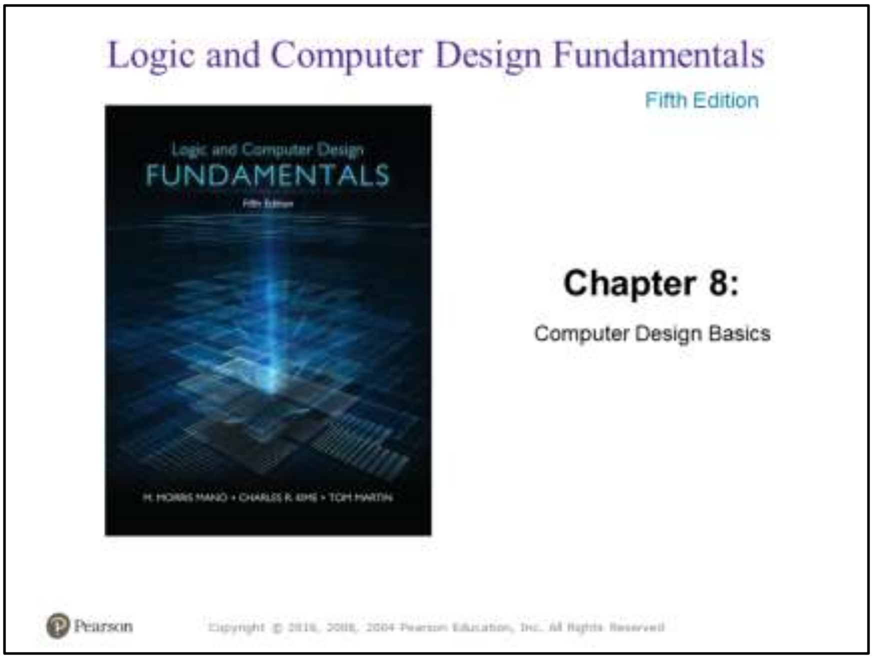


Computer Design Basics

M. Morris Mano

M. Morris Mano is an Emeritus Professor at California State University, Los Angeles. His subject is computer engineering; He is the author of many books pertaining to Digital Circuits, which deals with the primary digital logic circuits in an accessible manner. His books are popular for the basic-level learning.

Political theory, political theory by Rajeev Bhargava, modern political theory books, modern political theory, introduction of political theory



Logic and Computer Design Fundamentals

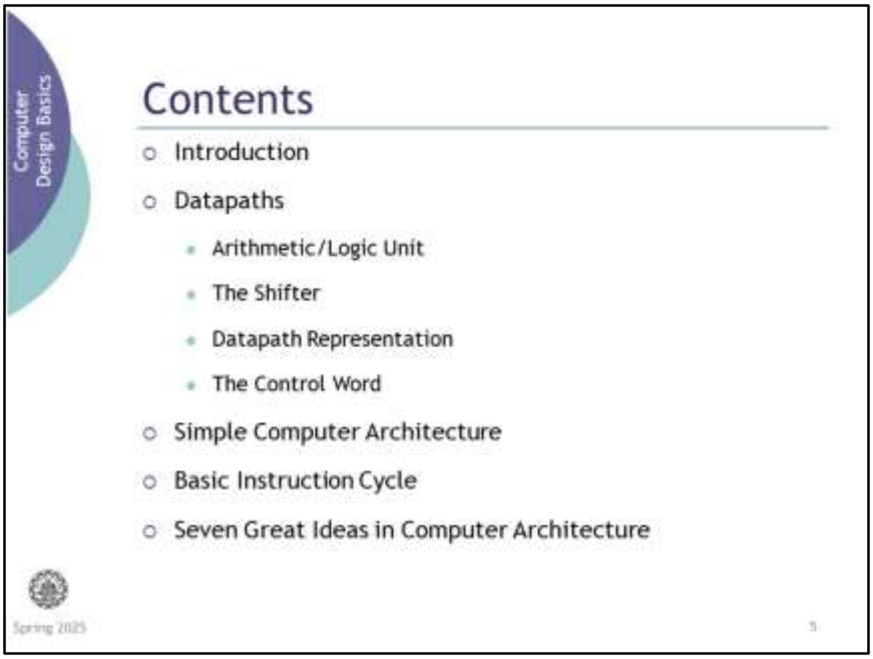
Fifth Edition

Logic and Computer Design
FUNDAMENTALS
Fifth Edition
M. MORRIS MANO • CHARLES R. KIME • TOM MARTIN

Chapter 8:

Computer Design Basics

Pearson



Computer Design Basics

Contents

- Introduction
- Datapaths
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- Simple Computer Architecture
- Basic Instruction Cycle
- Seven Great Ideas in Computer Architecture

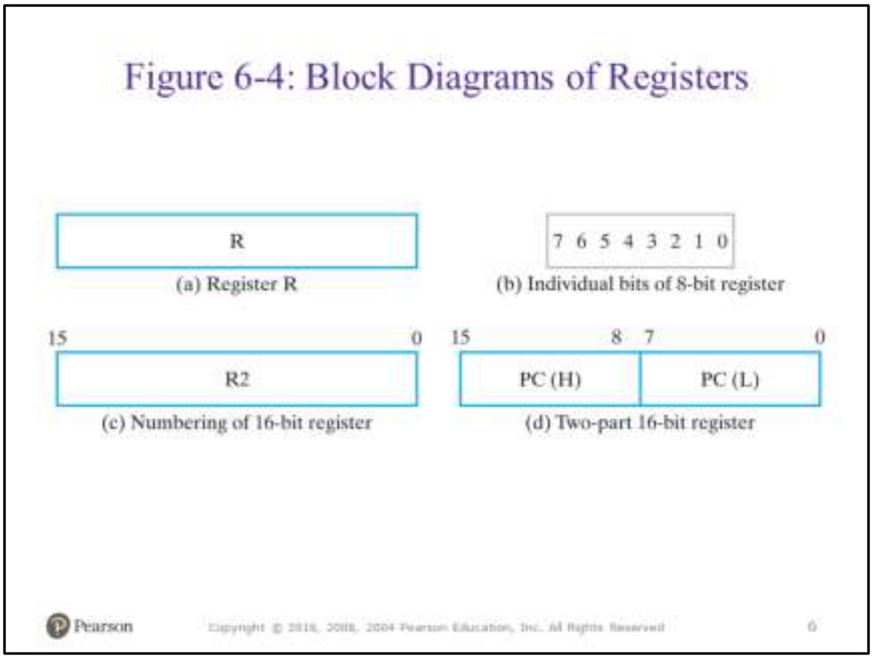


Figure 6-4: Block Diagrams of Registers

(a) Register R

(b) Individual bits of 8-bit register

(c) Numbering of 16-bit register

(d) Two-part 16-bit register

Registers in a digital system are denoted by uppercase letters (sometimes followed by numerals) that indicate the function of the register, e.g.:

AR for *address register*,

PC for *program counter*,

IR for *instruction register*,

and *R*2 for register 2.

The individual flip-flops in an *n*-bit register are typically numbered in sequence from 0 to *n* -1, starting with 0 in the least significant.

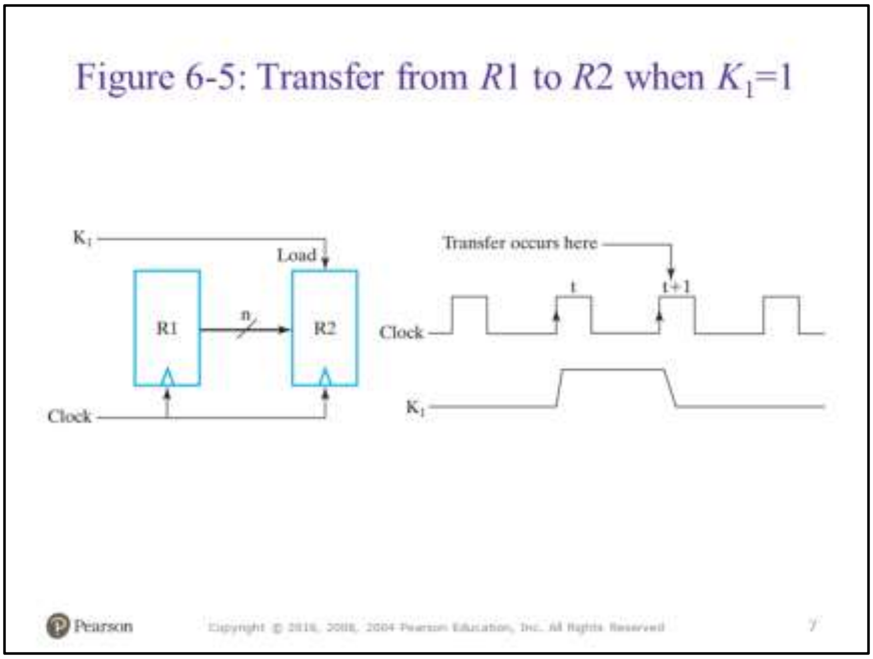


Figure 6-5: Transfer from $R1$ to $R2$ when $K_1=1$

Pearson

7

Data transferfrom one register to the other:

K1: R2←R1

K1: condition

R1: source

R2: destination

*K*1is 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*2are loaded into the register in parallel.

*K*1 returns to 0 on the positive clock transition at time *t* + 1, so that only a single transfer from *R*1 to *R*2 occurs.

Note that the **clock is not included** as a variable in the register-transfer statements.

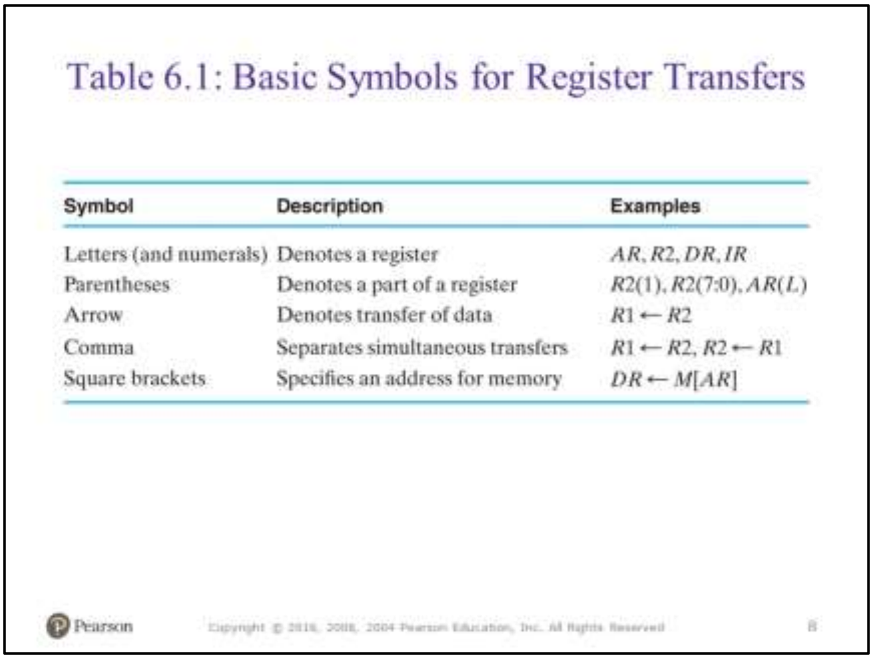


Table 6.1: Basic Symbols for Register Transfers

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	$DR \leftarrow M[AR]$

Pearson

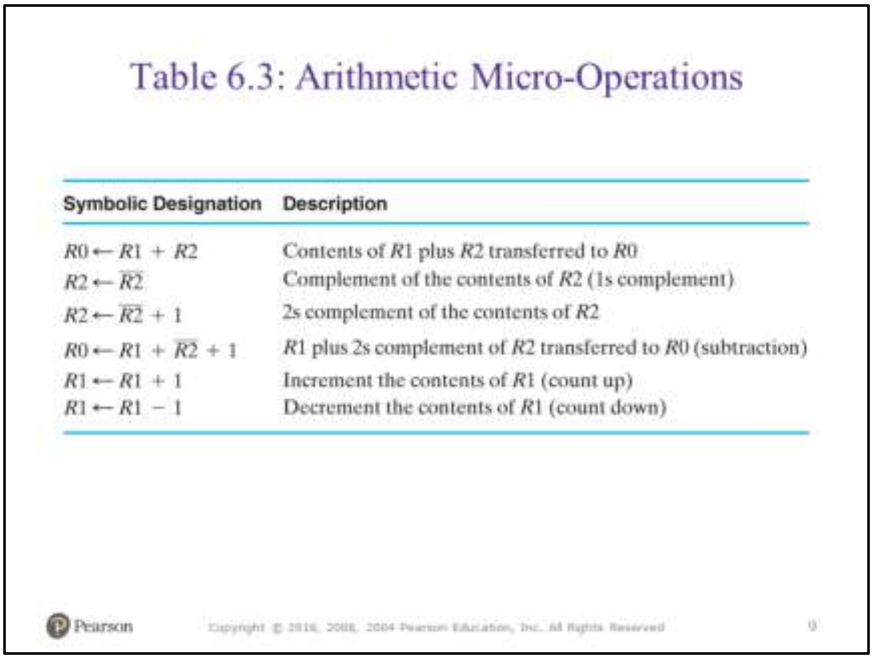


Table 6.3: Arithmetic Micro-Operations

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)

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 operations on data in registers.

3. *Logic* microoperations, which perform bit manipulation on data in registers.

4. *Shift* microoperations, which shift data in registers.

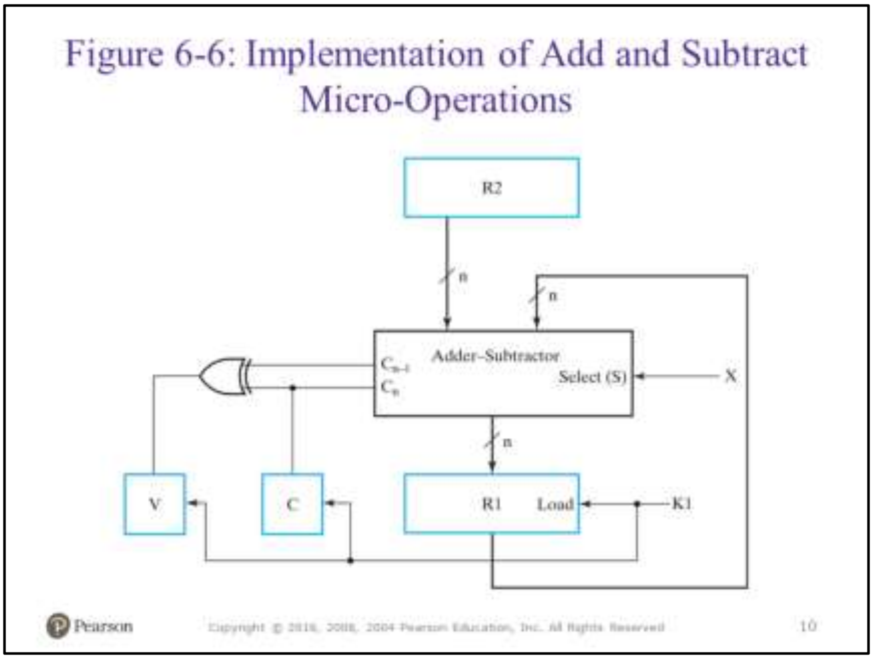


Figure 6-6: Implementation of Add and Subtract Micro-Operations

Pearson

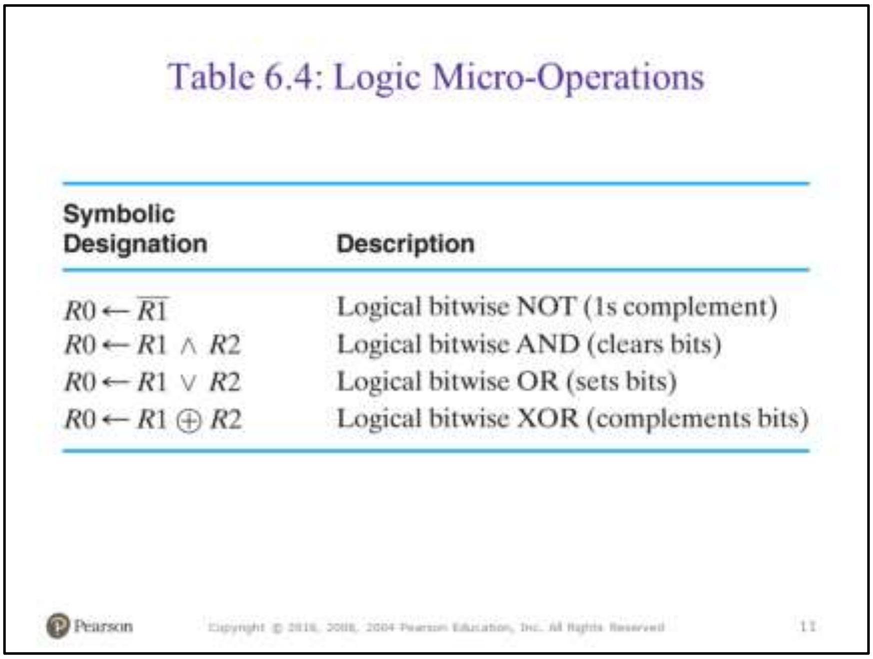


Table 6.4: Logic Micro-Operations

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)

Pearson

Logic microoperations are useful in manipulating the bits stored in a register.

The AND microoperation can be used for clearing one or more bits in a register to 0. (*selective clear/mask*)

10101101 10101011 *R*1(data)

00000000 11111111 *R*2(mask)

00000000 10101011 *R*1←*R*1 and *R*2

We also can apply *selective set*/*selective complement* via OR/XOR microoperations.

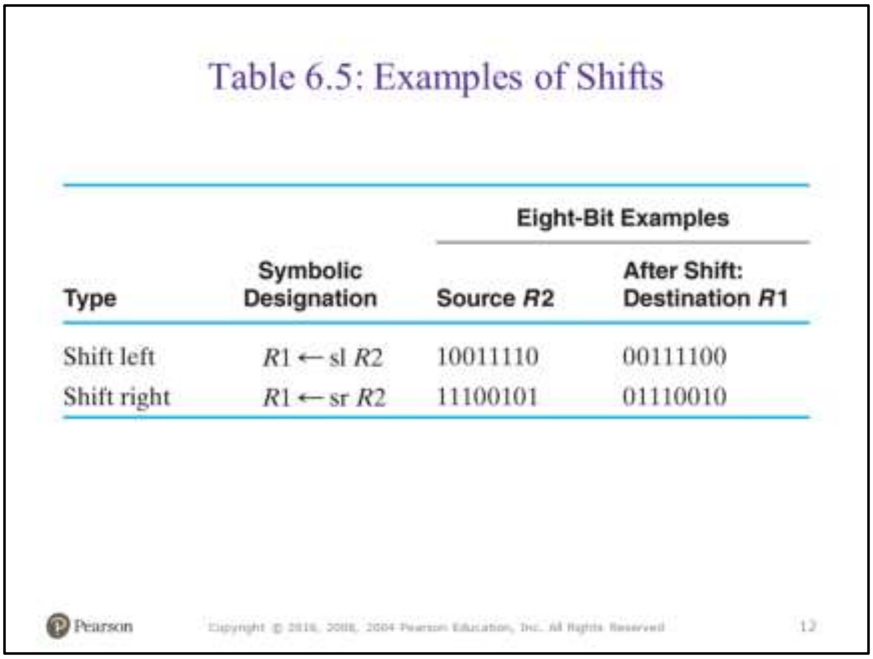


Table 6.5: Examples of Shifts

		Eight-Bit Examples	
Type	Symbolic Designation	Source *R*2	After Shift: Destination *R*1
Shift left	$R1 \leftarrow \mathrm{sl}\ R2$	10011110	00111100
Shift right	$R1 \leftarrow \mathrm{sr}\ R2$	11100101	01110010

Pearson

Shift microoperations are used for lateral movement of data.

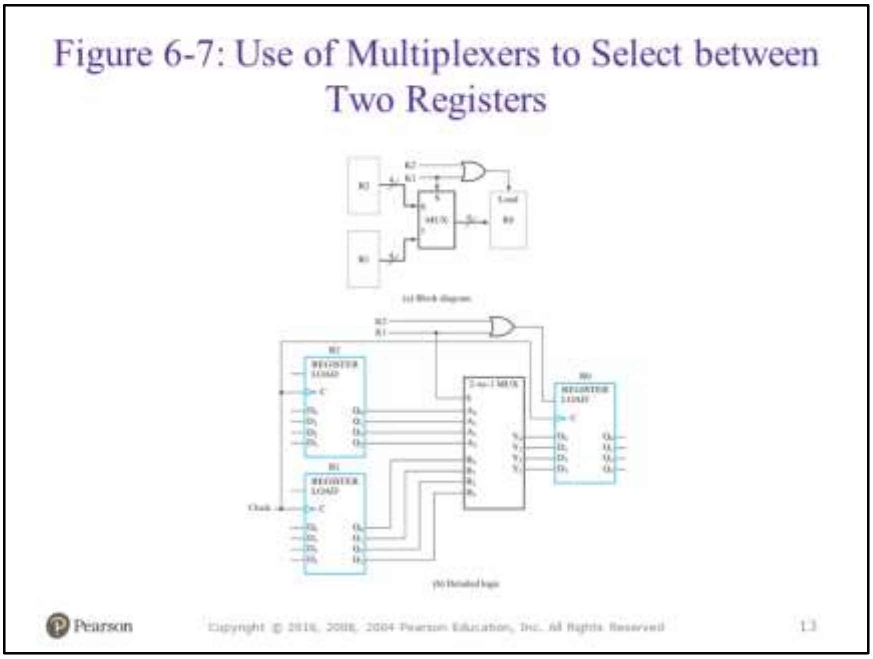


Figure 6-7: Use of Multiplexers to Select between Two Registers

(a) Block diagram

(b) Detailed logic

Pearson

13

if (*K*1= 1) then (*R*0←*R*1) else if (*K*2 = 1) then (*R0*←*R*2)

Equivalently:

K1: R0←R1, K1'K2: R0←R2

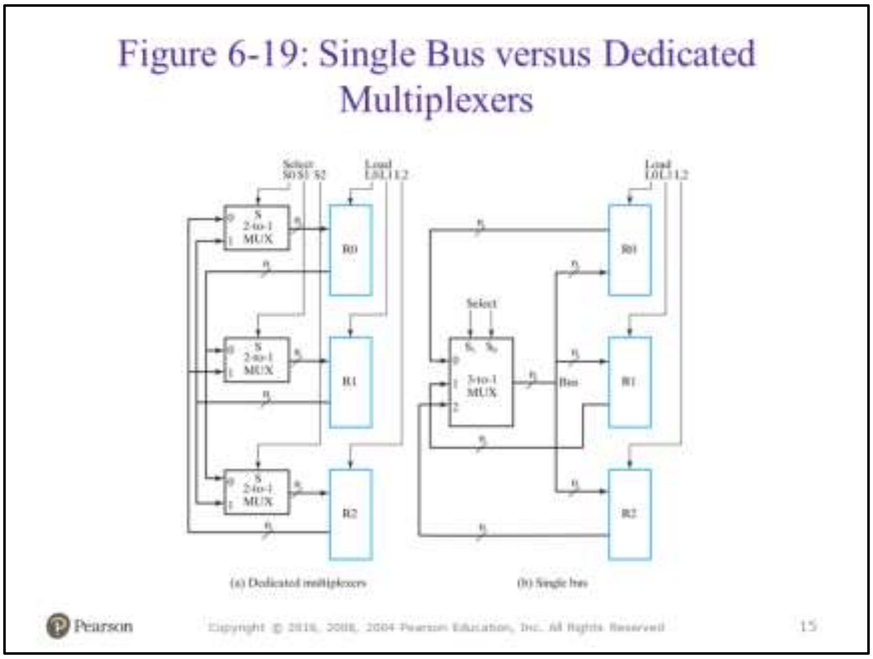


Figure 6-19: Single Bus versus Dedicated Multiplexers

(a) Dedicated multiplexers

(b) Single bus

Pearson

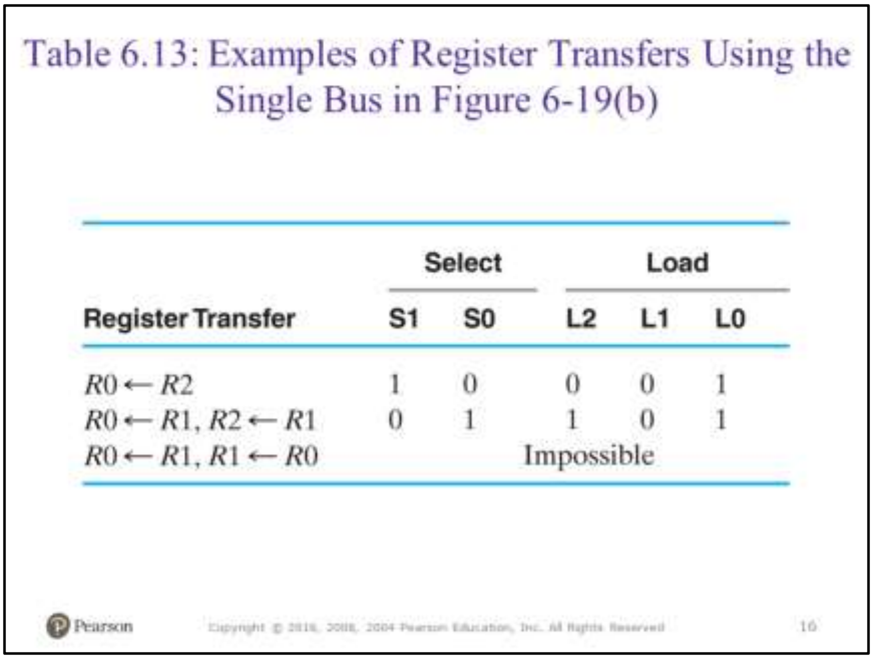


Table 6.13: Examples of Register Transfers Using the Single Bus in Figure 6-19(b)

	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		

Pearson

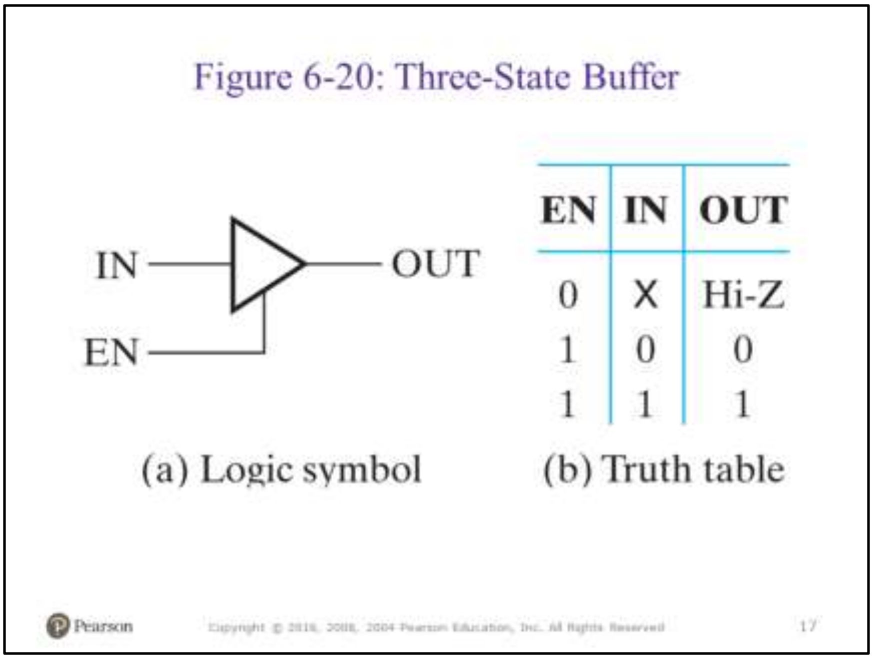


Figure 6-20: Three-State Buffer

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

(a) Logic symbol

(b) Truth table

High-Impedance Outputs:

The three-state buffer provides a third output value referred to as the *high-impedance state* and denoted by Hi-Z or just plain Z or z.

The Hi-Z value behaves as an open circuit, which means that, looking back into the circuit, we find that the output appears to be disconnected internally.

Thus, the output appears not to be there at all and, thus, is incapable of driving any attached inputs.

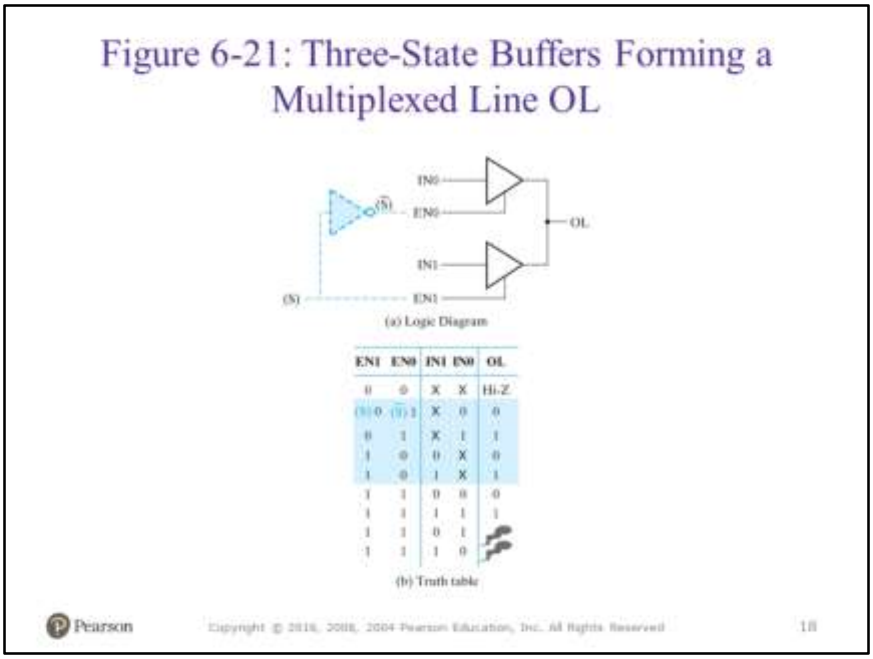


Figure 6-21: Three-State Buffers Forming a Multiplexed Line OL

(a) Logic Diagram

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

Pearson

Hi-Z outputs can be connected together provided that no two or more gates drive the line at the same time to opposite 0 and 1 values.

(gates with only logic 0 and logic 1 outputs cannot have their outputs connected together)

One way to avoid confliction is to use a **decoder** to generate the *EN* signals.

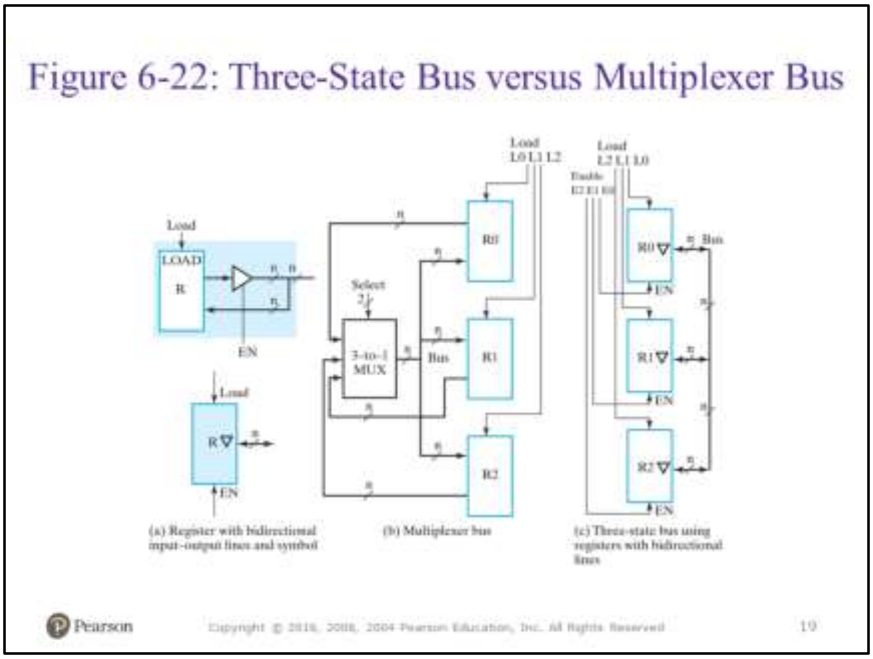


Figure 6-22: Three-State Bus versus Multiplexer Bus

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

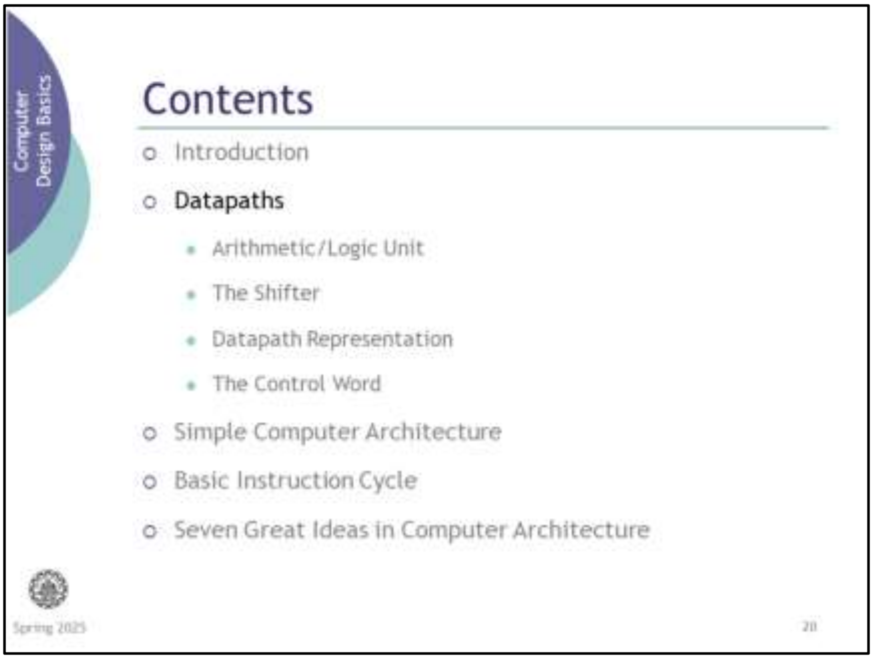
(b) Multiplexer bus

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

Pearson

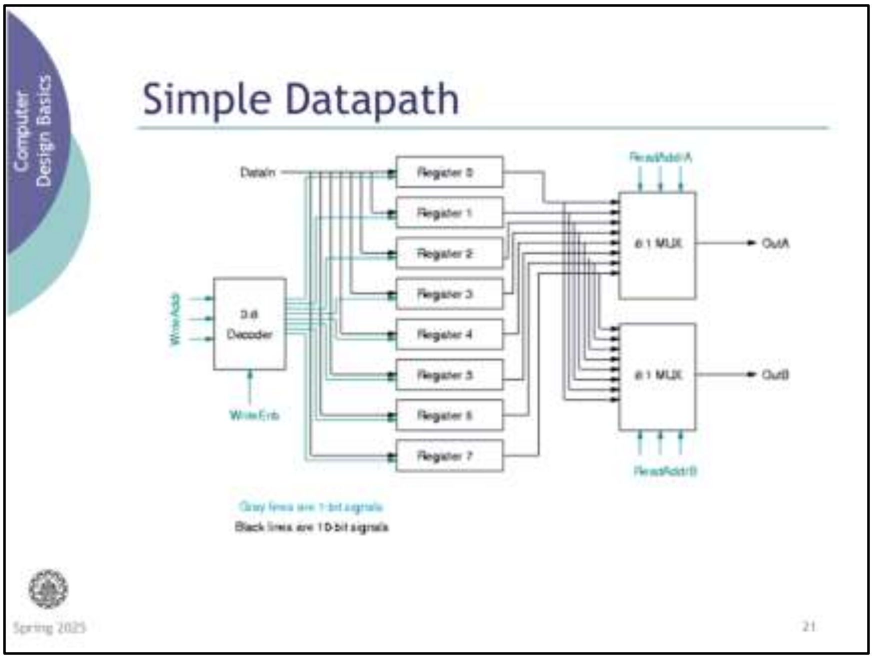
Bidirectional input/output

an output in the Hi-Z states, since it appears as an open circuit, can have an input attached to it internally, so that the Hi-Z output can act as both an output and an input



Contents

- Introduction
- **Datapaths**
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- Simple Computer Architecture
- Basic Instruction Cycle
- Seven Great Ideas in Computer Architecture



Computer Design Basics

Simple Datapath

Gray lines are 1-bit signals

Black lines are 16-bit signals

Spring 2025

21

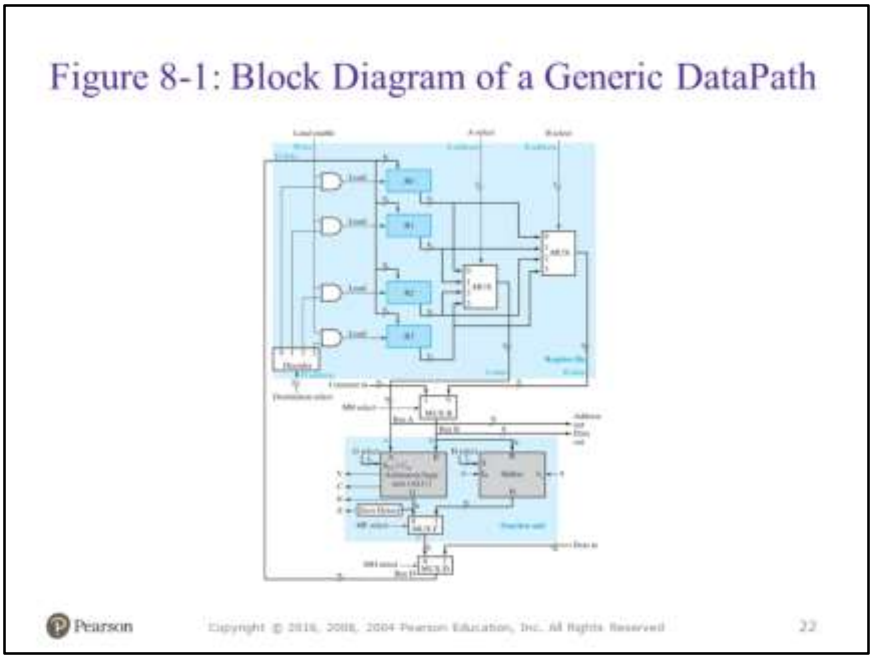


Figure 8-1: Block Diagram of a Generic DataPath

Pearson

22

The combination of a set of registers with a shared ALU and interconnecting paths is the datapath for the system.

Control inputs:

1. *A select*, to place the contents of *R*2 onto *A data* and, hence, Bus *A*.

2. *B select*, to place the contents of *R*3 onto the 0 input of MUX B; and *MB select*, to put the 0 input of MUX B onto Bus *B*.

3. *G select*, to provide the arithmetic operation *A* + *B*.

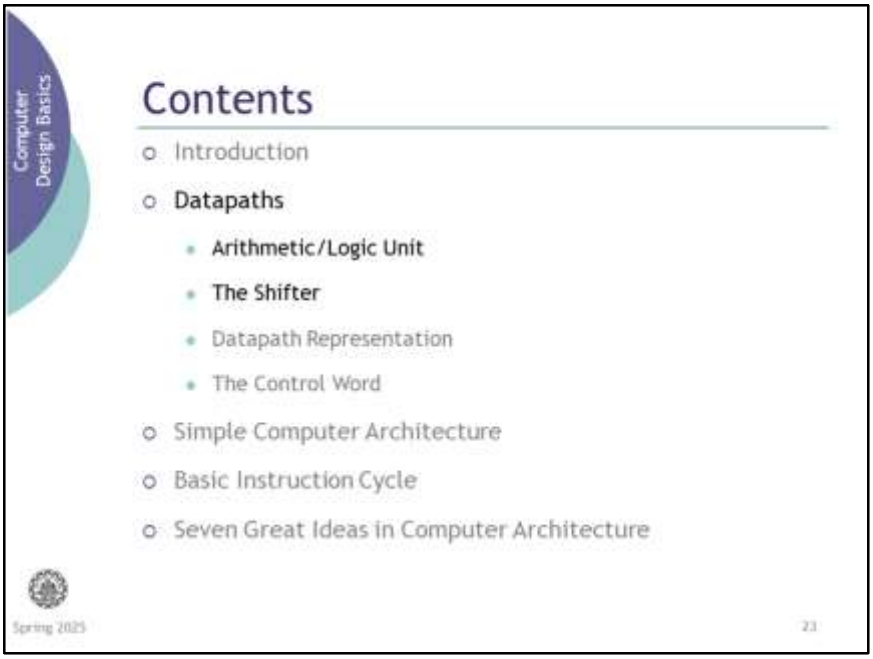
4. *H select*, to provide the shift operation.

5. *MF select*, to place the ALU output on the MUX F output.

6.. *MD select*, to place the MUX F output onto Bus *D*.

7. *Destination select*, to select *R*1 as the destination of the data on Bus *D*.

8. *Load enable*, to enable a register—in this case, *R*1—to be loaded.



Computer Design Basics

Contents

- Introduction
- Datapaths
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- Simple Computer Architecture
- Basic Instruction Cycle
- Seven Great Ideas in Computer Architecture

Spring 2025

23

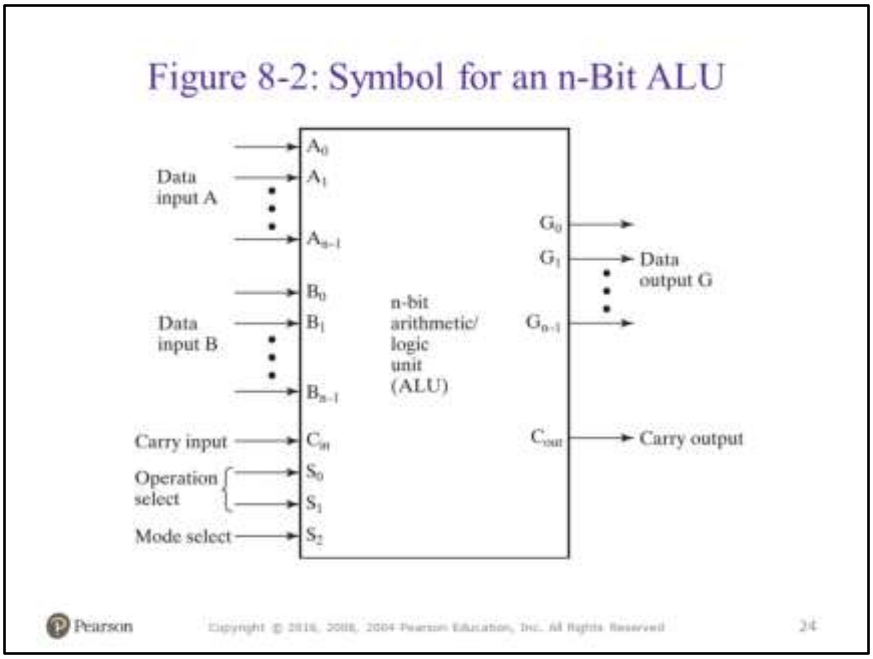


Figure 8-2: Symbol for an n-Bit ALU

Pearson Copyright © 2016, 2008, 2004 Pearson Education, Inc. All Rights Reserved 24

The ALU is a combinational circuit that performs a set of basic arithmetic and logicmicrooperations.

The selection lines are decoded within the ALU, so that k selection lines can specify up to 2kdistinct operations.

The mode-select input S2distinguishes between arithmetic and logic operations.

First, we design the arithmetic section. Then we design the logic section, and finally, we combine the two sections to form the ALU.

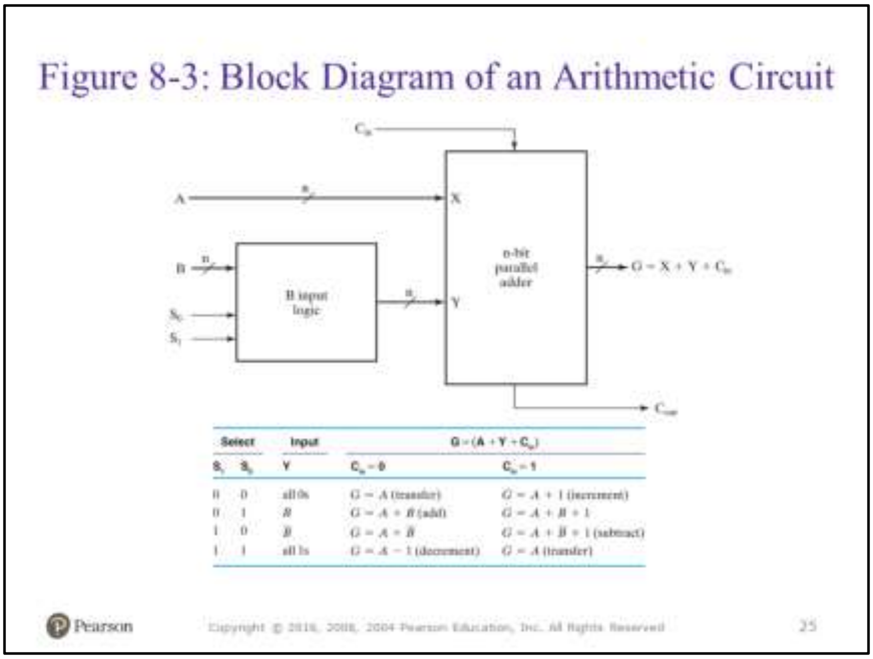


Figure 8-3: Block Diagram of an Arithmetic Circuit

Select		Input	$G = (A + Y + C_{in})$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\bar{B}$	$G = A + \bar{B}$	$G = A + \bar{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

Pearson

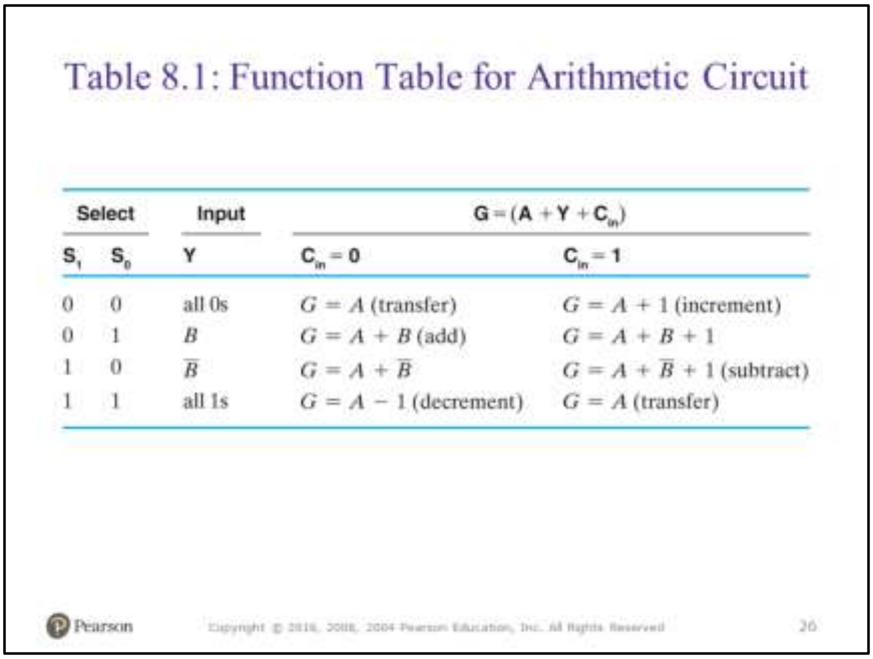


Table 8.1: Function Table for Arithmetic Circuit

Select		Input	$G = (A + Y + C_{in})$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

Pearson

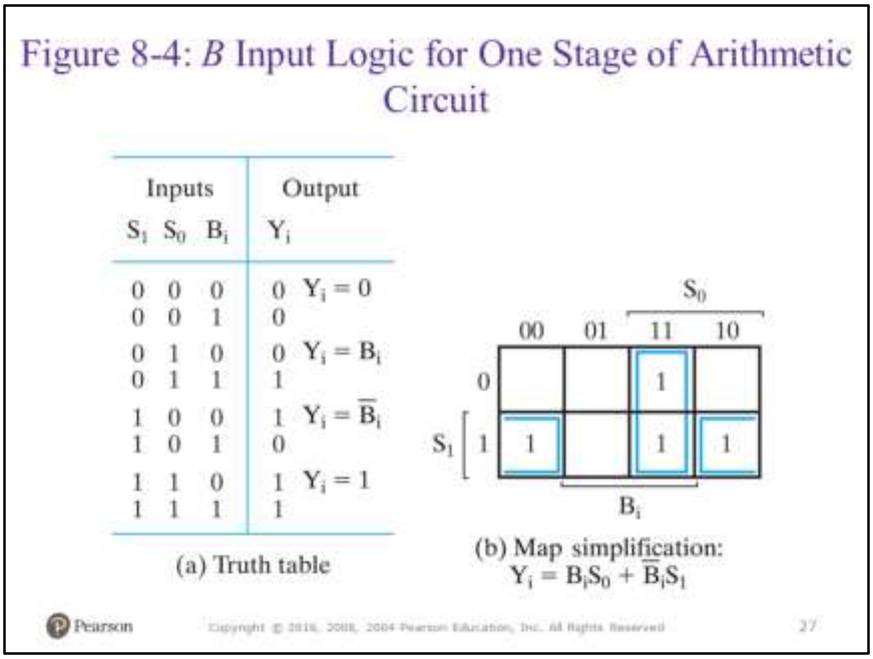


Figure 8-4: *B* Input Logic for One Stage of Arithmetic Circuit

Inputs			Output	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B}_i$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

(a) Truth table

S_1	00	01	11	10
0			1	
1	1		1	1

(b) Map simplification:
$Y_i = B_iS_0 + \overline{B}_iS_1$

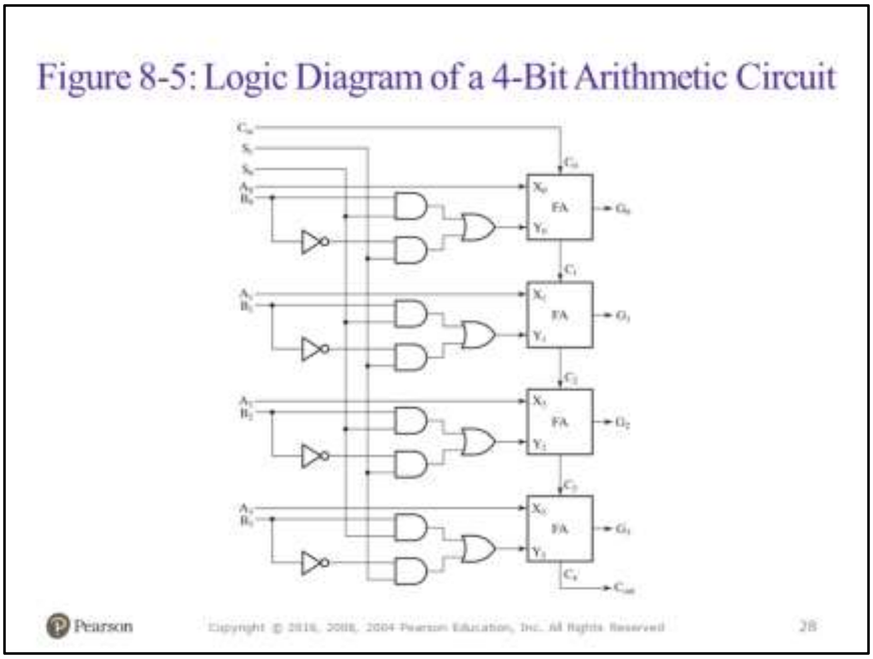


Figure 8-5: Logic Diagram of a 4-Bit Arithmetic Circuit

Pearson

28

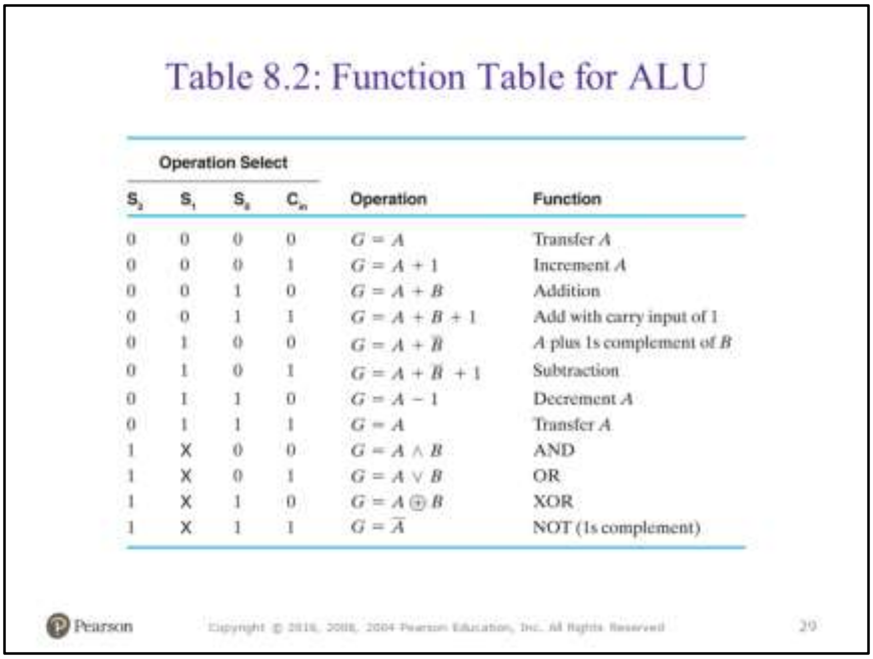


Table 8.2: Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

Pearson

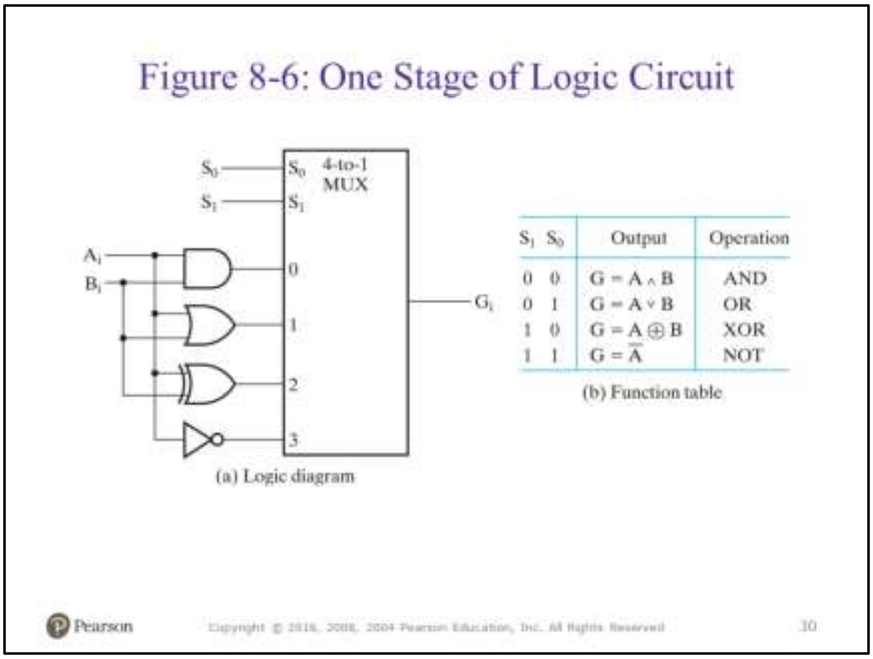


Figure 8-6: One Stage of Logic Circuit

(a) Logic diagram

S_1 S_0	Output	Operation
0 0	$G = A \wedge B$	AND
0 1	$G = A \vee B$	OR
1 0	$G = A \oplus B$	XOR
1 1	$G = \overline{A}$	NOT

(b) Function table

Pearson

30

The S_0S_1 pair in this figure are different from those in the previous slide!

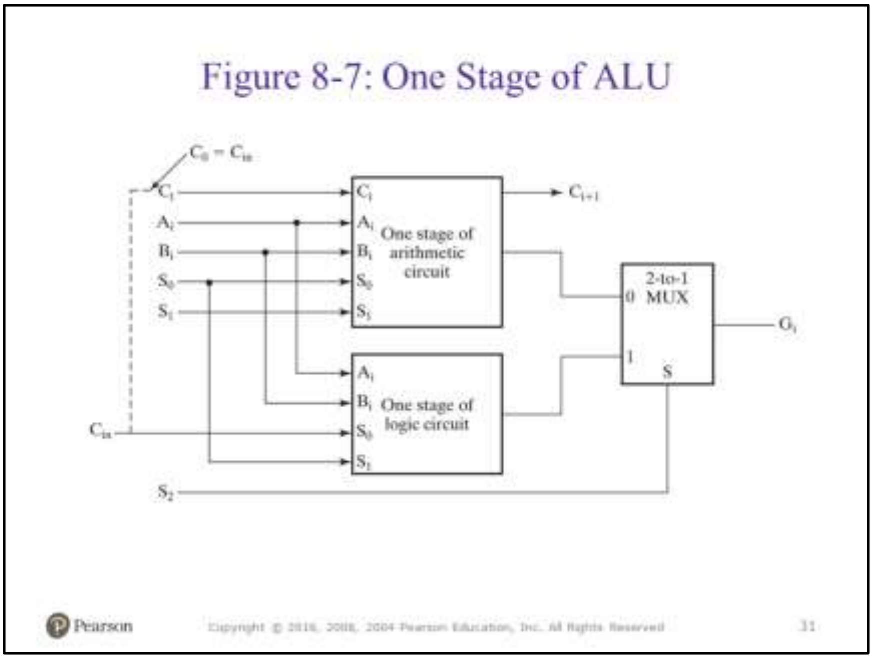


Figure 8-7: One Stage of ALU

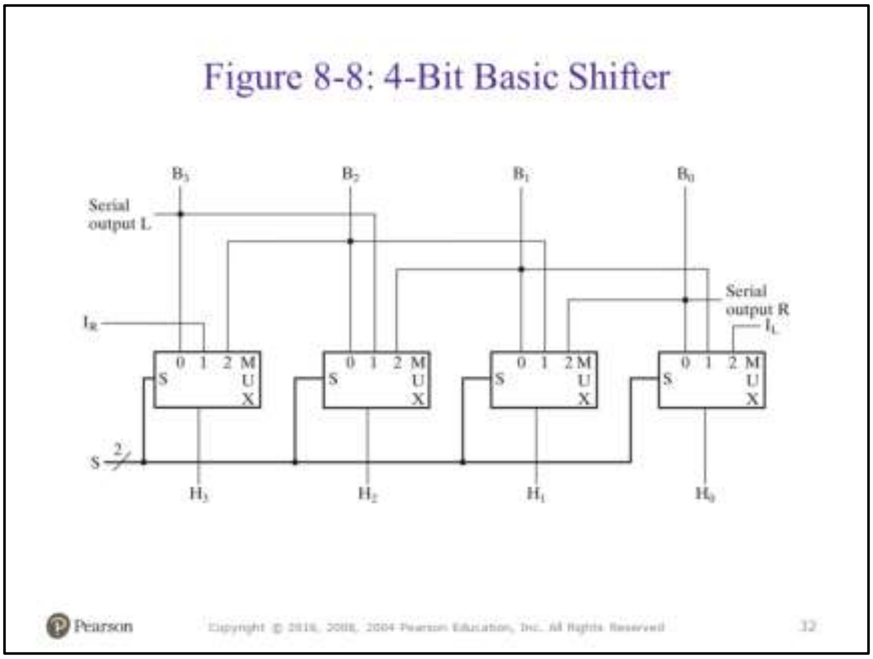


Figure 8-8: 4-Bit Basic Shifter

Pearson

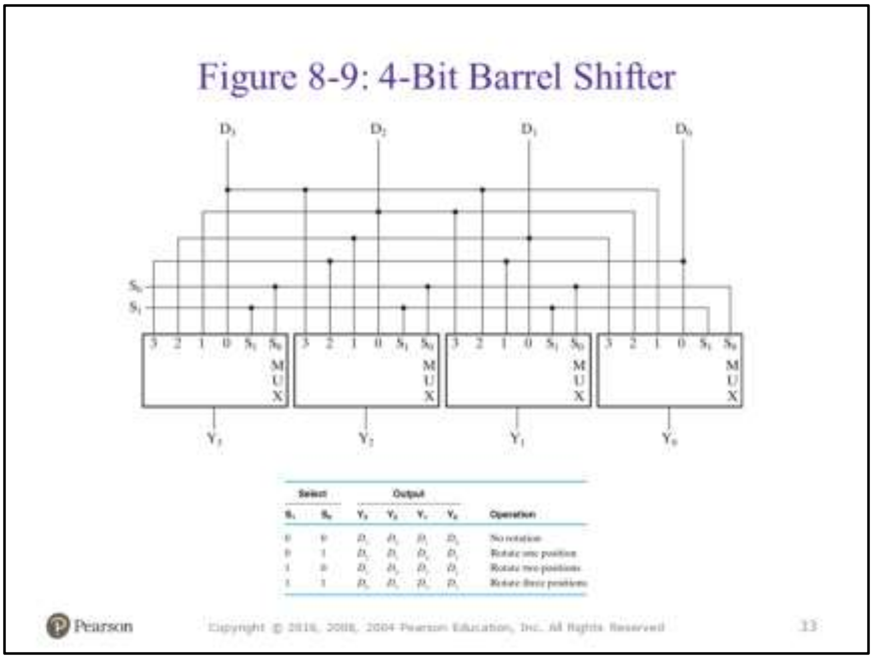


Figure 8-9: 4-Bit Barrel Shifter

Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

Pearson

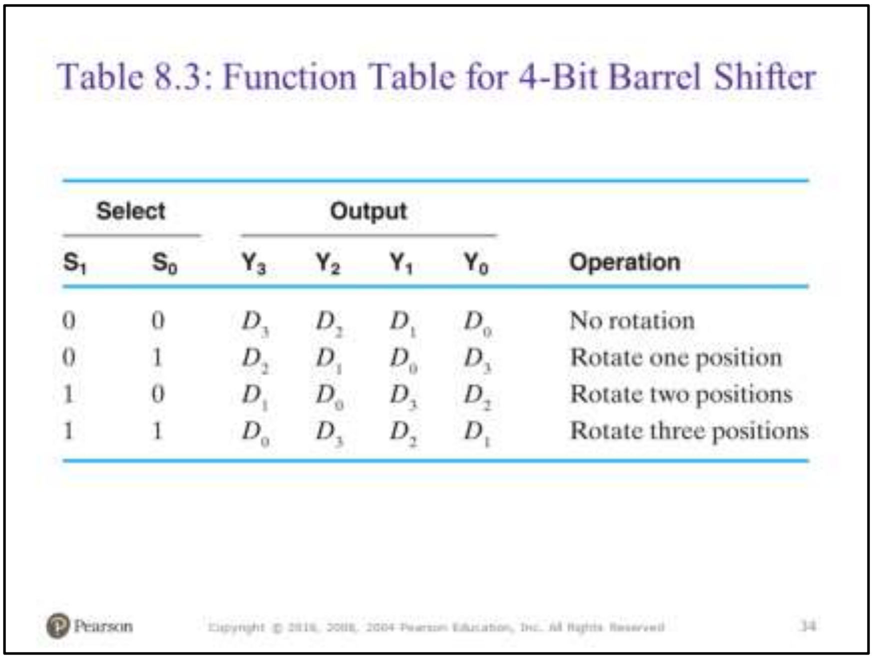
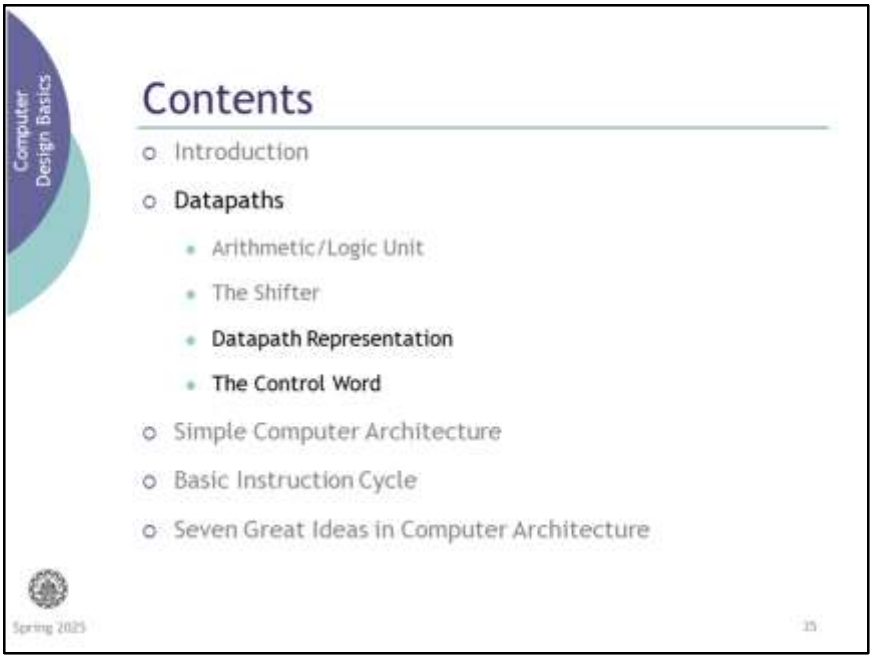


Table 8.3: Function Table for 4-Bit Barrel Shifter

Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

Pearson



Computer Design Basics

Contents

- Introduction
- Datapaths
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- Simple Computer Architecture
- Basic Instruction Cycle
- Seven Great Ideas in Computer Architecture

Spring 2025

35

Selection variables control the addresses for

- the data read from the register file,
- the function performed by the function unit,
- and the data loaded into the register file,
- as well as the selection of external data.

Their combined values specify a *control word.*

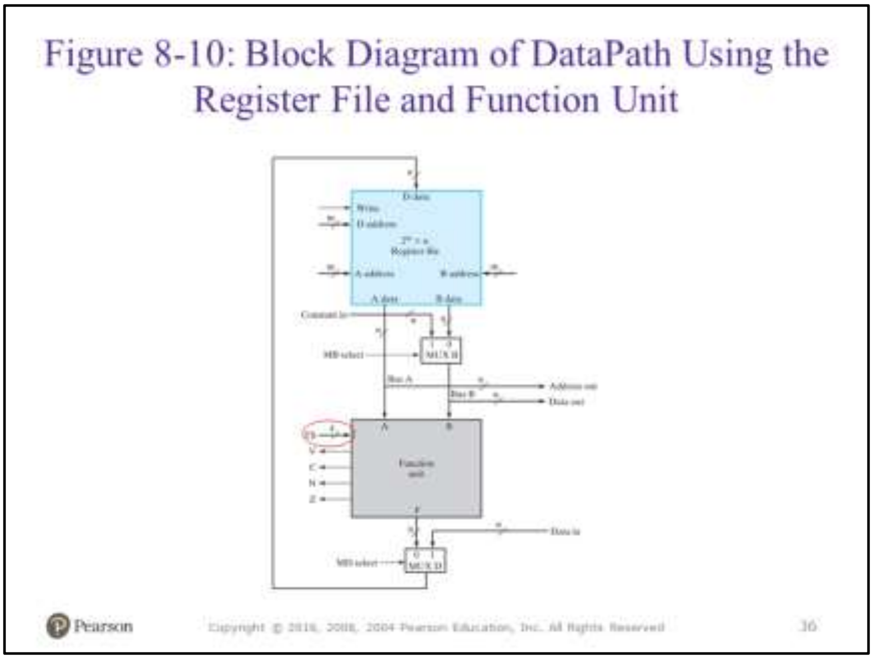


Figure 8-10: Block Diagram of DataPath Using the Register File and Function Unit

36

The typical register file is a special type of fast memory that permits one or more words to be read and one or more words to be written, all simultaneously.

The *A address* accesses a word to be read onto *A data*, the *B address* accesses a second word to be read onto *B data*, and the *D address* accesses a word to be written into from *D data.*

All of these accesses occur in the same clock cycle.

The status bit implementation depends on the specific implementation that has been used for the arithmetic circuit. Alternative implementations may not produce the same results.

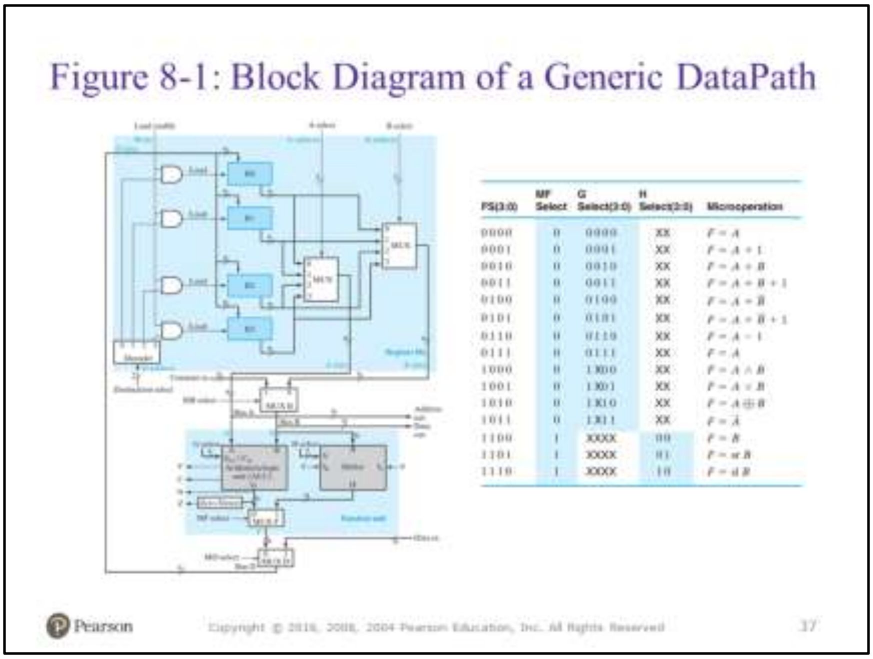


Figure 8-1: Block Diagram of a Generic DataPath

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

Pearson

37

The combination of a set of registers with a shared ALU and interconnecting paths is the datapath for the system.

Control inputs:

1. *A select*, to place the contents of *R*2 onto *A data* and, hence, Bus *A*.

2. *B select*, to place the contents of *R*3 onto the 0 input of MUX B; and *MB select*, to put the 0 input of MUX B onto Bus *B*.

3. *G select*, to provide the arithmetic operation *A* + *B*.

4. *H select*, to provide the shift operation.

5. *MF select*, to place the ALU output on the MUX F output.

6.. *MD select*, to place the MUX F output onto Bus *D*.

7. *Destination select*, to select *R*1 as the destination of the data on Bus *D*.

8. *Load enable*, to enable a register—in this case, *R*1—to be loaded.

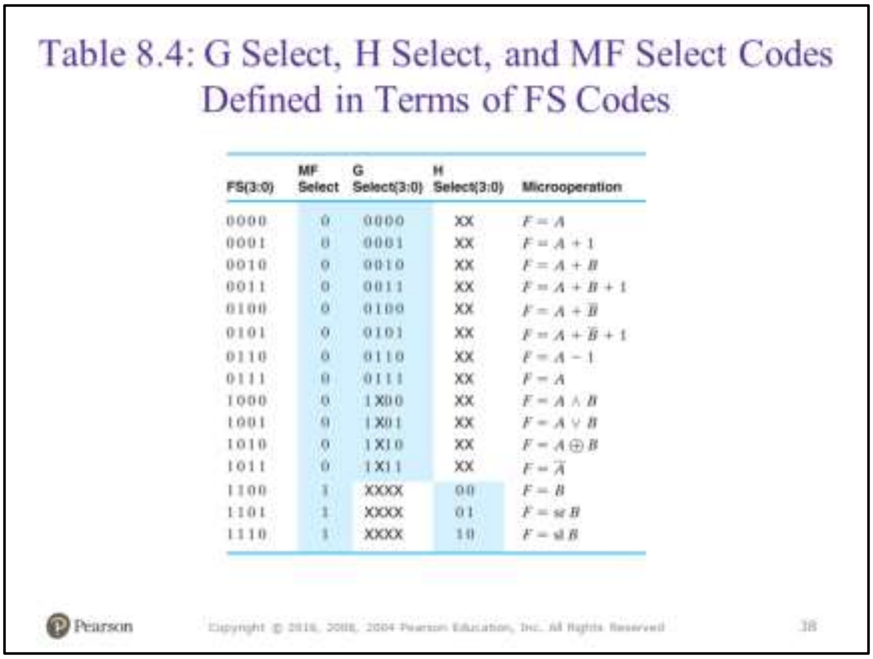


Table 8.4: G Select, H Select, and MF Select Codes Defined in Terms of FS Codes

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

Pearson

In Figure 8-1, there are three sets of select inputs: the *G select*, *H select*, and *MF select*.

In Figure 8-10, there is a single set of select inputs labeled FS, for “function select.”

To fully specify the function unit symbol in the figure, all of the codes for *MF select*, *G select*, and *H select* must be defined in terms of the codes for *FS*

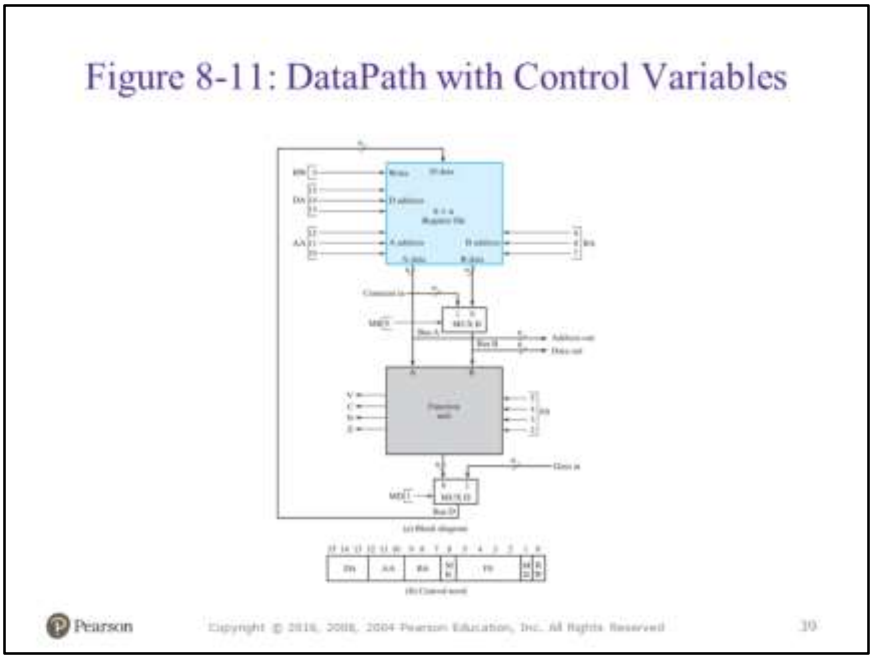


Figure 8-11: DataPath with Control Variables

Pearson

39

The combined values of control inputs specify a *control word.*

It consists of seven parts called *fields.*

The three register fields are three bits each. The remaining fields have one or four bits.

The three bits of DA select one of eight destination registers for the result of the microoperation.

The three bits of AA select one of eight source registers for the Bus *A* input to the ALU.

The three bits of BA select a source register for the 0 input of the MUX B.

The single MB bit determines whether Bus *B* carries the contents of the selected source register or a constant value.

The 4-bit FS field controls the operation of the function unit. (Table 8-4)

The single bit of MD selects the function unit output or the data on *Data in* as the input to Bus *D*.

The final field, RW, determines whether a register is written or not.

When applied to the control inputs, the 16-bit control word specifies a particular microoperation.

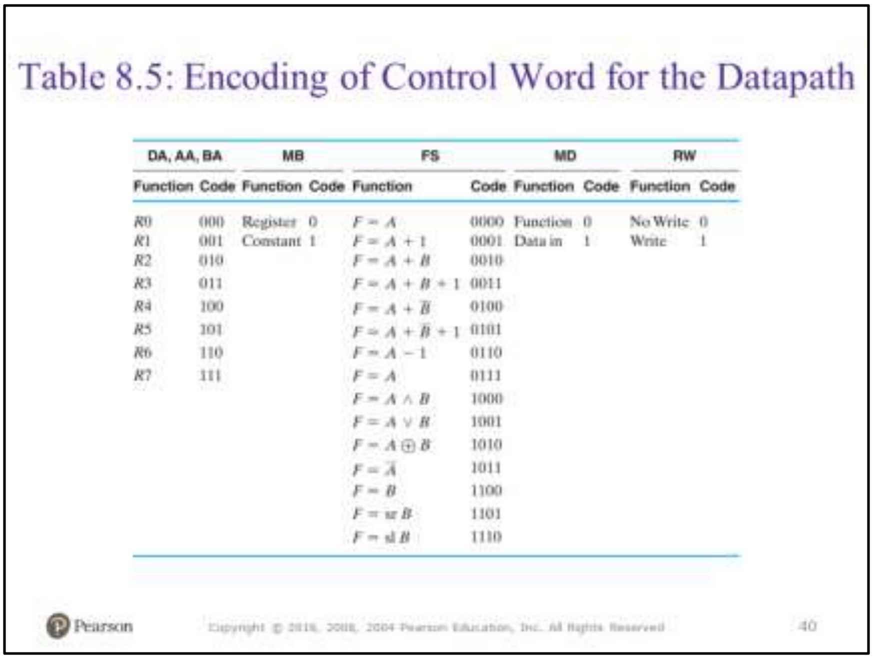


Table 8.5: Encoding of Control Word for the Datapath

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
$R0$	000	Register	0	$F = A$	0000	Function	0	No Write	0
$R1$	001	Constant	1	$F = A + 1$	0001	Data in	1	Write	1
$R2$	010			$F = A + B$	0010				
$R3$	011			$F = A + B + 1$	0011				
$R4$	100			$F = A + \overline{B}$	0100				
$R5$	101			$F = A + \overline{B} + 1$	0101				
$R6$	110			$F = A - 1$	0110				
$R7$	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = \text{sr } B$	1101				
				$F = \text{sl } B$	1110				

Pearson

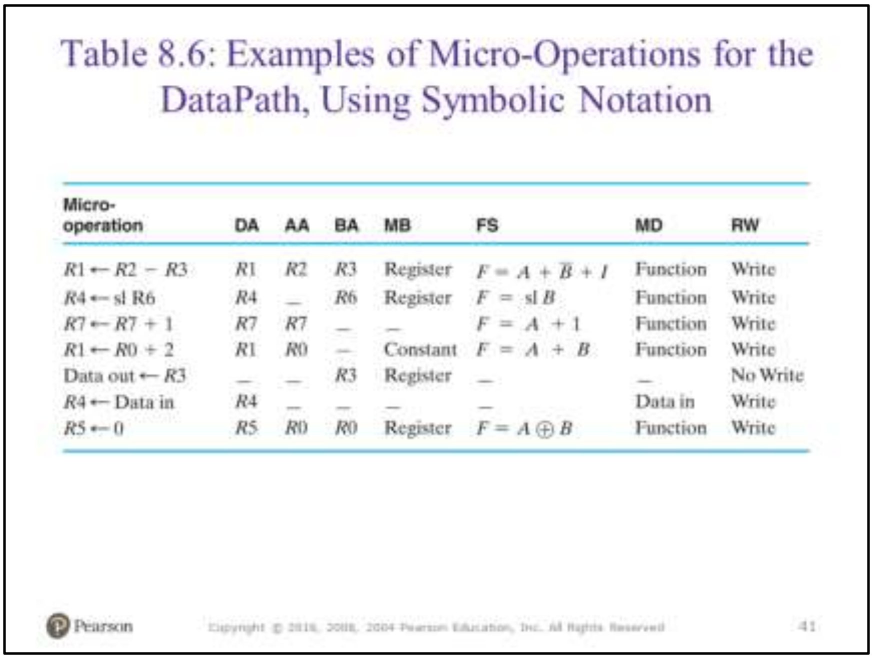


Table 8.6: Examples of Micro-Operations for the DataPath, Using Symbolic Notation

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	—	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	—	—	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	—	—	$R3$	Register	—	—	No Write
$R4 \leftarrow$ Data in	$R4$	—	—	—	—	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Pearson

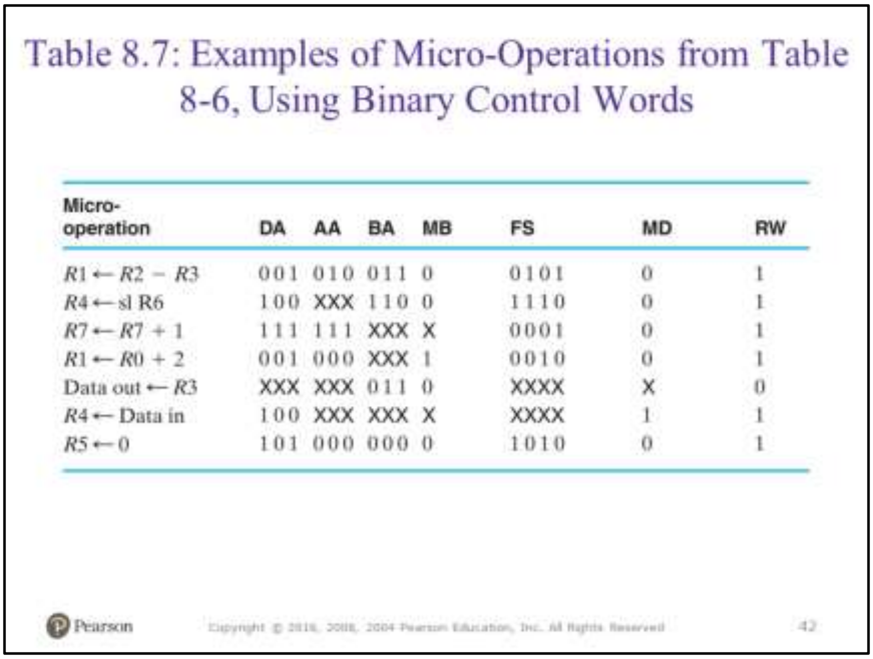
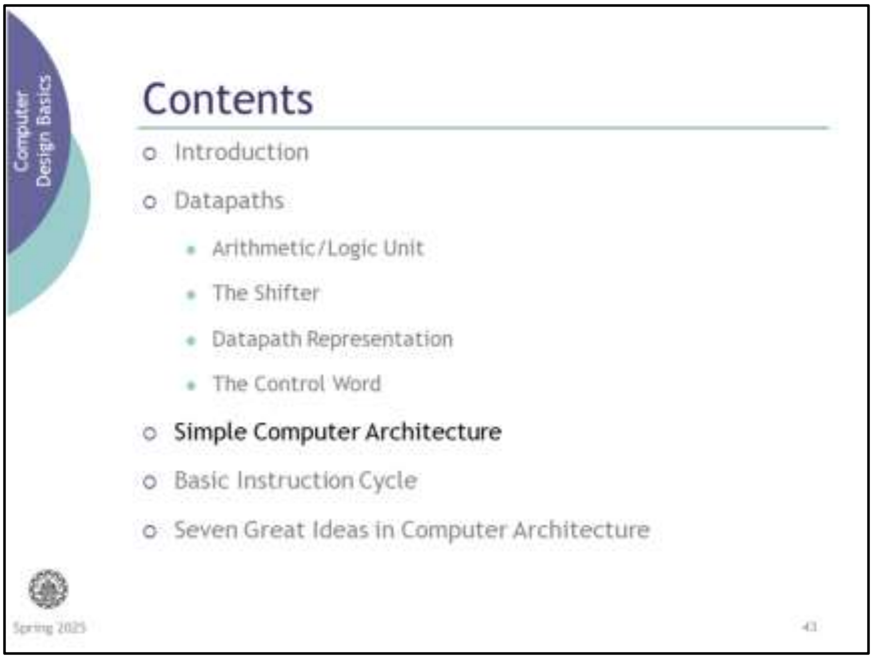


Table 8.7: Examples of Micro-Operations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

Pearson



Computer Design Basics

Contents

- Introduction
- Datapaths
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- **Simple Computer Architecture**
- Basic Instruction Cycle
- Seven Great Ideas in Computer Architecture

Spring 2025

43

Selection variables control

- the addresses for the data read from the register file,
- the function performed by the function unit,
- and the data loaded into the register file,
- as well as the selection of external data.

Their combined values specify a *control word.*

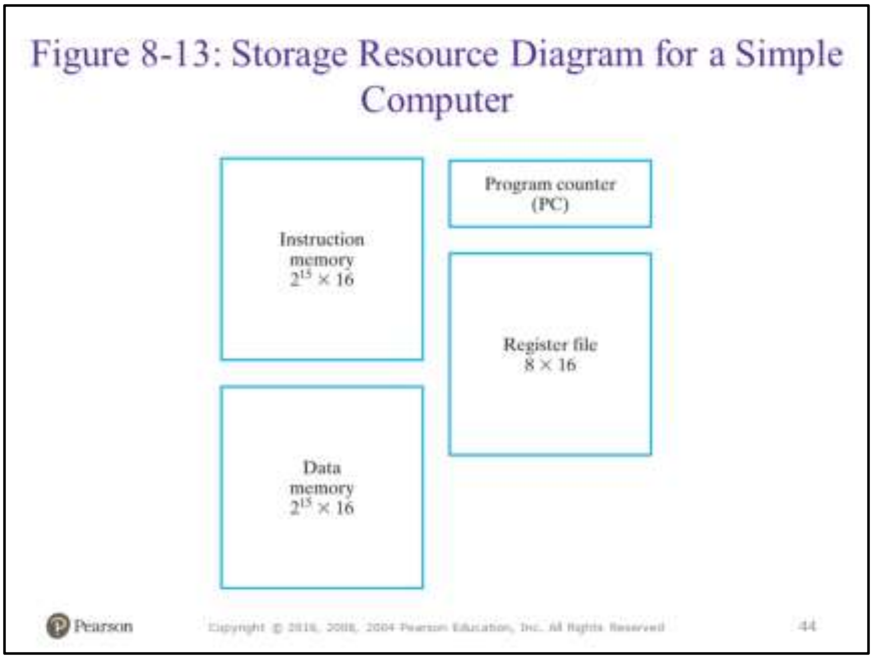


Figure 8-13: Storage Resource Diagram for a Simple Computer

Instruction memory $2^{15} \times 16$

Program counter (PC)

Register file 8×16

Data memory $2^{15} \times 16$

Pearson

Separate Instruction and Data Memory

Memory word size: 16 bits

Memory is both word addressable and word accessible

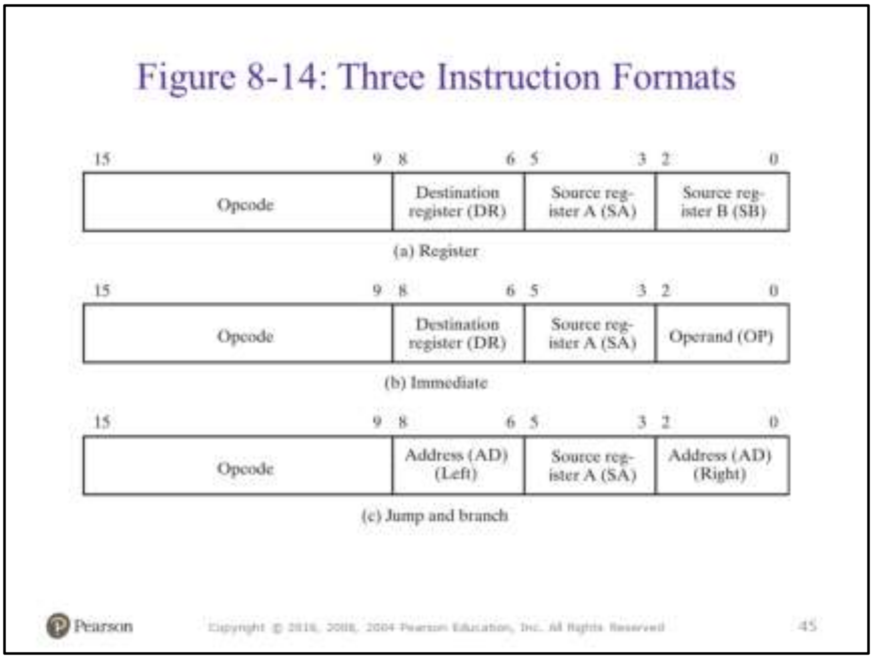


Figure 8-14: Three Instruction Formats

15 9	8 6	5 3	2 0
Opcode	Destination register (DR)	Source register A (SA)	Source register B (SB)

(a) Register

15 9	8 6	5 3	2 0
Opcode	Destination register (DR)	Source register A (SA)	Operand (OP)

(b) Immediate

15 9	8 6	5 3	2 0
Opcode	Address (AD) (Left)	Source register A (SA)	Address (AD) (Right)

(c) Jump and branch

45

The bits are divided into groups called *fields*.

Typical fields found in instruction formats:

1. An *opcode field*, which specifies the operation to be performed.

2. An *address field*, which provides either a memory address or an address that selects a processor register.

3. A *mode field*, which specifies the way the address field is to be interpreted.

Other special fields:

a field that gives the number of positions to shift in a shift-type instruction

an operand field in an immediate operand instruction.

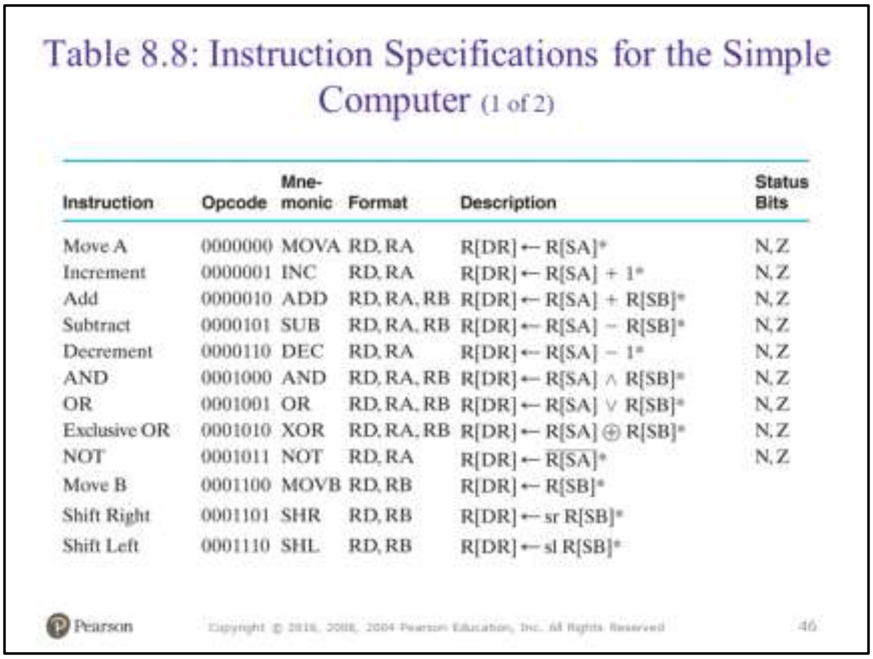


Table 8.8: Instruction Specifications for the Simple Computer (1 of 2)

Instruction	Opcode	Mne- monic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	$R[DR] \leftarrow R[SA]^*$	N, Z
Increment	0000001	INC	RD, RA	$R[DR] \leftarrow R[SA] + 1^*$	N, Z
Add	0000010	ADD	RD, RA, RB	$R[DR] \leftarrow R[SA] + R[SB]^*$	N, Z
Subtract	0000101	SUB	RD, RA, RB	$R[DR] \leftarrow R[SA] - R[SB]^*$	N, Z
Decrement	0000110	DEC	RD, RA	$R[DR] \leftarrow R[SA] - 1^*$	N, Z
AND	0001000	AND	RD, RA, RB	$R[DR] \leftarrow R[SA] \wedge R[SB]^*$	N, Z
OR	0001001	OR	RD, RA, RB	$R[DR] \leftarrow R[SA] \vee R[SB]^*$	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	$R[DR] \leftarrow R[SA] \oplus R[SB]^*$	N, Z
NOT	0001011	NOT	RD, RA	$R[DR] \leftarrow \overline{R[SA]}^*$	N, Z
Move B	0001100	MOVB	RD, RB	$R[DR] \leftarrow R[SB]^*$	
Shift Right	0001101	SHR	RD, RB	$R[DR] \leftarrow \text{sr } R[SB]^*$	
Shift Left	0001110	SHL	RD, RB	$R[DR] \leftarrow \text{sl } R[SB]^*$	

Pearson

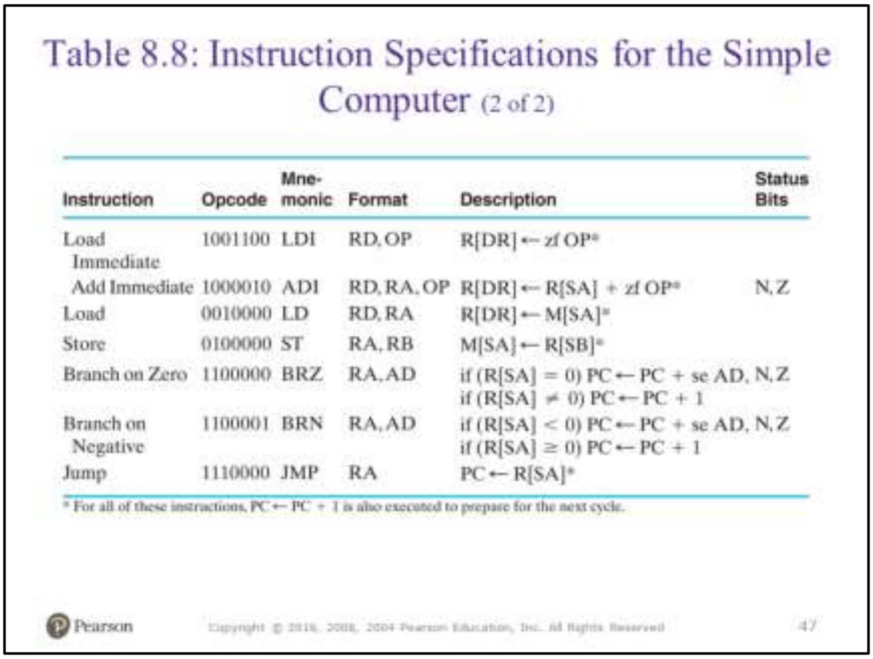


Table 8.8: Instruction Specifications for the Simple Computer (2 of 2)

Instruction	Opcode	Mne- monic	Format	Description	Status Bits
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	

Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD,
				if (R[SA] ≠ 0) PC ← PC + 1
					N, Z

Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD,
				if (R[SA] ≥ 0) PC ← PC + 1
					N, Z

Jump	1110000	JMP	RA	PC ← R[SA]*	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Pearson

zf: zero fill

se: sign extension

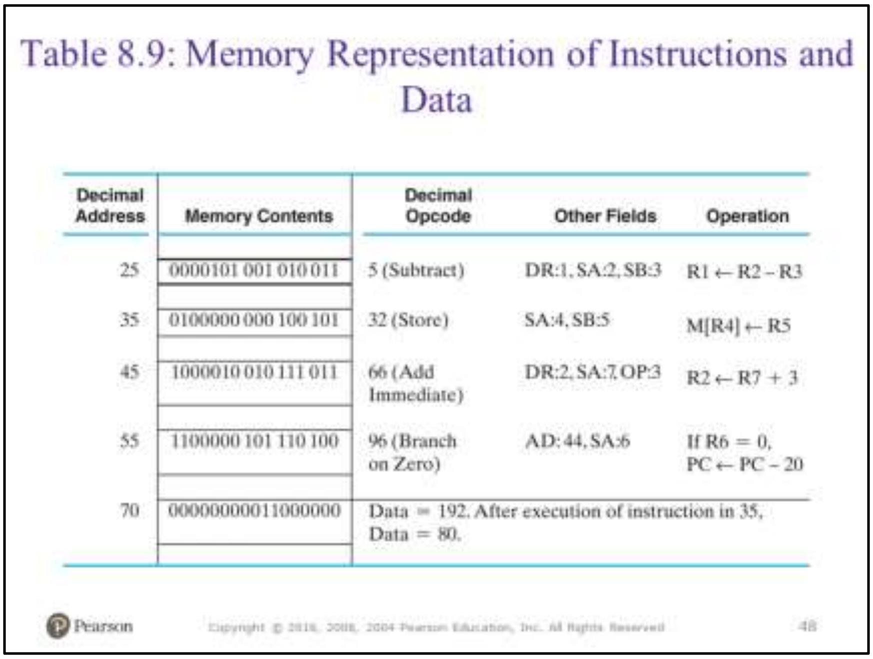
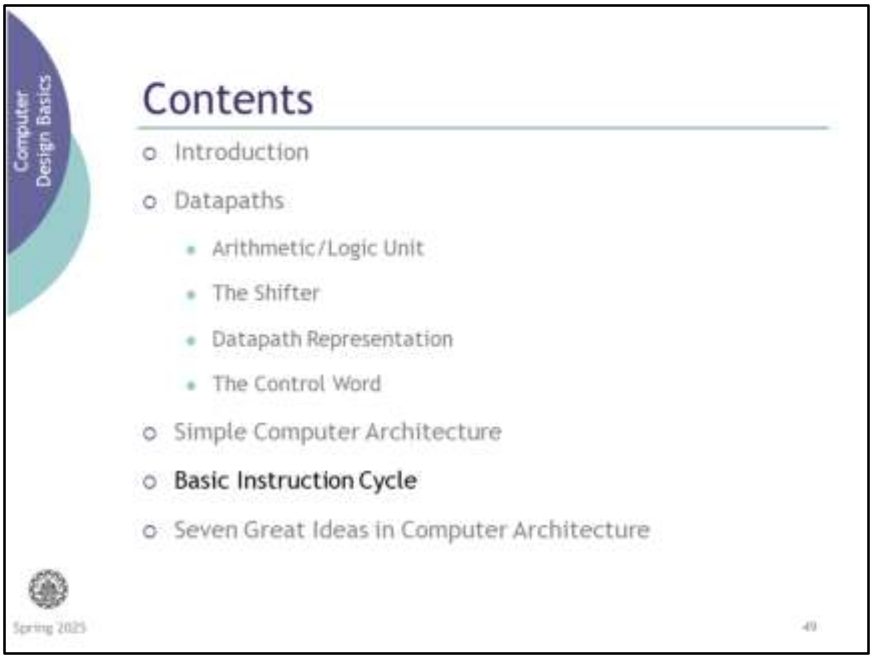


Table 8.9: Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

Pearson



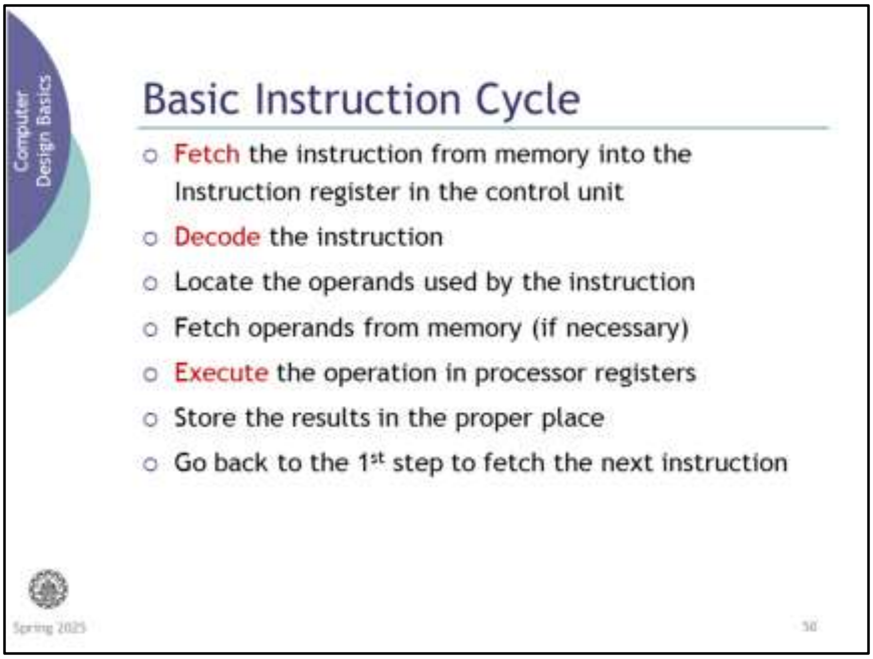
Computer Design Basics

Contents

- Introduction
- Datapaths
  - Arithmetic/Logic Unit
  - The Shifter
  - Datapath Representation
  - The Control Word
- Simple Computer Architecture
- **Basic Instruction Cycle**
- Seven Great Ideas in Computer Architecture

Spring 2025

49



Computer Design Basics

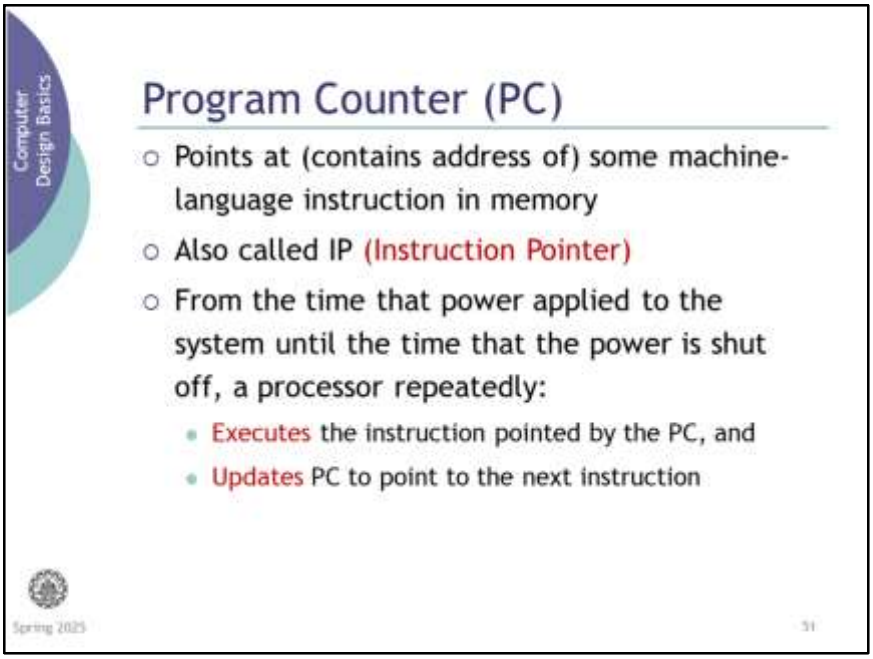
Basic Instruction Cycle

- Fetch the instruction from memory into the Instruction register in the control unit
- Decode the instruction
- Locate the operands used by the instruction
- Fetch operands from memory (if necessary)
- Execute the operation in processor registers
- Store the results in the proper place
- Go back to the 1st step to fetch the next instruction

Spring 2025

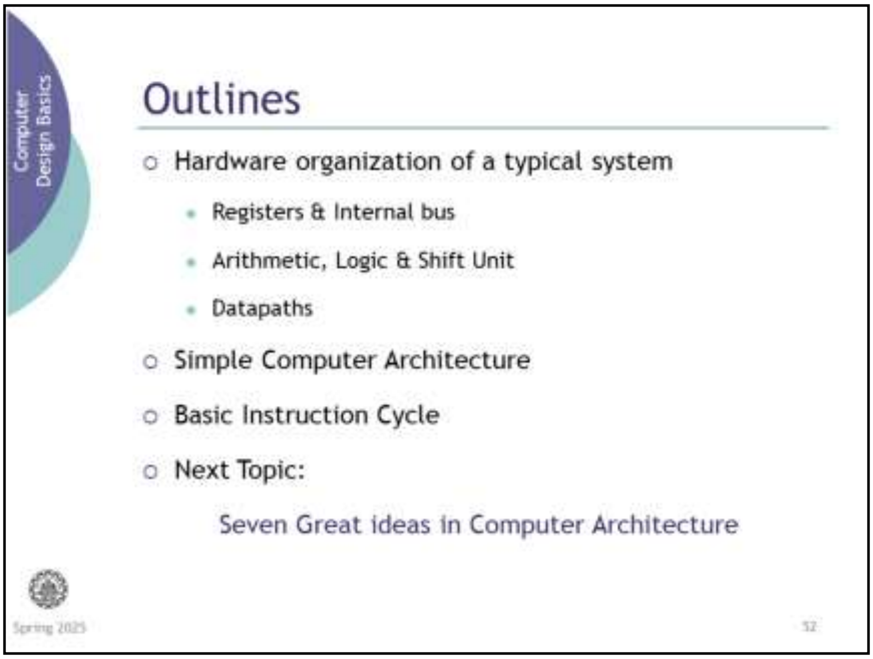
50

A register in the computer called the program counter (PC) keeps track of the instructions in the program stored in memory



Program Counter (PC)

- Points at (contains address of) some machine-language instruction in memory
- Also called IP (Instruction Pointer)
- From the time that power applied to the system until the time that the power is shut off, a processor repeatedly:
  - Executes the instruction pointed by the PC, and
  - Updates PC to point to the next instruction



Computer Design Basics

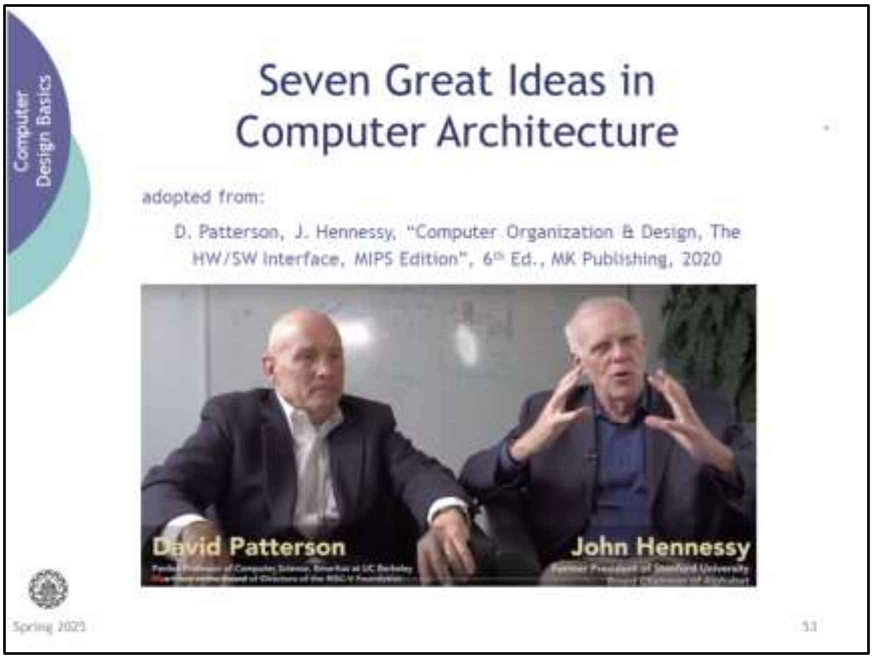
Outlines

- ○ Hardware organization of a typical system
  - Registers & Internal bus
  - Arithmetic, Logic & Shift Unit
  - Datapaths
- ○ Simple Computer Architecture
- ○ Basic Instruction Cycle
- ○ Next Topic:

Seven Great ideas in Computer Architecture

Spring 2025

52



Computer Design Basics

Seven Great Ideas in Computer Architecture

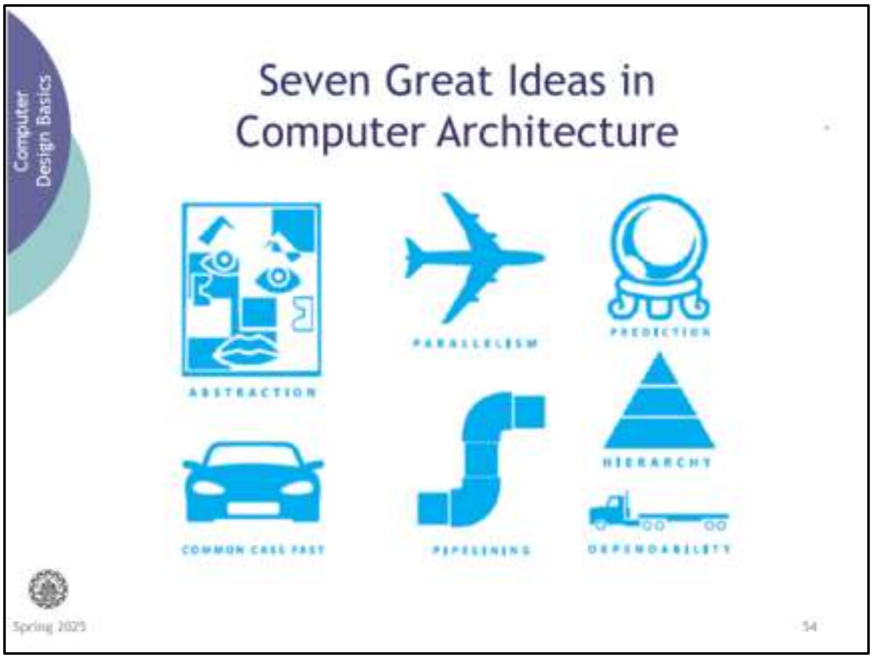
adopted from:

D. Patterson, J. Hennessy, "Computer Organization & Design, The HW/SW Interface, MIPS Edition", 6th Ed., MK Publishing, 2020

David Patterson
Pardee Professor of Computer Science, Emeritus at UC Berkeley
[illegible] of the Board of Directors of the RISC-V Foundation
John Hennessy
Former President of Stanford University
Board Chairman of Alphabet

Spring 2025

53

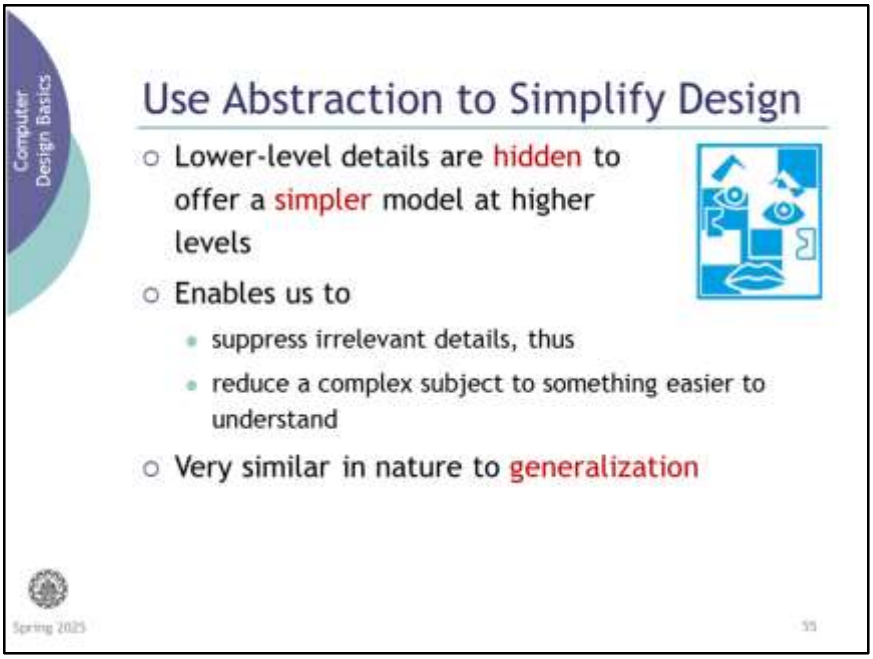


Computer Design Basics

Seven Great Ideas in Computer Architecture

Spring 2025

54



Computer Design Basics

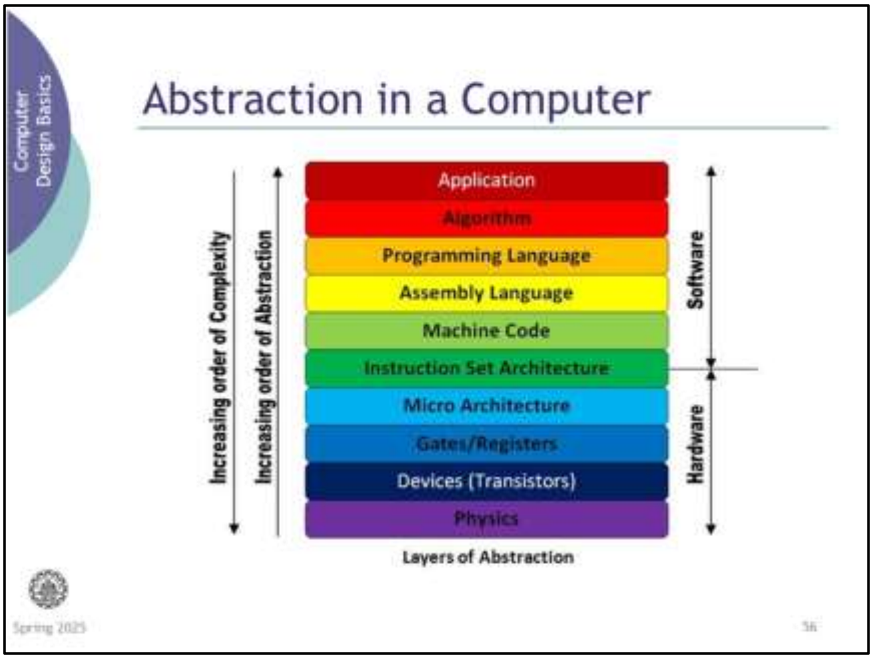
Use Abstraction to Simplify Design

- Lower-level details are hidden to offer a simpler model at higher levels
- Enables us to
  - suppress irrelevant details, thus
  - reduce a complex subject to something easier to understand
- Very similar in nature to generalization

Spring 2025

55

Abstraction: The process of removing physical, spatial, or temporal details or attributes in the study of objects or systems in order to more closely attend to other details of interest



Computer Design Basics

Abstraction in a Computer

Increasing order of Complexity

Increasing order of Abstraction

Application

Algorithm

Programming Language

Assembly Language

Machine Code

Instruction Set Architecture

Micro Architecture

Gates/Registers

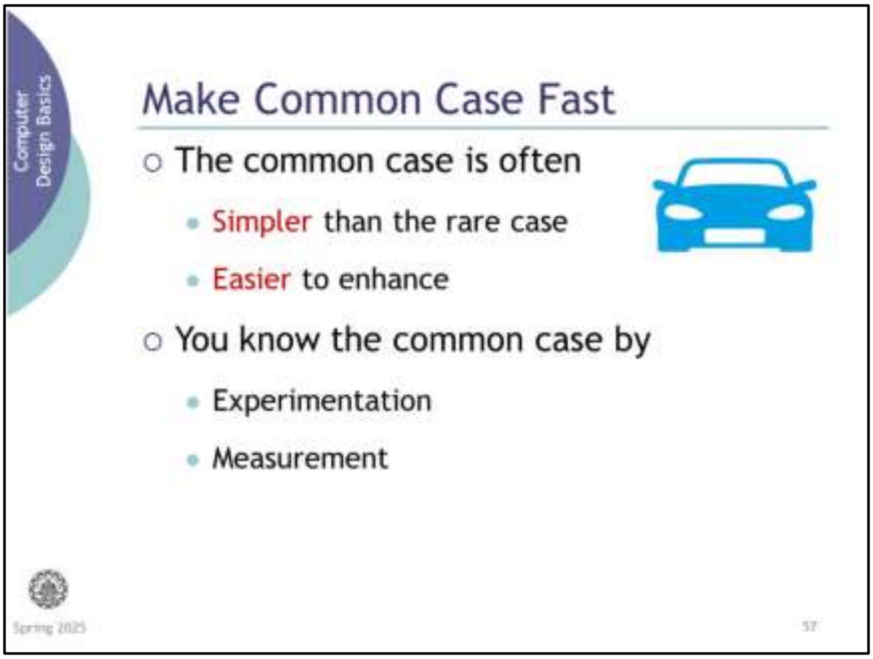
Devices (Transistors)

Physics

Software

Hardware

Layers of Abstraction



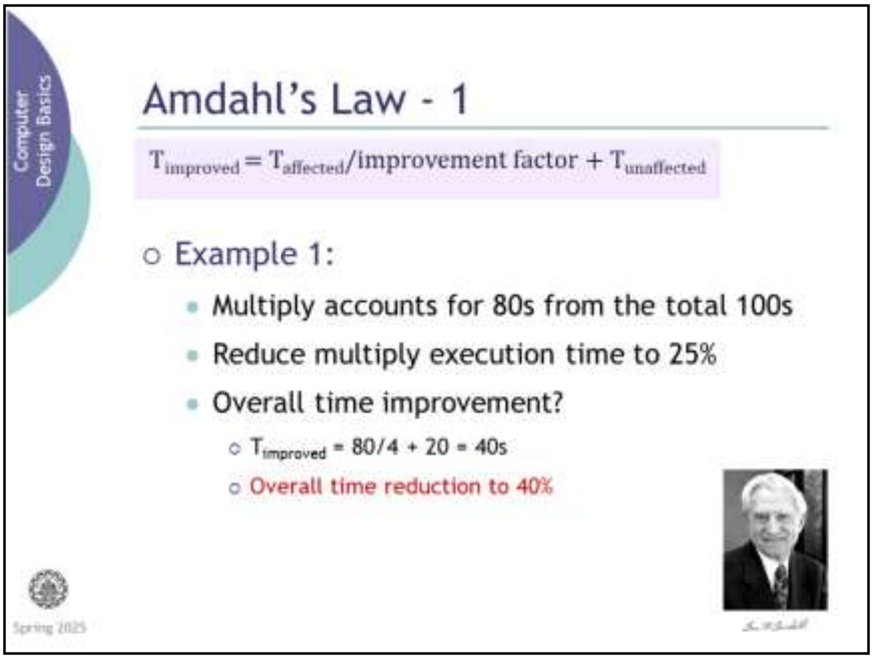
Computer Design Basics

Make Common Case Fast

- The common case is often
  - Simpler than the rare case
  - Easier to enhance
- You know the common case by
  - Experimentation
  - Measurement

Spring 2025

57



Amdahl’s Law - 1

$T_{improved} = T_{affected}/\text{improvement factor} + T_{unaffected}$

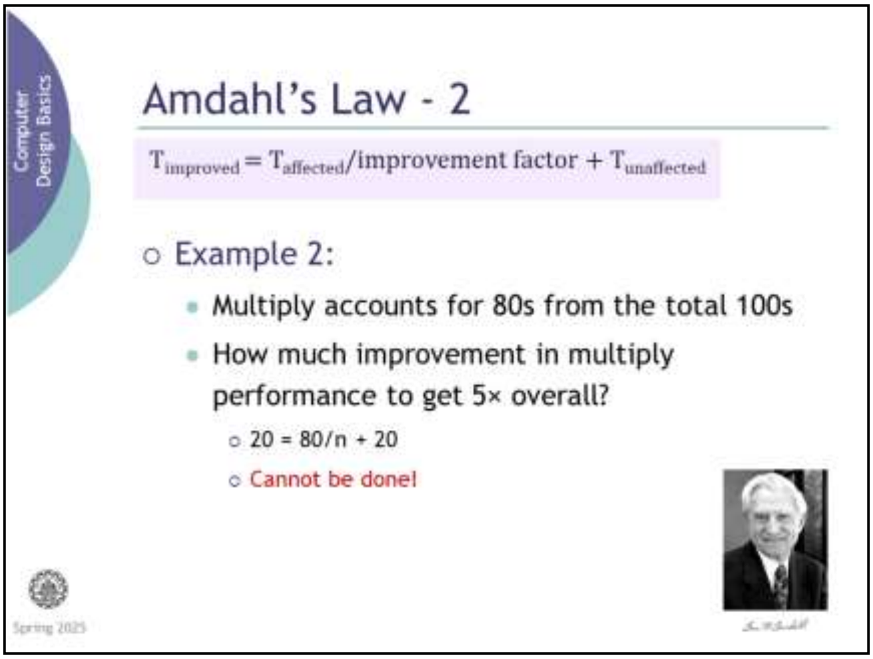
- Example 1:
  - Multiply accounts for 80s from the total 100s
  - Reduce multiply execution time to 25%
  - Overall time improvement?
    - $T_{improved}$ = 80/4 + 20 = 40s
    - Overall time reduction to 40%

Spring 2025

Gene Amdahl, 1922–. Most famous for Amdahl’s Law, an observation he made in 1965.

While in graduate school, he began designing computers in his free time. This side work earned him his Ph.D. in theoretical physics in 1952.

He joined IBM immediately after graduation, and later went on to found three companies, including one called Amdahl Corporation in 1970



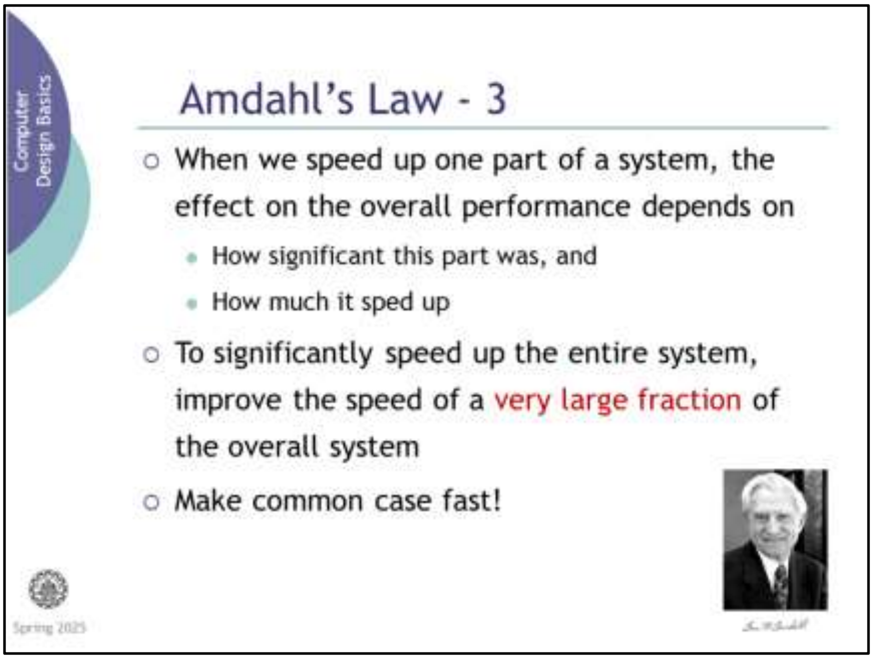
Computer Design Basics

Amdahl's Law - 2

$T_{\text{improved}} = T_{\text{affected}}/\text{improvement factor} + T_{\text{unaffected}}$

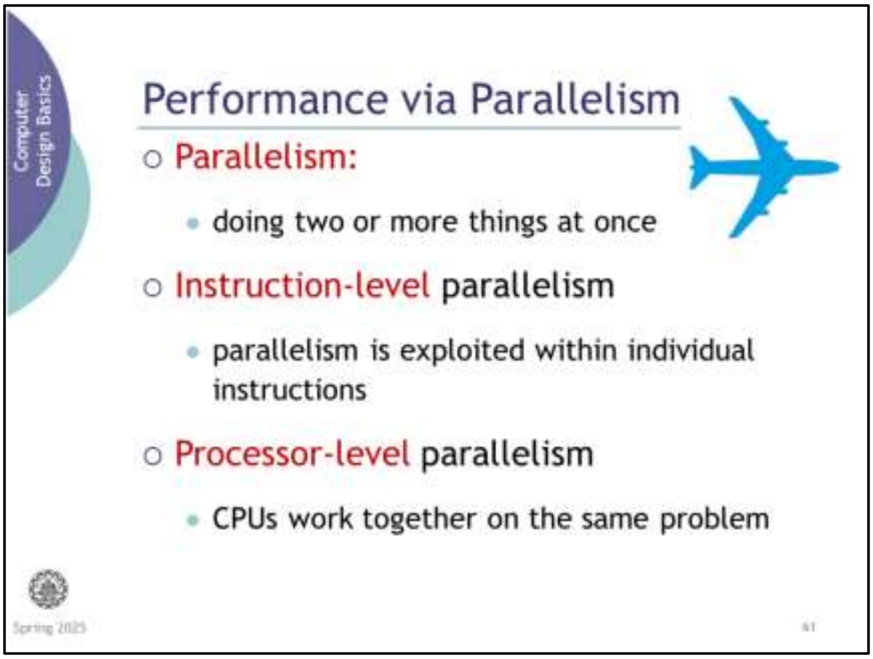
- Example 2:
  - Multiply accounts for 80s from the total 100s
  - How much improvement in multiply performance to get 5× overall?
    - $20 = 80/n + 20$
    - Cannot be done!

Spring 2025



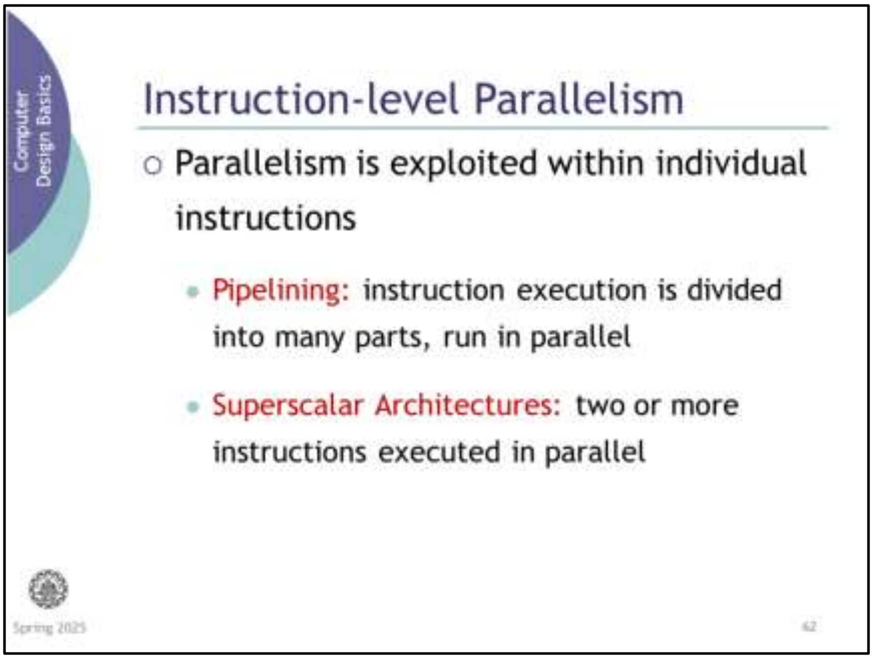
Amdahl's Law - 3

- When we speed up one part of a system, the effect on the overall performance depends on
  - How significant this part was, and
  - How much it sped up
- To significantly speed up the entire system, improve the speed of a very large fraction of the overall system
- Make common case fast!



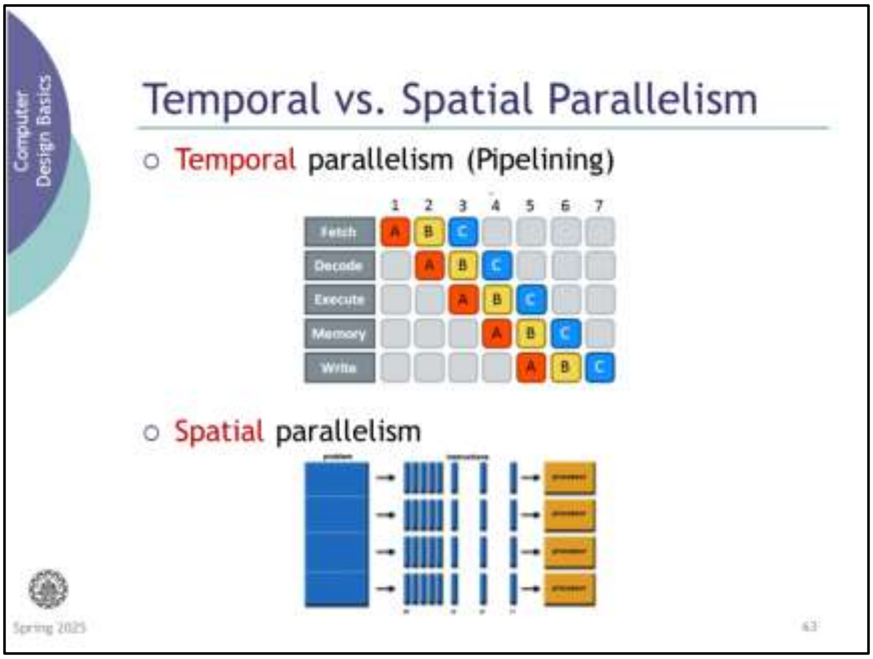
Performance via Parallelism

- ○ Parallelism:
  - doing two or more things at once
- ○ Instruction-level parallelism
  - parallelism is exploited within individual instructions
- ○ Processor-level parallelism
  - CPUs work together on the same problem



Instruction-level Parallelism

- Parallelism is exploited within individual instructions
  - Pipelining: instruction execution is divided into many parts, run in parallel
  - Superscalar Architectures: two or more instructions executed in parallel



Computer Design Basics

Temporal vs. Spatial Parallelism

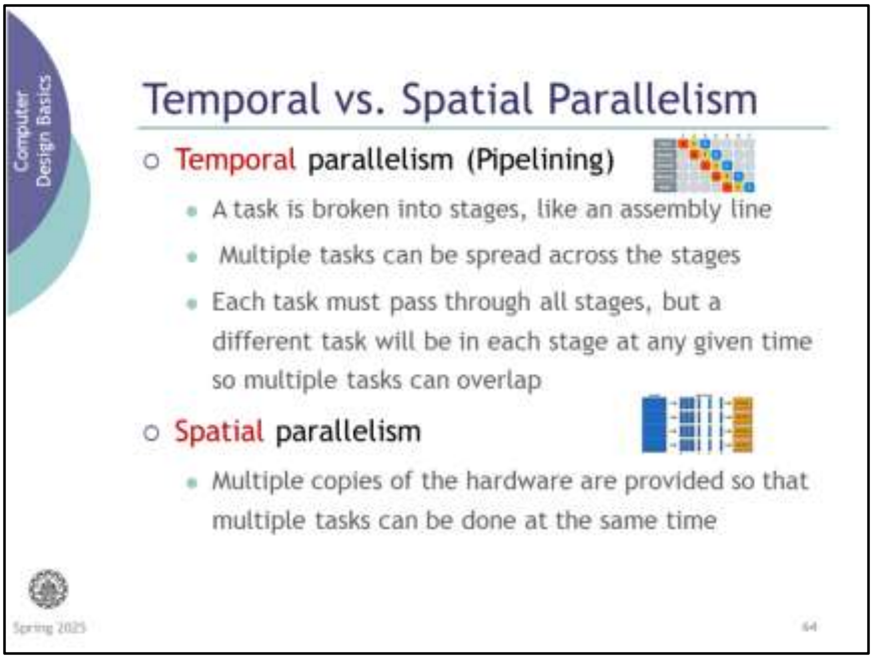
- Temporal parallelism (Pipelining)

	1	2	3	4	5	6	7
Fetch	A	B	C				
Decode		A	B	C			
Execute			A	B	C		
Memory				A	B	C	
Write					A	B	C

- Spatial parallelism

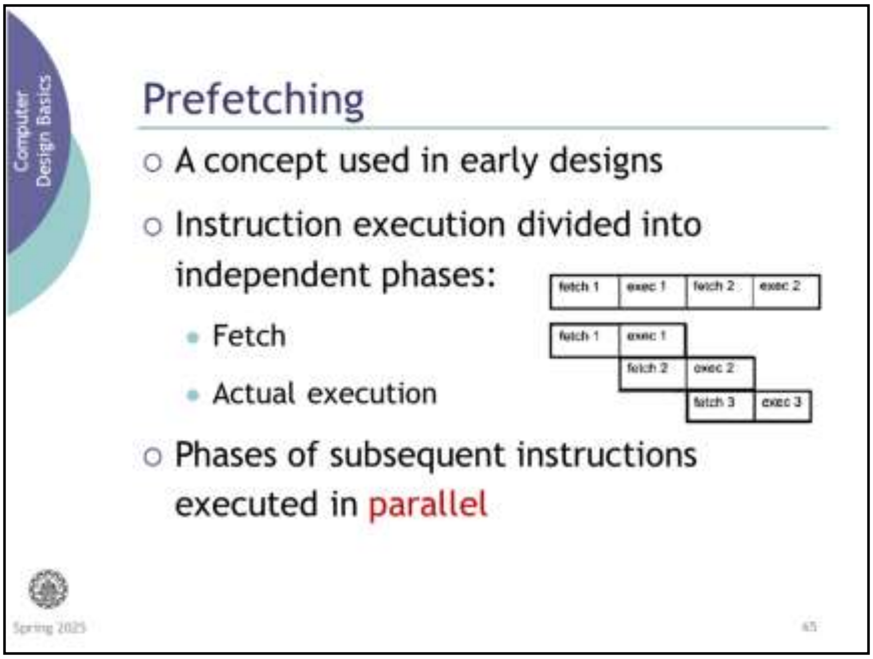
Spring 2025

63



Temporal vs. Spatial Parallelism

- ○ Temporal parallelism (Pipelining)
  - A task is broken into stages, like an assembly line
  - Multiple tasks can be spread across the stages
  - Each task must pass through all stages, but a different task will be in each stage at any given time so multiple tasks can overlap
- ○ Spatial parallelism
  - Multiple copies of the hardware are provided so that multiple tasks can be done at the same time



Computer Design Basics

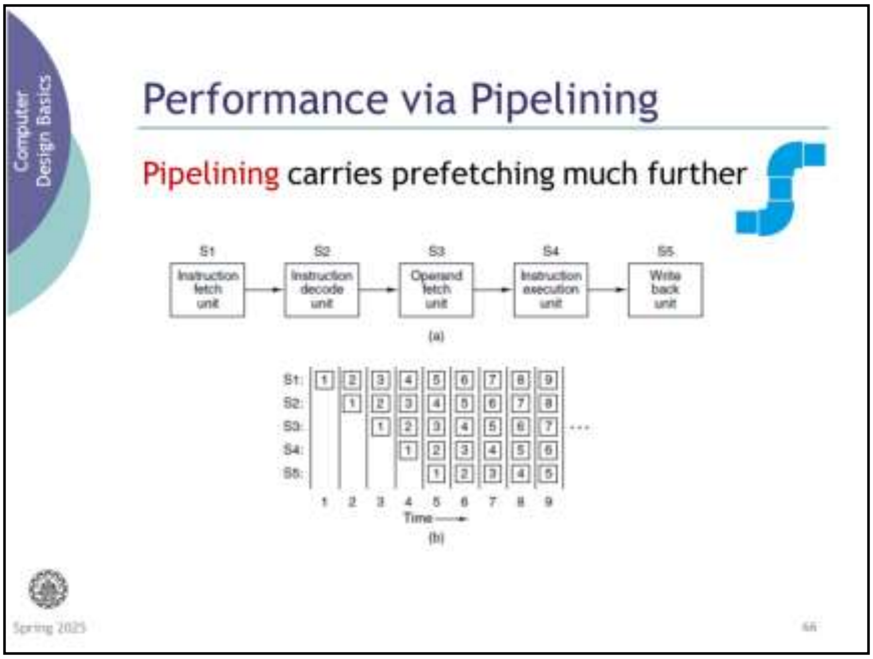
Prefetching

- A concept used in early designs
- Instruction execution divided into independent phases:
  - Fetch
  - Actual execution
- Phases of subsequent instructions executed in parallel

fetch 1	exec 1	fetch 2	exec 2

fetch 1	exec 1		
	fetch 2	exec 2	
		fetch 3	exec 3

Spring 2025



Computer Design Basics

Performance via Pipelining

Pipelining carries prefetching much further

S1 Instruction fetch unit → S2 Instruction decode unit → S3 Operand fetch unit → S4 Instruction execution unit → S5 Write back unit

(a)

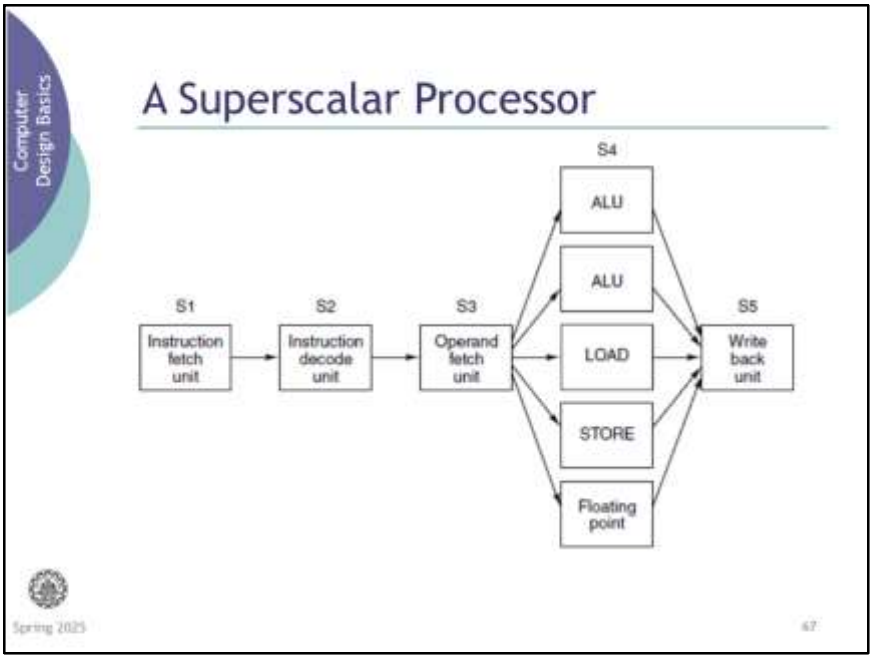
	1	2	3	4	5	6	7	8	9	
S1:	1	2	3	4	5	6	7	8	9	
S2:		1	2	3	4	5	6	7	8	
S3:			1	2	3	4	5	6	7	...
S4:				1	2	3	4	5	6	
S5:					1	2	3	4	5	

Time ⟶

(b)

Spring 2025

66



Computer Design Basics

A Superscalar Processor

S1

Instruction fetch unit

S2

Instruction decode unit

S3

Operand fetch unit

S4

ALU

ALU

LOAD

STORE

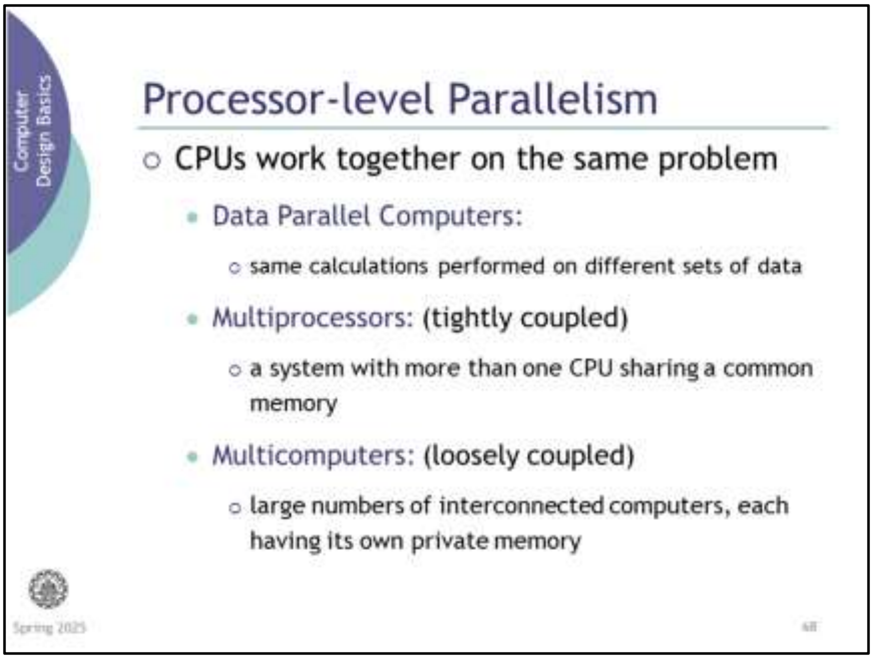
Floating point

S5

Write back unit

Spring 2025

67



Computer Design Basics

Processor-level Parallelism

- CPUs work together on the same problem
  - Data Parallel Computers:
    - same calculations performed on different sets of data
  - Multiprocessors: (tightly coupled)
    - a system with more than one CPU sharing a common memory
  - Multicomputers: (loosely coupled)
    - large numbers of interconnected computers, each having its own private memory

Spring 2025

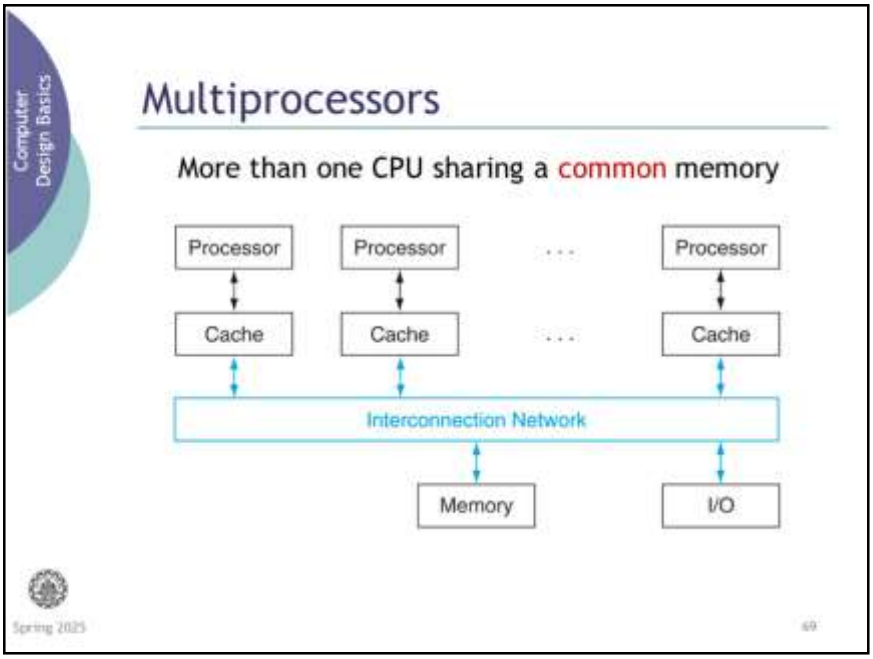
68

Data Parallel Computers:

SIMD processors: consist of a large number of identical processors that perform the same sequence of instructions on different sets of data. (e.g. GPUs)

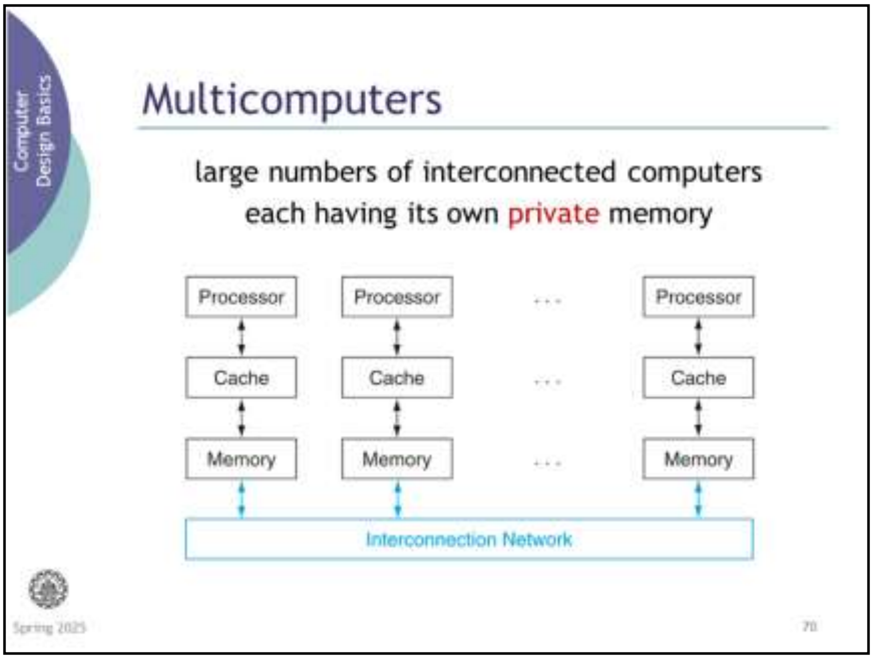
Vector processors: unlike a SIMD processor, all of the operations are performed in a single, heavily pipelined functional unit

Multiprocessors are **MIMD** (Multiple Instruction Multiple Data) computers



Multiprocessors

More than one CPU sharing a common memory



Multicomputers

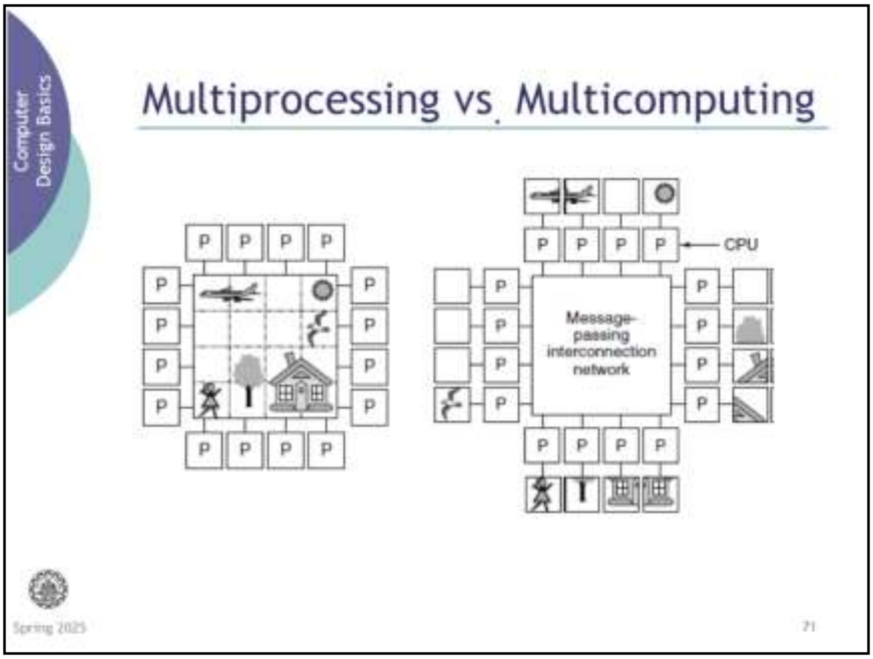
large numbers of interconnected computers each having its own private memory

Processor Processor . . . Processor

Cache Cache . . . Cache

Memory Memory . . . Memory

Interconnection Network

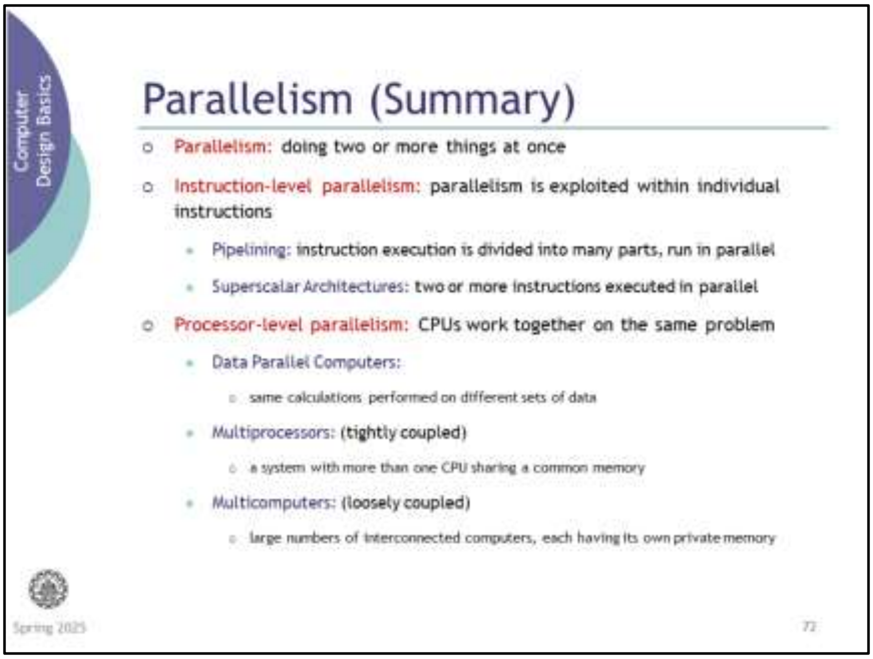


Computer Design Basics

Multiprocessing vs. Multicomputing

Spring 2025

71



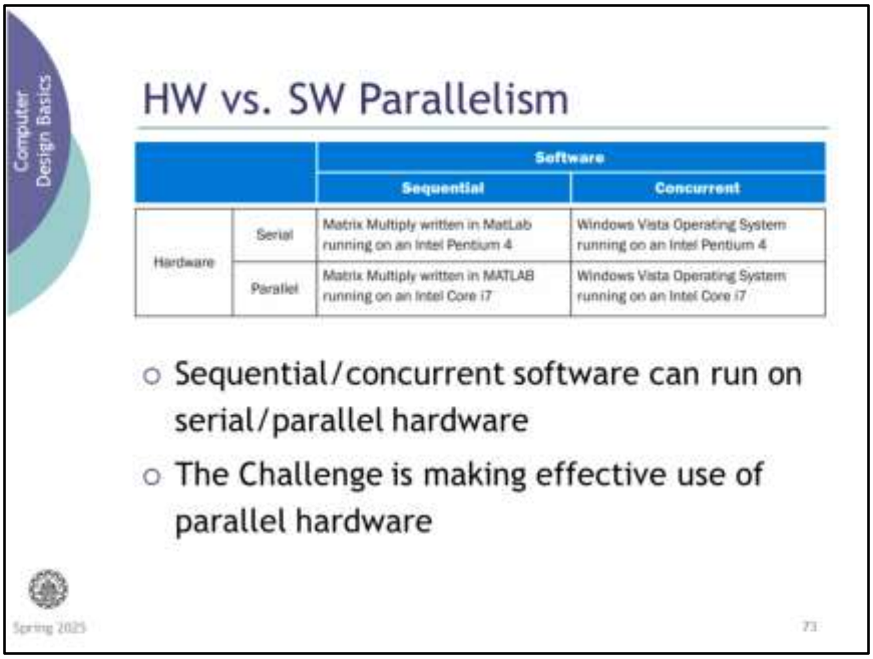
Computer Design Basics

Parallelism (Summary)

- Parallelism: doing two or more things at once
- Instruction-level parallelism: parallelism is exploited within individual instructions
  - Pipelining: instruction execution is divided into many parts, run in parallel
  - Superscalar Architectures: two or more instructions executed in parallel
- Processor-level parallelism: CPUs work together on the same problem
  - Data Parallel Computers:
    - same calculations performed on different sets of data
  - Multiprocessors: (tightly coupled)
    - a system with more than one CPU sharing a common memory
  - Multicomputers: (loosely coupled)
    - large numbers of interconnected computers, each having its own private memory

Spring 2025

72



Computer Design Basics

HW vs. SW Parallelism

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

- Sequential/concurrent software can run on serial/parallel hardware
- The Challenge is making effective use of parallel hardware

Spring 2025

73

This figure tries to clarify the terms serial, parallel, sequential, and concurrent.

The columns of this figure represent the software, which is either inherently sequential or concurrent.

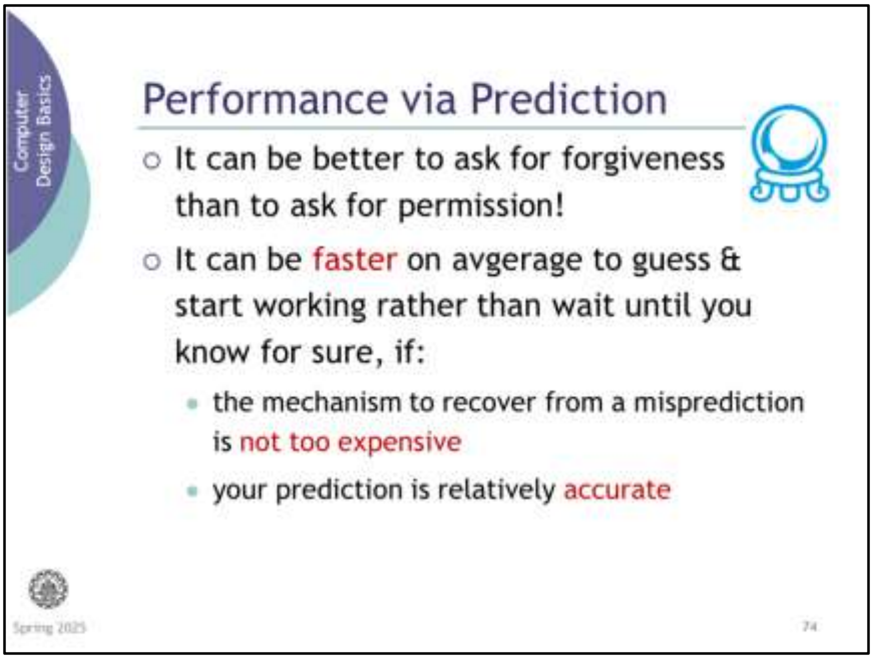
The rows of the figure represent the hardware, which is either serial or parallel.

For example, the programmers of compilers think of them as sequential programs: the steps include parsing, code generation, optimization, and so on.

In contrast, the programmers of operating systems normally think of them as concurrent programs: cooperating processes handling I/O events due to independent jobs running on a computer.

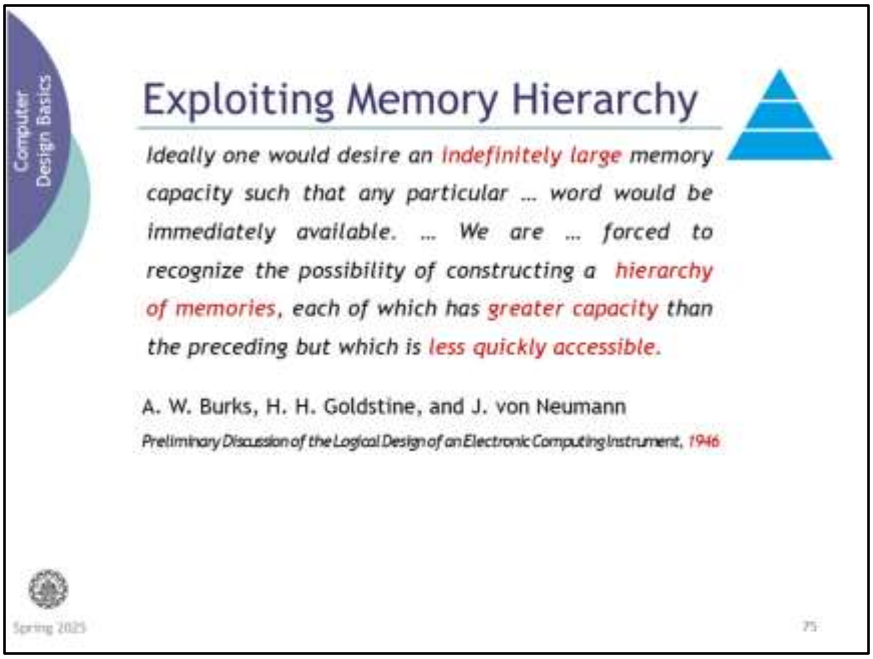
The point of these two axes of the figure is that concurrent software can run on serial hardware, such as operating systems for the Intel Pentium 4 uniprocessor, or on parallel hardware, such as an OS on the more recent Intel Core i7.

The same is true for sequential software. For example, the MATLAB programmer writes a matrix multiply thinking about it sequentially, but it could run serially on the entium4 or in parallel on the Intel Core i7.



Performance via Prediction

- It can be better to ask for forgiveness than to ask for permission!
- It can be faster on avgerage to guess & start working rather than wait until you know for sure, if:
  - the mechanism to recover from a misprediction is not too expensive
  - your prediction is relatively accurate

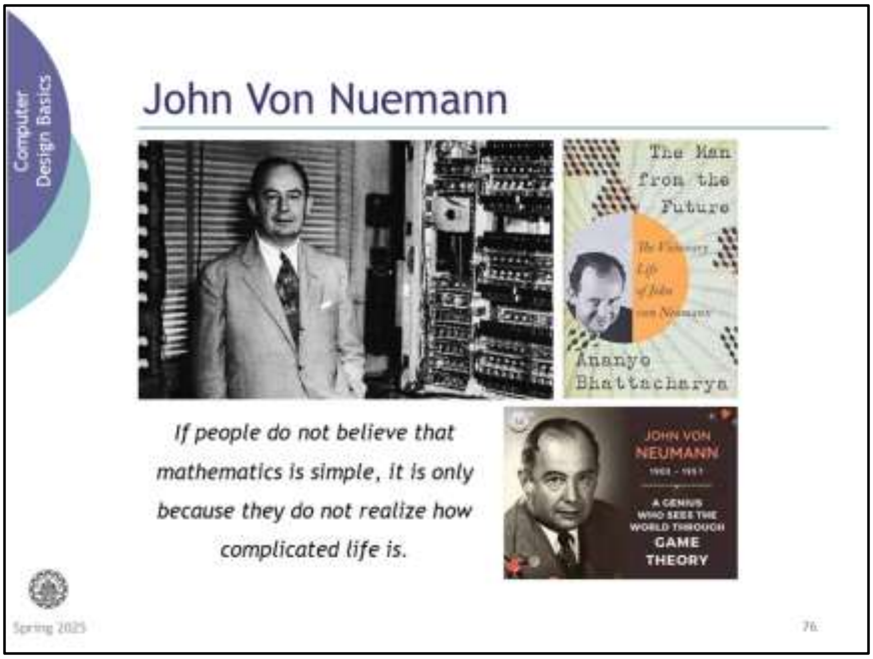


Exploiting Memory Hierarchy

Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available. … We are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

A. W. Burks, H. H. Goldstine, and J. von Neumann

Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946

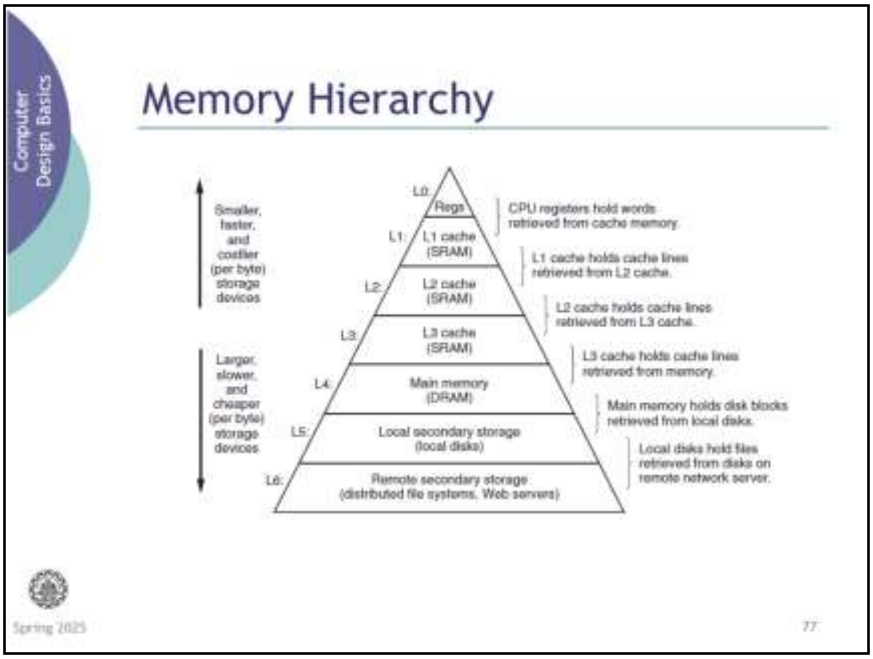


Computer Design Basics

John Von Nuemann

The Man from the Future
The Visionary Life of John von Neumann
Ananyo Bhattacharya
JOHN VON NEUMANN
1903 - 1957
A GENIUS WHO SEES THE WORLD THROUGH GAME THEORY

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.



Memory Hierarchy

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

L6: Remote secondary storage (distributed file systems, Web servers)

CPU registers hold words retrieved from cache memory.

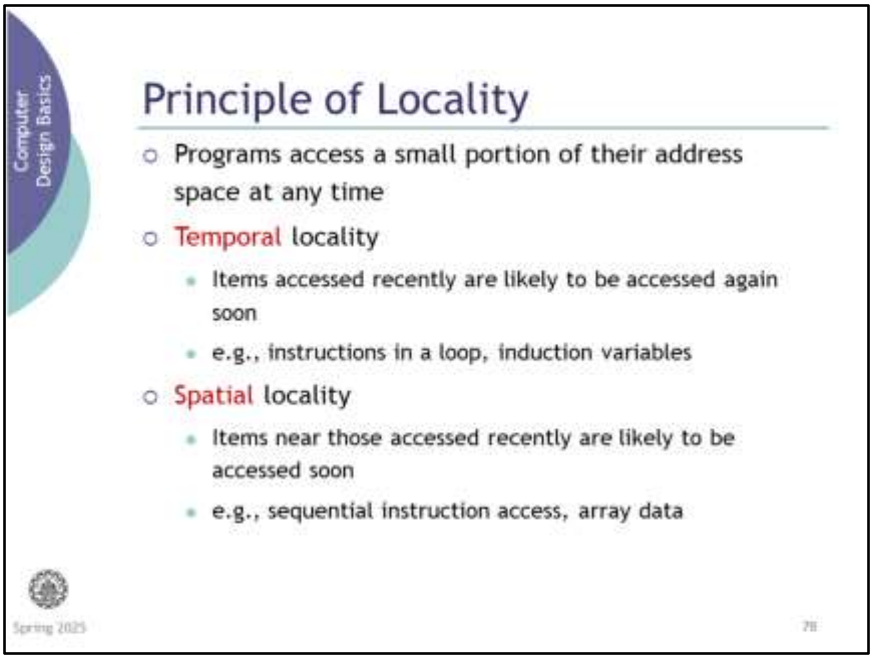
L1 cache holds cache lines retrieved from L2 cache.

L2 cache holds cache lines retrieved from L3 cache.

L3 cache holds cache lines retrieved from memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network server.

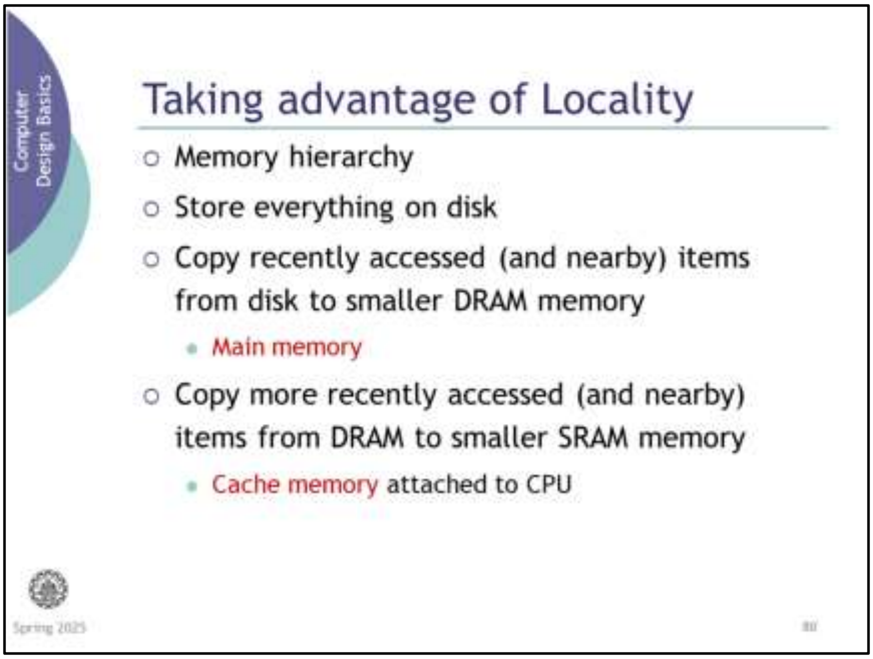


Computer Design Basics

Principle of Locality

- Programs access a small portion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - e.g., sequential instruction access, array data

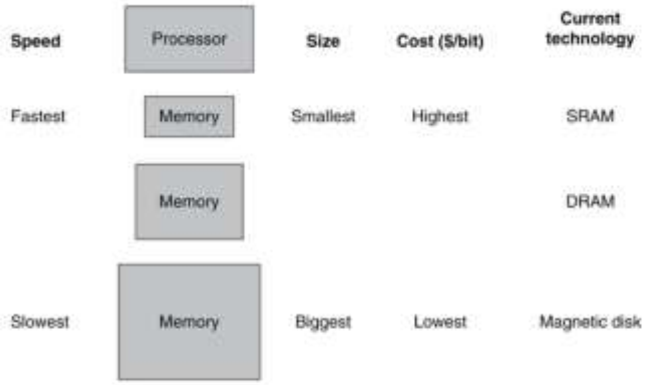
Suppose you were a student writing a term paper on important historical developments in computer hardware. You are sitting at a desk in a library with a collection of books that you have pulled from the shelves and are examining … check the whole analogy in the book.



Computer Design Basics

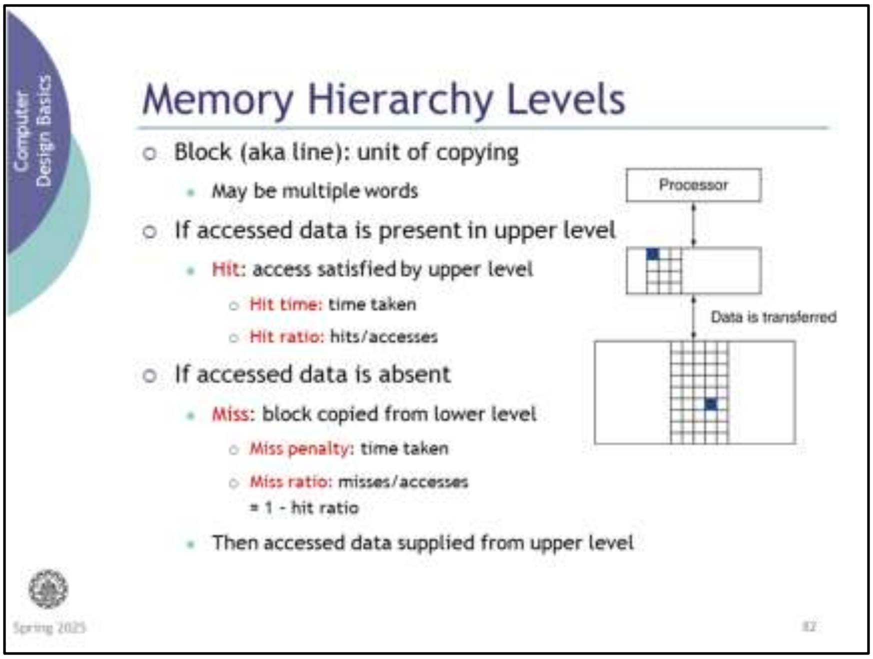
Taking advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU



Speed	Processor	Size	Cost ($/bit)	Current technology
Fastest	Memory	Smallest	Highest	SRAM
	Memory			DRAM
Slowest	Memory	Biggest	Lowest	Magnetic disk

The basic structure of a memory hierarchy. By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory. Flash memory has replaced disks in many personal mobile devices, and may lead to a new level in the storage hierarchy for desktop and server computers.



Computer Design Basics

Memory Hierarchy Levels

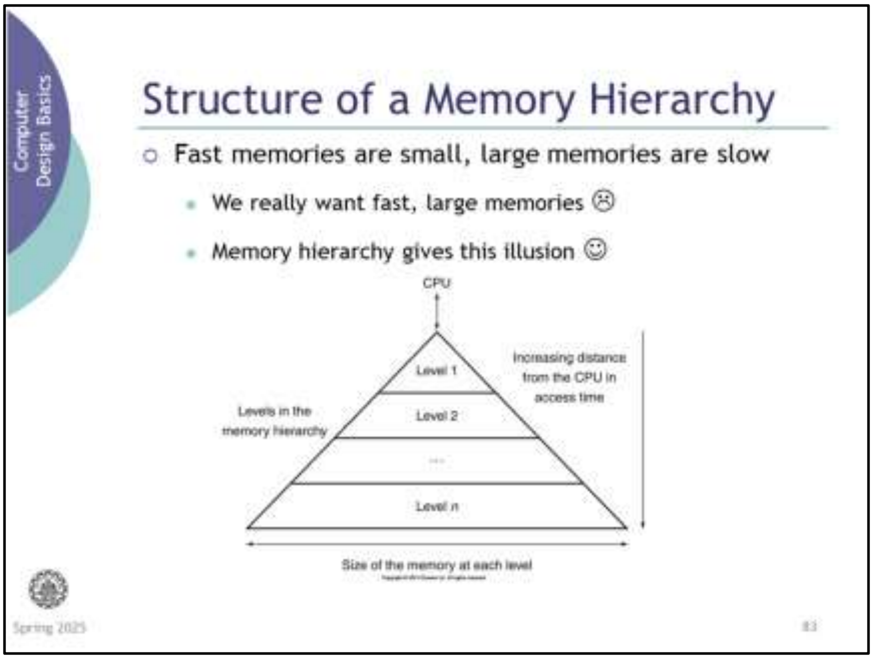
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit time: time taken
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Miss penalty: time taken
    - Miss ratio: misses/accesses
      = 1 - hit ratio
  - Then accessed data supplied from upper level

Spring 2025

82

Hit ratio: نرخ برخورد

Miss ratio: نرخ فقدان

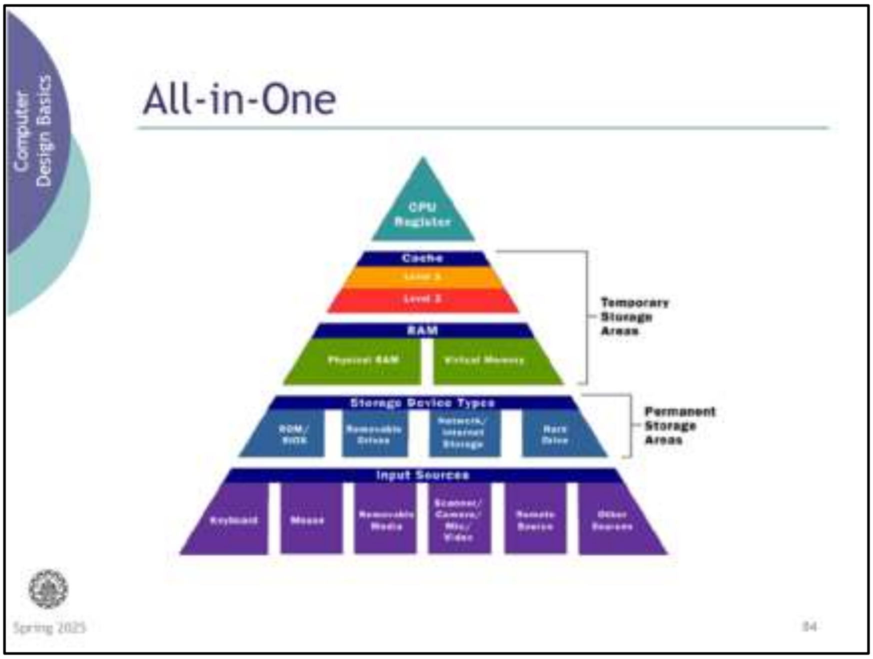


Computer Design Basics

Structure of a Memory Hierarchy

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Memory hierarchy gives this illusion ☺

CPU
Level 1
Level 2
...
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level



Computer Design Basics

All-in-One

Spring 2025

84

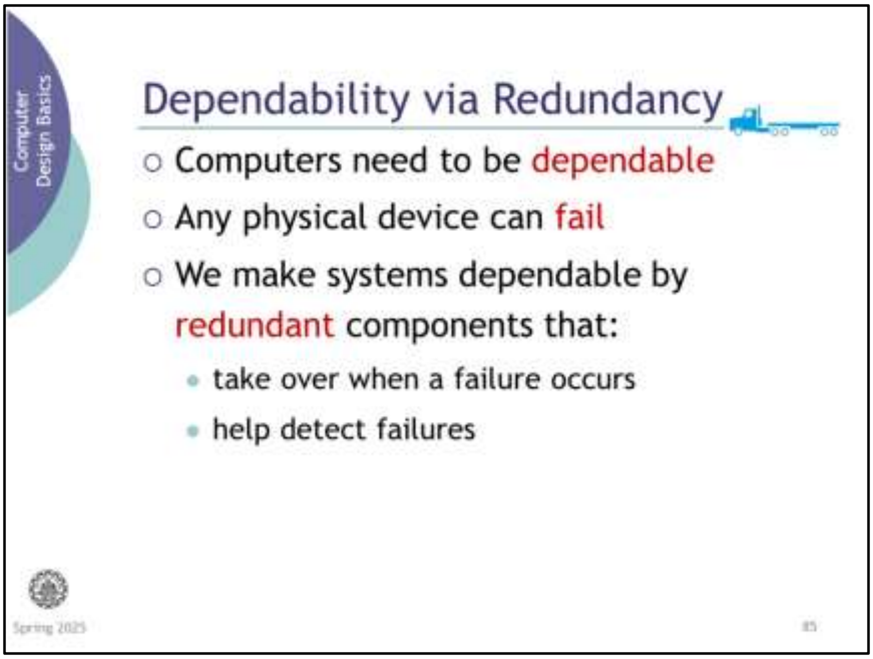
CPU registers hold words retrieved from Level1 (L1) cache

Level1 cache holds cache lines retrieved from the Level2 cache memory

Level2 cache holds cache lines retrieved from Main Memory

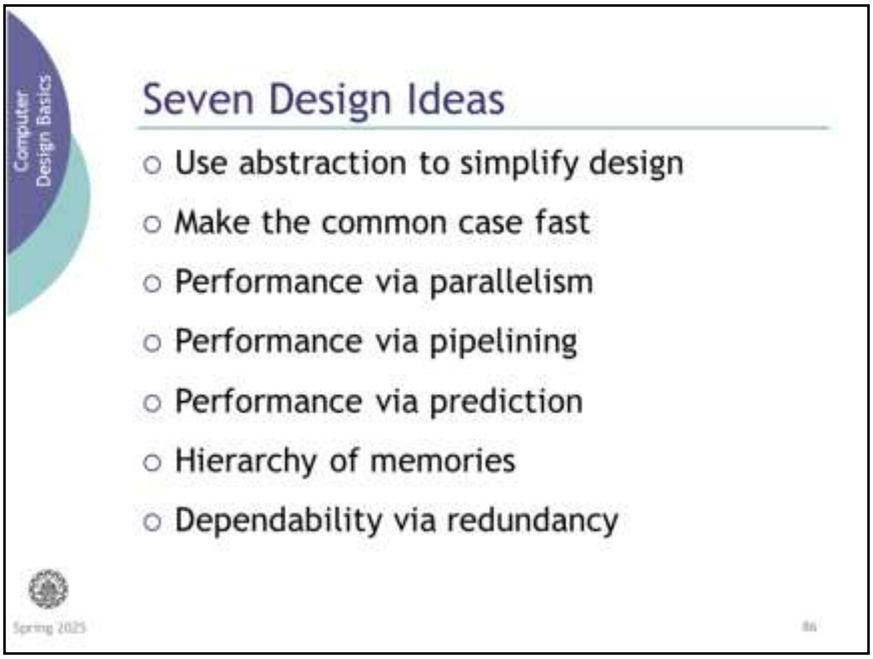
Main Memory holds disks blocks retrieved from local disks

Local disks hold files retrieved from disks on remote network servers



Dependability via Redundancy

- Computers need to be dependable
- Any physical device can fail
- We make systems dependable by redundant components that:
  - take over when a failure occurs
  - help detect failures



Seven Design Ideas

- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy