

Digital Design and Computer Organization

Course Code	1BCS303	Scheme	2025
Type of Course	PCC	Semester	3
Teaching Hours/Week (L:T:P)	4:0:0	CIE Marks	50
Total Hours of Pedagogy per semester L /T/P/SL&TW:	56:0:0:64:120	SEE Marks	50
Credits	4	Total Marks	100
Examination type (SEE)	Theory	Exam Hours	03

Module-1

Boolean Algebra and Logic Gates: Basic Theorems And Properties of Boolean Algebra, Boolean Functions, Digital Logic Gates, Integrated Circuits. Gate-Level Minimization: Introduction, The Map Method, Four-Variable K-Map, Product of sums simplification, Don't-Care Conditions, NAND and NOR Implementation.

Module-2

Combinational Logic: Introduction, Combinational Circuits, Analysis of Combinational Circuits, Design Procedure, Binary Adder – Subtractor, Decoders, Encoders, Multiplexers. Synchronous Sequential Logic: Introduction, Sequential Circuits, Storage Elements: Latches, Flip-Flops.

Module-3

Basic Structure of Computers: Functional Units, Basic Operational Concepts, Bus structure, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Instruction Set: CISC and RISC, Performance Measurement. Machine Instructions and Programs: Memory Location and Addresses, Memory Operations, Instruction and Instruction sequencing, Addressing Modes.

Module-4

Input/Output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Direct Memory Access, Buses. The Memory System: Some Basic Concepts, Semiconductor RAM Memories, Read Only Memory, Speed, size and Cost of memory systems. Cache Memories – Mapping Functions.

Module-5

Basic Processing Unit: Some Fundamental Concepts, Execution of a Complete Instruction, Multiple Bus Organization. Hardwired Control. Pipelining: Basic concepts, Role of Cache memory, Pipeline Performance.

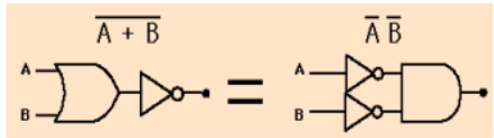
Module-1

Chapter 1: Boolean Algebra and Logic Gates

Basic Theorems and Properties of Boolean Algebra

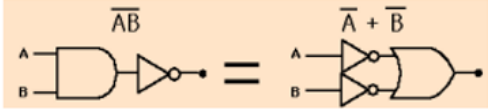
De Morgan's First Theorem: The complement of a sum equals the product of the complements.

A	B	A+B	$\overline{A+B}$	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0



De Morgan's Second Theorem: The complement of a product equals the sum of the complements.

A	B	AB	$\overline{AB}$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0



NAND Gate
Bubbled OR Gate

Duality Theorem: Starting with a Boolean relation, can derive another Boolean relation by

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign
3. Complementing any 0 or 1 appearing in the expression.

Laws of Boolean Algebra

- 1) Commutative Law: $A + B = B + A$ and $A \cdot B = B \cdot A$
- 2) Associative Law: $A + (B + C) = (A + B) + C$ and $A \cdot (BC) = (AB) \cdot C$
- 3) Distributive Law: $A(B + C) = AB + AC$
- 4) In relation to OR operation, the following laws hold good: $A + 0 = A$

$$A + A = A$$

$$A + 1 = 1 \text{ and } A + A' = 1$$

- 5) In relation to AND operation, the following laws hold good: $A \cdot 1 = A$

$$A \cdot A = A$$

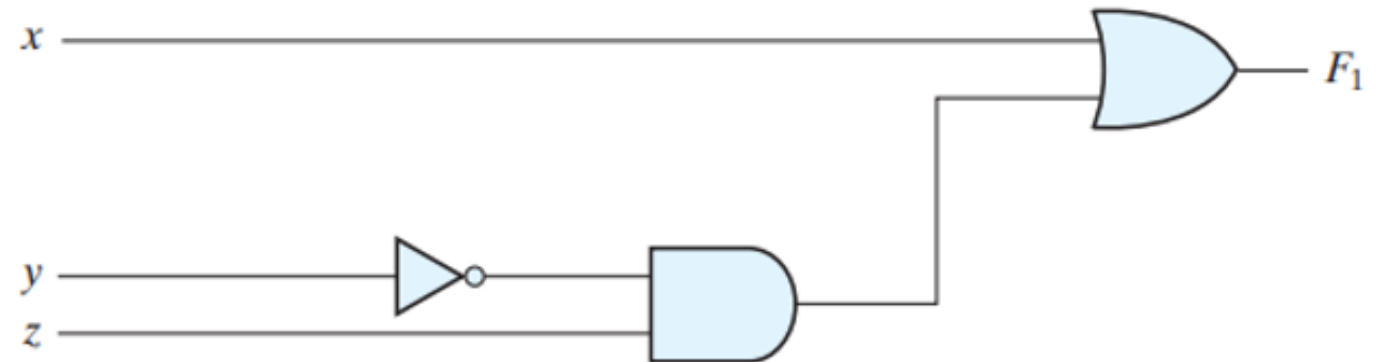
$$A \cdot 0 = 0$$

$$A \cdot A' = 0 \text{ and } A'' = A$$

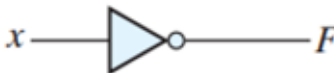
Boolean Functions

$$F_1 = x + y'z.$$

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	

Digital Logic Gates

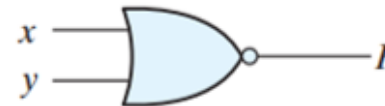
NAND



$$F = (xy)'$$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

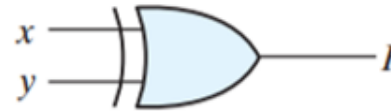
NOR



$$F = (x + y)'$$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR
(XOR)

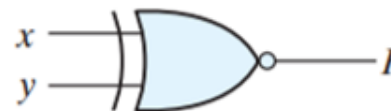


$$F = xy' + x'y$$

$$= x \oplus y$$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR
or
equivalence

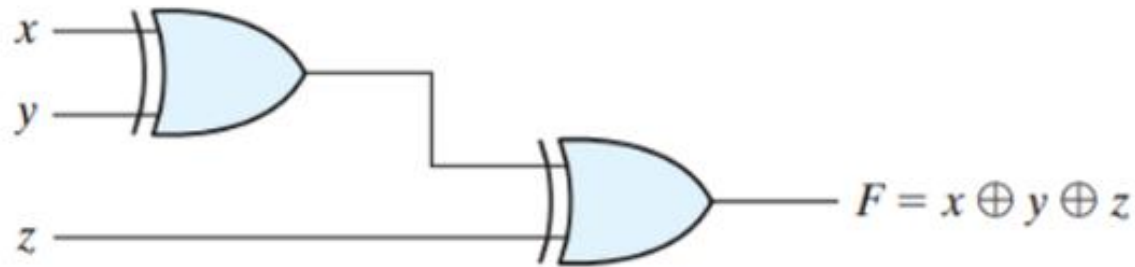


$$F = xy + x'y'$$

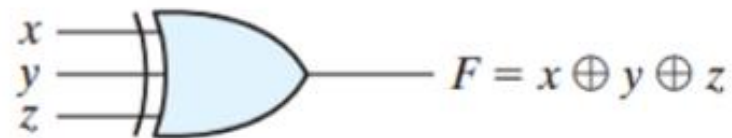
$$= (x \oplus y)'$$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Extension to Multiple Inputs



(a) Using 2-input gates

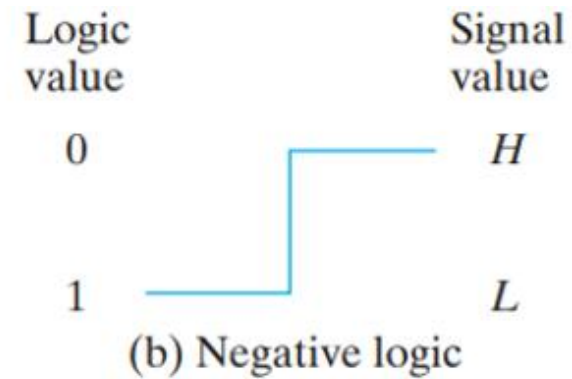
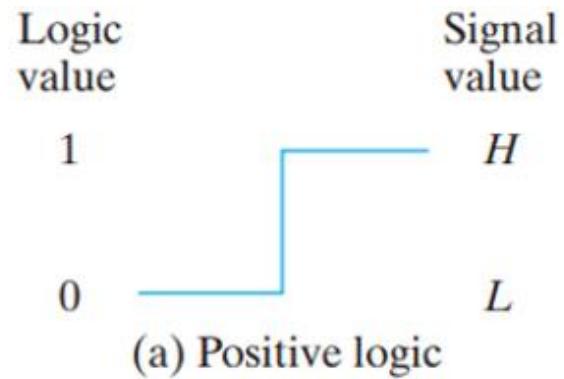


(b) 3-input gate

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Truth table

Positive and Negative Logic



Levels of Integration

- Small-scale integration (SSI)
- Medium-scale integration (MSI)
- Large-scale integration (LSI)
- Very large-scale integration (VLSI)

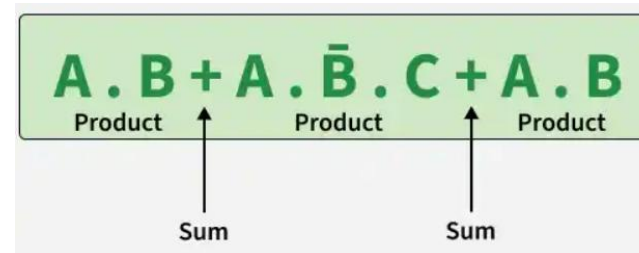
Digital Logic Families

- TTL transistor–transistor logic
- ECL emitter-coupled logic
- MOS metal-oxide semiconductor
- CMOS complementary metal-oxide semiconductor.

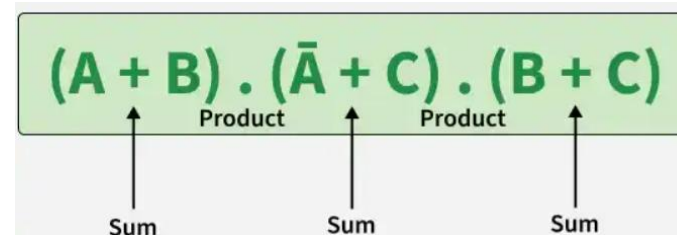
Chapter 2: Gate-Level Minimization

Minimum Forms of Switching Functions

- A minimum sum-of-products expression for a function is defined as a sum of product terms which
 - (a) has a minimum number of terms and
 - (b) of all those expressions which have the same minimum number of terms, has a minimum number of literals.

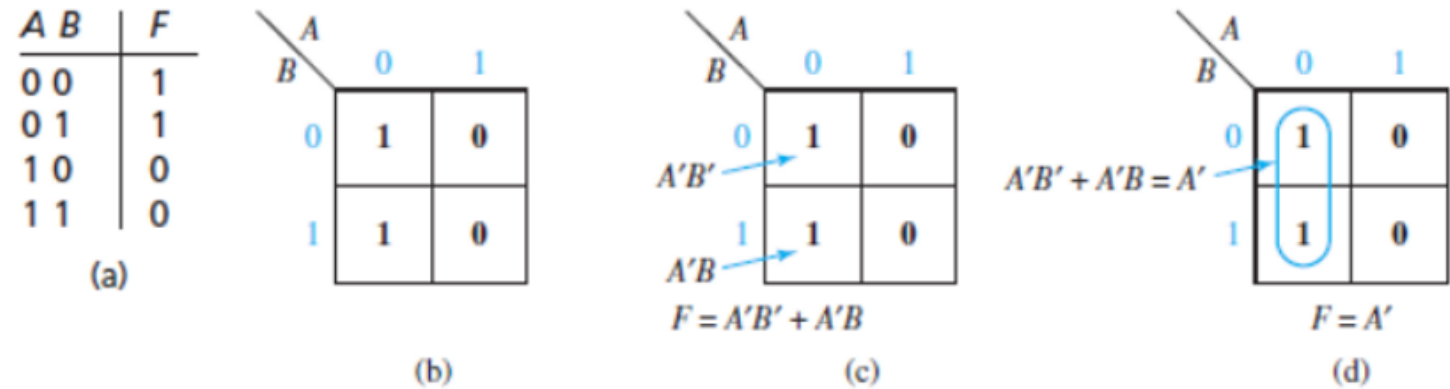


- A minimum product-of-sums expression for a function is defined as a product of sum terms which
 - (a) has a minimum number of terms, and
 - (b) of all those expressions which have the same number of terms, has a minimum number of literals.

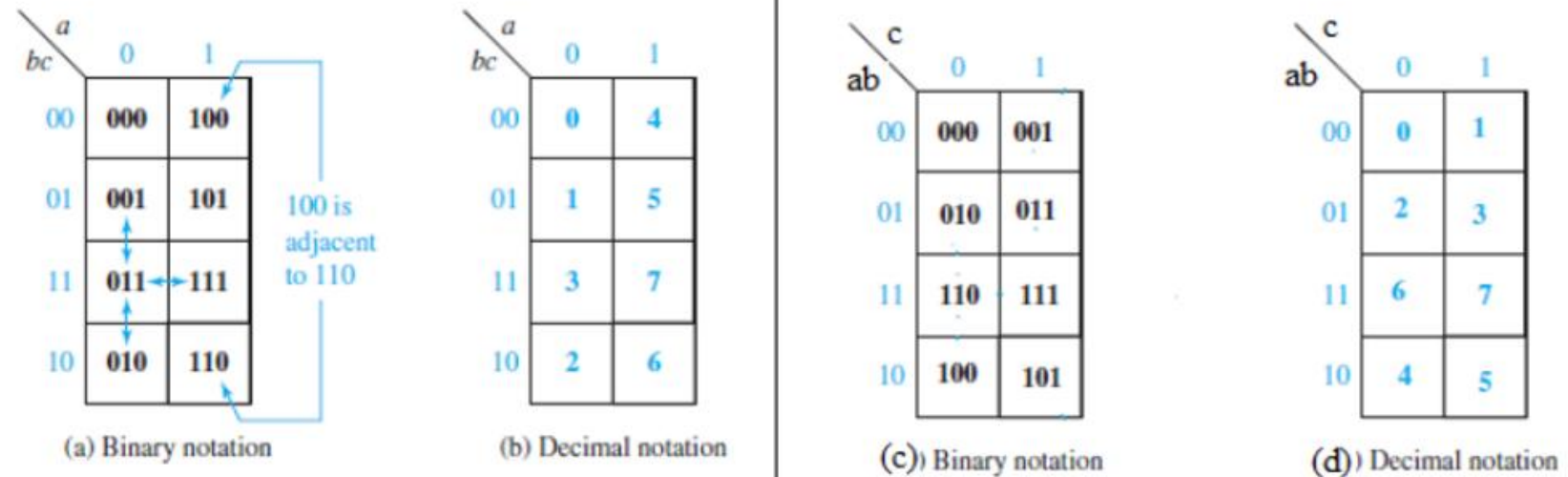


The Map Method

Two variable K-Map



Three Variable K-Map



Four-Variable K-Map

AB \ CD	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

ab \ cd	00	01	11	10
00		1	1	
01	1	1	1	
11	1			
10				1

$$f_1 = \sum m(1, 3, 4, 5, 10, 12, 13)$$

$$= bc' + a'b'd + ab'cd'$$

ab \ cd	00	01	11	10
00	1			1
01		1		
11	1	1	1	1
10	1	1	1	1

$$f_2 = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$$

$$= c + b'd + a'bd$$

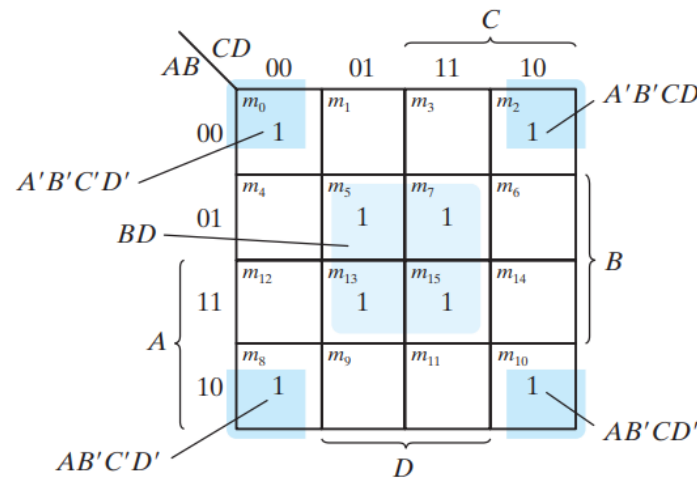
Four corner terms combine to give $b'd'$

$a'b'd$

Prime Implicants

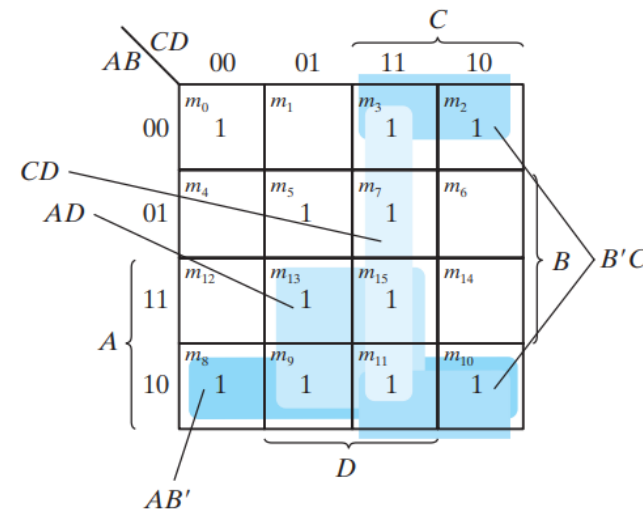
A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.

Boolean function: $F(A, B, C, D) = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
 $AB'C'D' + AB'CD' = AB'D'$
 $A'B'D' + AB'D' = B'D'$

(a) Essential prime implicants
BD and B'D'



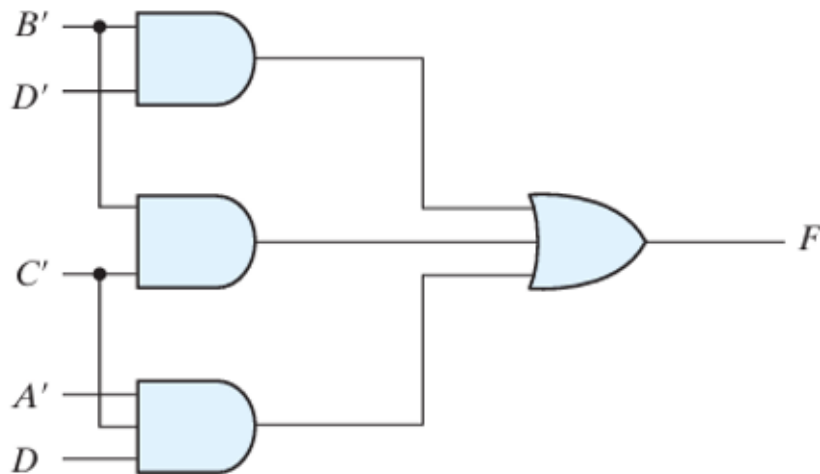
(b) Prime implicants CD, B'C,
AD, and AB'

Product of sums simplification

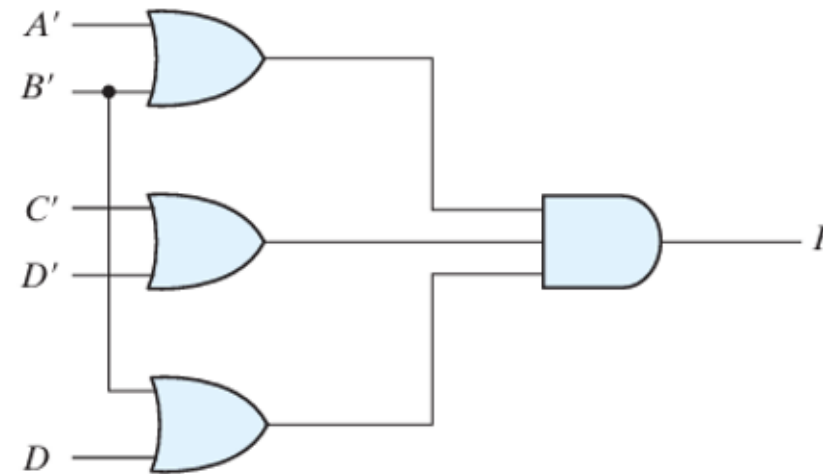
$$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$

AB \ CD		C			
		00	01	11	10
00	m_0	1	1	0	1
	m_1	1	1	0	1
01	m_4	0	1	0	0
	m_5	0	1	0	0
11	m_{12}	0	0	0	0
	m_{13}	0	0	0	0
10	m_8	1	1	0	1
	m_9	1	1	0	1
		D			

Groupings: $BC'D'$ (m₀, m₁), BCD' (m₄, m₅), AB (m₈, m₉), CD (m₀, m₁), B (m₄, m₅, m₁₂, m₁₃), A (m₁₂, m₁₃, m₈, m₉).



(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

Don't-Care Conditions

Simplify the Boolean function $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$ which has the don't-care conditions

$$d(w, x, y, z) = \sum(0, 2, 5)$$

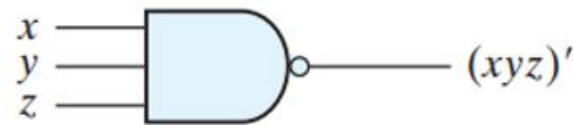
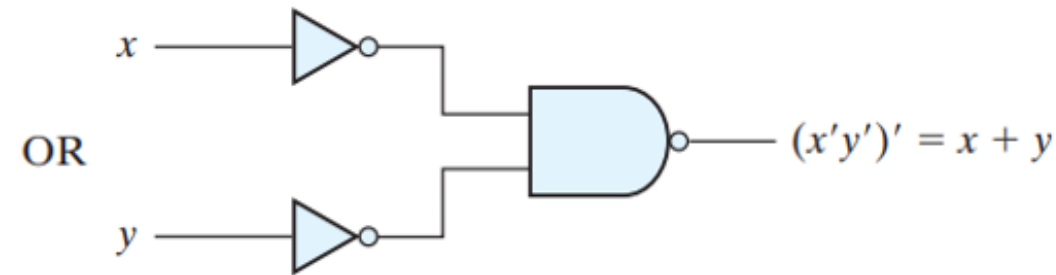
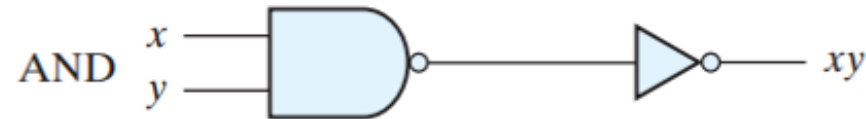
		y			
		yz		11	10
wx	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
w'x'	00	X	1	1	X
	01	0	X	1	0
w	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}
				z	yz

(a) $F = yz + w'x'$

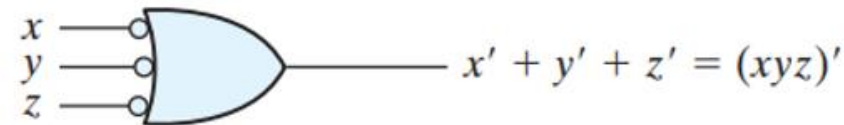
		y			
		yz		11	10
wx	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
w'z	00	X	1	1	X
	01	0	X	1	0
w	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}
				z	yz

(b) $F = yz + w'z$

NAND Implementation

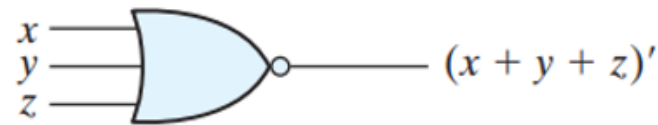
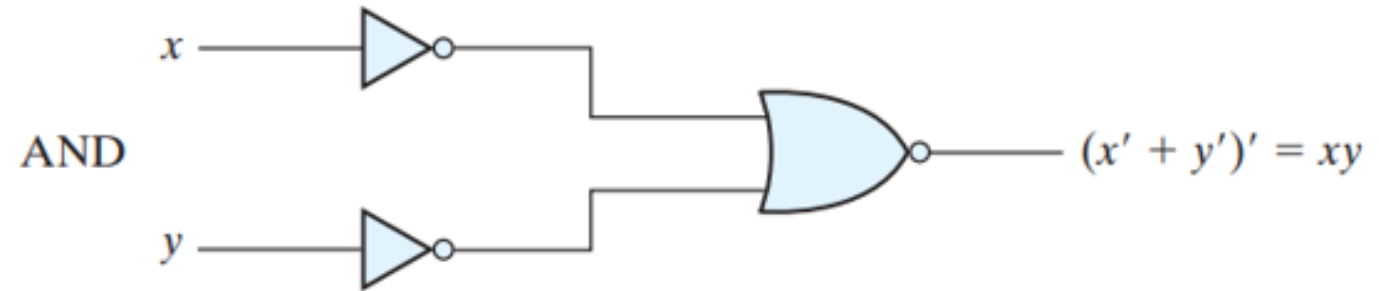


(a) AND-invert

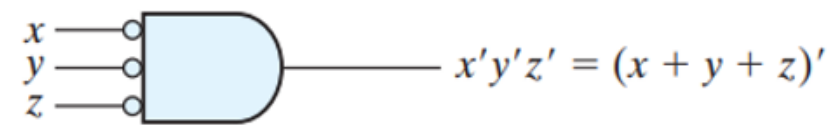


(b) Invert-OR

NOR Implementation



(a) OR-invert



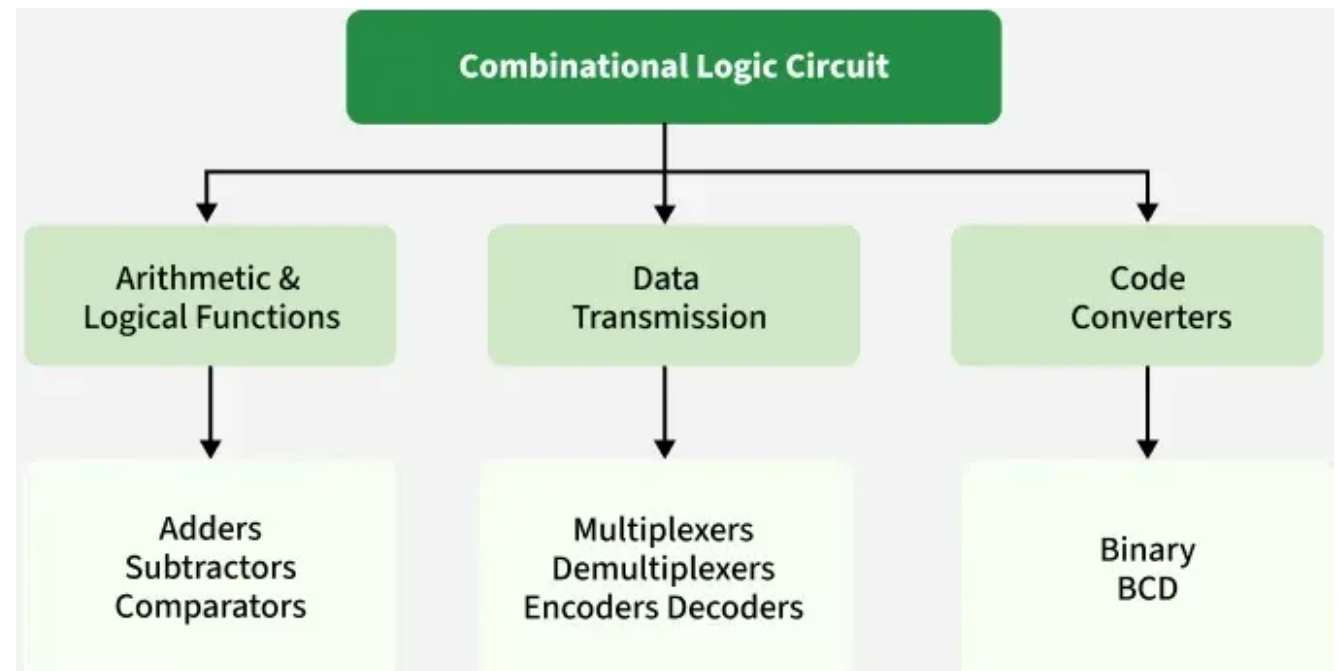
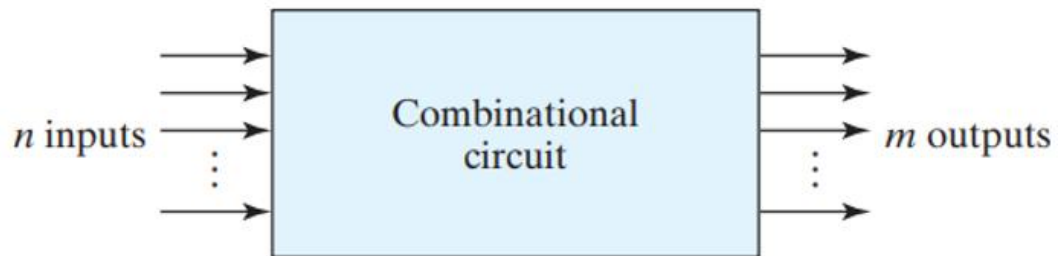
(b) Invert-AND

Module-2

Chapter 1: Combinational Logic

Introduction

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.



Design Procedure

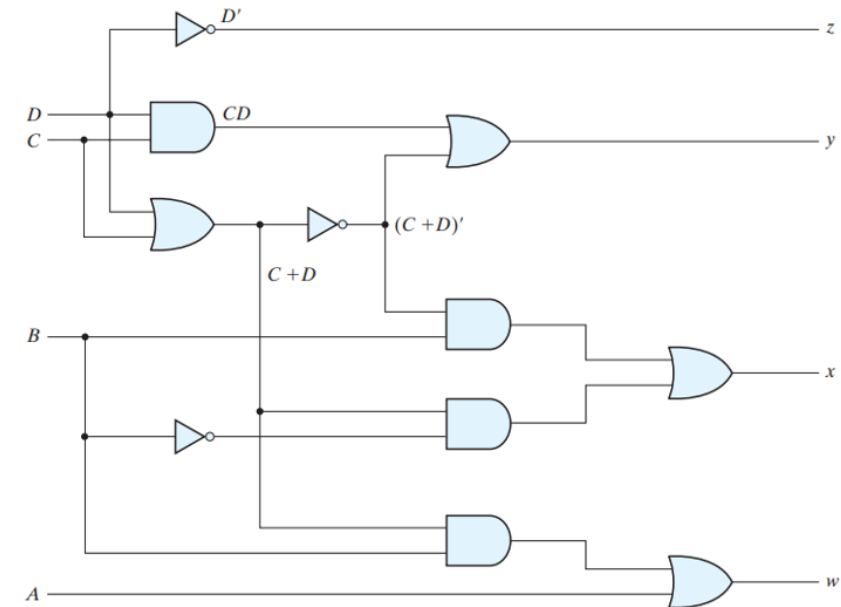
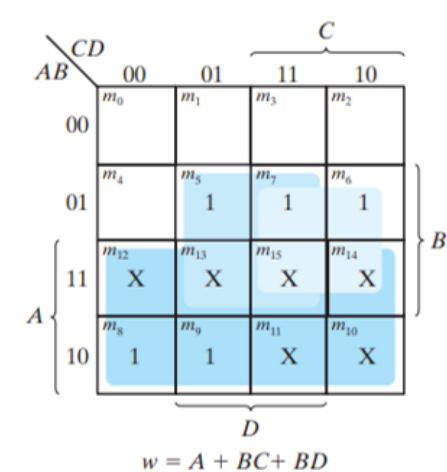
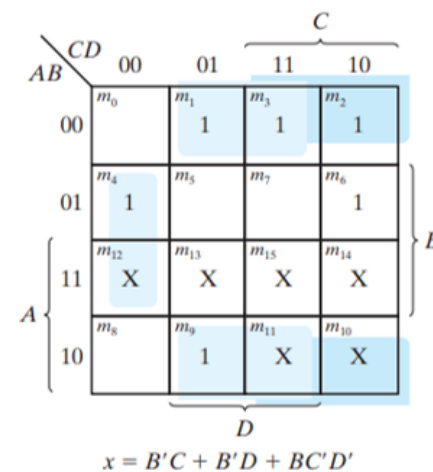
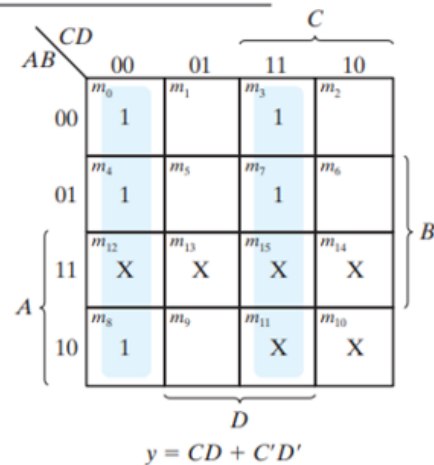
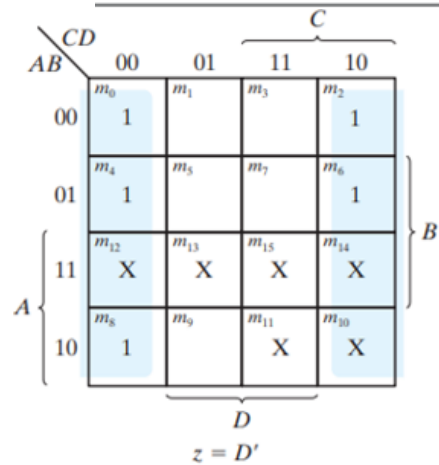
The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Code Conversion Example

Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



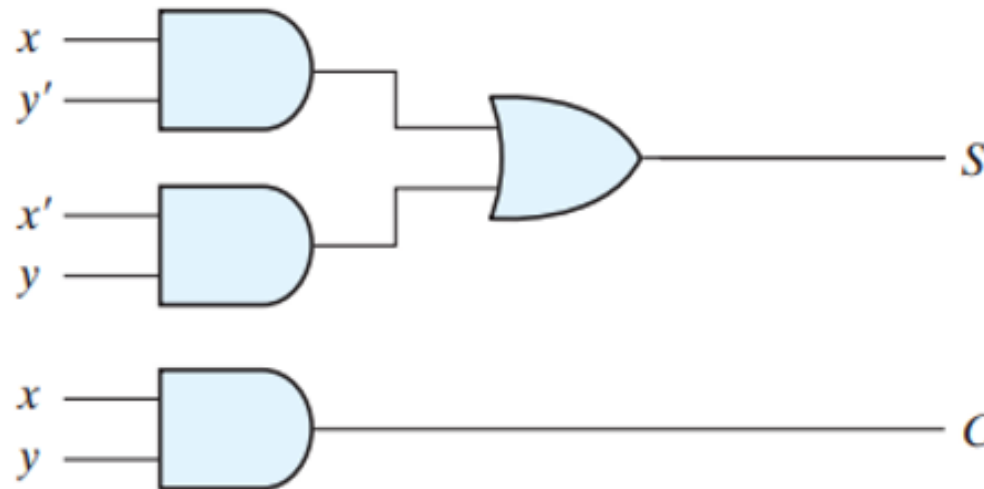
Binary Adder – Subtractor

Half Adder

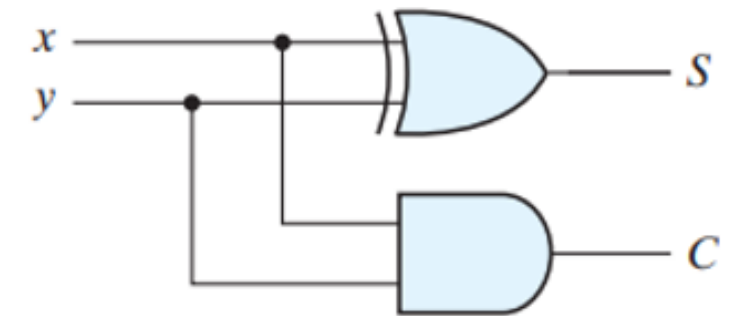
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy'$$

$$C = xy$$



(a) $S = xy' + x'y$
 $C = xy$



(b) $S = x \oplus y$
 $C = xy$

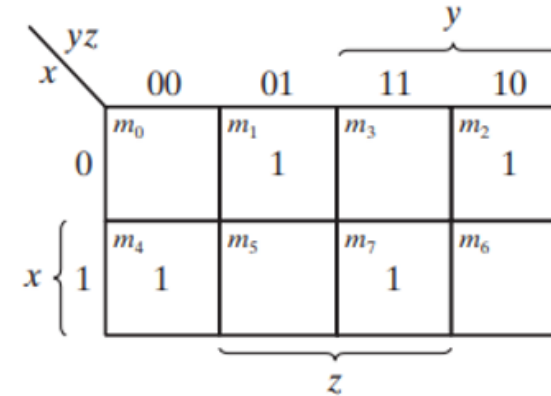
Binary Adder – Subtractor

Full Adder

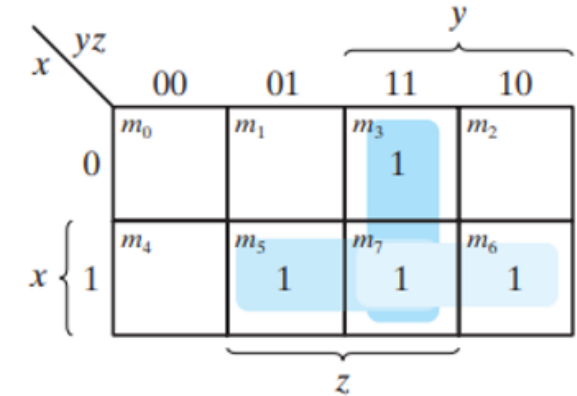
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = x'y'z + x'yz' + xy'z' + xyz$$

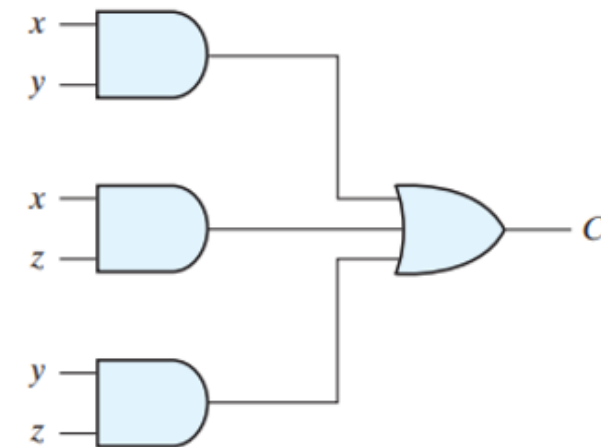
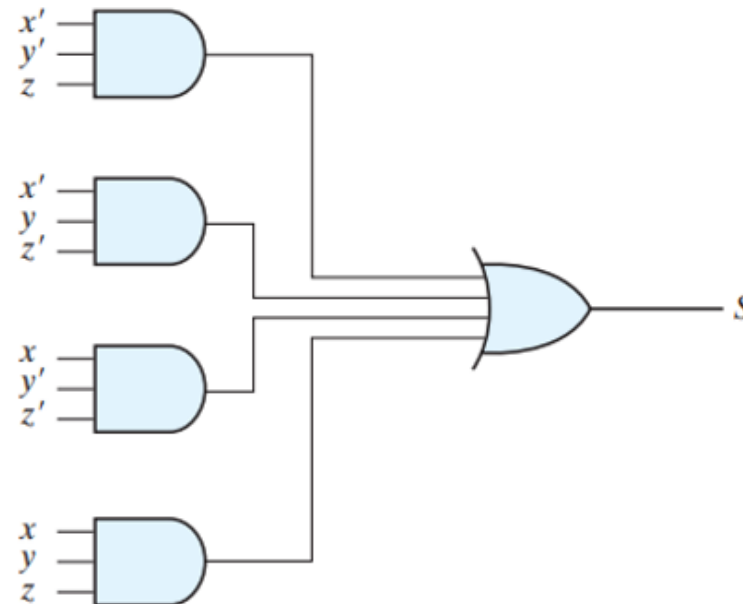
$$C = xy + xz + yz$$



$$(a) S = x'y'z + x'yz' + xy'z' + xyz$$

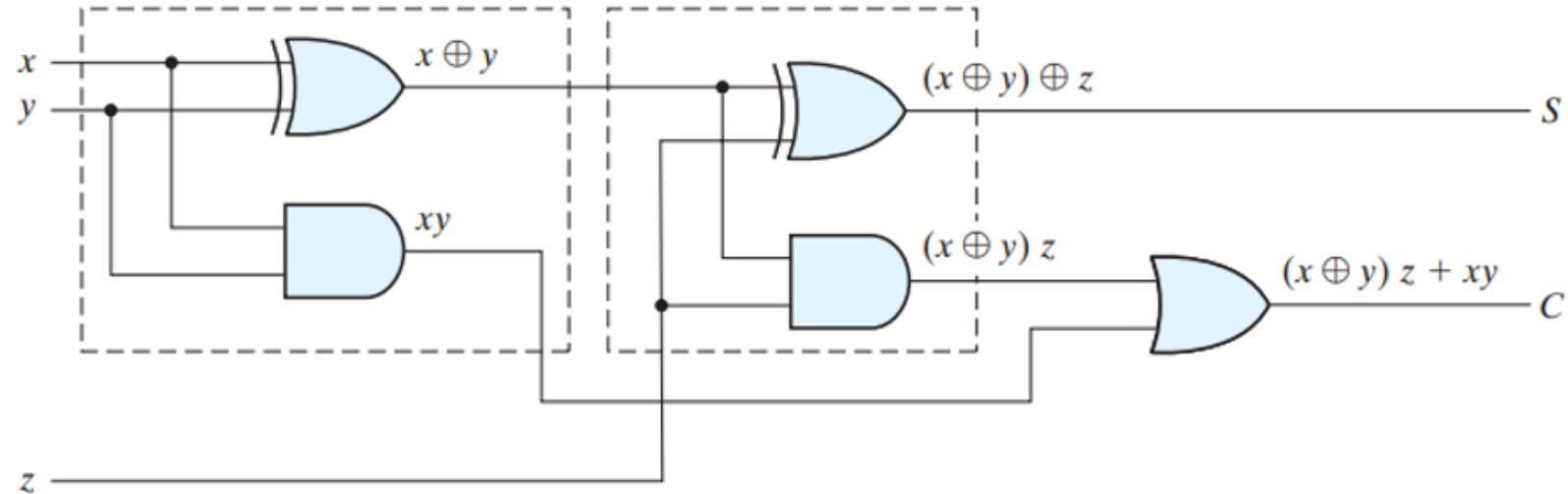


$$(b) C = xy + xz + yz$$



Binary Adder – Subtractor

Full Adder

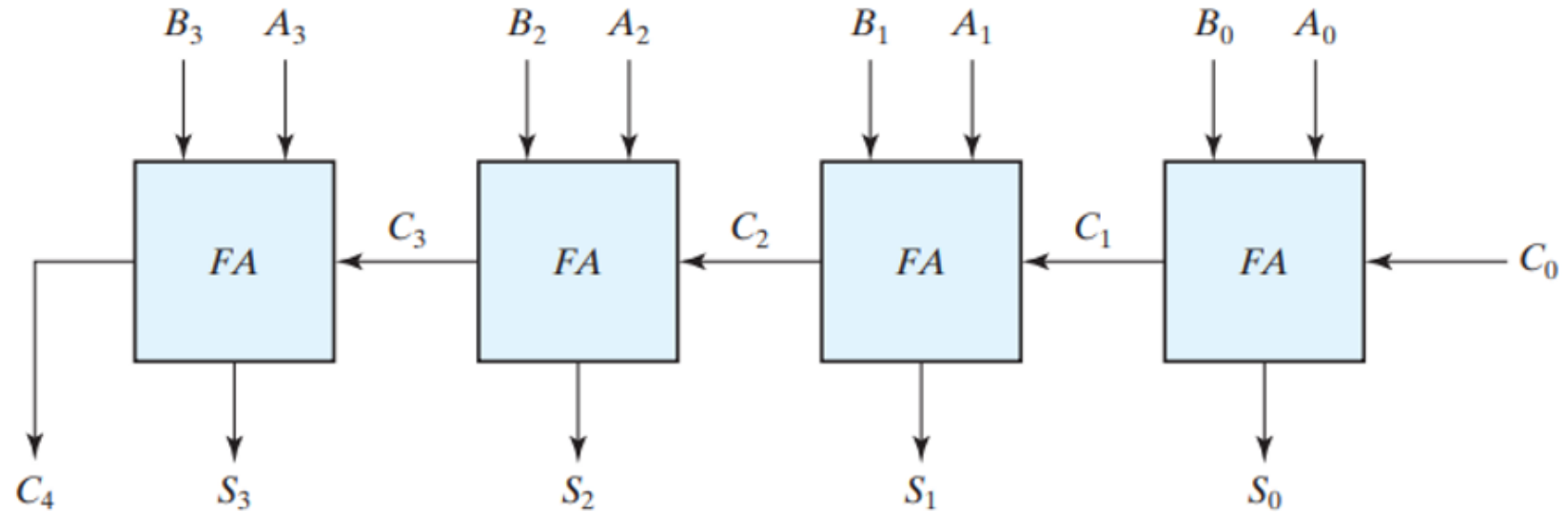


$$\begin{aligned}
 S &= z \oplus (x \oplus y) \\
 &= z'(xy' + x'y) + z(xy' + x'y)' \\
 &= z'(xy' + x'y) + z(xy + x'y') \\
 &= xy'z' + x'yz' + xyz + x'y'z
 \end{aligned}$$

The carry output is

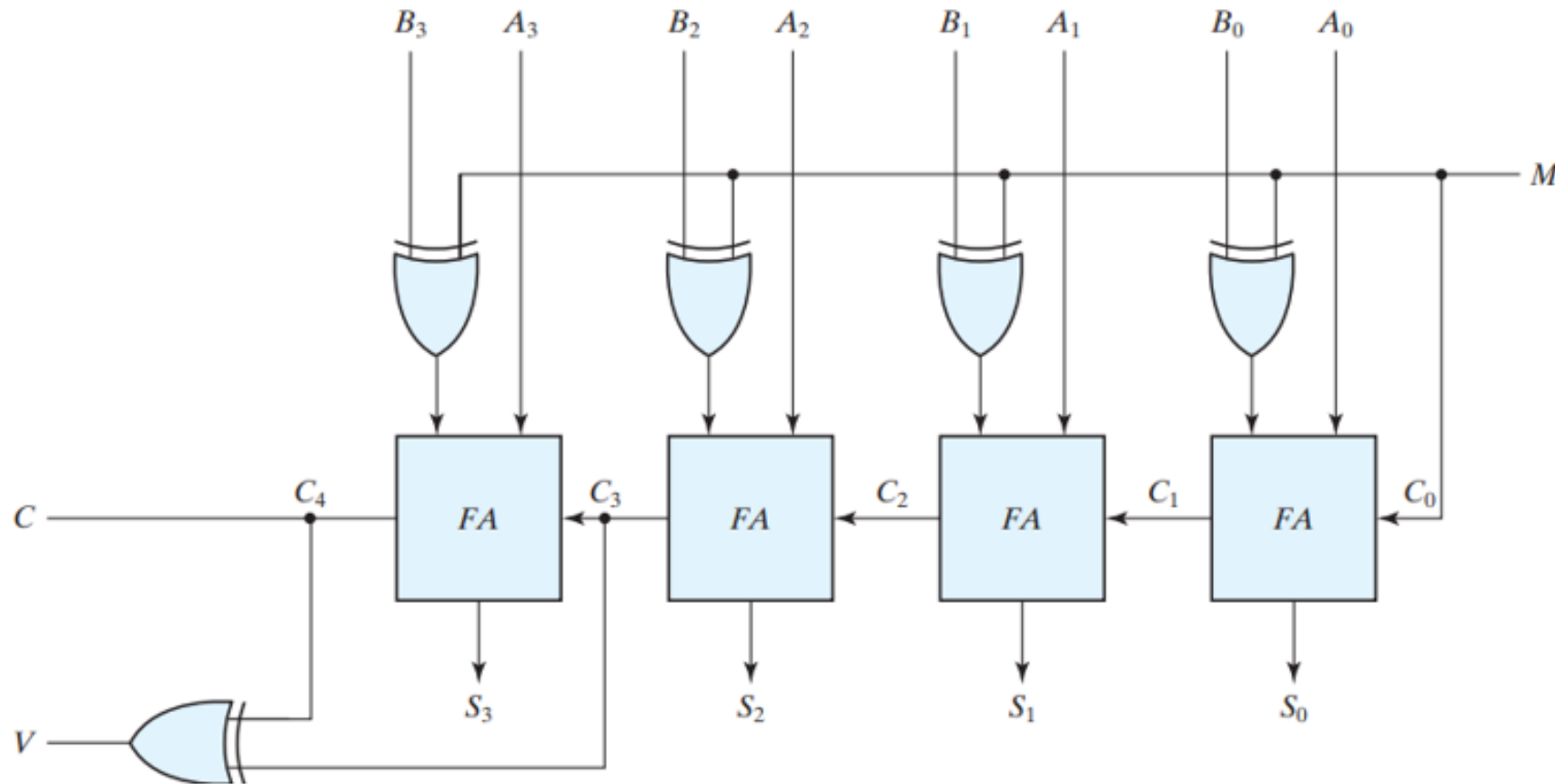
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Binary Adder



Subscript i :	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

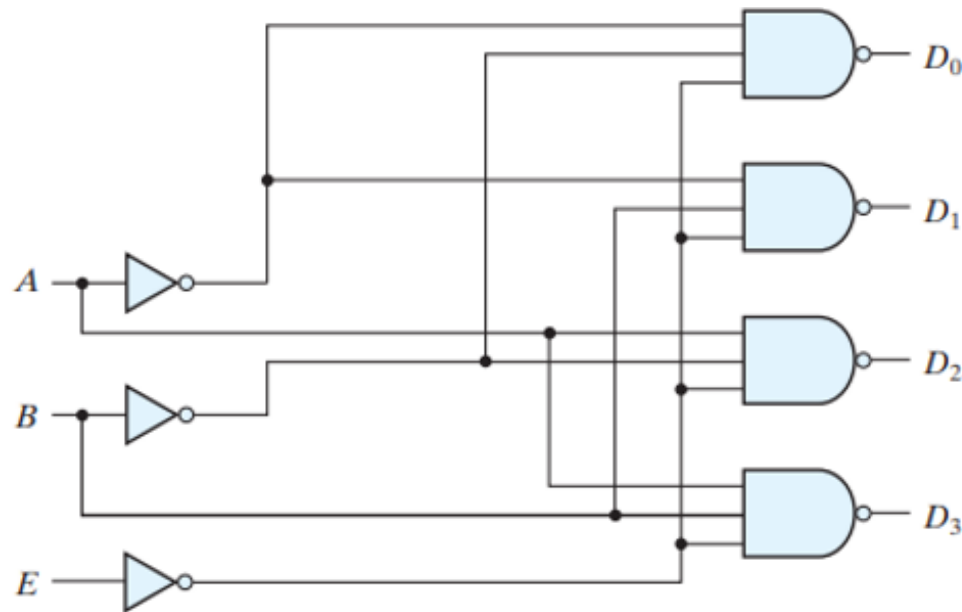
Four-bit Adder-subtractor



Decoders

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

Two-to-four-line decoder with an enable input constructed with NAND gates



(a) Logic diagram

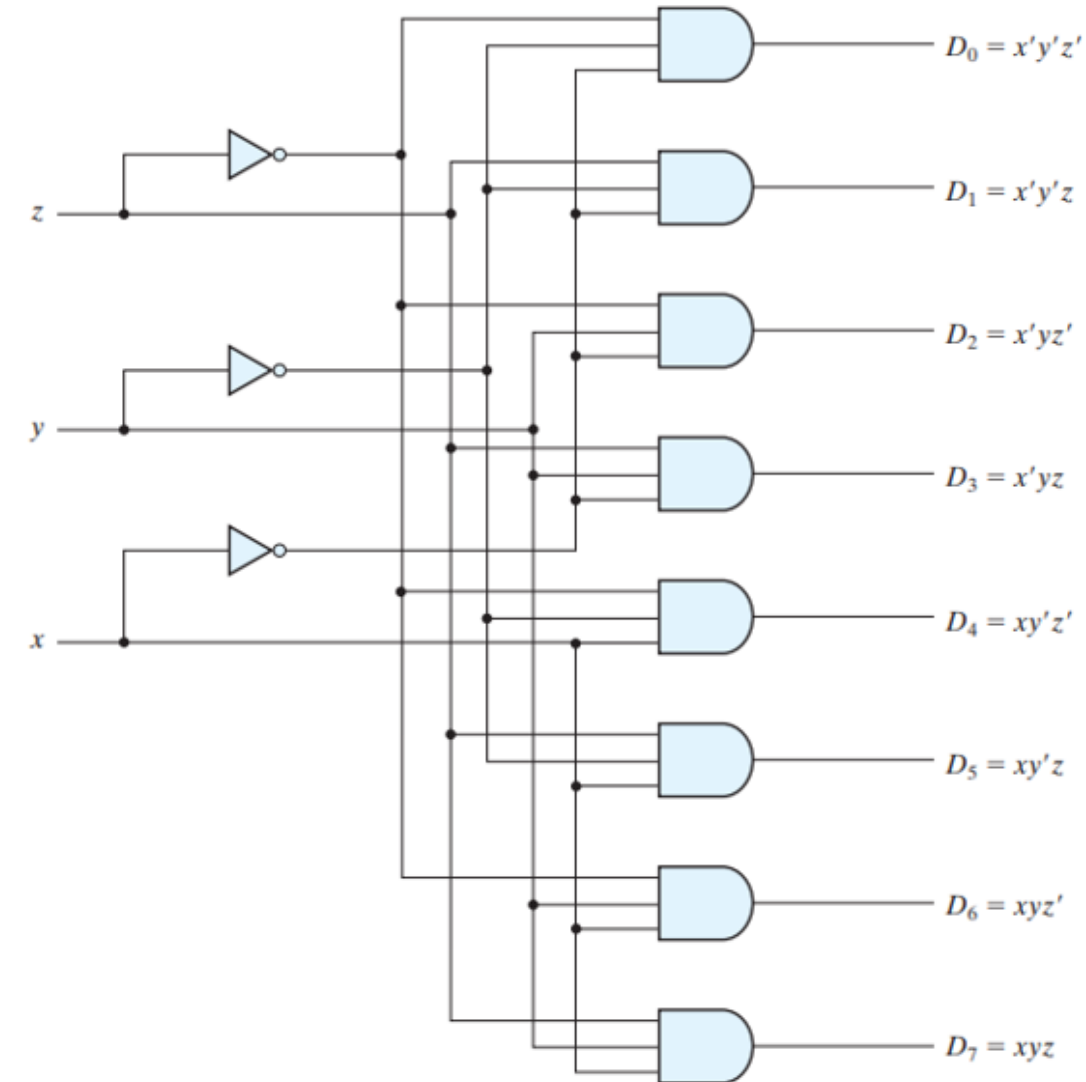
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

Decoders

Three-to-eight-line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

Octal-to-binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

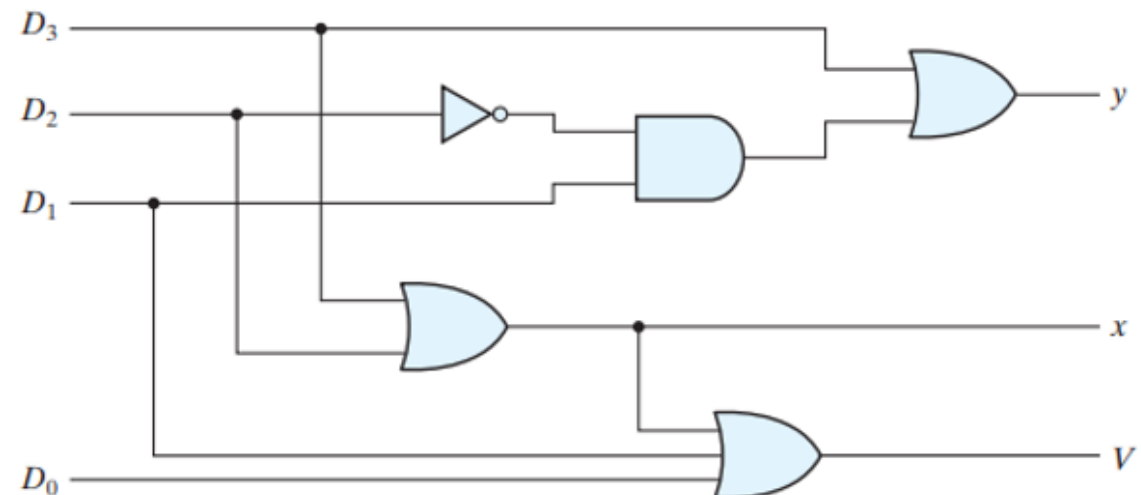
Four-input Priority Encoder

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

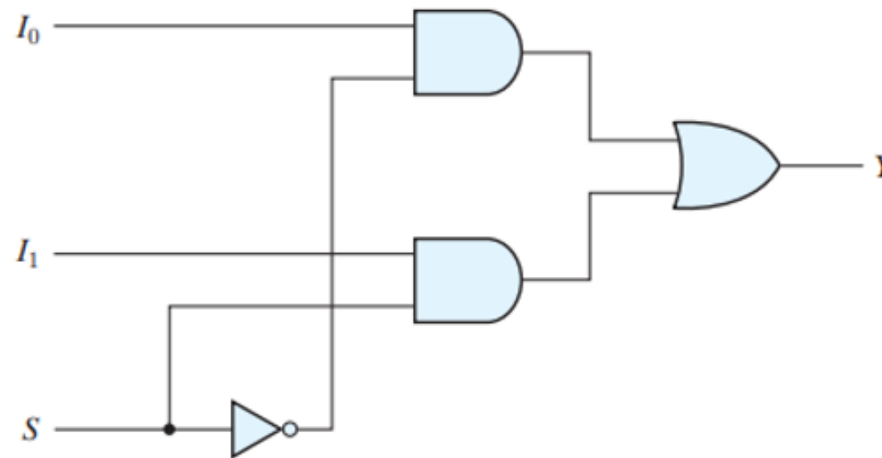
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1



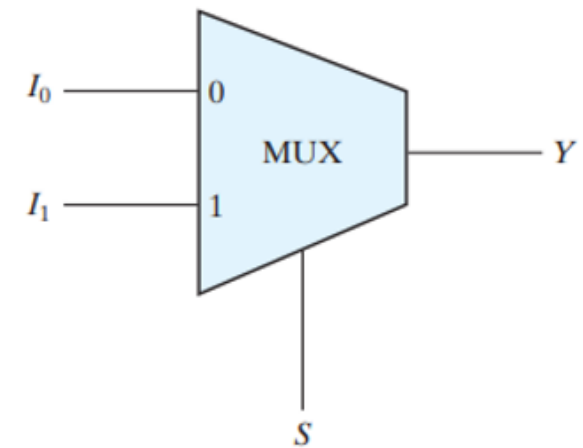
Multiplexers

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

Two-to-one-line Multiplexer



(a) Logic diagram

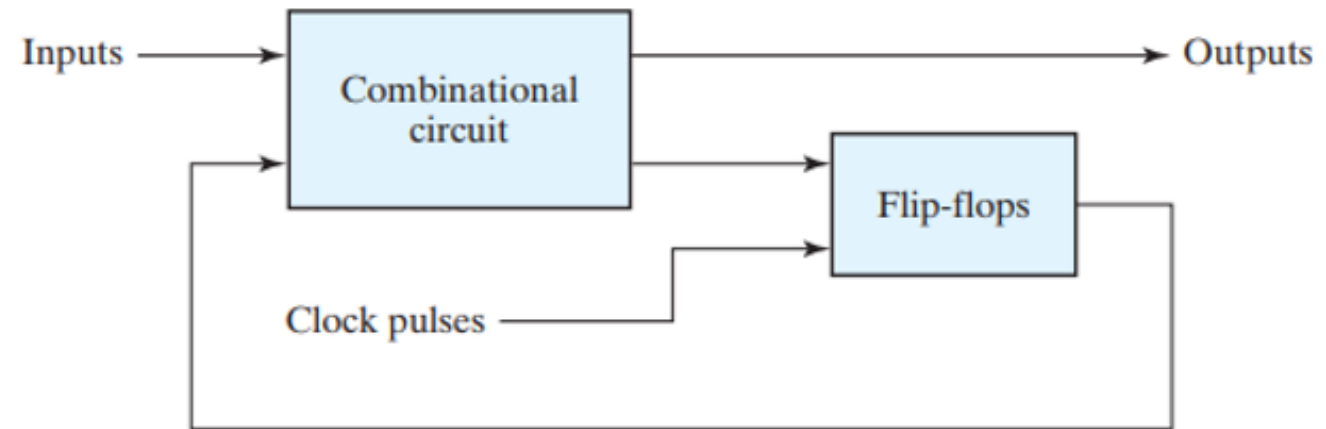
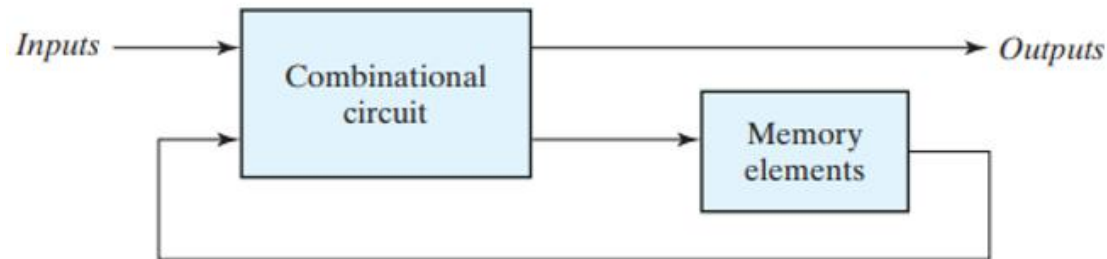


(b) Block diagram

Chapter 2: Synchronous Sequential Logic

Sequential Circuits

Sequential circuits are digital logic circuits whose outputs depend on both current input values and previous state information stored in internal memory



(a) Block diagram

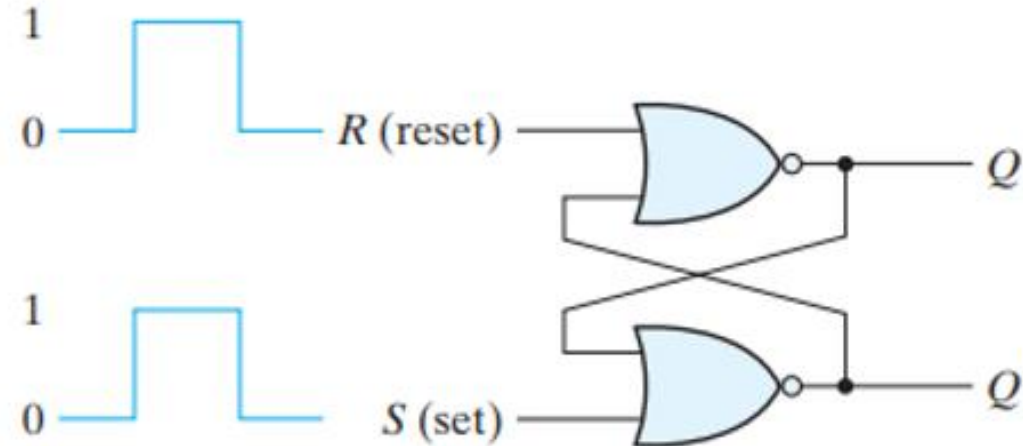


(b) Timing diagram of clock pulses

Latches

- A storage element in a digital circuit can maintain a binary state indefinitely, until directed by an input signal to switch states.
- The major differences among various types of storage elements are in the number of inputs they possess and in the way the inputs affect the binary state.
- Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops.
- Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.

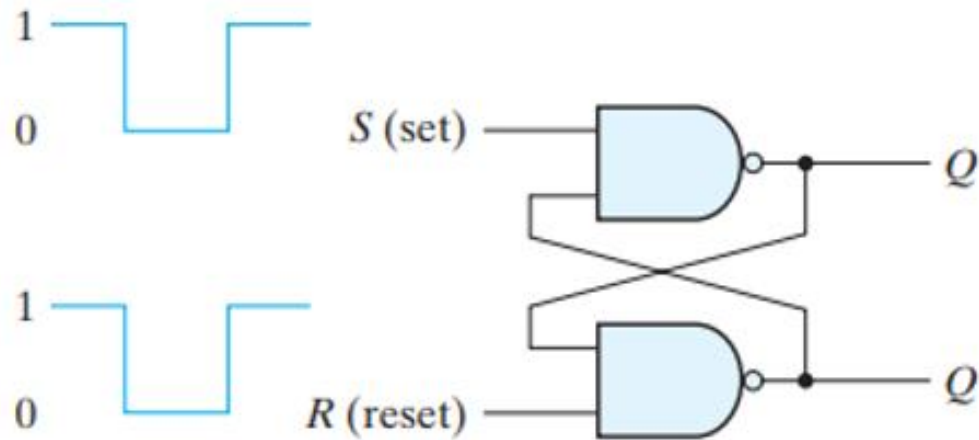
SR Latch



(a) Logic diagram

S	R	Q	Q'
1	0	1	0
0	0	1	0 (after $S = 1, R = 0$)
0	1	0	1
0	0	0	1 (after $S = 0, R = 1$)
1	1	0	0 (forbidden)

(b) Function table

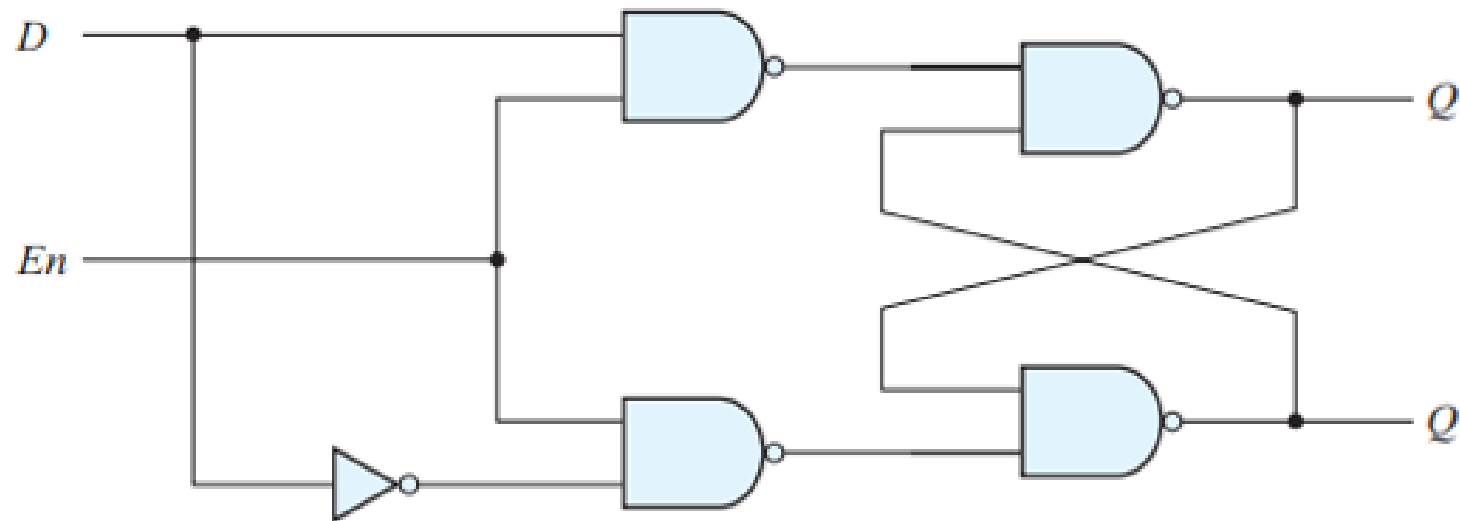


(a) Logic diagram

S	R	Q	Q'
1	0	0	1
1	1	0	1 (after $S = 1, R = 0$)
0	1	1	0
1	1	1	0 (after $S = 0, R = 1$)
0	0	1	1 (forbidden)

(b) Function table

D Latch



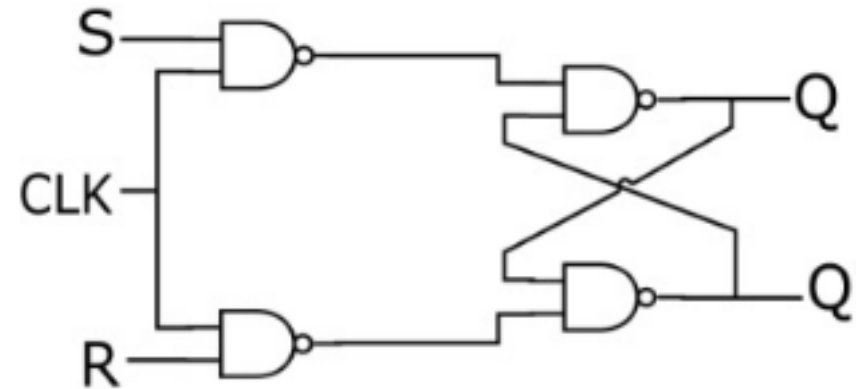
(a) Logic diagram

En	D	Next state of Q
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(b) Function table

Flip-flops

S-R Flip-Flop

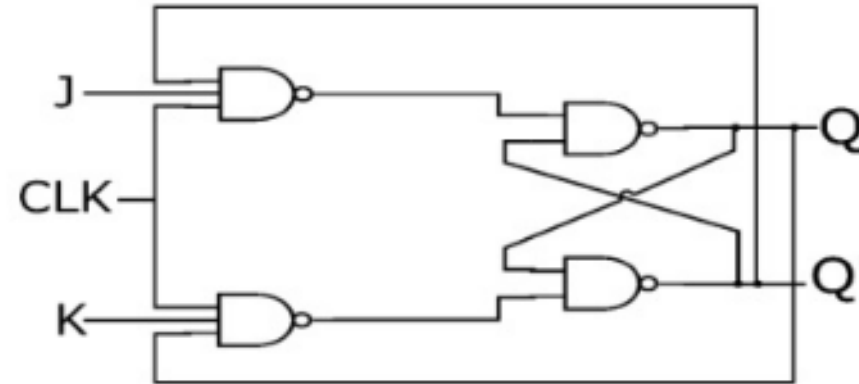


Truth Table of S-R Flip-Flop

S	R	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set
1	1	X	

Flip-flops

J-K Flip-Flop

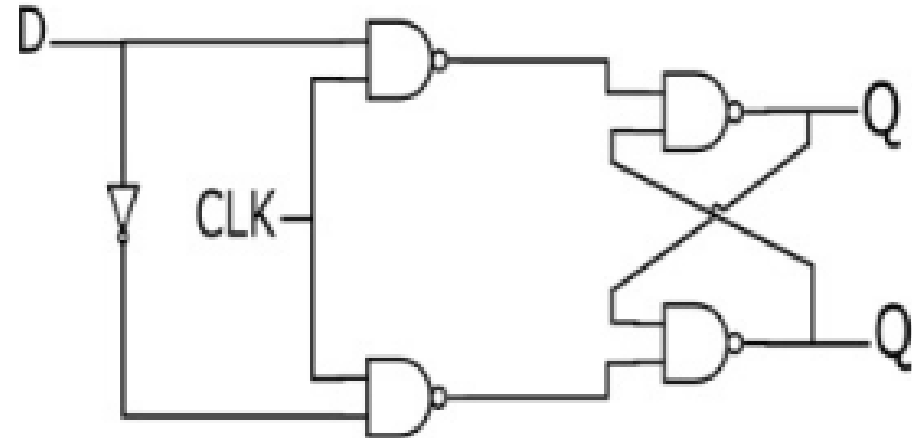


Truth Table of JK Flip-Flop

J	K	Q	State
0	0	0	No Change
0	1	0	Reset
1	0	1	Set
1	1	Toggles	Toggle

Flip-flops

D Flip-Flop

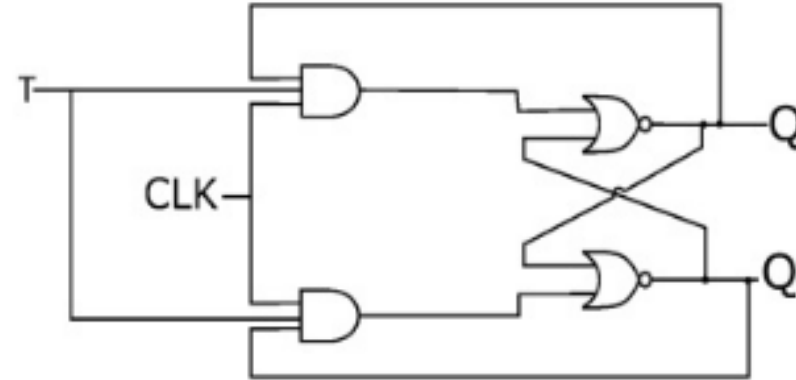


Truth Table of D Flip-Flop

D	Q
0	0
1	1

Flip-flops

T Flip-Flop



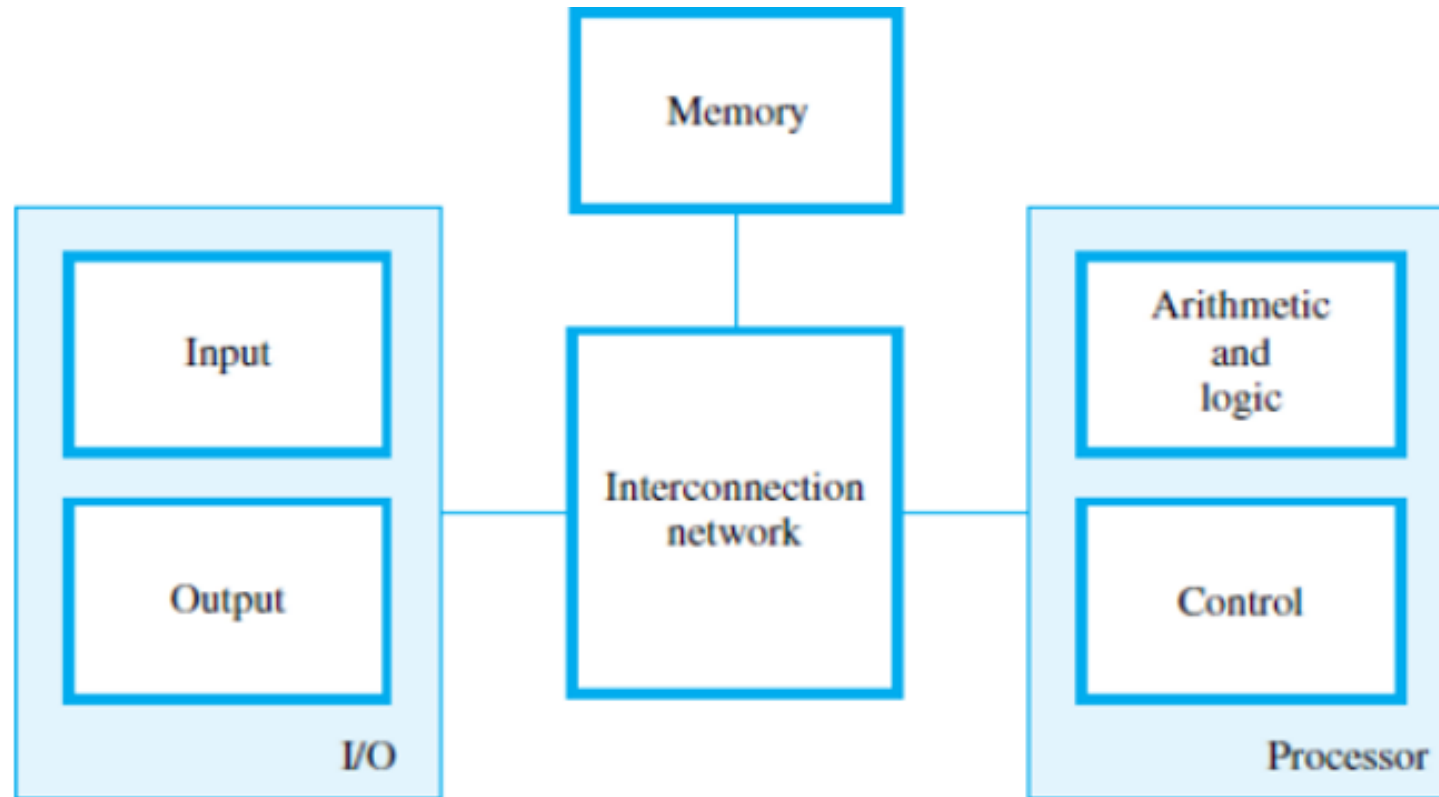
Truth Table of T Flip-Flop

T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Module-3

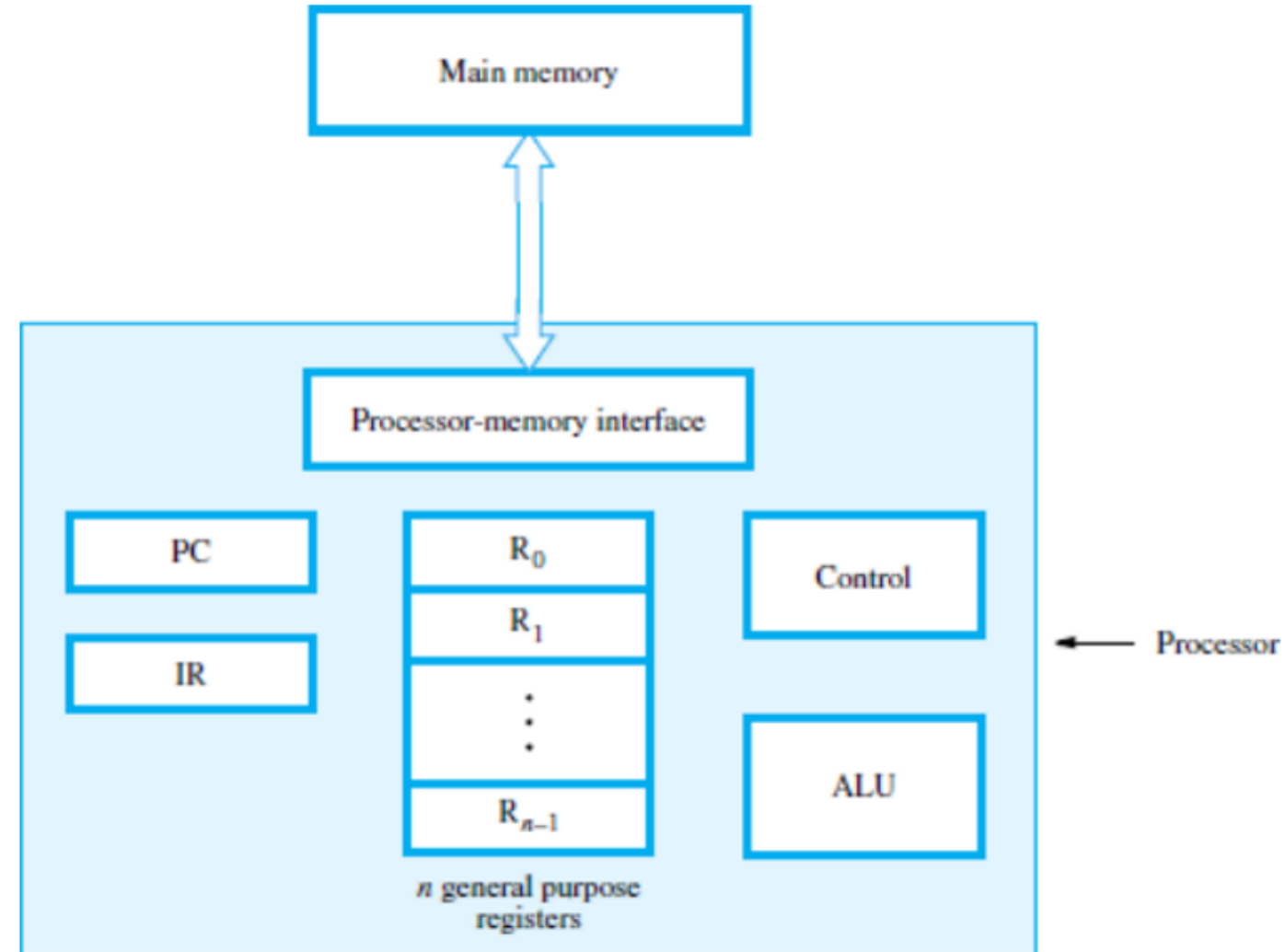
Chapter 1: Basic Structure of Computers

Functional Units



Basic Operational Concepts

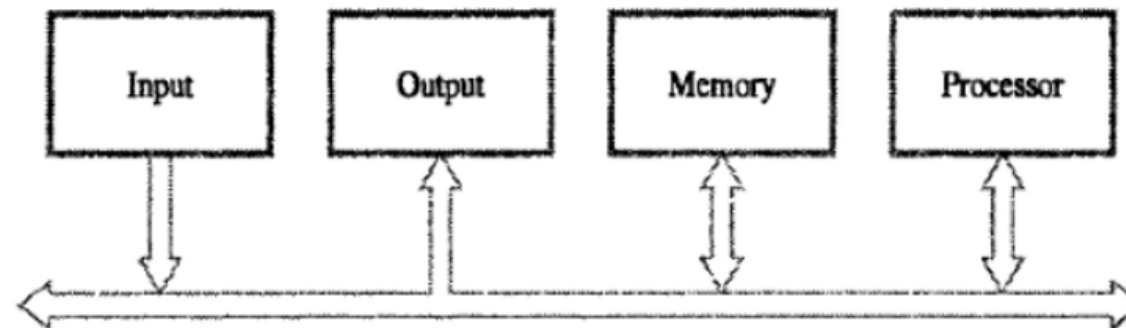
ADD LOCA, R0



Bus structure

A bus is a group of lines that serves as a connecting path for several devices. A bus may be lines or wires. The lines carry data or address or control signal. There are 2 types of Bus structures:

- 1) **Single Bus Structure:** Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

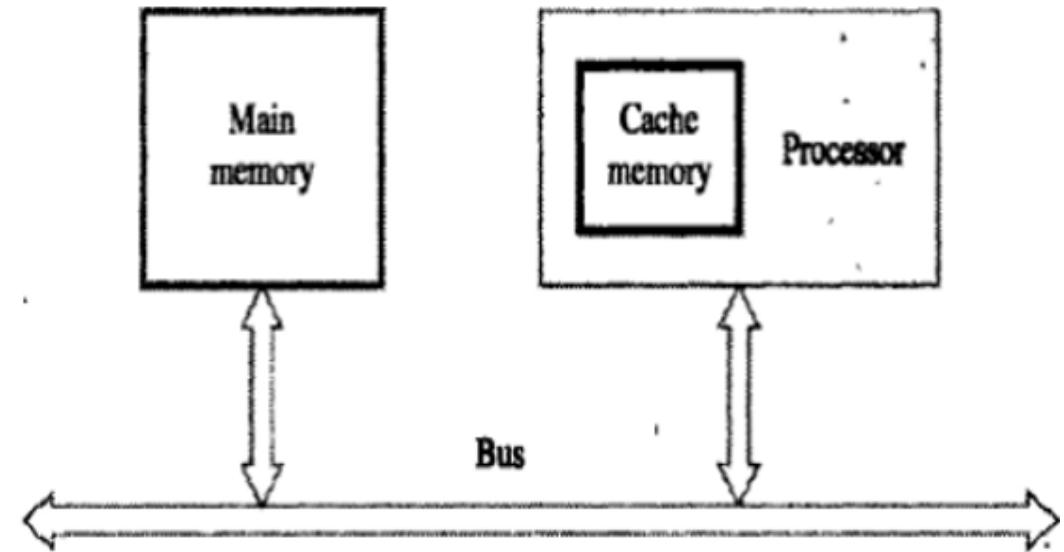


- 2) **Multiple Bus Structure:** Systems that contain multiple buses achieve more concurrency in operations. Two or more transfers can be carried out at the same time.

Performance – Processor Clock

- Processor circuits are controlled by a timing signal called a Clock.
- The clock defines regular time intervals called Clock Cycles.
- To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps such that each step can be completed in one clock cycle.

Let P = Length of one clock cycle R = Clock rate.
Relation between P and R is given by $R = (1/P)$, R is measured in cycles per second. Cycles per second is also called Hertz (Hz).



Basic Performance Equation

Let

T = Processor time required to execute a program.

N = Actual number of instruction executions.

S = Average number of basic steps needed to execute one machine instruction.

R = Clock rate in cycles per second.

The program execution time is given by $T = (N * S) / R$ -----(1)

Equ1 is referred to as the basic performance equation.

Clock Rate

There are 2 possibilities for increasing the clock rate R :

- 1) Improving the IC technology makes logic-circuits faster. This reduces the time needed to compute a basic step. (IC integrated circuits). This allows the clock period P to be reduced and the clock rate R to be increased.
- 2) Reducing the amount of processing done in one basic step also reduces the clock period P . In presence of a cache, the percentage of accesses to the main memory is small. Hence, much of performance-gain expected from the use of faster technology can be realized. The value of T will be reduced by same factor as R is increased, S & N are not affected.

Performance Measurement

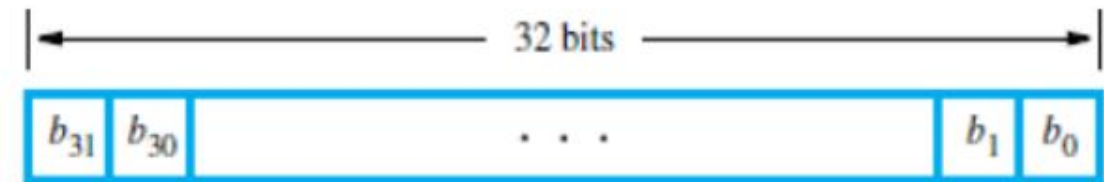
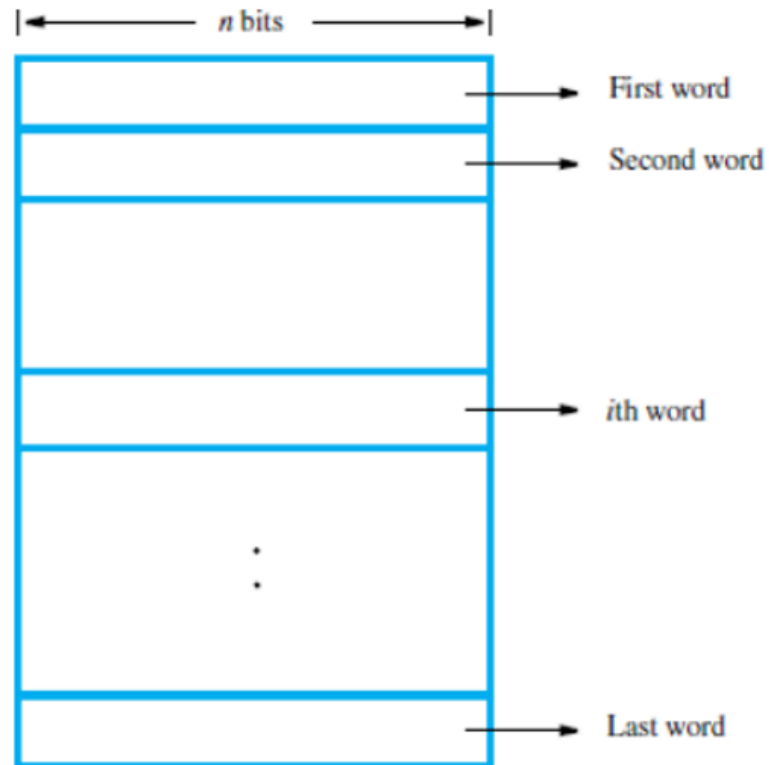
- Benchmark refers to standard task used to measure how well a processor operates.
- The Performance Measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC System Performance Evaluation Corporation).
- SPEC Rating is given by $\text{SPEC rating} = (\text{Running time on the reference computer}) / (\text{Running time on the computer under test})$

$$\text{SPEC rating} = \frac{\text{running time on reference computer}}{\text{running time on computer undertest}}$$

$$\text{SPEC rating} = (\prod_{i=1}^n \text{SPEC}_i)^{1/n}$$

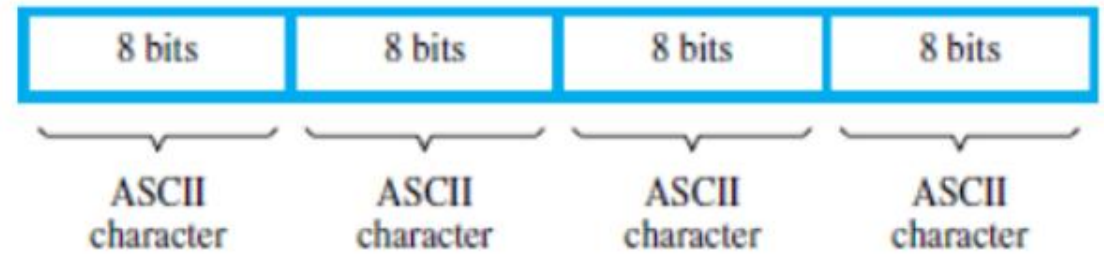
Chapter 2: Machine Instructions and Programs

Memory Location and Addresses



Sign bit: $b_{31} = 0$ for positive numbers
 $b_{31} = 1$ for negative numbers

(a) A signed integer

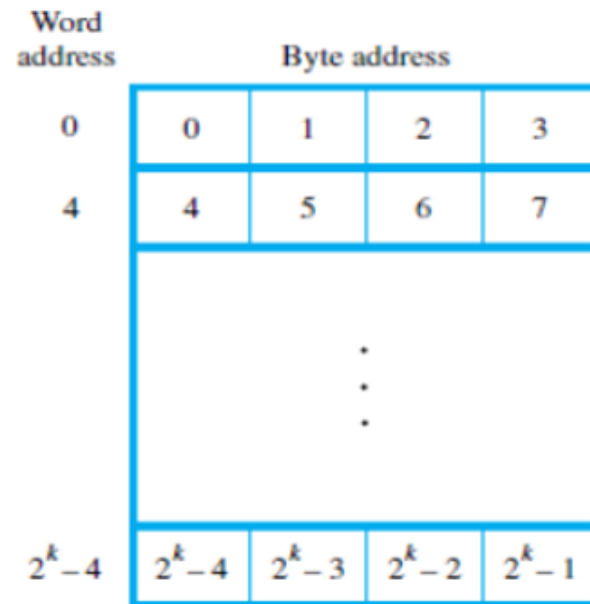


(b) Four characters

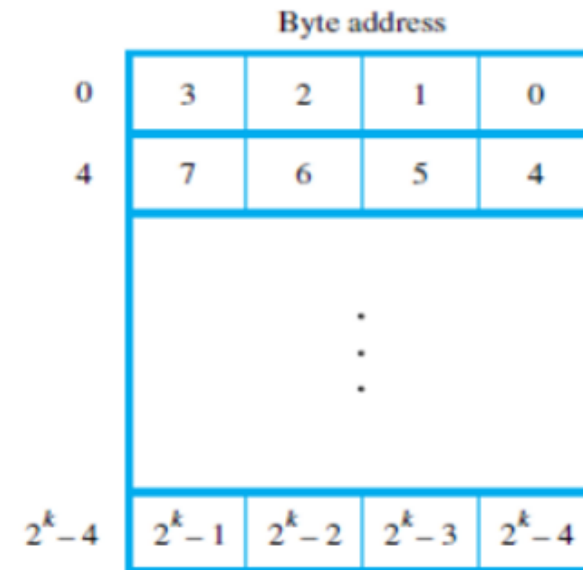
Byte-Addressability

There are two ways in which byte-addresses are arranged.

- 1) Big-Endian: Lower byte-addresses are used for the more significant bytes of the word.
- 2) Little-Endian: Lower byte-addresses are used for the less significant bytes of the word



(a) Big-endian assignment



(b) Little-endian assignment

Memory Operations

Two memory operations are: 1) Load (Read/Fetch) & 2) Store (Write).

The Load operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged. Steps for Load operation:

- 1) Processor sends the address of the desired location to the memory.
- 2) Processor issues 'read' signal to memory to fetch the data.
- 3) Memory reads the data stored at that address.
- 4) Memory sends the read data to the processor.

The Store operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location. Steps for Store operation is:

- 1) Processor sends the address of the memory-location where it wants to store data.
- 2) Processor issues 'write' signal to memory to store the data.
- 3) Content of register (MDR) is written into the specified memory-location.

Instruction and Instruction sequencing

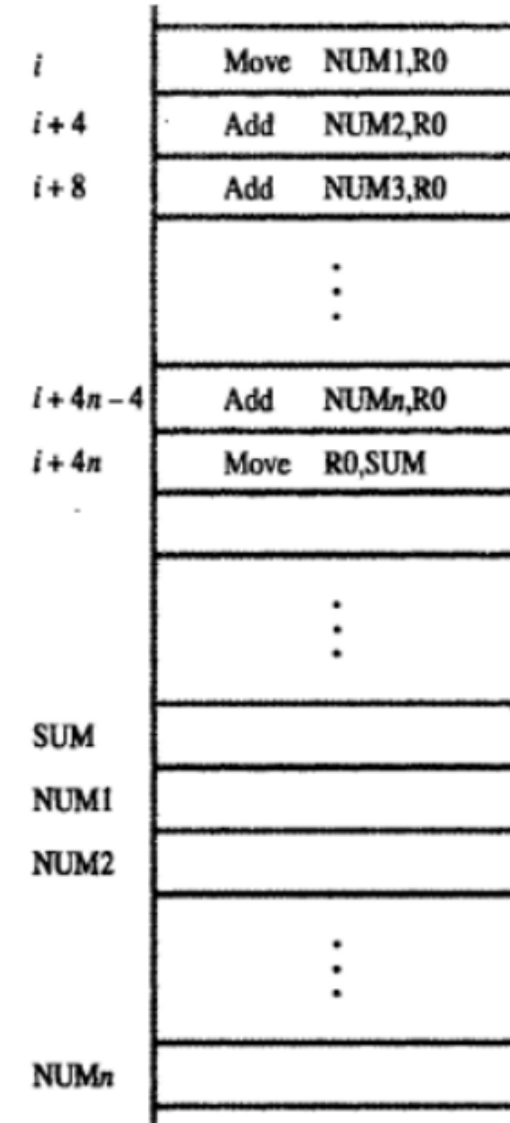
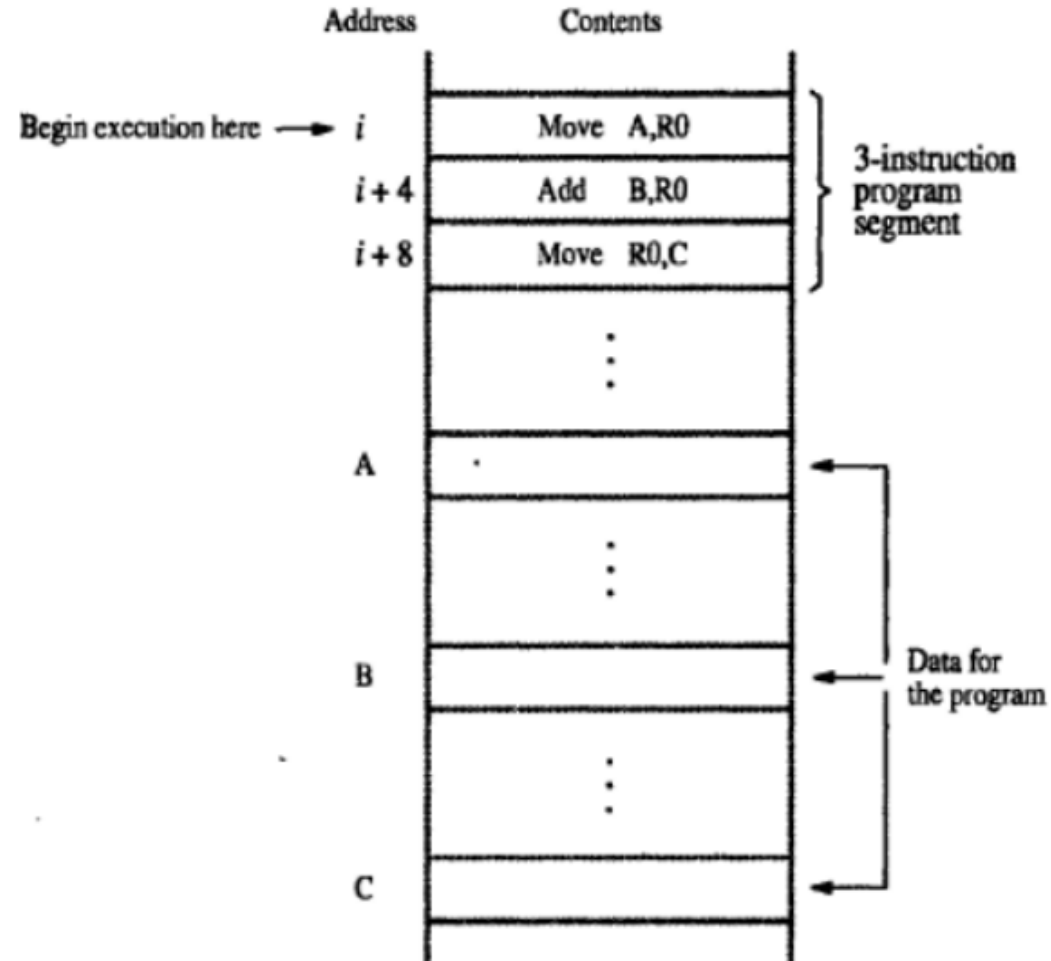
Register Transfer Notation (RTN)

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	$R1 \leftarrow [LOC]$	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1 ,R2	$[R3] \leftarrow [R1] + [R2]$	Add the contents of register R1 & R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	$R1 \leftarrow DATAIN$	Contents of I/O register DATAIN are transferred into register R1.

Assembly Language Notation

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

Instruction Execution & Straight-Line Sequencing



Addressing Modes

1. Register Mode: The operand is the contents of a register.

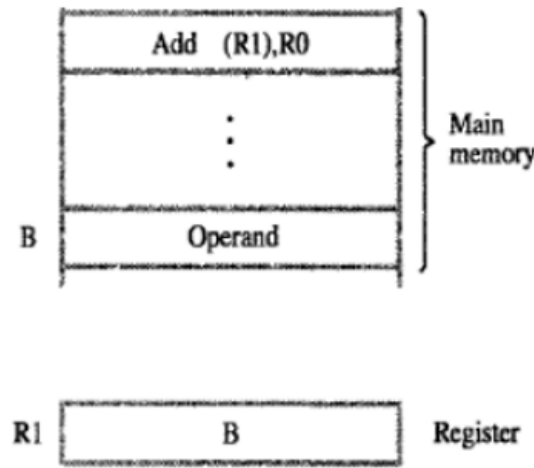
Example: Move R1, R2; Copy content of register R1 into register R2.

2. Absolute (Direct) Mode: The operand is in a memory-location.

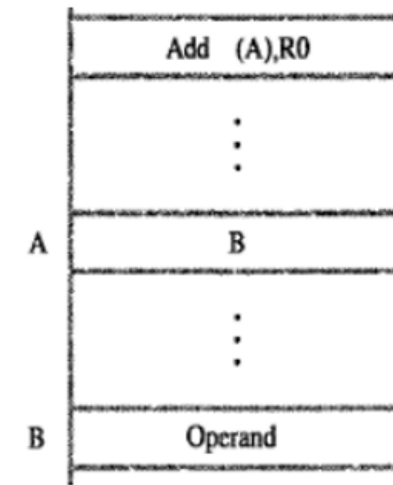
Example: Move LOC, R2; Copy content of memory-location LOC into register R2.

3. Indirect Mode: The EA of the operand is the contents of a register (or memory-location).

Example: Add (R1),R0;



(a) Through a general-purpose register



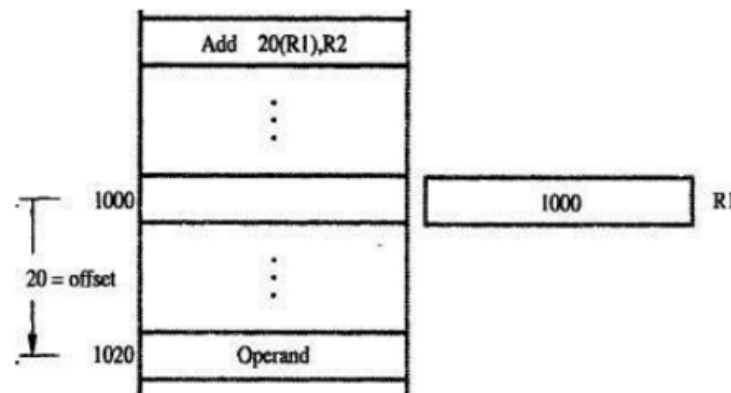
(b) Through a memory location

Addressing Modes

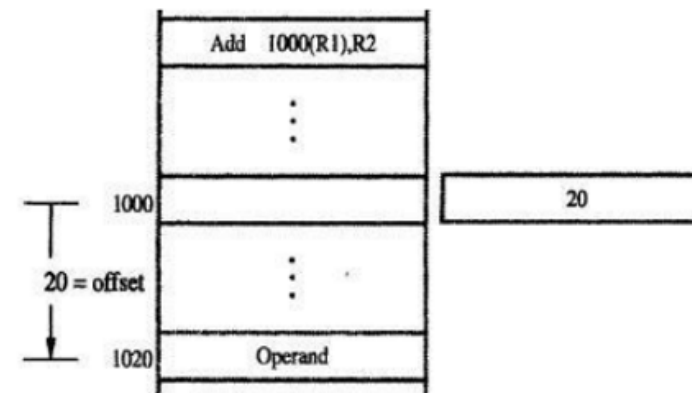
4. Index mode: the operation is indicated as $X(R_i)$ where X =the constant value which defines an offset (also called a displacement). R_i =the name of the index registers which contains address of a new location.

Example: Add 20(R_1), R_2

5. Base with Index Mode: Another version of the Index mode uses 2 registers which can be denoted as (R_i, R_j). Here, a second register may be used to contain the offset X . The second register is usually called the base register. The effective address of the operand is given by $EA = [R_i] + [R_j]$.



(a) Offset is given as a constant



(b) Offset is in the index register

Addressing Modes

6. Base with Index & Offset Mode: Another version of the Index mode uses 2 registers plus a constant, which can be denoted as $X(R_i, R_j)$. The effective address of the operand is given by $EA = X + [R_i] + [R_j]$.
7. Relative Mode: This is similar to index-mode with one difference: The effective-address is determined using the PC in place of the general-purpose register R_i . The operation is indicated as $X(PC)$. $X(PC)$ denotes an effective address of the operand which is X locations above or below the current contents of PC.

Module-4

Chapter 1: Input/Output Organization

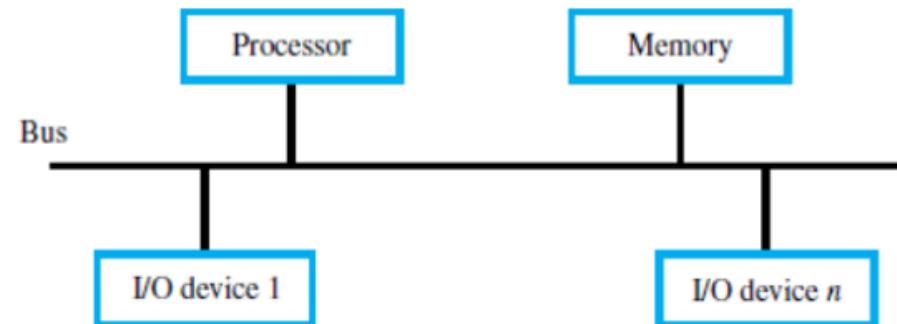
Accessing I/O Devices

There are 2 ways to deal with I/O-devices:

1) Memory-Mapped I/O: Memory and I/O-devices share a common address-space. Any data-transfer instruction (like Move, Load) can be used to exchange information.

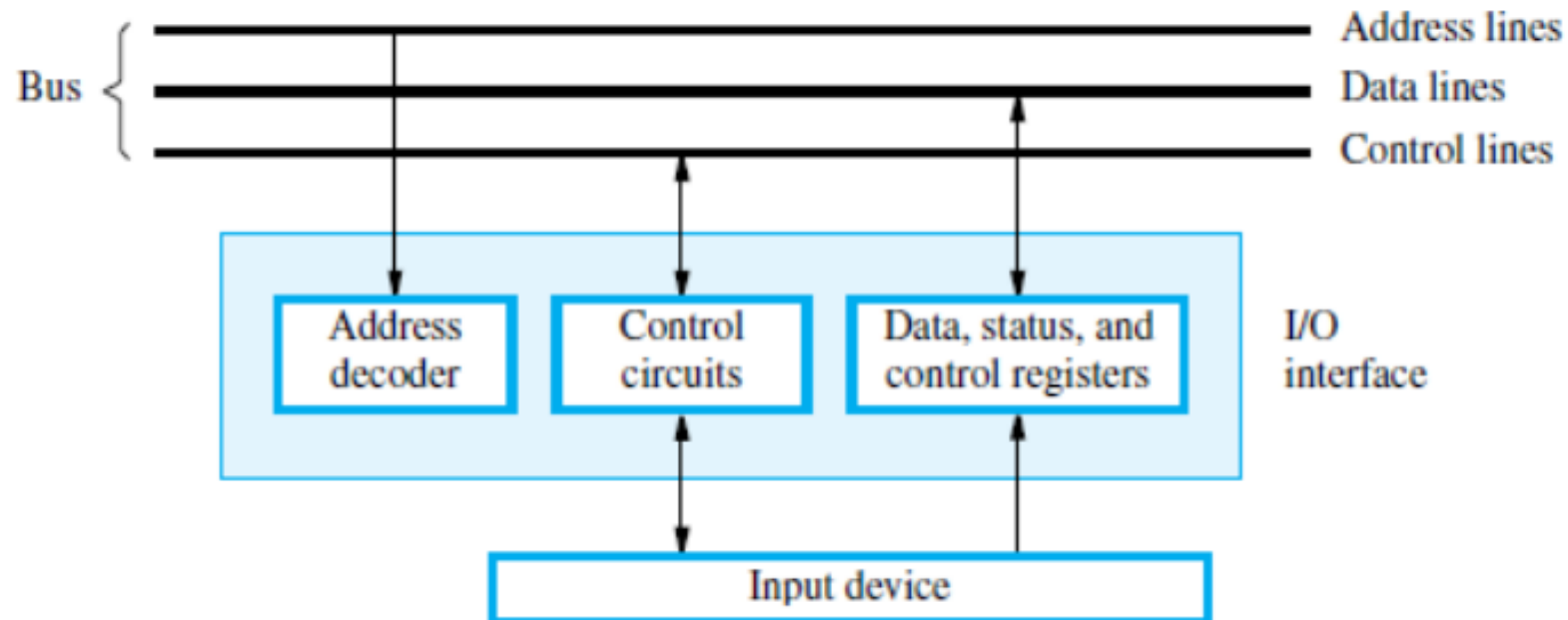
For example, Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0. Here, DATAIN address of the input-buffer of the keyboard.

2) I/O-Mapped I/O: Memory and I/O address-spaces are different. A special instruction named IN and OUT are used for data-transfer. Ex: Processor in Intel family. Advantage of separate I/O space: I/O-devices deal with fewer address-lines.



I/O Interface for an Input Device

- 1) Address Decoder: enables the device to recognize its address when this address appears on the address-lines.
- 2) Status Register: contains information relevant to operation of I/O-device.
- 3) Data Register: holds data being transferred to or from processor.

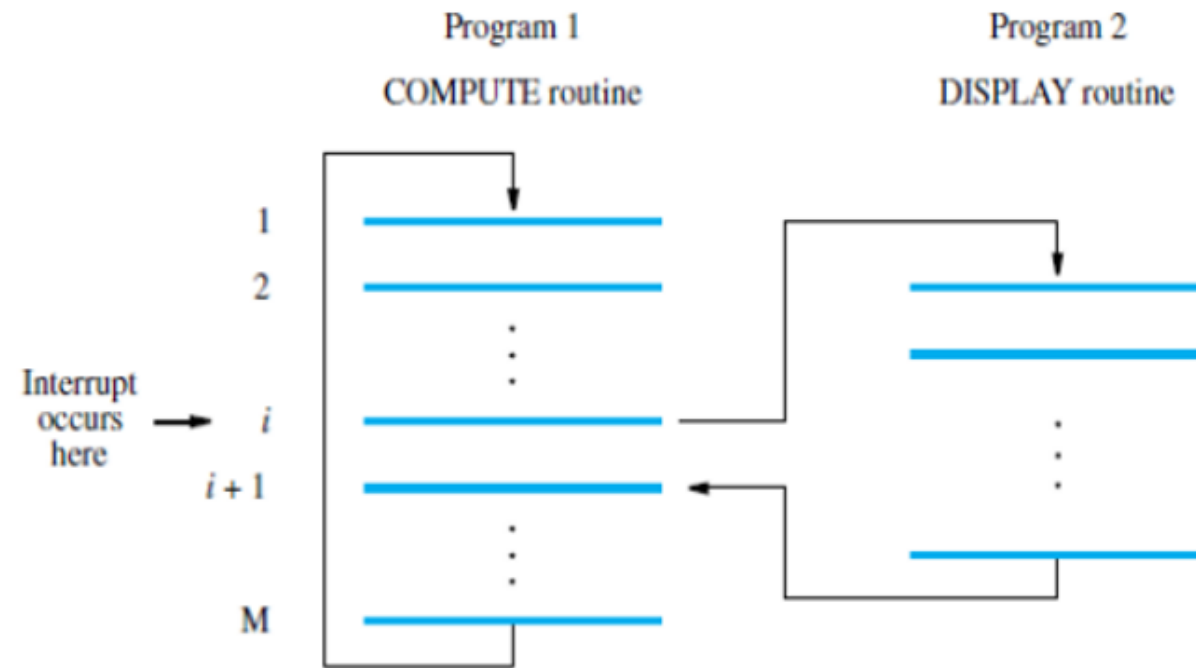


Mechanisms Used for Interfacing I/O-Devices

- 1) Program Controlled I/O: Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/O device. Main drawback: The processor wastes time in checking status of device before actual data-transfer takes place.
- 2) Interrupt I/O: I/O-device initiates the action instead of the processor. I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation. Like this, required synchronization is done between processor & I/O device.
- 3) Direct Memory Access (DMA): Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor. DMA is a technique used for high speed I/O-device.

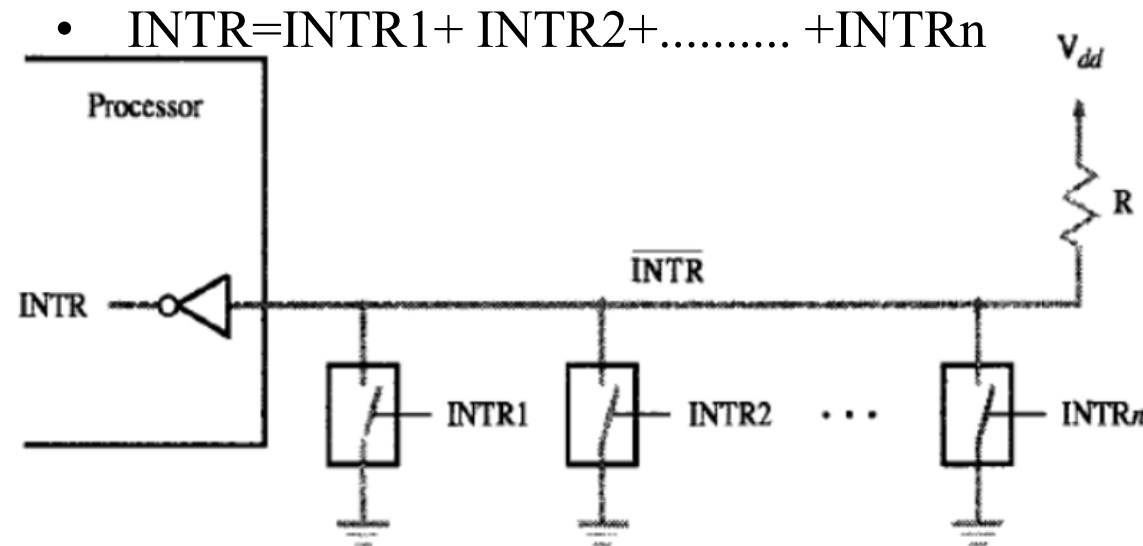
Interrupts

- There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
- A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready.
- Interrupt-signal is sent on the interrupt-request line.
- The processor can be performing its own task without the need to continuously check the I/O-device.
- The routine executed in response to an interrupt-request is called ISR.
- The processor must inform the device that its request has been recognized by sending INTA signal.



Interrupt Hardware

- Most computers have several I/O devices that can request an interrupt.
- A single interrupt-request (IR) line may be used to serve n devices. All devices are connected to IR line via switches to ground. To request an interrupt, a device closes its associated switch.
- Thus, if all IR signals are inactive, the voltage on the IR line will be equal to V_{dd}. When a device requests an interrupt, the voltage on the line drops to 0. This causes the INTR received by the processor to go to 1. The value of INTR is the logical OR of the requests from individual devices.



Enabling and Disabling Interrupts

All computers fundamentally should be able to enable and disable interrupts as desired. Processors generally provide the ability to enable and disable such interrupts as desired. The problem of infinite loop occurs due to successive interruptions of active INTR signals. There are 3 mechanisms to solve problem of infinite loop:

- 1) Processor should ignore the interrupts until execution of first instruction of the ISR by the processor hardware. By using an Interrupt-disable instruction as the first instruction in the interrupt-service routine, the programmer can ensure that no further interruptions will occur until an Interrupt-enable instruction is executed.
- 2) Processor should automatically disable interrupts before starting the execution of the ISR. The second option, which is suitable for a simple processor with only one interrupt request line, is to have the processor automatically disable interrupts before starting the execution of the interrupt-service routine.
- 3) In the third option, the processor has a special interrupt-request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line is said to be edge triggered.

Handling Multiple Devices

The two methods to handle multiple devices are: Polling and Vectored Interrupts

Polling

Information needed to determine whether device is requesting interrupt is available in status-register.

Following condition-codes are used:

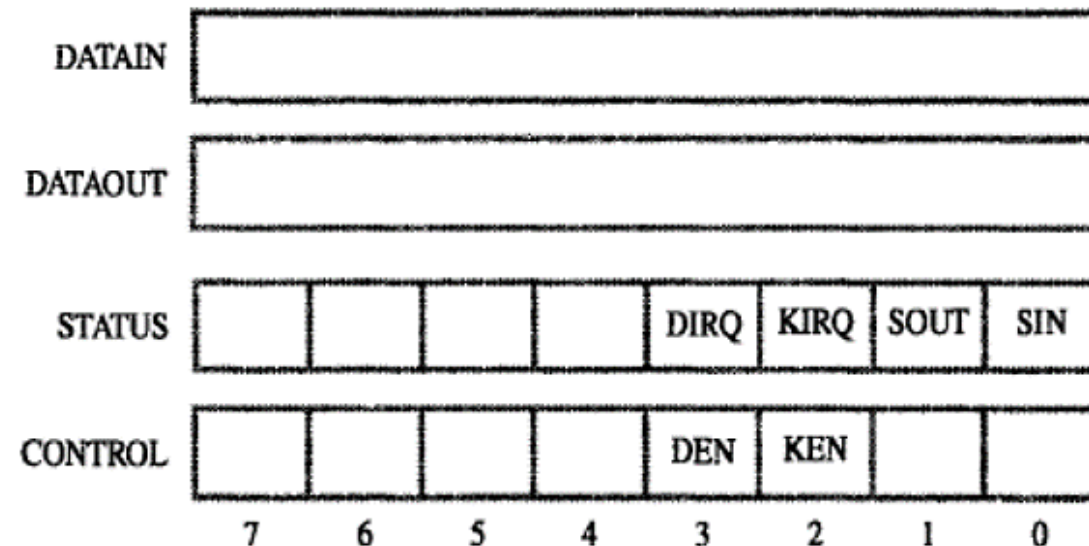
DIRQ Interrupt-request for display.

KIRQ Interrupt-request for keyboard.

KEN keyboard enable.

DEN Display Enable.

SIN, SOUT status flags.



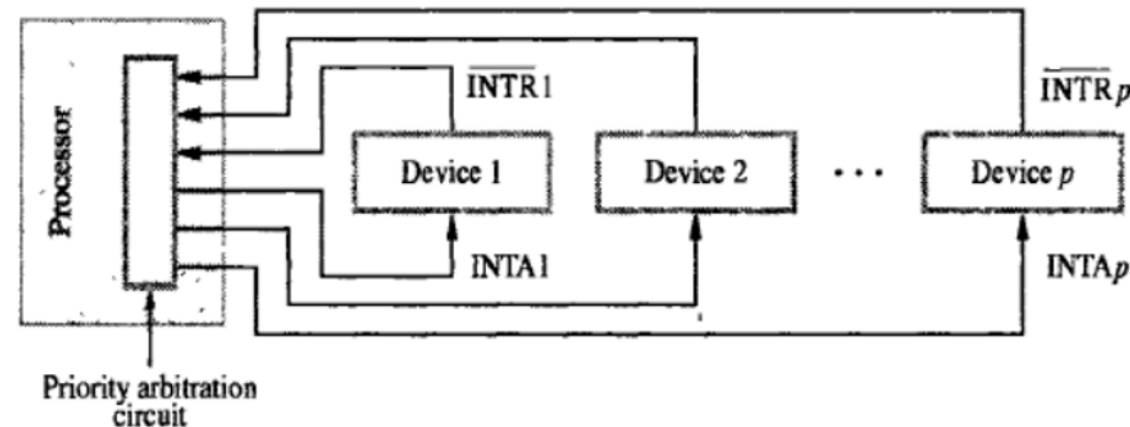
Handling Multiple Devices

Vectored Interrupts

- A device requesting an interrupt identifies itself by sending a special code to processor over bus. Then, the processor starts executing the ISR.
- The special-code indicates starting-address of ISR. The special-code length ranges from 4 to 8 bits. The location pointed to by the interrupting-device is used to store the starting address to ISR.
- The starting address to ISR is called the interrupt vector. Processor loads interrupt-vector into PC & executes appropriate ISR.
- When processor is ready to receive interrupt-vector code, it activates INTA line. Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal.
- The interrupt vector also includes a new value for the Processor Status Register.

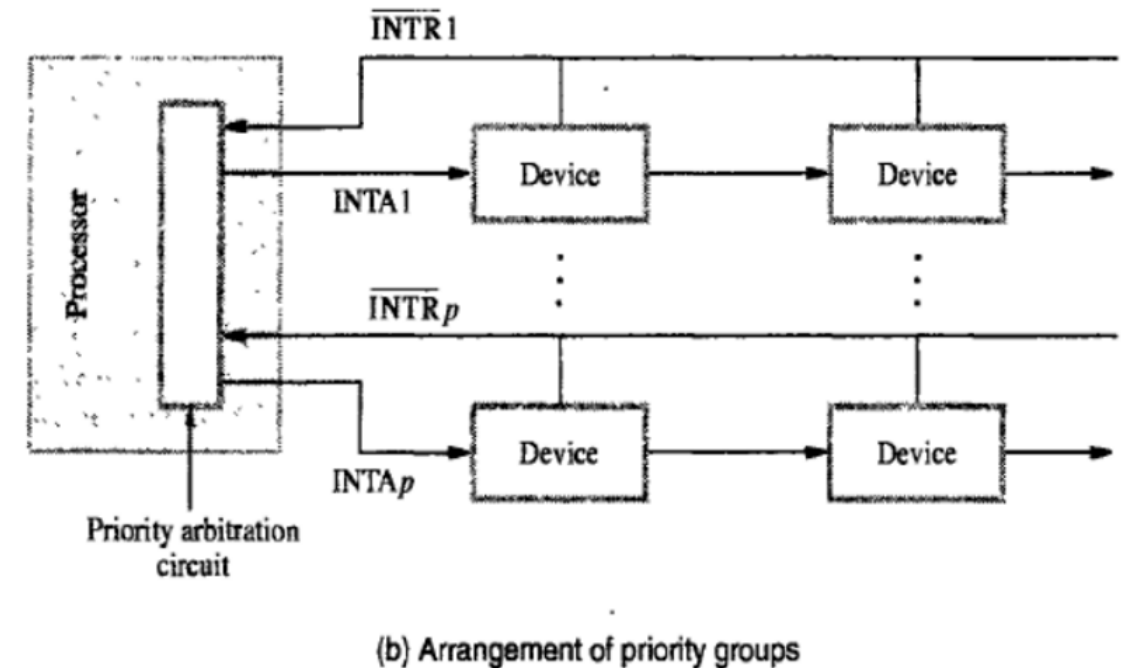
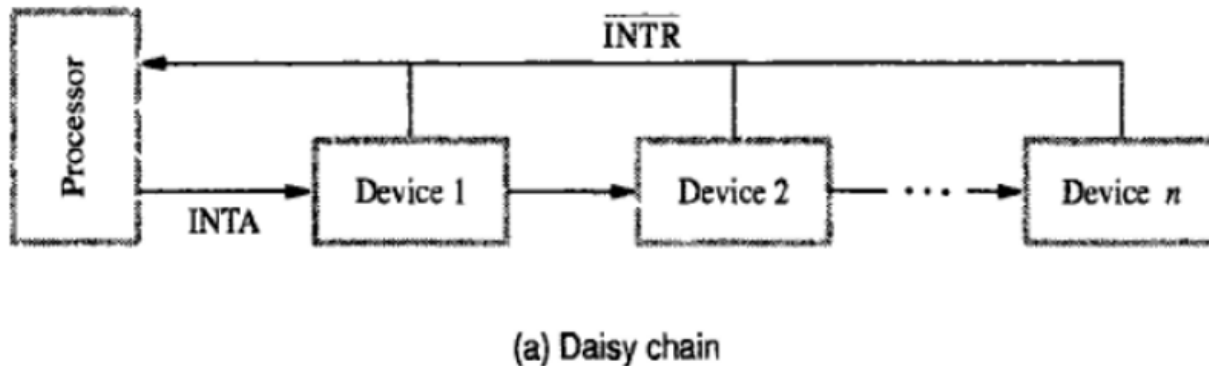
Interrupt Nesting

- Interrupt-service routines are typically short, and the delay they may cause is acceptable for most simple devices. A multiple-priority scheme is implemented by using separate INTR & INTA lines for each device.
- Each INTR line is assigned a different priority-level. Priority-level of processor is the priority of program that is currently being executed.
- Processor accepts interrupts only from devices that have higher priority than its own. At the time of execution of ISR for some device, priority of processor is raised to that of the device. Thus, interrupts from devices at the same level of priority or lower are disabled.



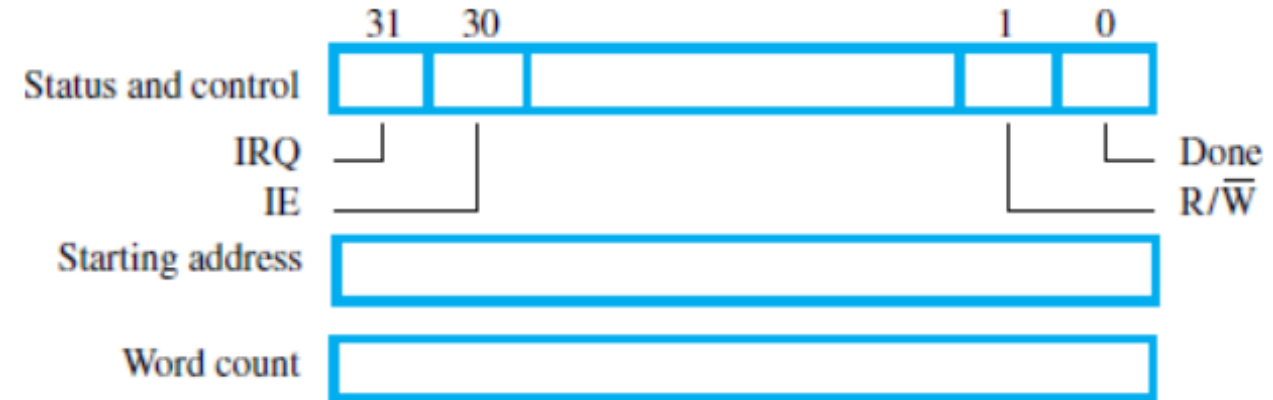
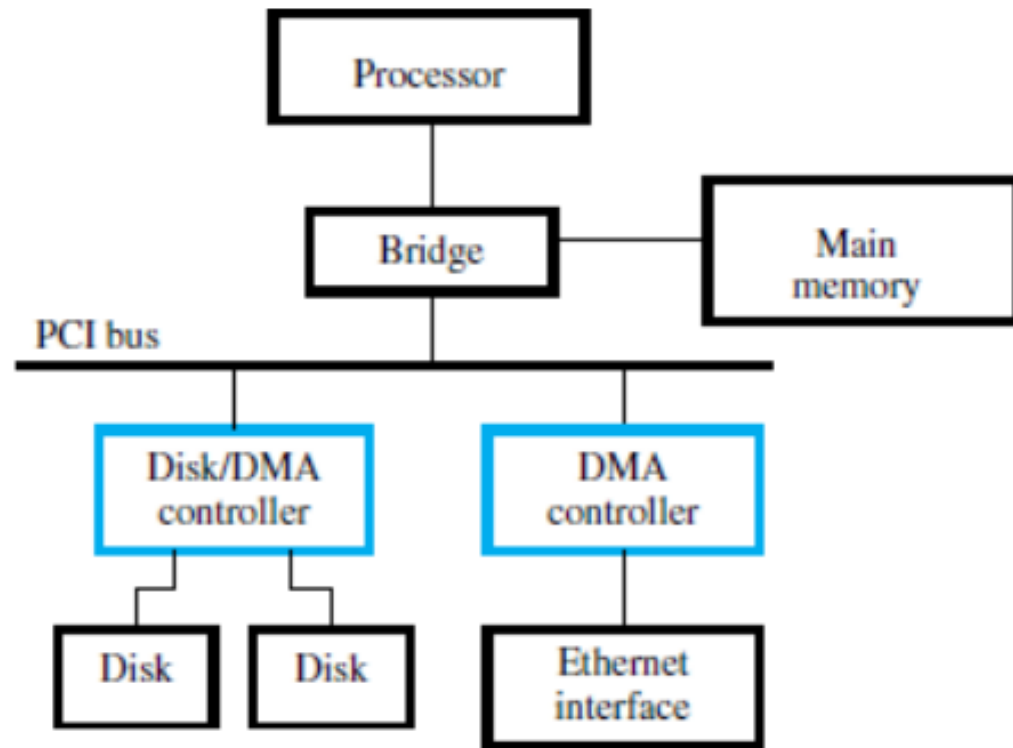
Simultaneous Requests

The processor must have some mechanisms to decide which request to service when simultaneous requests arrive. A widely used scheme is to connect the devices to form a daisy chain, INTR line is common to all devices. The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices. INTA signal propagates serially through devices.



Direct Memory Access

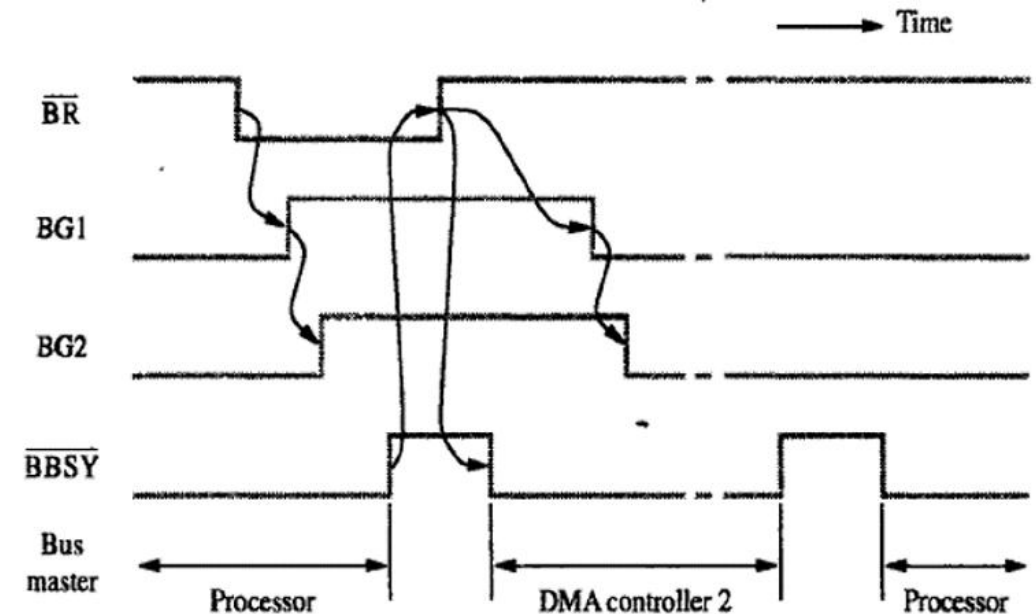
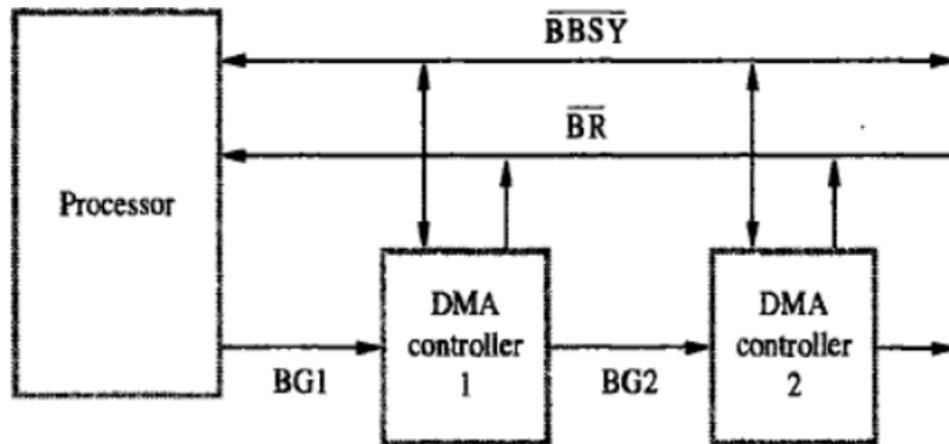
The transfer of a block of data directly between an external device & main-memory without continuous involvement by processor is called DMA.



Bus Arbitration

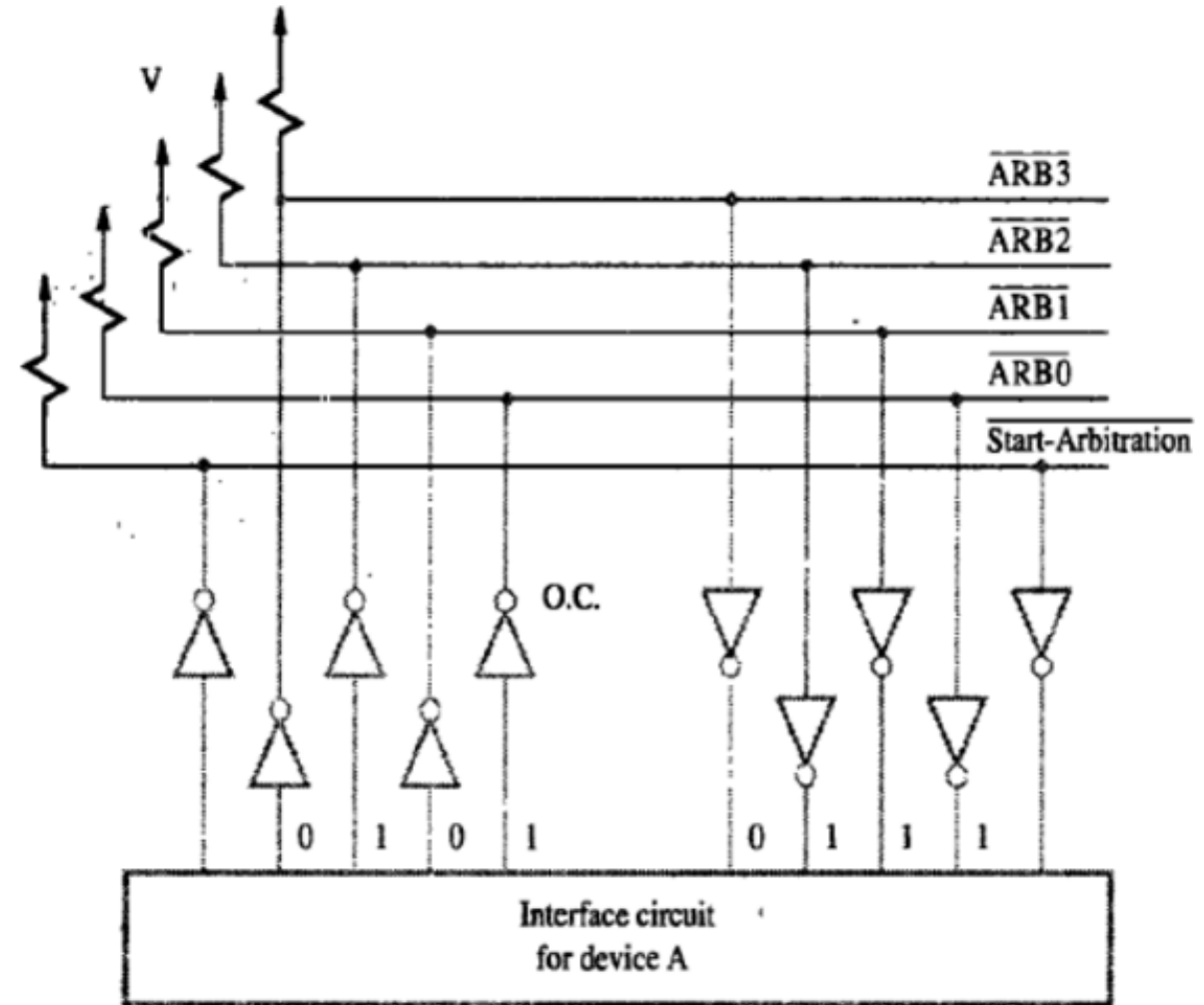
The device that is allowed to initiate data-transfers on bus at any given time is called bus-master. There can be only one bus-master at any given time. Bus Arbitration is the process by which next device to become the bus-master is selected & bus mastership is transferred to that device.

The two approaches are: 1) Centralized Arbitration: A single bus-arbiter performs the required arbitration. 2) Distributed Arbitration: All devices participate in selection of next bus-master.



Bus Arbitration

Distributed Arbitration

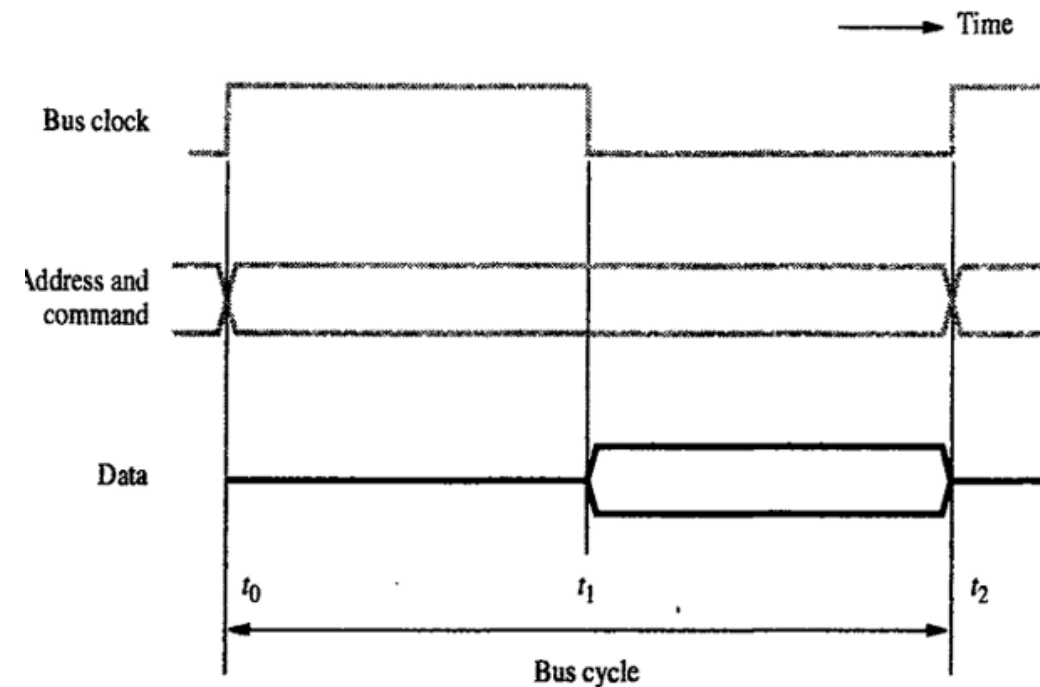


Buses

The processor, main memory, and I/O devices can be interconnected by means of a common bus whose primary function is to provide a communications path for the transfer of data. The bus includes the lines needed to support interrupts and arbitration.

1. Synchronous Bus

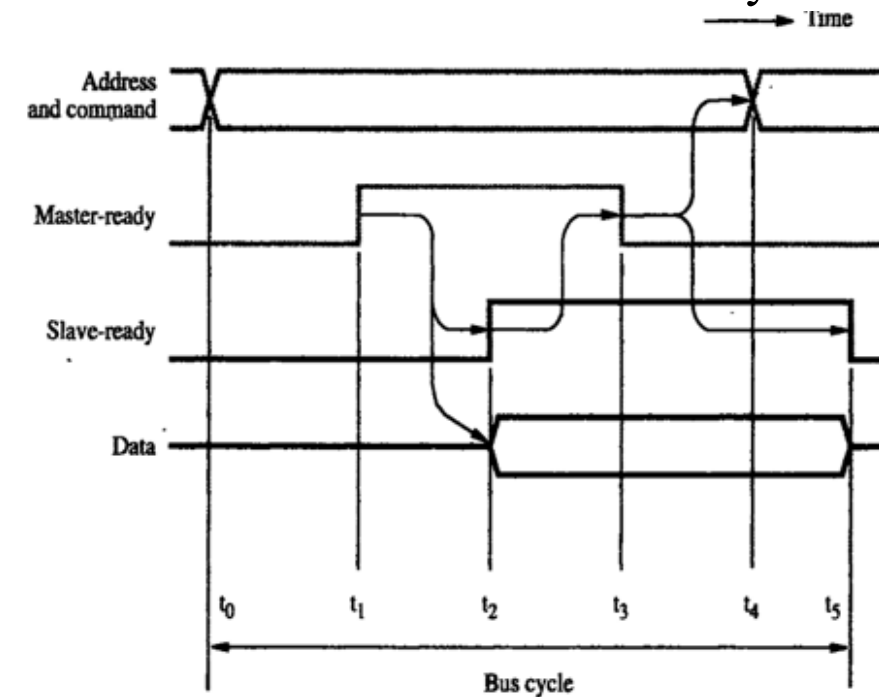
In a synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place.



Buses

2. Asynchronous Bus

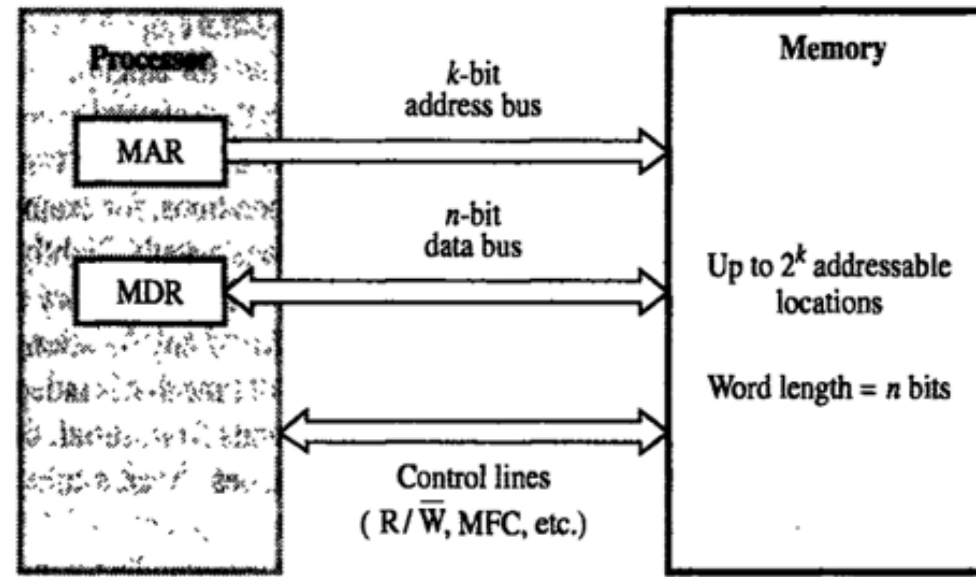
An alternative scheme for controlling data transfers on the bus is based on the use of a handshake between the master and the slave. The concept of a handshake is a generalization of the idea of the Slave-ready signal in Figure above. The common clock is replaced by two timing control lines, Master-ready and Slave-ready. The first is asserted by the master to indicate that it is ready for a transaction, and the second is a response from the slave.



Chapter 2: The Memory System

Some Basic Concepts

Data transfer between the memory and the processor takes place through the use of two processor registers, usually called MAR (memory address register) and MDR (memory data register).

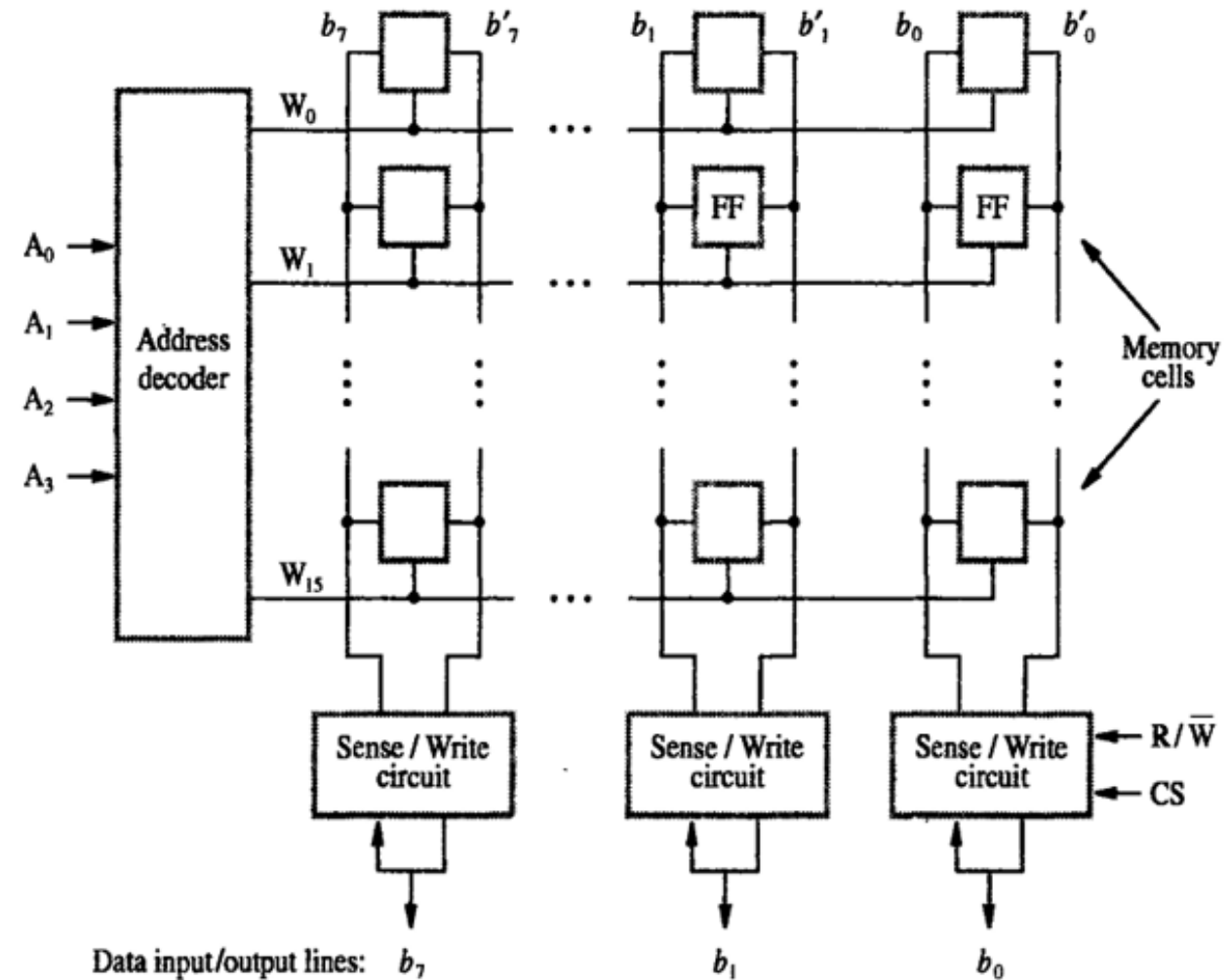


Semiconductor RAM Memories

- Semiconductor memories are available in a wide range of speeds.
- Their cycle times range from 100 ns to less than 10 ns.
- When first introduced in the late 1960s, they were much more expensive than the magnetic-core memories they replaced. Because of rapid advances in VLSI (Very Large-Scale Integration) technology, the cost of semiconductor memories has dropped dramatically.

Internal Organization of Memory Chips

Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information.



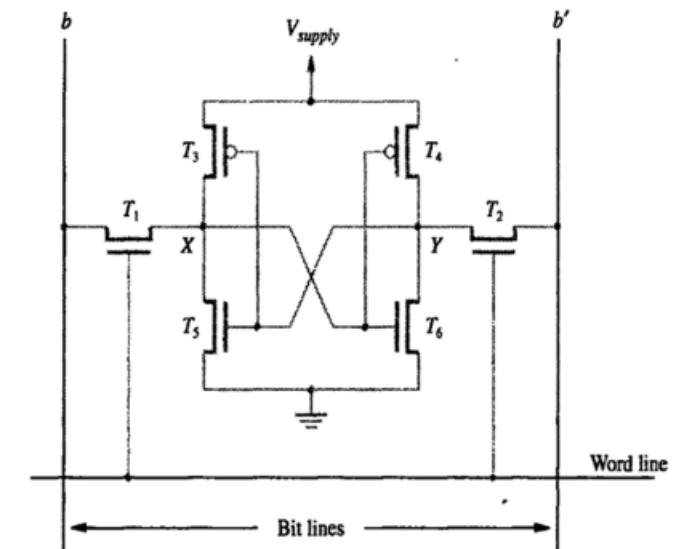
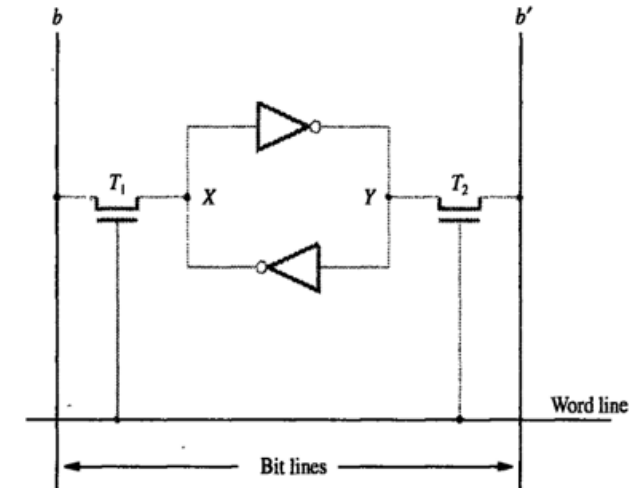
Semiconductor RAM Memories

Static Memories

Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories.

CMOS cell

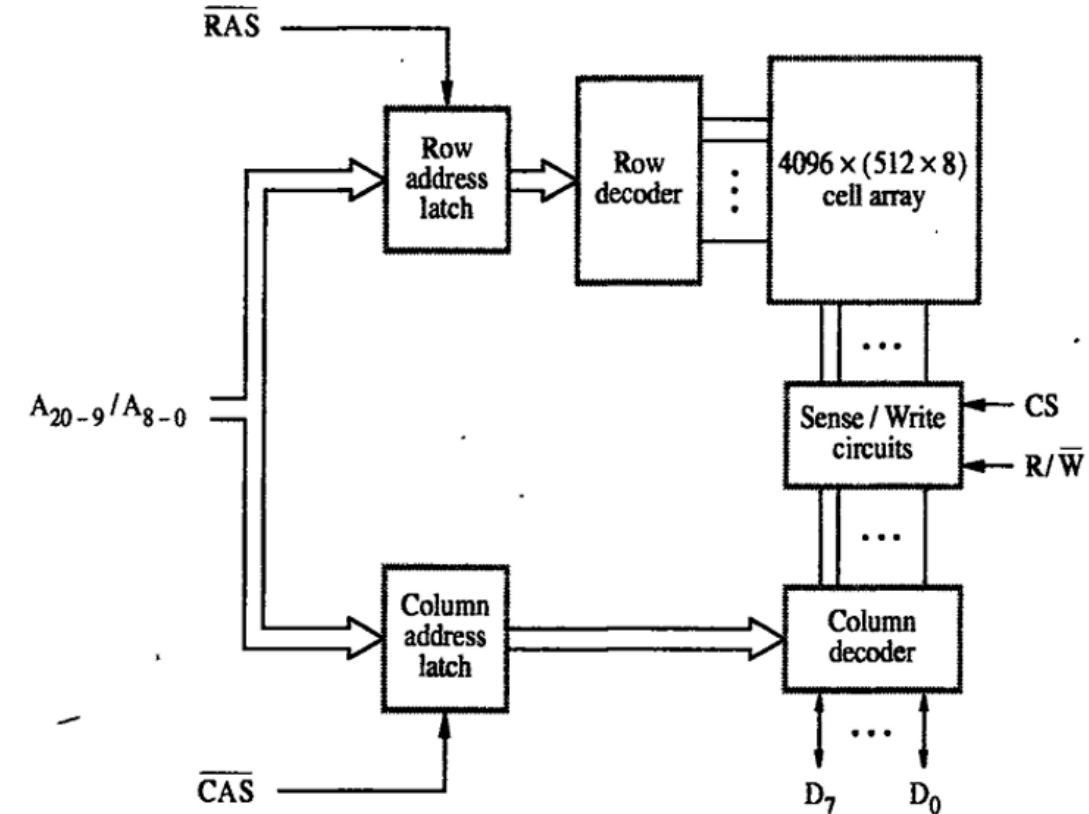
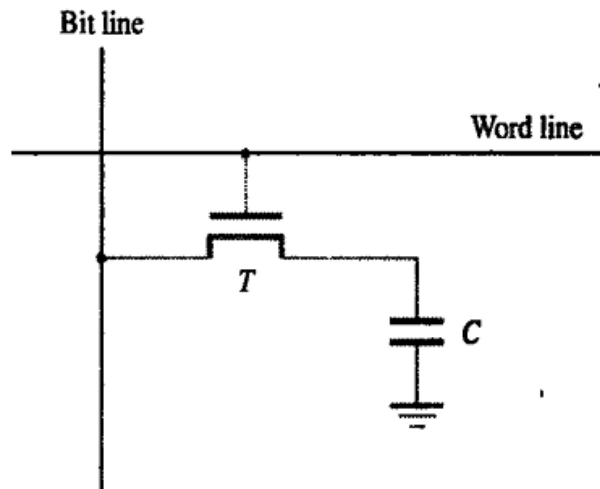
A CMOS realization of the cell is given above. Transistor pairs (T_3 , T_5) and (T_4 , T_6) form the inverters in the latch. The power supply voltage, V_{supply} , is 5 V in older CMOS SRAMs or 3.3 V in new low-voltage versions.



Semiconductor RAM Memories

Asynchronous DRAMs

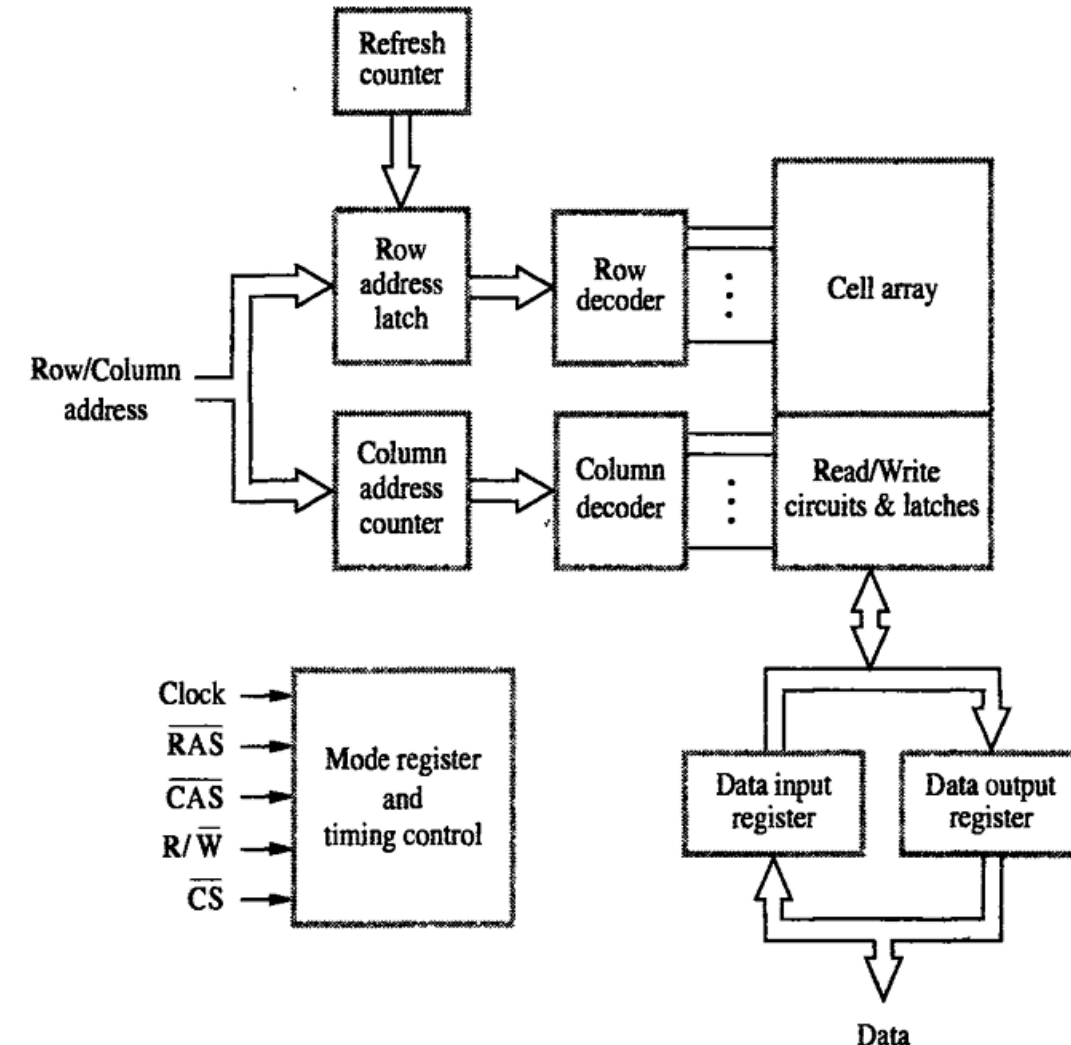
Static RAMs are fast, but they come at a high cost because their cells require several transistors. Less expensive RAMs can be implemented if simpler cells are used. However, such cells do not retain their state indefinitely; hence, they are called dynamic RAMs (DRAMs).



Semiconductor RAM Memories

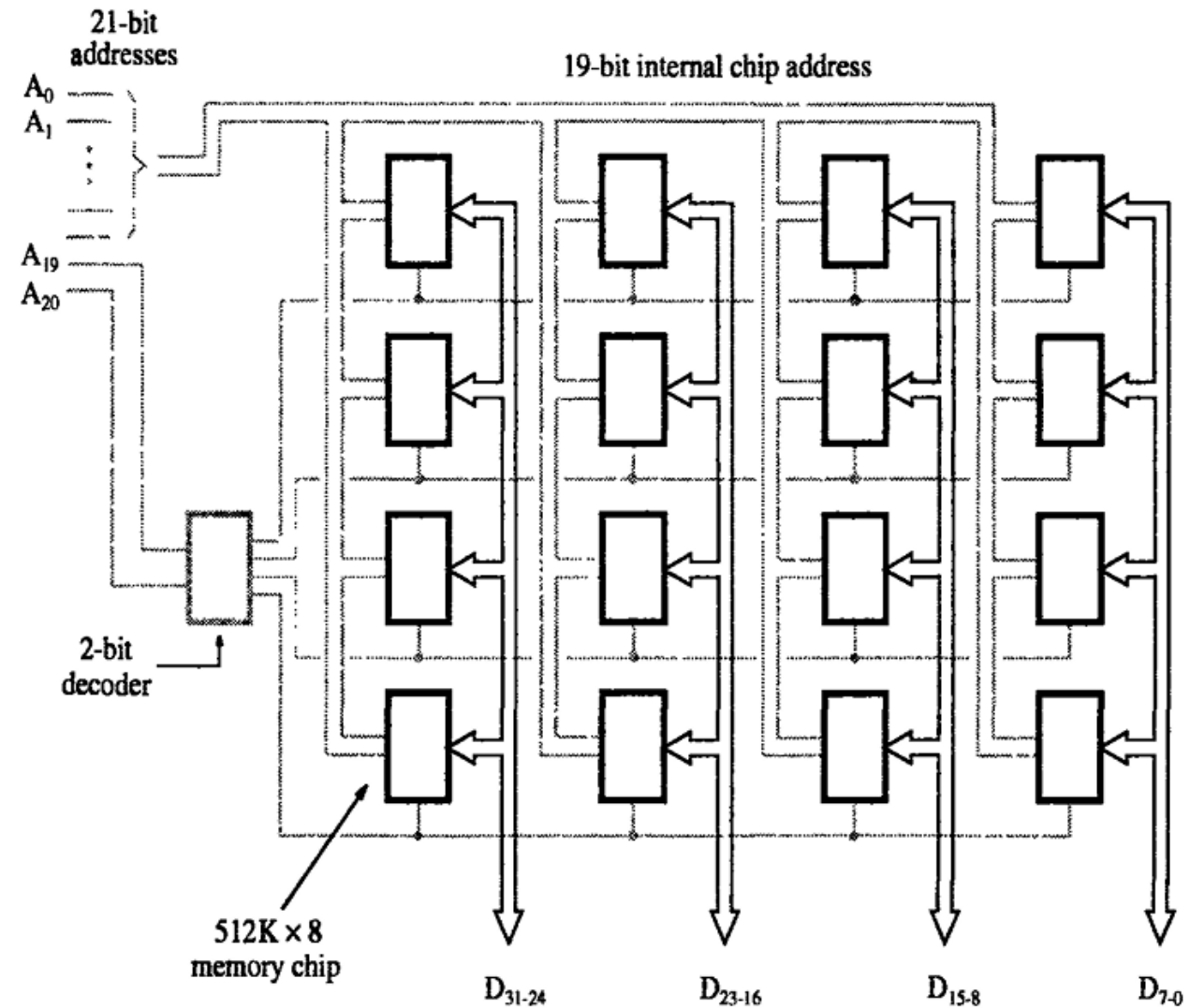
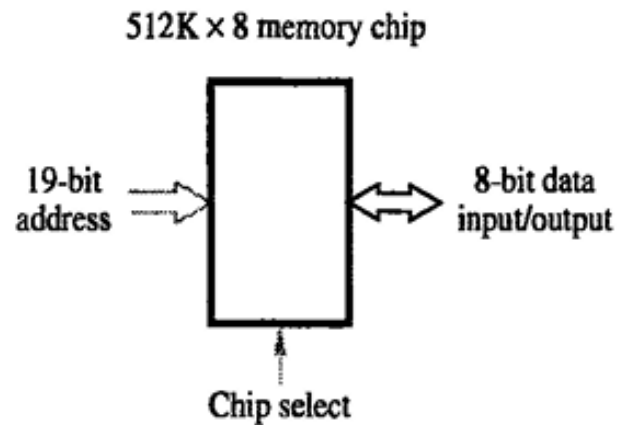
Synchronous DRAMs

More recent developments in memory technology have resulted in DRAMs whose operation is directly synchronized with a clock signal. Such memories are known as synchronous DRAMS (SDRAMs). Figure below indicates the structure of an SDRAM. The cell array is the same as in asynchronous DRAMs. The address and data connections are buffered by means of registers.



Semiconductor RAM Memories

Static Memory System

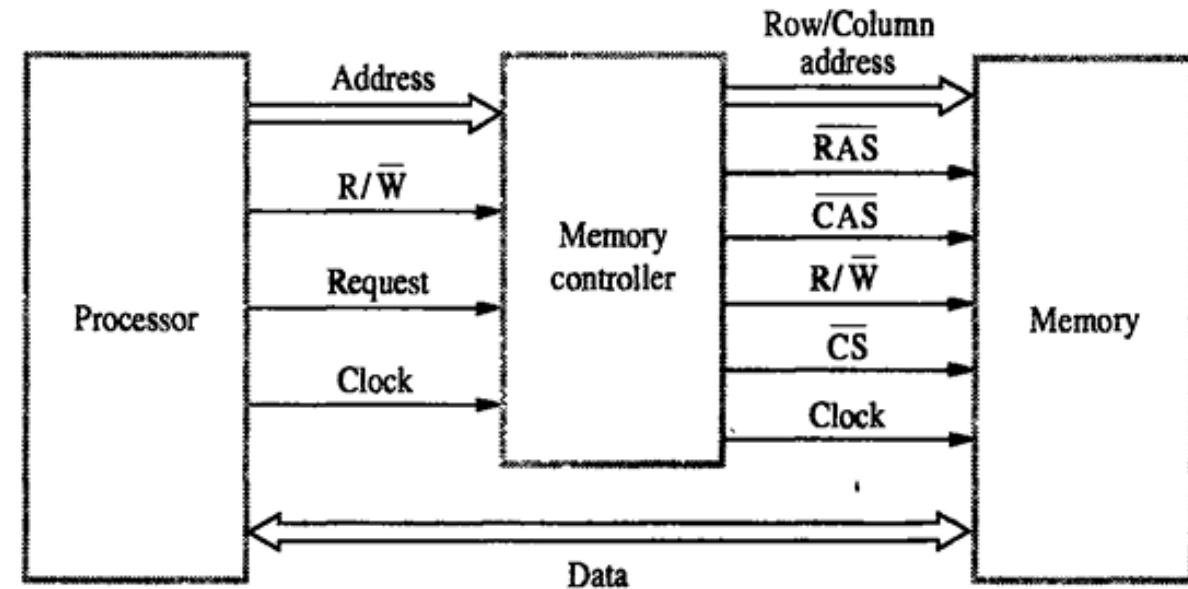


Semiconductor RAM Memories

Memory Controller

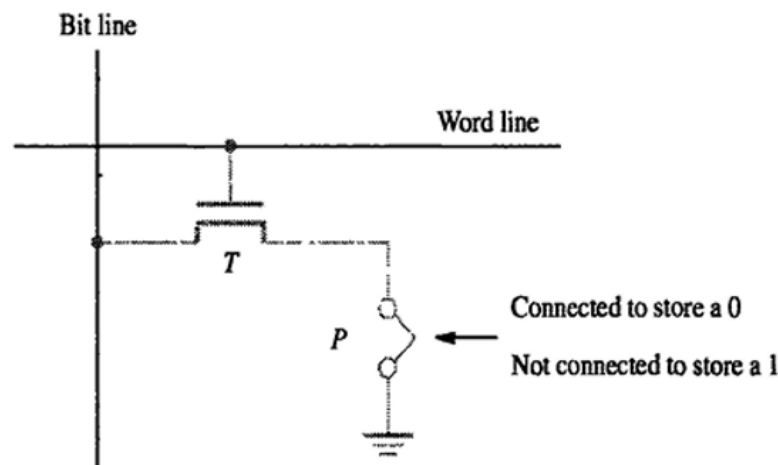
To reduce the number of pins, the dynamic memory chips use multiplexed address inputs. The address is divided into two parts. The high-order address bits, which select a row in the cell array, are provided first and latched into the memory chip under control of the RAS signal.

Then, the low-order address bits, which select a column, are provided on the same address pins and latched using the CAS signal.



Read Only Memory

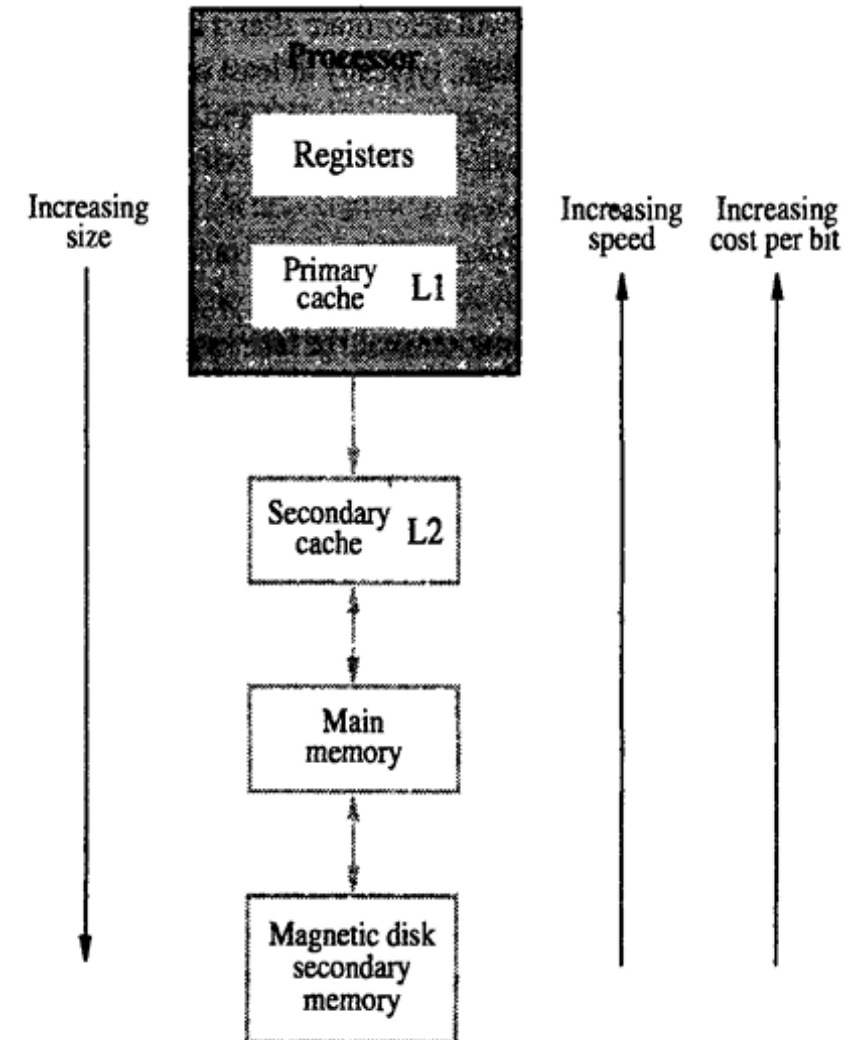
- Both SRAM and DRAM chips are volatile, which means that they lose the stored information if power is turned off. There are many applications that need memory devices which retain the stored information if power is turned off.
- A logic value 0 is stored in the cell if the transistor is connected to ground at point P; otherwise, a 1 is stored. The bit line is connected through a resistor to the power supply.
- To read the state of the cell, the word line is activated. Thus, the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage, indicating a 1.



Read Only Memory

- Programmable ROM (PROM): allow the data to be loaded by the user. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses at these locations using high-current pulses.
- Erasable, reprogrammable ROM (EPROM): Another type of ROM chip allows the stored data to be erased and new data to be loaded. It provides considerable flexibility during the development phase of digital systems. Since EPROMs are capable of retaining stored information for a long time, they can be used in place of ROMs while software is being developed. A significant disadvantage of EPROMs is that a chip must be physically removed from the circuit for reprogramming and that its entire contents are erased by the ultraviolet light.
- EEPROMs: It is possible to implement another version of erasable PROMs that can be both programmed and erased electrically. Do not have to be removed for erasure.
- A flash cell is based on a single transistor controlled by trapped charge, just like an EEPROM cell. In a flash device it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells. Prior to writing, the previous contents of the block are erased.

Speed, size and Cost of memory systems

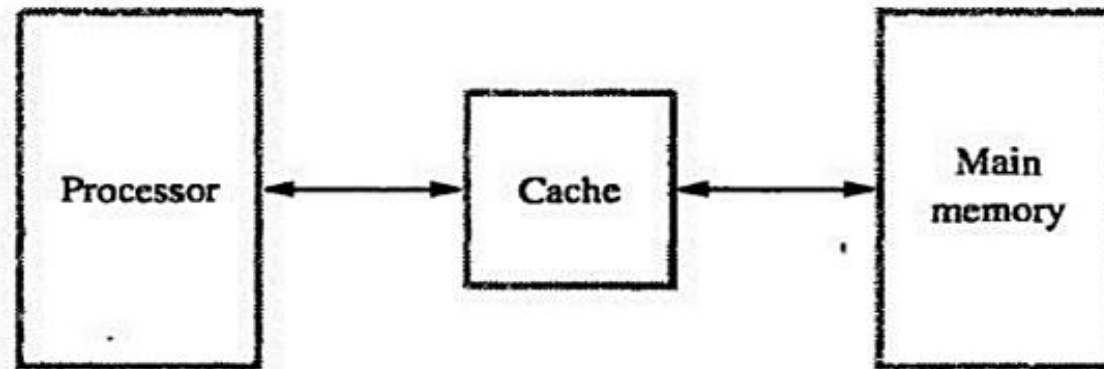


Cache Memories

Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is. Cache memory is based on the property of computer programs known as “locality of reference”.

Cache hit: Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner. If the data is in the cache it is called a Read or Write hit.

Cache miss: If the data is not present in the cache, then a Read miss or Write miss occurs.



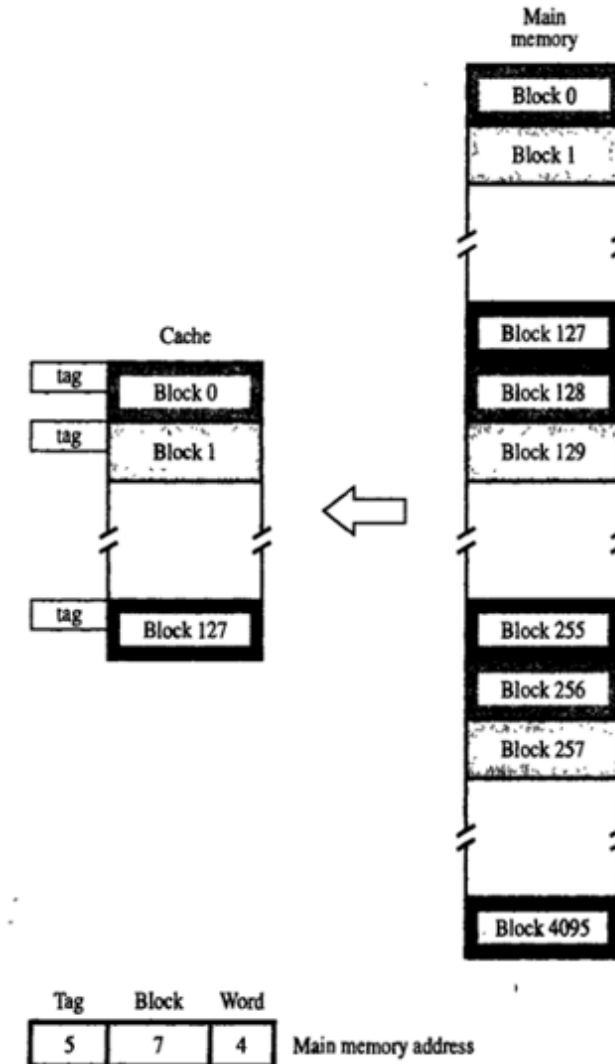
Mapping Functions

Mapping functions determine how memory blocks are placed in the cache.

Direct mapping

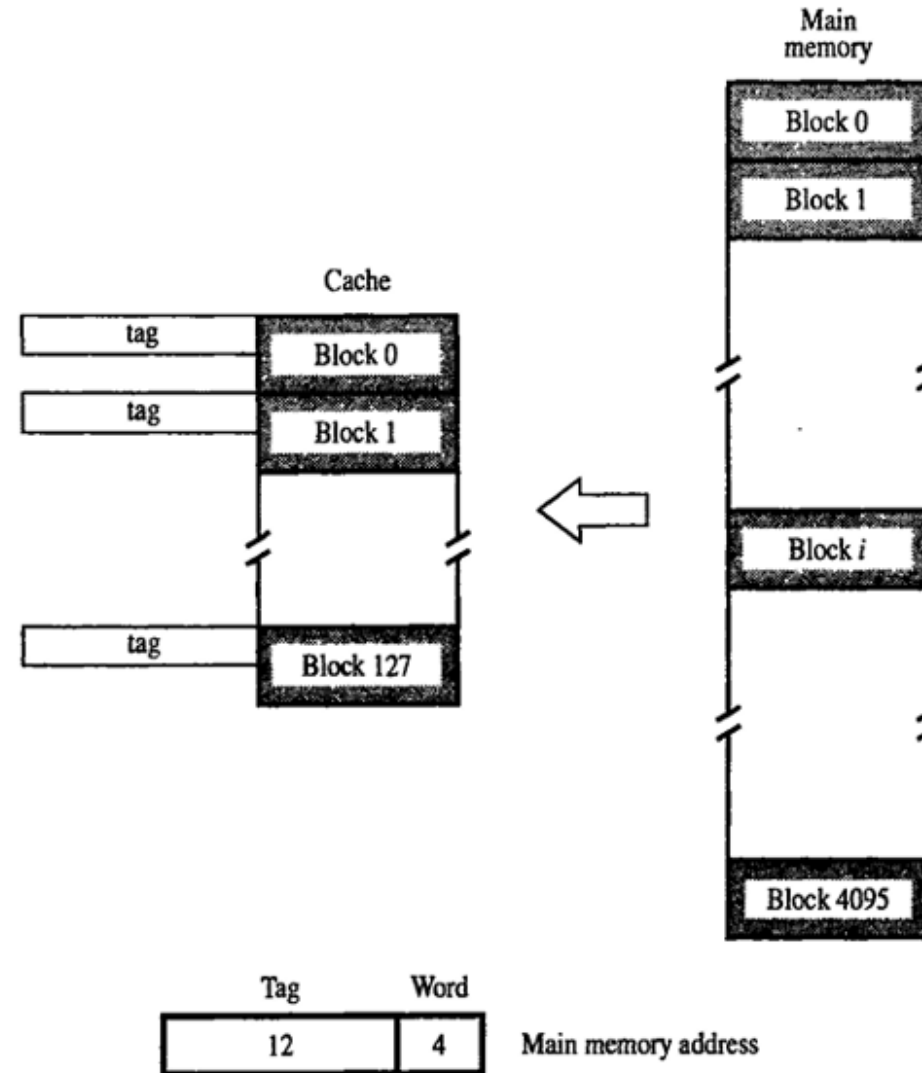
A simple processor example:

- Cache consisting of 128 blocks of 16 words each.
- Total size of cache is 2048 (2K) words.
- Main memory is addressable by a 16-bit address.
- Main memory has 64K words.
- Main memory has 4K blocks of 16 words each.



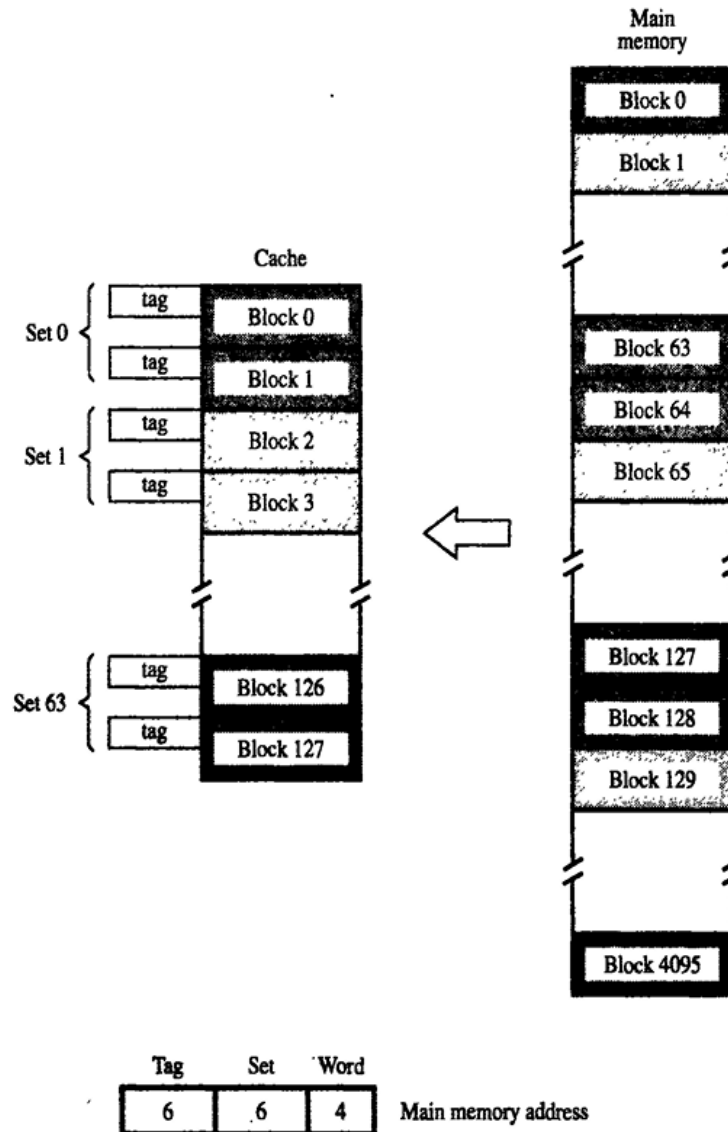
Mapping Functions

Associative mapping



Mapping Functions

Set-associative mapping



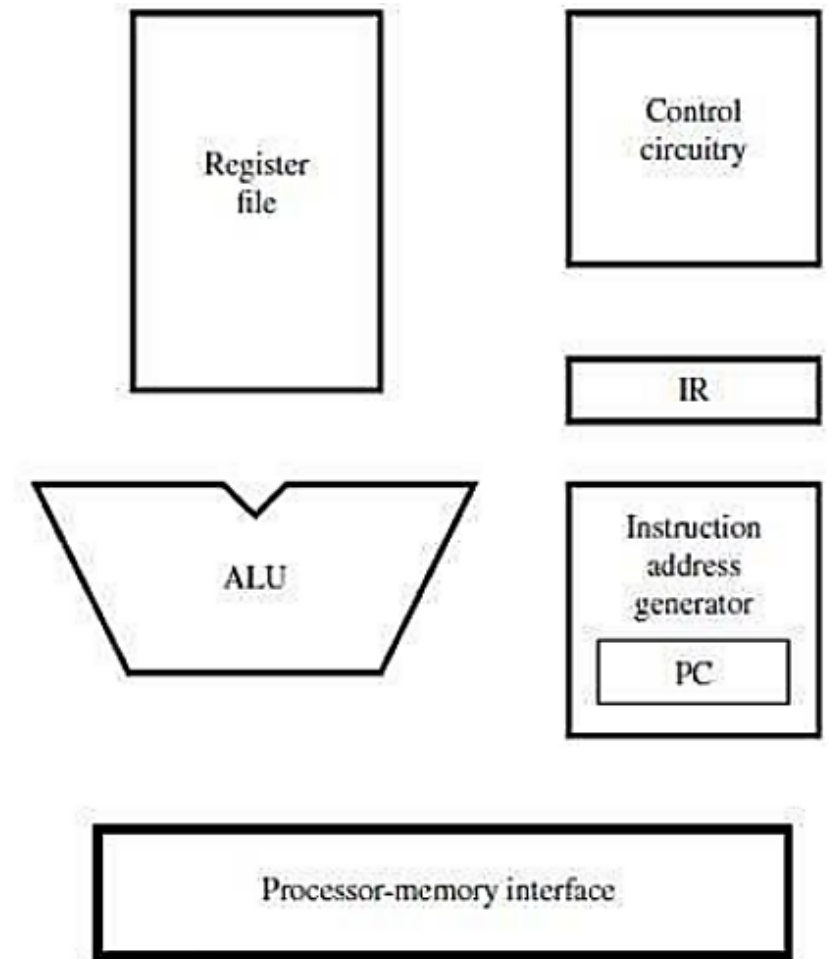
Module-5

Chapter 1: Basic Processing Unit

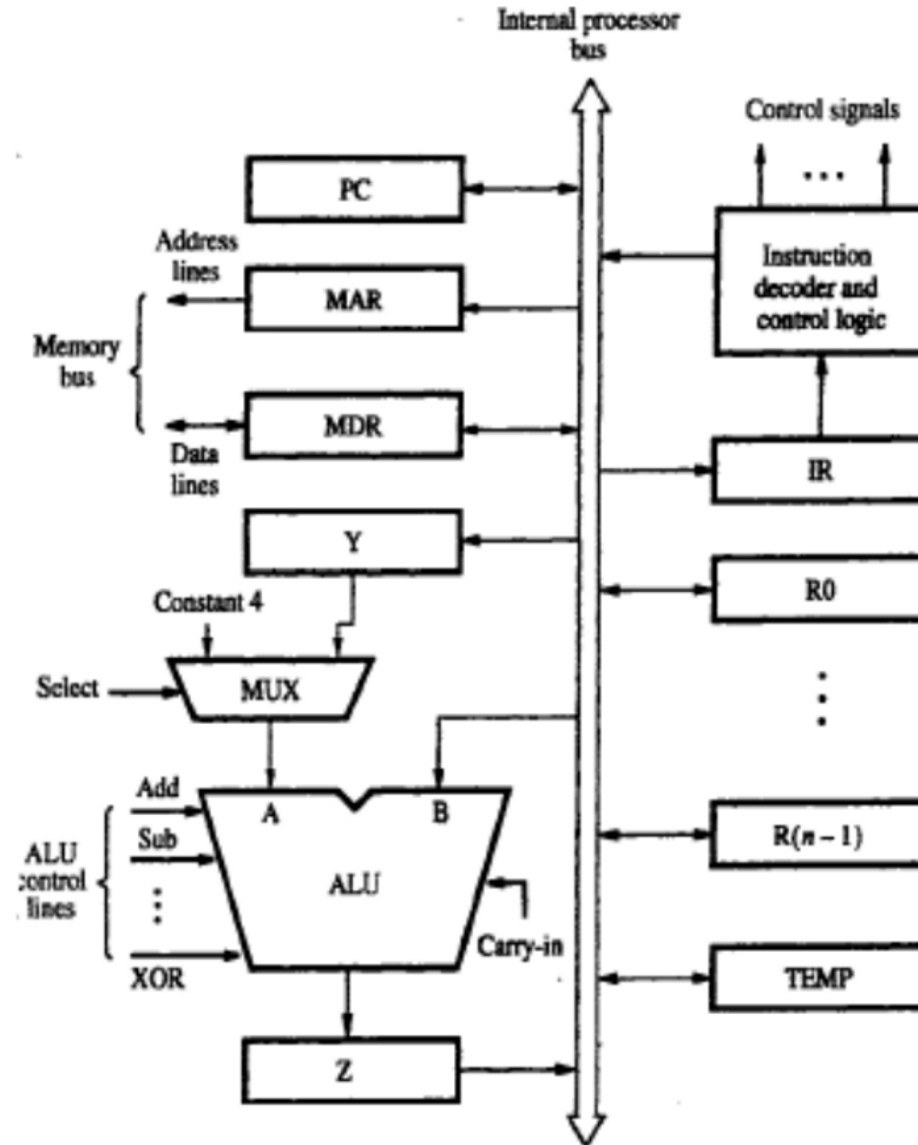
Some Fundamental Concepts

To execute an instruction, processor has to perform following 3 steps:

1. Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as: $IR \leftarrow [[PC]]$
2. Increment PC by 4. $PC \leftarrow [PC] + 4$
3. Carry out the actions specified by instruction (in the IR). The steps 1 and 2 are referred to as Fetch Phase. Step 3 is referred to as Execution Phase.



Single Bus Organization



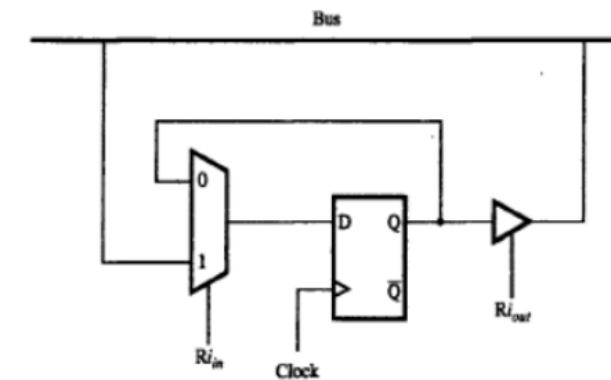
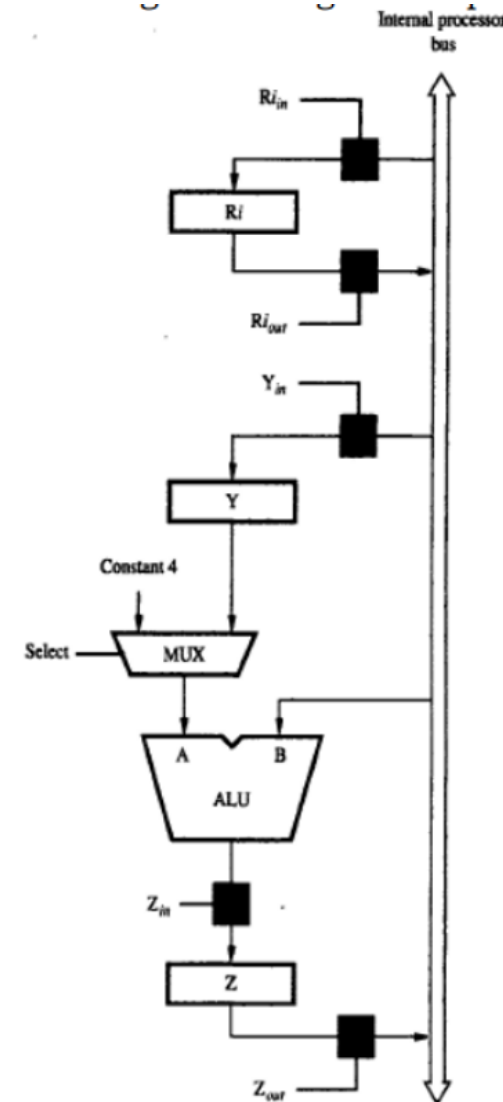
Register Transfers

Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control-signals are used: Ri_{in} & Ri_{out} . These are called Gating Signals.

$Ri_{in}=1$ data on bus is loaded into Ri .

$Ri_{out}=1$ content of Ri is placed on bus.

$Ri_{out}=0$, bus can be used for transferring data from other registers.



Performing ALU operations

The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs. One of the operands is output of MUX; And the other operand is obtained directly from processor-bus. The result (produced by the ALU) is stored temporarily in register Z.

The sequence of operations for $[R3] = [R1] + [R2]$ is as follows:

- 1) R1out, Yin
- 2) R2out, SelectY, Add, Zin
- 3) Zout, R3in

Instruction execution proceeds as follows:

Step 1: Contents from register R1 are loaded into register Y.

Step 2: Contents from Y and from register R2 are applied to the A and B inputs of ALU; Addition is performed & Result is stored in the Z register.

Step 3: The contents of Z register is stored in the R3 register. The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

Fetching a word from Memory

To fetch instruction/data from memory, processor transfers required address to MAR. At the same time, processor issues Read signal on control-lines of memory-bus. When requested data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers. The response time of each memory access varies (based on cache miss, memory mapped I/O). To accommodate this, MFC Memory Function Completed is used.

Example: Move (R1),R2.

The sequence of steps is:

- 1) R1out, MARin, Read; desired address is loaded into MAR & Read command is issued.
- 2) MDRinE, WMFC; load MDR from memory-bus & Wait for MFC response from memory.
- 3) MDRout, R2in; load R2 from MDR. where WMFC=control-signal that causes processor's control. circuitry to wait for arrival of MFC signal.

Storing a word in memory

Example: Move R2, (R1).

This requires the following sequence:

- 1) R1out, MARin; desired address is loaded into MAR.
- 2) R2out, MDRin, Write ;data to be written are loaded into MDR & Write command is issued.
- 3) MDRoutE, WMFC ;load data into memory-location pointed by R1 from MDR.

Execution of a Complete Instruction

Example: Add (R3), R1 which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

- 1) Fetch the instruction.
- 2) Fetch the first operand.
- 3) Perform the addition &
- 4) Load the result into R1.

Step	Action
1	$PC_{out}, MAR_{in}, \text{Read}, \text{Select4}, \text{Add}, Z_{in}$
2	$Z_{out}, PC_{in}, Y_{in}, \text{WMFC}$
3	MDR_{out}, IR_{in}
4	$R3_{out}, MAR_{in}, \text{Read}$
5	$R1_{out}, Y_{in}, \text{WMFC}$
6	$MDR_{out}, \text{SelectY}, \text{Add}, Z_{in}$
7	$Z_{out}, R1_{in}, \text{End}$

Branching Instructions

Instruction execution proceeds as follows:

Step 1-3: The processing starts & the fetch phase ends in step3.

Step 4: The offset-value is extracted from IR by instruction-decoding circuit. Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.

Step 5: the result, which is the branch-address, is loaded into the PC. The branch instruction loads the branch target address in PC so that PC will fetch the next instruction from the branch target address. The branch target address is usually obtained by adding the offset in the contents of PC. The offset X is usually the difference between the branch target-address and the address immediately following the branch instruction.

Step Action

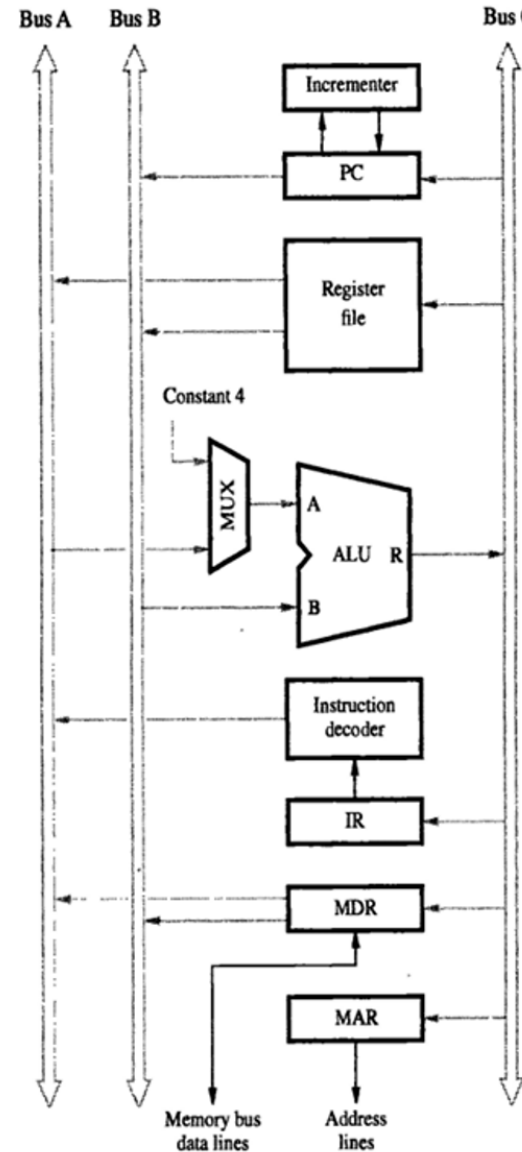
1	$PC_{out}, MAR_{in}, \text{Read}, \text{Select4}, \text{Add}, Z_{in}$
2	$Z_{out}, PC_{in}, Y_{in}, \text{WMFC}$
3	MDR_{out}, IR_{in}
4	$\text{Offset-field-of-IR}_{out}, \text{Add}, Z_{in}$
5	$Z_{out}, PC_{in}, \text{End}$

Multiple Bus Organization

Example: Add R4, R5, R6

Step Action

- 1 $PC_{out}, R=B, MAR_{in}, Read, IncPC$
- 2 WMFC
- 3 $MDR_{outB}, R=B, IR_{in}$
- 4 $R4_{outA}, R5_{outB}, SelectA, Add, R6_{in}, End$



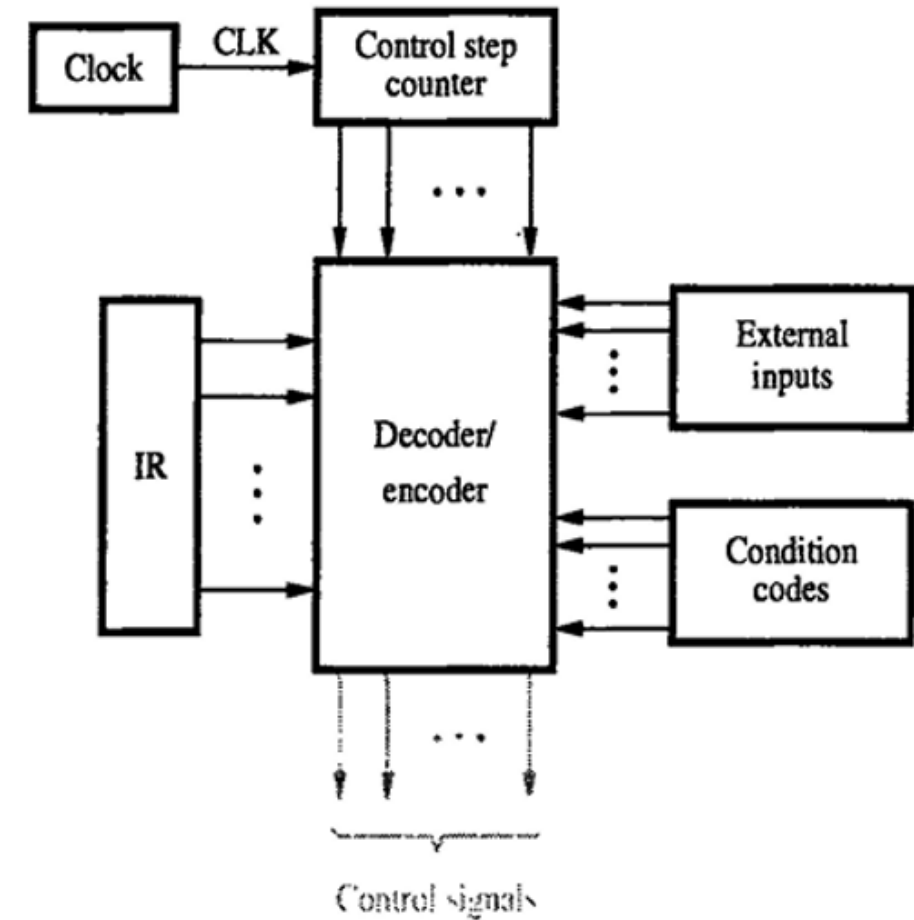
Hardwired Control

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. Computer designers use a wide variety of techniques to solve this problem. The approaches used: hardwired control and microprogrammed control.

Example: Add (R3),R1.

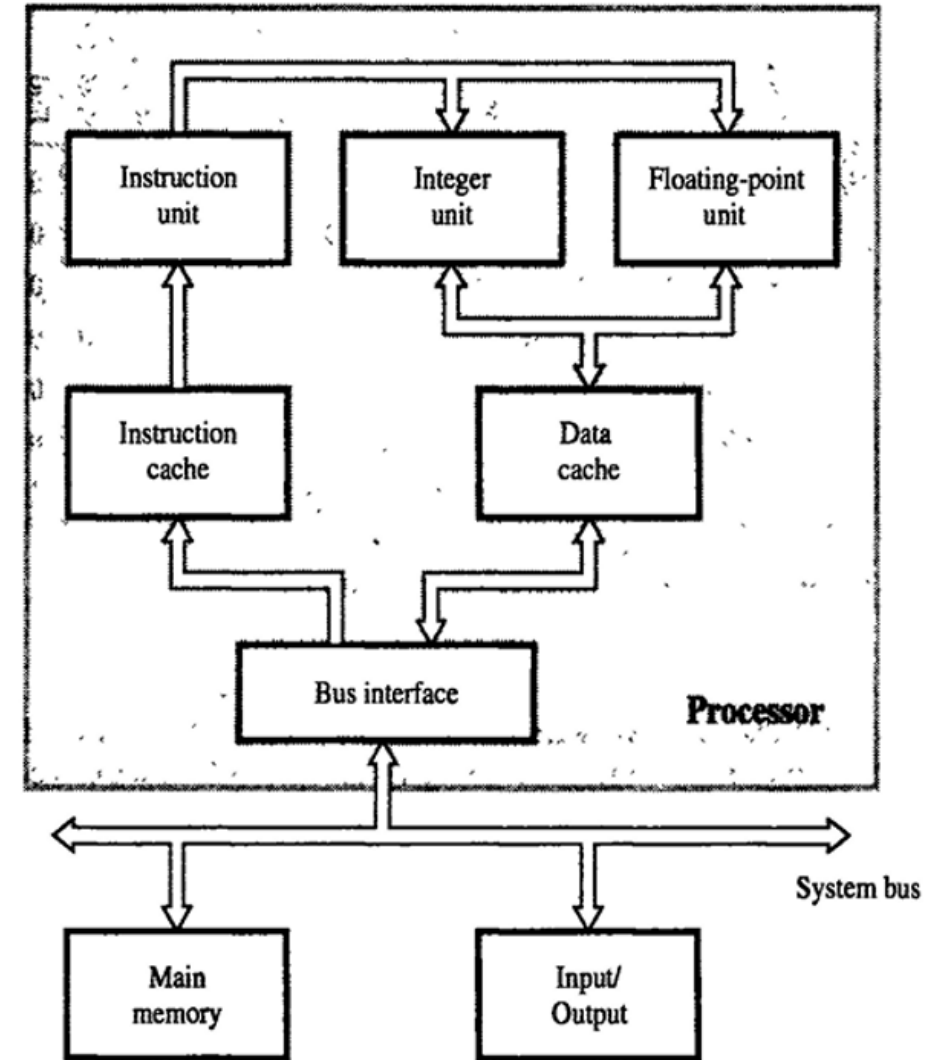
Each step in this sequence is completed in one clock period. A counter may be used to keep track of the control steps. Each state, or count, of this counter corresponds to one control step. The required control signals are determined by the following information:

- Contents of the control step counter
- Contents of the instruction register
- Contents of the condition code flags
- External input signals, such as MFC and interrupt requests



A Complete Processor

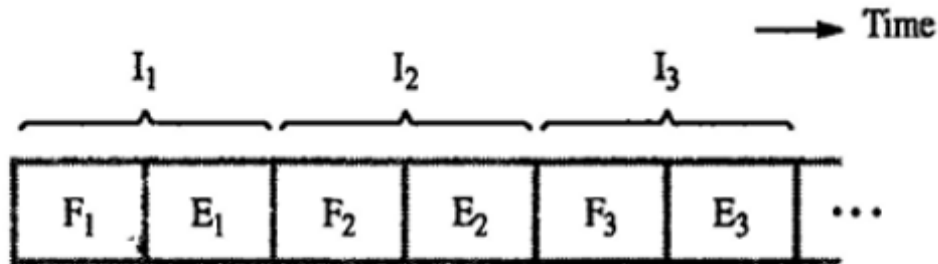
- A complete processor structure has an instruction unit that fetches instructions from an instruction cache or from the main memory when the desired instructions are not already in the cache.
- It has separate processing units to deal with integer data and floating-point data.
- A data cache is inserted between these units and the main memory.
- Using separate caches for instructions and data is common practice in many processors today.



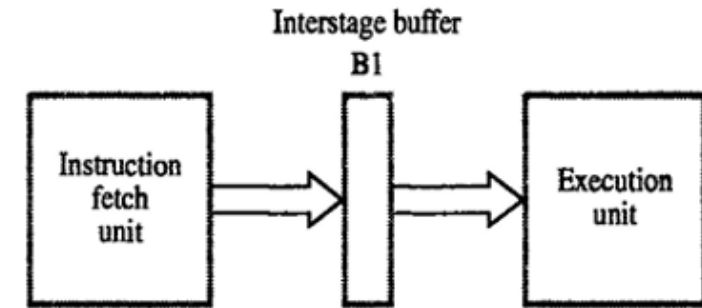
Chapter 2: Pipelining

Basic concepts

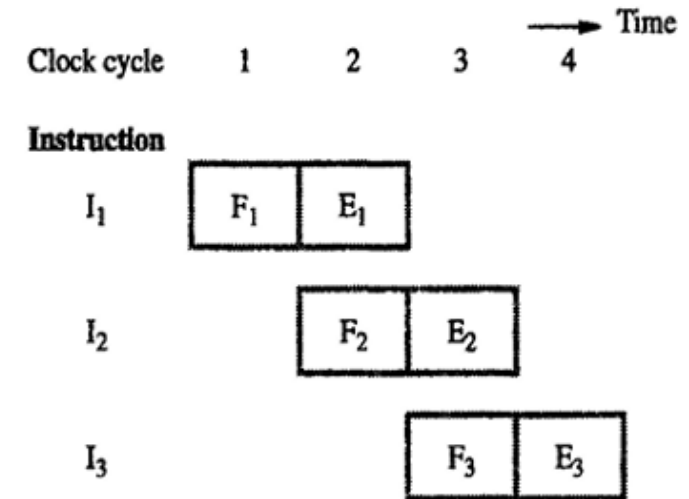
Pipelining is a hardware technique that improves computer processor performance by overlapping the execution of multiple instructions



(a) Sequential execution

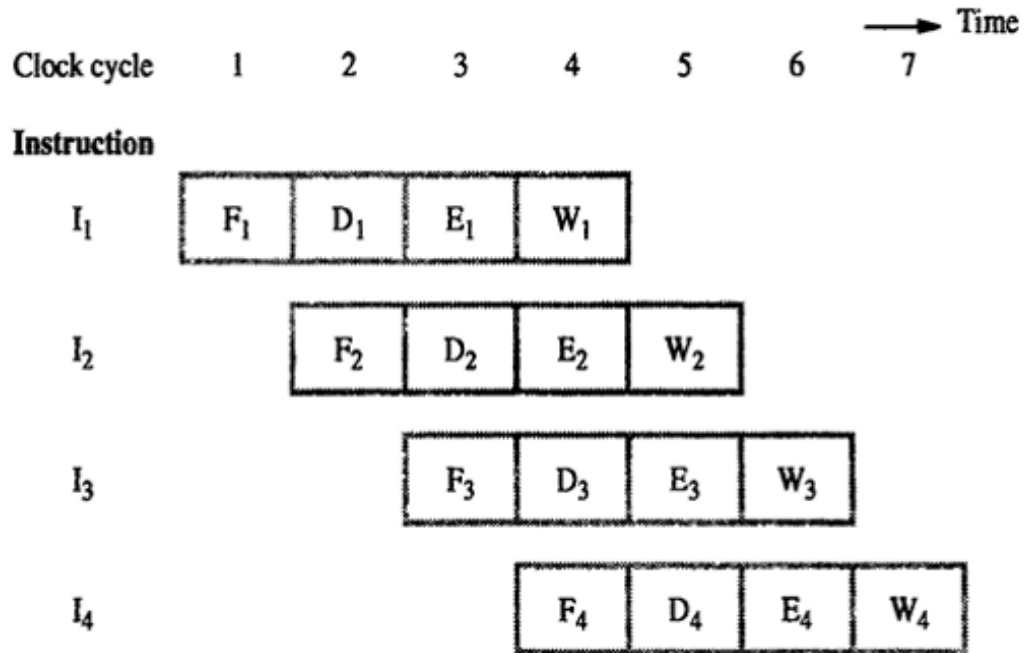


(b) Hardware organization

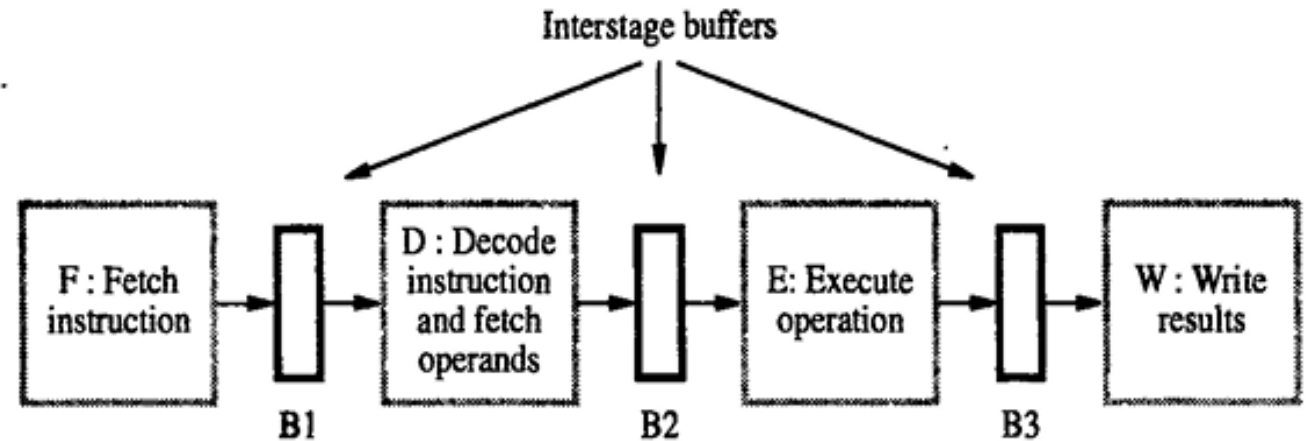


(c) Pipelined execution

Role of Cache memory

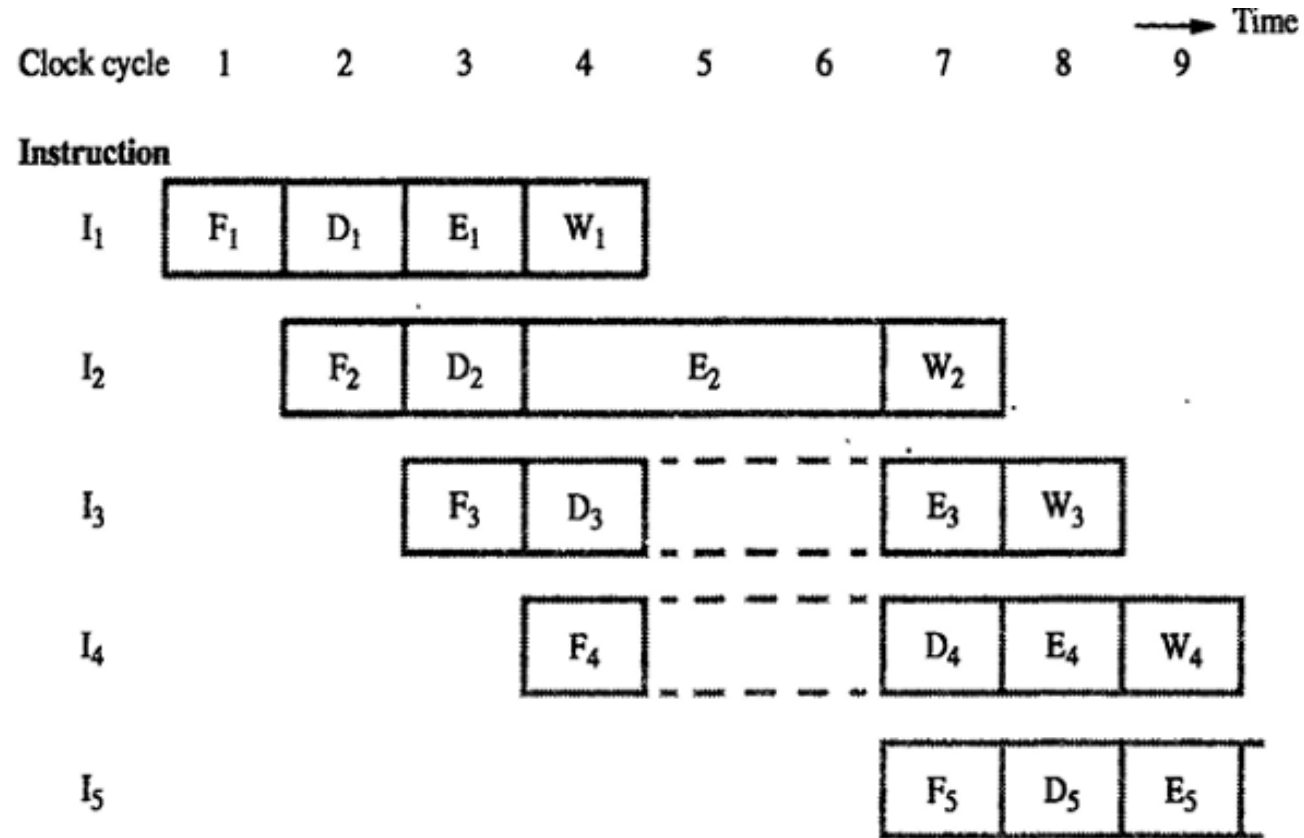


(a) Instruction execution divided into four steps



(b) Hardware organization

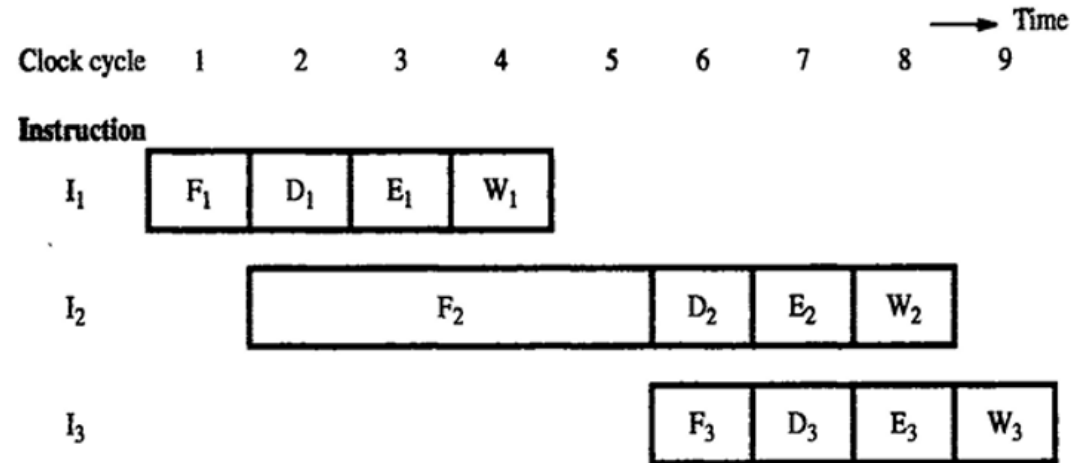
Pipeline Performance



Effect of an execution operation taking more than one clock cycle

Pipeline Performance

Pipeline stall caused by a cache miss in F2



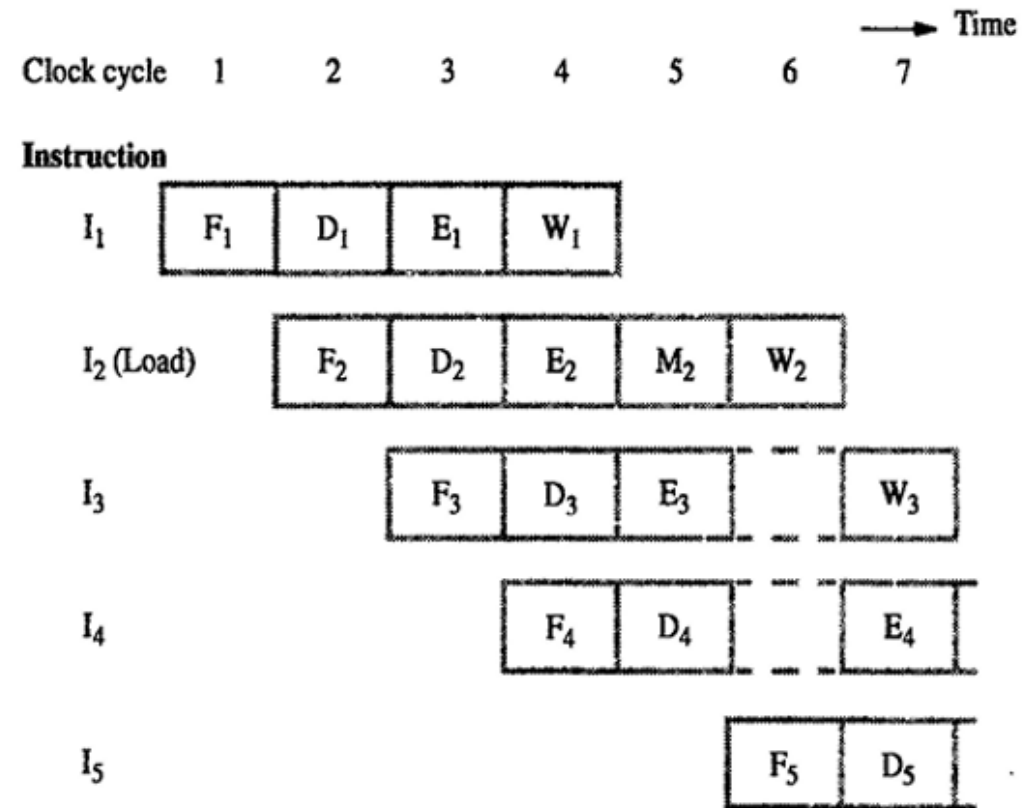
(a) Instruction execution steps in successive clock cycles



(b) Function performed by each processor stage in successive clock cycles

Pipeline Performance

Example: Load X(R1),R2



Thank you