

COMPUTER ORGANIZATION AND ARCHITECTURE

Hamsashree M K
Asst. Professor
Dept of ECE
BGSIT

COMPUTER ORGANIZATION AND ARCHITECTURE(18EC33)

- ◉ Basic Structure of Computers, Machine Instructions and Programs
- ◉ Addressing Modes
- ◉ Input/output Organization
- ◉ Memory System
- ◉ Basic Processing Units

Module 1

Basic Structure of Computers, Machine Instructions and Programs

- ◉ Computer Types
- ◉ Functional Units
- ◉ Basic Operational Concepts
- ◉ Bus Structures
- ◉ Software
- ◉ Performance-Processor clock
- ◉ Basic Performance Equation

- Numbers
- Arithmetic Operations and Characters
- IEEE Standard for floating point Numbers
- Memory Location and Addresses
- Memory Operations
- Instructions and Instruction Sequencing

MODULE-2

- ◉ Addressing Modes
- ◉ Assembly Language
- ◉ Basic input and output operations
- ◉ Stacks and Queues
- ◉ Subroutines
- ◉ Additional Sequencing

MODULE-3

INPUT/OUTPUT ORGANIZATION

- ◉ Accessing I/O Devices
- ◉ Interrupts-Interrupt Hardware
- ◉ Enabling and Disabling Interrupts
- ◉ Controlling Device Requests
- ◉ Direct Memory Access

MODULE-4

MEMORY SYSTEM

- ◉ Basic Concepts
- ◉ Semiconductor RAM Memories
- ◉ Internal Organization of memory chips
- ◉ Static memories
- ◉ Asynchronous DRAMS
- ◉ Read only memories
- ◉ Cache memories
- ◉ Virtual Memories
- ◉ Secondary Storage-Magnetic Hard Disks

MODULE-5

BASIC PROCESSING UNIT

- ◉ Some Fundamental Concepts
- ◉ Execution of Complete Instruction
- ◉ Multiple Bus Organization
- ◉ Hardwired Control
- ◉ Micro Programmed Control

Text Book:

- ◉ Computer Organization-Fifth Edition, Tata McGraw Hill, 2002

Reference Books:

- ◉ David A Patterson, John L, Hannessy: Computer Orgaization and Design-The Hardware /Software Interface.

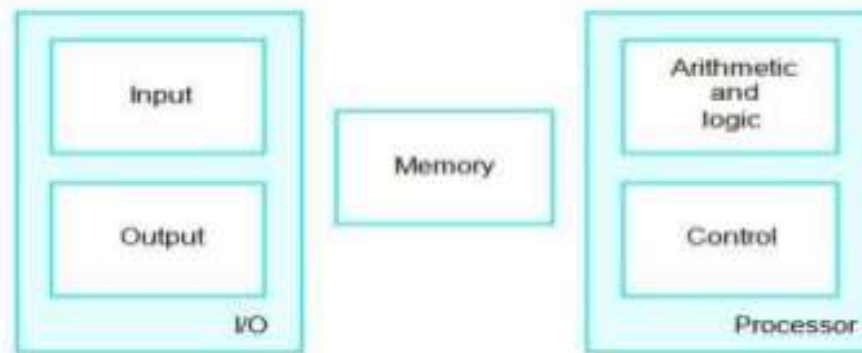
BASIC STRUCTURE OF COMPUTERS

- ❑ Computer Architecture (CA) is concerned with the structure and behaviour of the computer.
- ❑ CA includes the information formats, the instruction set and techniques for addressing memory.
- ❑ In general covers, CA covers 3 aspects of computer-design namely:
 - 1) Computer Hardware
 - 2) Instruction set Architecture
 - 3) Computer Organization.

FUNCTIONAL UNITS

- ◉ A computer consists of 5 functionally independent main parts:
 - ◉ 1) Input
 - ◉ 2) Memory
 - ◉ 3) ALU
 - ◉ 4) Output &
 - ◉ 5) Control units.

Functional Units



1. Input Unit

- Input units are used by a computer, which read the data.

- ◉ Output Unit:

Its function is to send the processed results to the outside world

- ◉ Arithmetic and logic Unit:

Is used to perform Arithmetic and Logical Operations

- ◉ Memory Unit:

The function of the memory unit is to store programs and Data

- Control Unit:

All activities inside the machine are directed by the control unit.

BASIC OPERATIONAL CONCEPTS

- ◉ Load
- ◉ Store

Add LOCA,R0

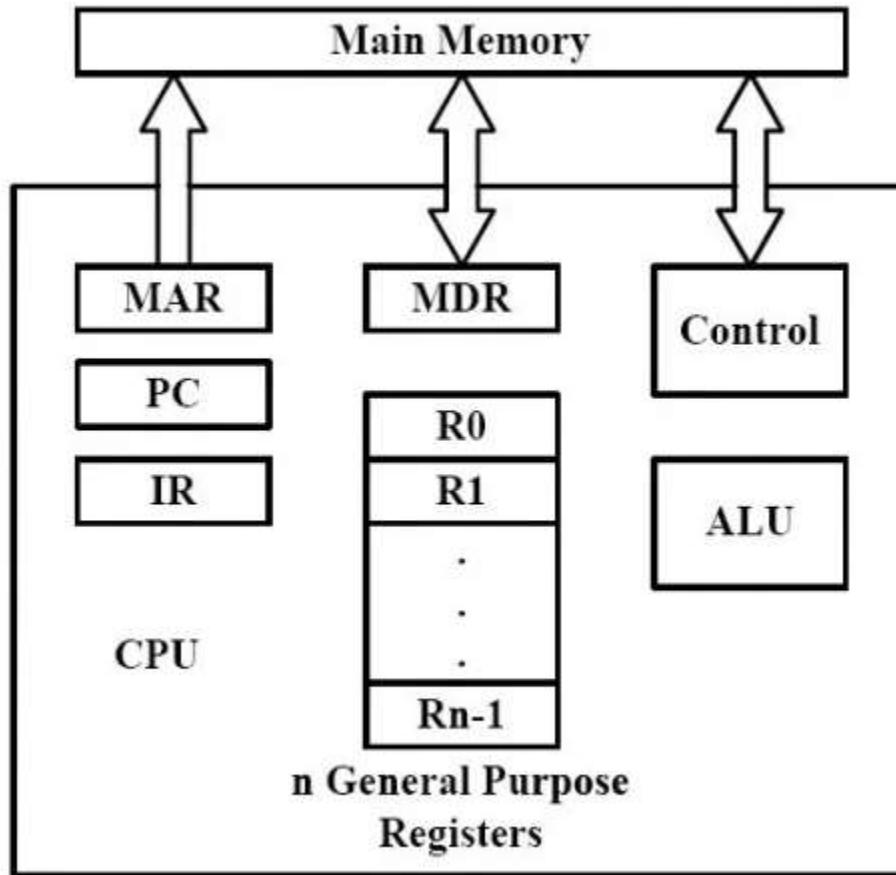
This instruction adds the operand at memory location LOCA to the operand which will be present in the Register R0

Load LOCA,R1

Add R1,R0

- First instruction sends the contents of the memory location LOCA into processor Register R0, and meanwhile the second instruction adds the contents of Register R1 and R0 and places the output in the Register R1.

CONNECTIONS BETWEEN THE PROCESSOR AND THE MEMORY



- ⦿ Processors have various registers to perform various functions :-
- ⦿ Program Counter :- It contains the memory address of next instruction to be fetched.
- ⦿ Instruction Register:- It holds the instruction which is currently being executed.
- ⦿ MDR :- It facilitates communication with memory. It contains the data to be written into or read out of the addressed location.
- ⦿ MAR :- It holds the address of the location that is to be accessed
- ⦿ There are n general purpose registers that is R0 to Rn-1

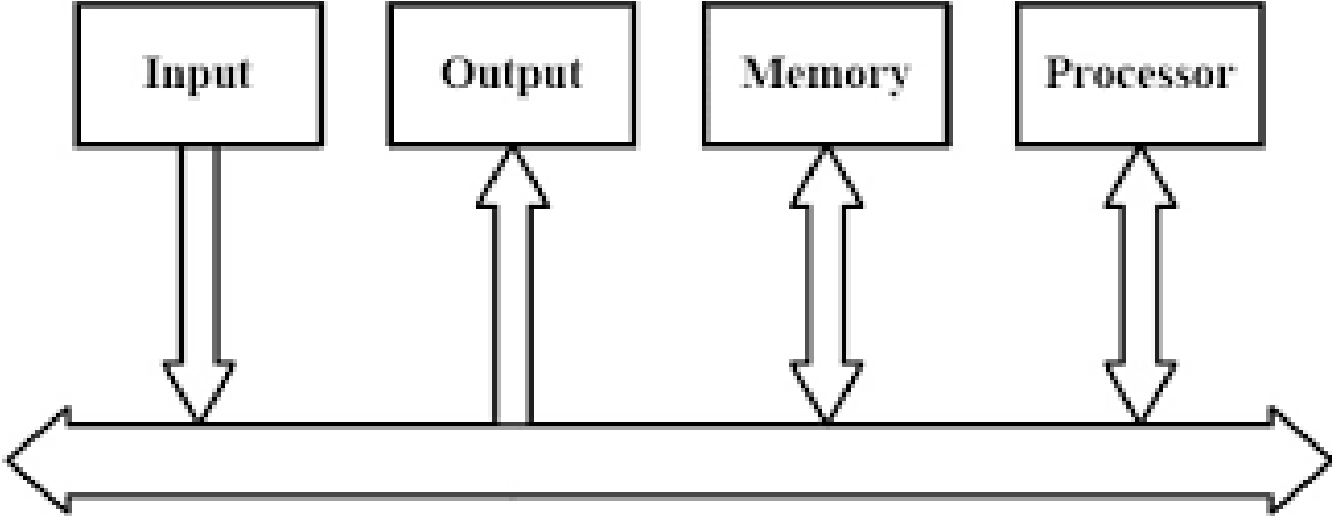
- These registers are:
 - 1) MAR (Memory Address Register)
 - 2) MDR (Memory Data Register)
- Memory Address Register:
- The address of the location to be accessed is held by MAR.
- Memory Data Register:
- It contains the data to be written into or to be read out of the addressed location.

WORKING EXPLANATION

- ⦿ A **PC** is set to point to the first instruction of the program. The contents of the **PC** are transferred to the **MAR** and a Read control signal is sent to the memory. The addressed word is fetched from the location which is mentioned in the **MAR** and loaded into **MDR**.

- ◉ 1) Memory
 - ◉ 2) MAR
 - ◉ 3) MDR
 - ◉ 4) PC
 - ◉ 5) IR
 - ◉ 6) General Purpose Registers
 - ◉ 7) Control Unit
 - ◉ 8) ALU
- ◉ The instruction that is currently being executed is held by the Instruction Register.
 - ◉ IR output is available to the control circuits, which generates the timing signal that control the various processing elements involved in executing the instruction.
 - ◉ The Memory address of the next instruction to be fetched and executed is contained by the Program Counter.
 - ◉ It is a specialized register.
 - ◉ It keeps the record of the programs that are executed.
 - ◉ Role of these registers is to handle the data available in the instructions. They store the data temporarily.
 - ◉ Two registers facilitate the communication with memory.

SINGLE BUS STRUCTURE



BUS STRUCTURE

- ◉ A bus is a group of lines that serves as a connecting path for several devices.

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.
- ◉ Advantages: 1) Low cost & 2) Flexibility for attaching peripheral devices.

SOFTWARE

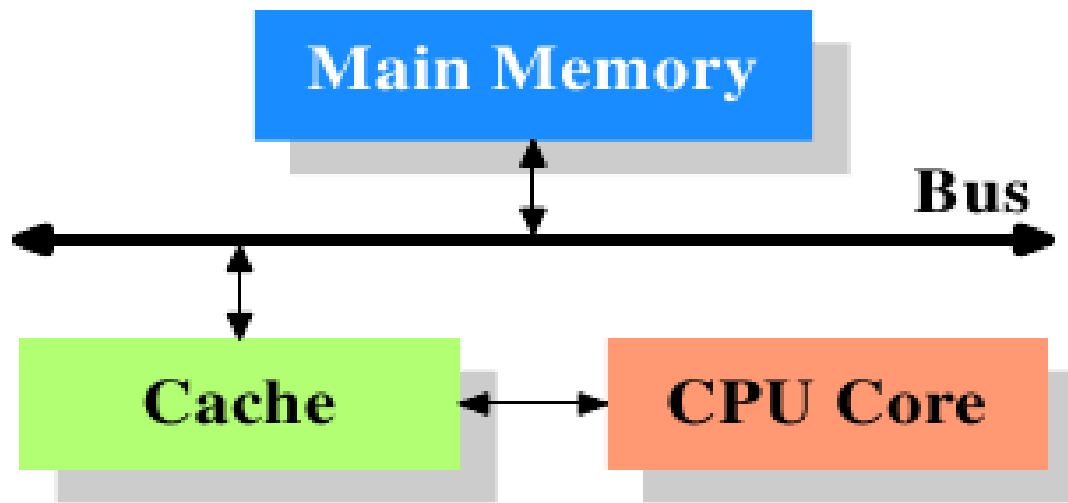
- ◉ Receiving and interpreting user commands
- ◉ Entering and editing
- ◉ Managing the storage and retrieval of files in secondary storage devices.

- ⦿ Running standard application programs.
- ⦿ Controlling I/O units.
- ⦿ Translating Programs.
- ⦿ Linking and Running.

PERFORMANCE

- ⦿ Performance means how quickly a program can be executed.
- ⦿ In order to get the best performance it is required to design the compiler, machine instruction set & hardware in a coordinated manner.

THE PROCESSOR CACHE



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 executed 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
$$T = N \times S / R$$

PROBLEMS

1. List the steps needed to execute the machine instruction:

◉ *Load R2, LOC*

Interms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC.

Solution:

- ⦿ Transfer the contents of register PC to register MAR.
- ⦿ Issue a Read command to memory.
- ⦿ And, then wait until it has transferred the requested word into register MDR.
- ⦿ Transfer the instruction from MDR into IR and decode it.
- ⦿ Transfer the address LOCA from IR to MAR.
- ⦿ Issue a Read command and wait until MDR is loaded.
- ⦿ Transfer contents of MDR to the ALU.
- ⦿ Transfer contents of R0 to the ALU.
- ⦿ Perform addition of the two operands in the ALU and transfer result into R0.
- ⦿ Transfer contents of PC to ALU.
- ⦿ Add 1 to operand in ALU and transfer incremented address to PC.

2. List the steps needed to execute the machine instruction:

⊙ *Add R4, R2, R3*

In terms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC.

Solution:

- ⦿ Transfer the contents of register PC to register MAR.
- ⦿ Issue a Read command to memory.
- ⦿ And, then wait until it has transferred the requested word into register MDR.
- ⦿ Transfer the instruction from MDR into IR and decode it.
- ⦿ Transfer contents of R1 and R2 to the ALU.
- ⦿ Perform addition of two operands in the ALU and transfer answer into R3.
- ⦿ Transfer contents of PC to ALU.
- ⦿ Add 1 to operand in ALU and transfer incremented address to PC.

3.a) Give a short sequence of machine instructions for the task “Add the contents of memory location A to those of location B, and place the answer in location C”. Instructions:

Load R_i , LOC Load A, R0
 Load B, R1
 Add R0, R1
 Store R1, C

and

Store R_i , LOC

4. A program contains 1000 instructions. Out of that 25% instructions requires 4 clock cycles, 40% instructions requires 5 clock cycles and remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine.

Solution:

- ⊙ $N = 1000$
- ⊙ 25% of $N = 250$ instructions require 4 clock cycles.
- ⊙ 40% of $N = 400$ instructions require 5 clock cycles. 35% of $N = 350$ instructions require 3 clock cycles.
- ⊙ $T = (N \cdot S) / R = (250 \cdot 4 + 400 \cdot 5 + 350 \cdot 3) / 1 \times 10^9$
 $= (1000 + 2000 + 1050) / 1 \cdot 10^9 = 4.05 \mu\text{s}.$

Machine Instructions and Programs

Numbers, Arithmetic Operations and characters

I. Numbers Representation

- **Sign-and-magnitude**
- **One's Complement**
- **Two's Complement**

b3 b2 b1 b0	Sign magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000 8	-0	-7	-8
1001 9	-1	-6	-7
1010 10	-2	-5	-6
1011 11	-3	-4	-5
1100 12	-4	-3	-4
1101 13	-5	-2	-3

- Addition of Positive numbers
- Addition and Subtraction of signed numbers
- Overflow in Integer Arithmetic

0111	+7	0000
1000		1111
1001		10000

MEMORY-LOCATIONS & ADDRESSES

- Memory consists of many millions of storage cells (flip-flops).
- Each group of n bits is referred to as a word of information, and n is called the **word length**.
- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a **byte**.

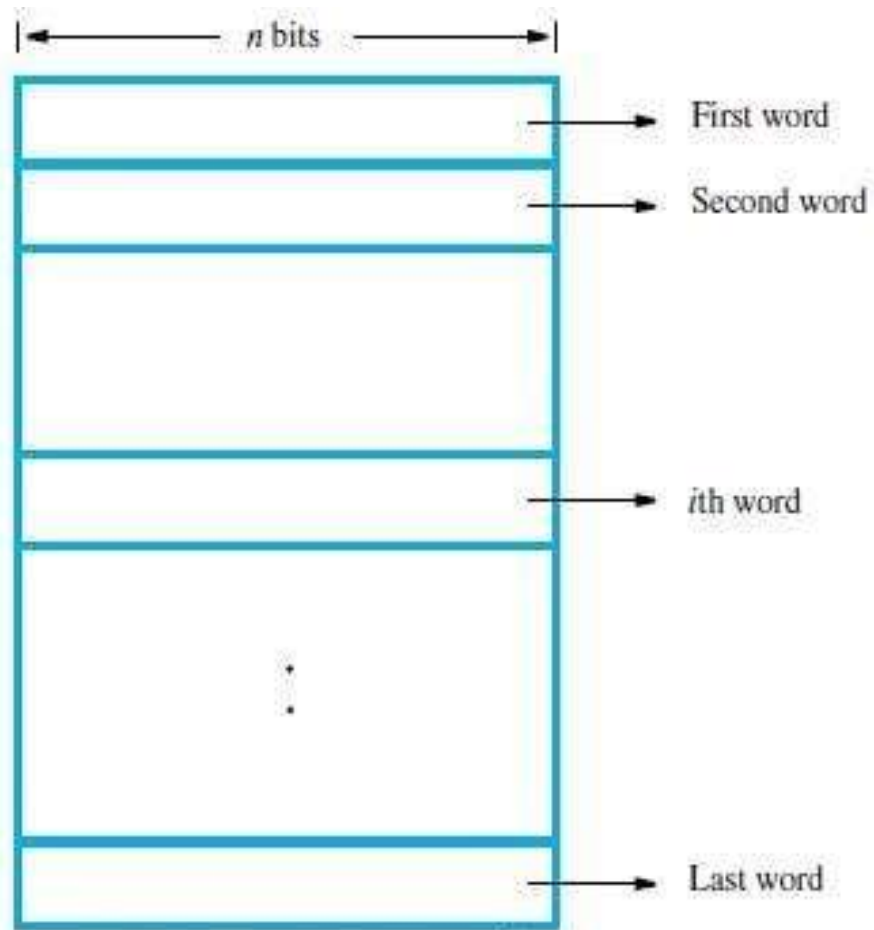
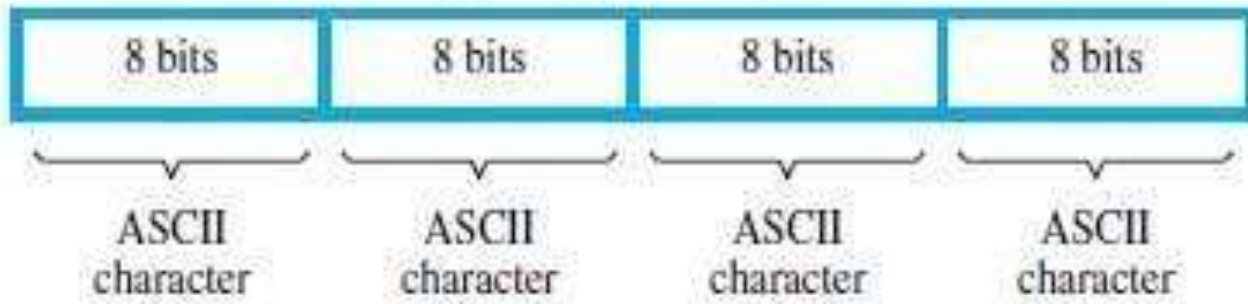


Figure 2.1 Memory words.



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

(a) A signed integer



(b) Four characters

Figure 2.2 Examples of encoded information in a 32-bit word.

Byte Addressability

In byte-addressable memory, successive addresses refer to successive byte locations in the memory.

There are 2 types

1. Big-Endian

2. Little-Endian

Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
	⋮			
$2^k - 4$	$2^k - 4$	$2^k - 3$	$2^k - 2$	$2^k - 1$

(a) Big-endian assignment

	Byte address			
0	3	2	1	0
4	7	6	5	4
	⋮			
$2^k - 4$	$2^k - 1$	$2^k - 2$	$2^k - 3$	$2^k - 4$

(b) Little-endian assignment

Figure 2.3

Byte and word addressing.

On little-endian, memory will look like

Address	Value
1001	78
1002	56
1003	34
1004	12

On big-endian, memory will look like

Address	Value
1001	12
1002	34
1003	56
1004	78

Word Alignment

Words are said to be **Aligned** in memory if they begin at a byte-address that is a multiple of the number of bytes in a word.

For example,

- If the word length is 16(2 bytes), aligned words begin at byte-addresses 0, 2, 4
- If the word length is 64(2 bytes), aligned words begin at byte-addresses 0, 8, 16

Accessing Numbers, Characters & Characters Strings

A number usually occupies one word. It can be accessed in the memory by specifying its word address.

There are two ways to indicate the length of the string:

- A special control character with the meaning "end of string" can be used as the last character in the string.
- A separate memory word location or register can contain a number indicating the length of the string in bytes.

Memory operations

- Load
- Store

The **Load** operation transfers a copy of the contents of a specific memory-location to the processor.

The **Store** operation transfers the information from the register to the specified memory location.

Instruction and Instruction Sequencing

A computer must have instructions capable of performing 4 types of operations:

- Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
- Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
- Program sequencing and control (CALL, RET, LOOP, INT).
- I/O transfers (IN, OUT).

Register Transfer Notation

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	$[R3] \leftarrow [R1] + [R2]$	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.

Basic Instruction Types

Instruction Type	Syntax	Example	Description	Instructions for Operation $C \leftarrow [A] + [B]$
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C

One Address	Opcode Source/Destination	Load A	Copy contents of memory- location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory- location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

Instruction Execution & Straight Line Sequencing

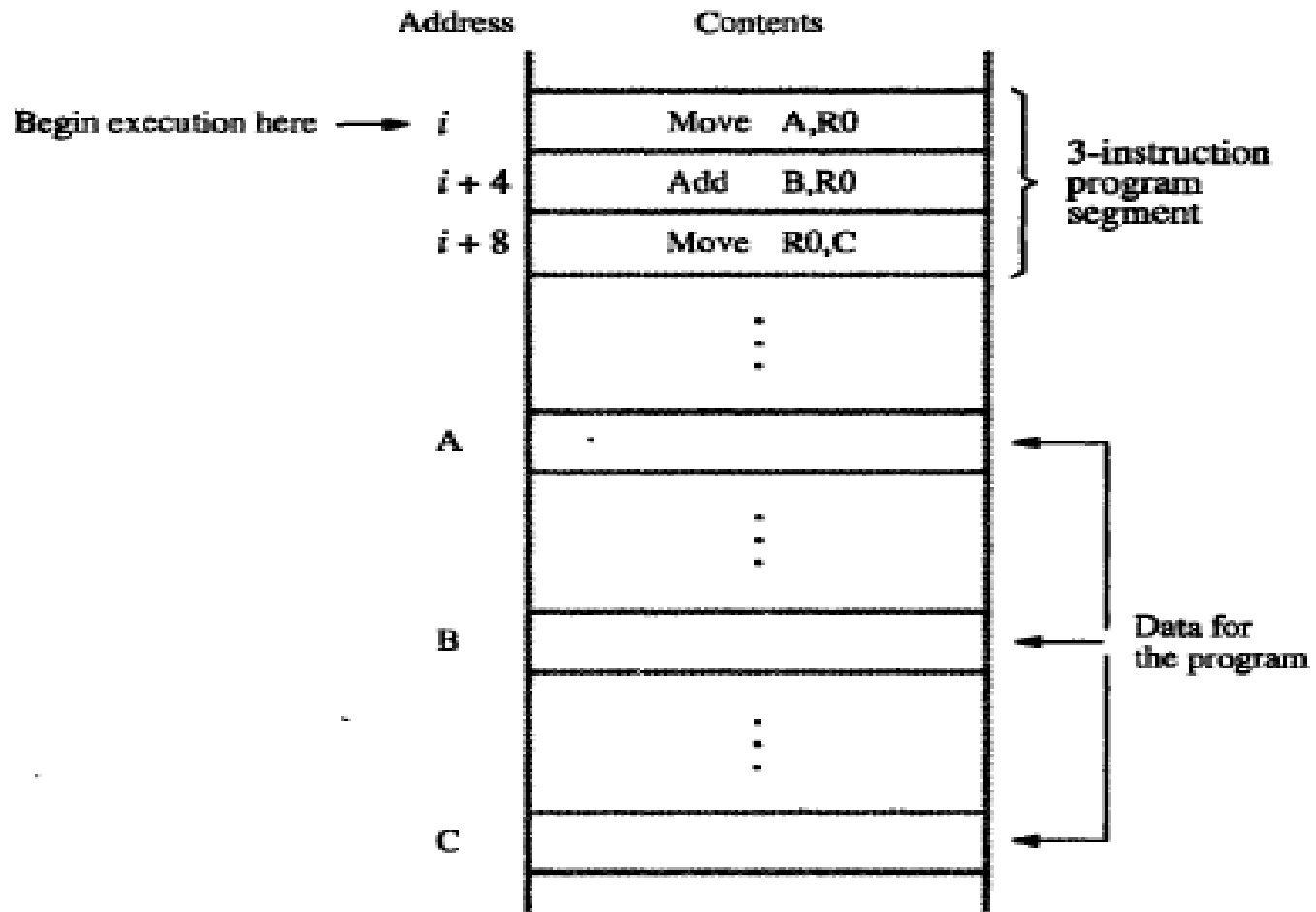


Figure 2.8 A program for $C \leftarrow [A] + [B]$.



The program is executed as follows:

- Initially, the address of the first instruction is loaded into PC
- Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called *Straight-Line sequencing*.
- During the execution of each instruction, PC is incremented by 4 to point to next instruction.

There are 2 phases for Instruction Execution:

- **Fetch Phase:** The instruction is fetched from the memory-location and placed in the IR.
- **Execute Phase:** The contents of IR is examined to determine which operation is to be performed.

i	Move	NUM1,R0
$i + 4$	Add	NUM2,R0
$i + 8$	Add	NUM3,R0
		⋮
$i + 4n - 4$	Add	NUM n ,R0
$i + 4n$	Move	R0,SUM
		⋮
SUM		
NUM1		
NUM2		
		⋮
NUM n		

Figure 2.9 A straight-line program for adding n numbers.

Program Explanation

- Consider the program for adding a list of n numbers.
- The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2.....NUM n .
- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory-location SUM.

Branching

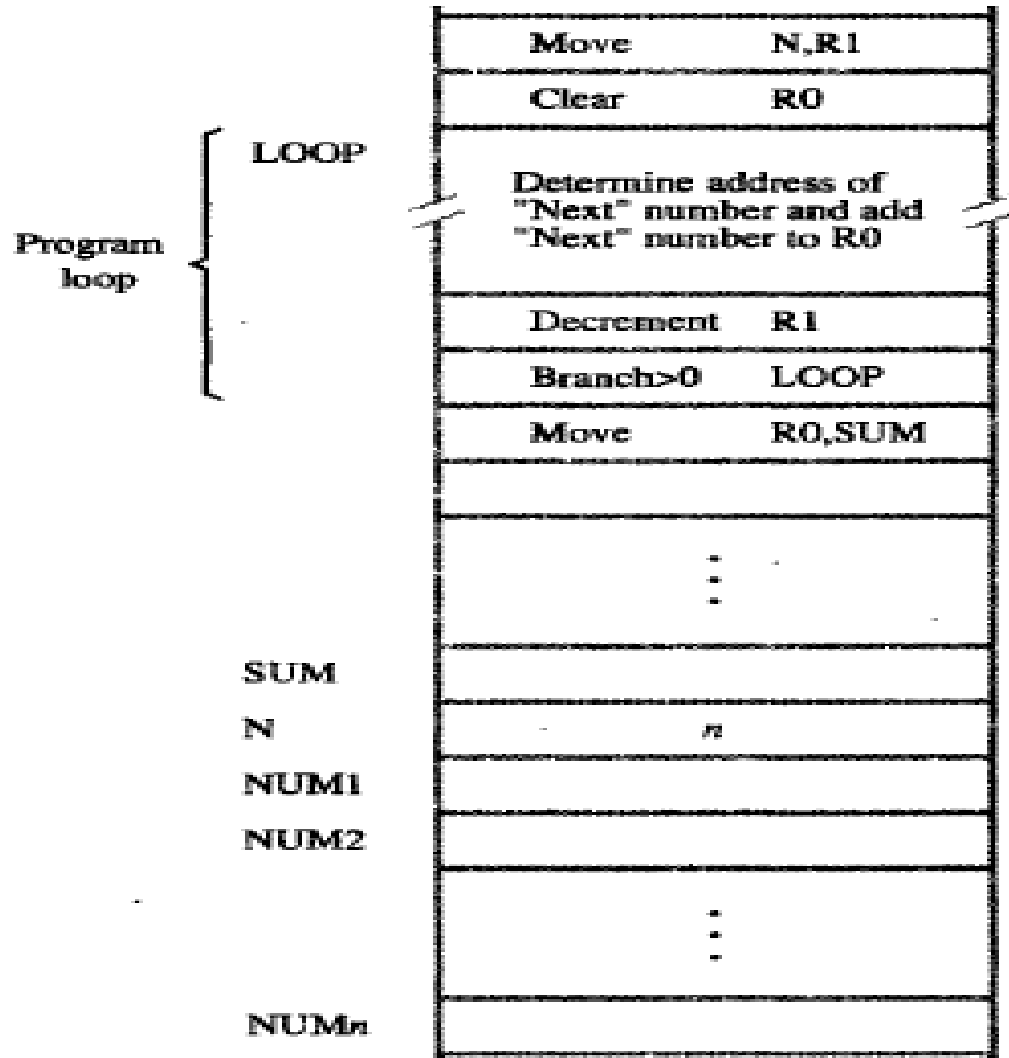



Figure 2.10 Using a loop to add n numbers.



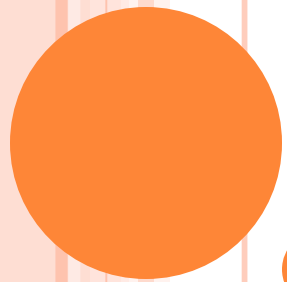
Branch Instruction loads a new value in the program counter. As a result, the processor fetches and executes the instruction at this new address called the **Branch Target**.

A **Conditional Branch Instruction** causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

Condition Codes

Four commonly used flags are:

- N (negative) set to 1 if the result is negative, otherwise cleared to 0.
- Z (zero) set to 1 if the result is 0; otherwise, cleared to 0.
- V (overflow) set to 1 if arithmetic overflow occurs; otherwise, cleared to 0.
- C (carry) set to 1 if a carry-out results from the operation; otherwise cleared to 0.



ASSEMBLY LANGUAGE

ASSEMBLY LANGUAGE

- We generally use symbolic-names to write a program.
- A complete set of symbolic-names and rules for their use constitute an **Assembly Language**.
- The set of rules for using the mnemonics in the specification of complete instructions and programs is called the **Syntax** of the language.



- **Assembler.**
- **Source Program**
- **Object Program.**
- For example:
- *MOVE R0,SUM*;The term MOVE represents OP code for operation performed by instruction.
- *ADD #5,R3*;

Adds number 5 to contents of register R3 & puts the result back into registerR3.



ASSEMBLER DIRECTIVES

- **Directives** are the assembler commands to the assembler concerning the program being assembled.
- These commands are not translated into machine opcode in the object-program.

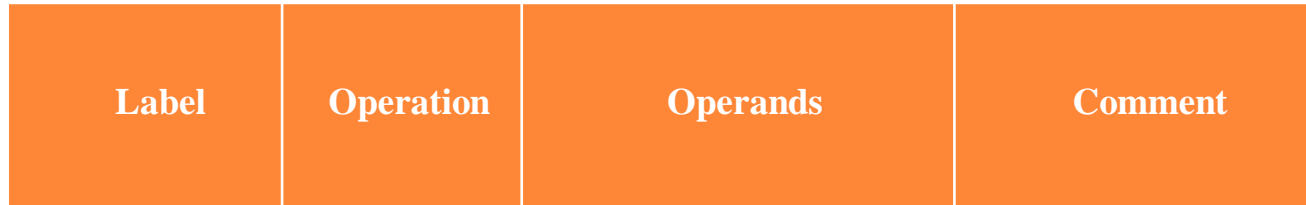


	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
Statements that generate machine instructions		ORIGIN	100
	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
Assembler directives		MOVE	R0,SUM
		RETURN	
		END	START

Figure 2.18 Assembly language representation for the program in Figure 2.17.



GENERAL FORMAT OF A LABEL



Label is an optional name associated with the memory-address

Operation Field contains the OP-code mnemonic of the desired instruction or assembler.

Operand Field contains addressing information for accessing one or more operands, depending on the type of instruction.

Comment Field is used for documentation purposes to make program easier to understand.



ASSEMBLY AND EXECUTION OF PROGRAMS

Assembler Program

- replaces all symbols denoting operations & addressing-modes with binary-codes used in machine instructions.
- replaces all names and labels with their actual values.
- assigns addresses to instructions & data blocks, starting at address given in ORIGIN directive
- inserts constants that may be given in DATAWORD directives.
- reserves memory-space as requested by RESERVE directives.



TWO PASS ASSEMBLER

- **First Pass:** Work out all the addresses of labels.

- **Second Pass:** Generate machine code, substituting values for the labels.



- **Debugger Program** is used to help the user find the programming errors.
- Debugger program enables the user
 - to stop execution of the object-program at some points of interest.
 - to examine the contents of various processor-registers and memory-location.



BASIC INPUT AND OUTPUT OPERATIONS

- When a key is pressed, the corresponding ASCII code is stored in a **DATAIN** register associated with the keyboard.
 - **SIN=1** When a character is typed in the keyboard. This informs the processor that a valid character is in **DATAIN**.
 - **SIN=0** When the character is transferred to the processor.



- An analogous process takes place when characters are transferred from the processor to the display.
 - **SOUT=1** When the display is ready to receive a character.
 - **SOUT=0** When the character is being transferred to DATAOUT.



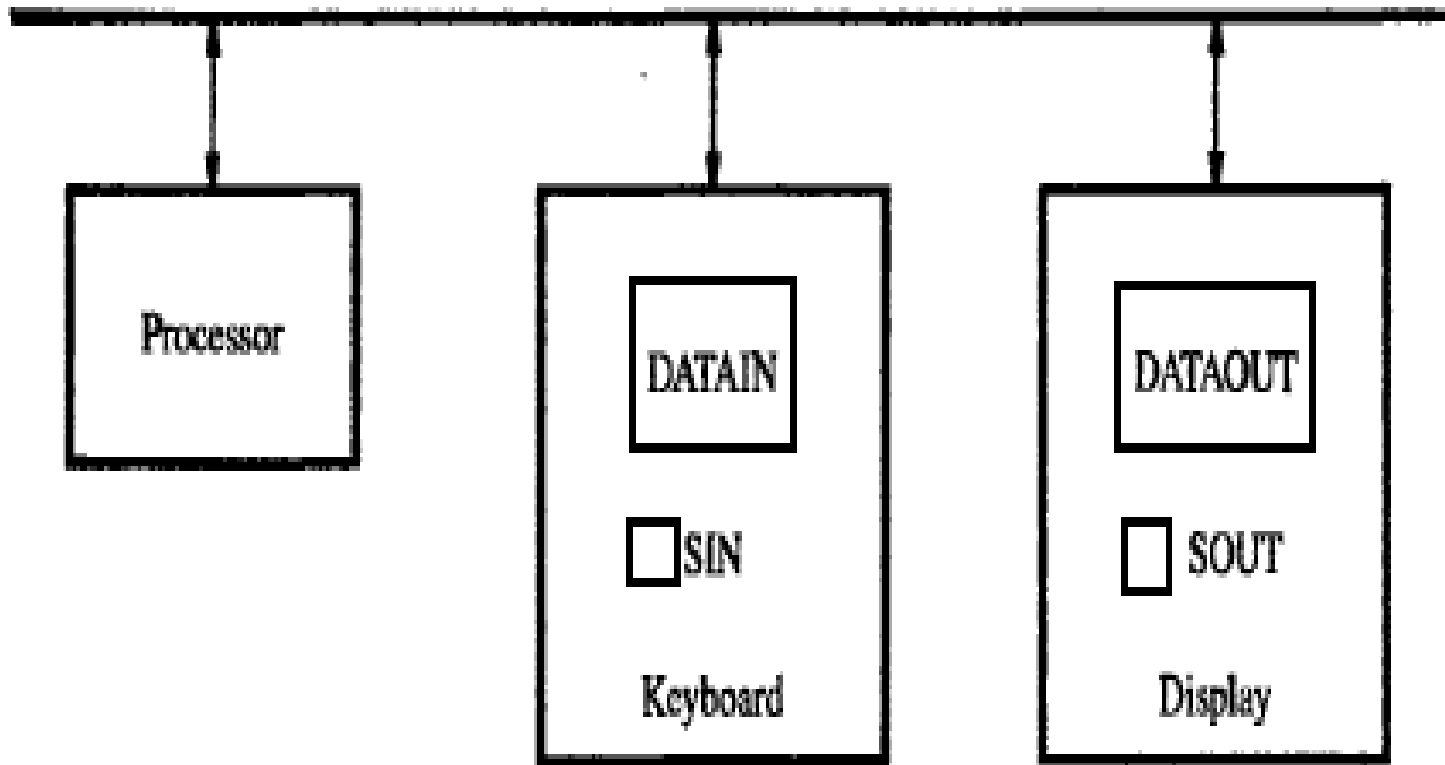


Figure 2.19 Bus connection for processor, keyboard, and display.



MEMORY MAPPED I/O

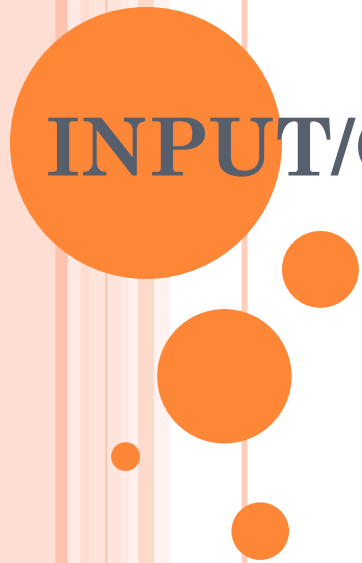
Program to read a line of characters and display it			
	Move	#LOC,R0	Initialize pointer register R0 to point to the address of the first location in memory where the characters are to be stored.
READ	TestBit	#3,INSTATUS	Wait for a character to be entered in the keyboard buffer DATAIN.
	Branch=0	READ	
	MoveByte	DATAIN,(R0)	Transfer the character from DATAIN into the memory (this clears SIN to 0).
ECHO	TestBit	#3,OUTSTATUS	Wait for the display to become ready.
	Branch=0	ECHO	
	MoveByte	(R0),DATAOUT	Move the character just read to the display buffer register (this clears SOUT to 0).
	Compare	#CR,(R0)+	Check if the character just read is CR (carriage return). If it is not CR, then branch back and read another character.
	Branch≠0	READ	Also, increment the pointer to store the next character.

Figure 2.20 A program that reads a line of characters and displays it.



MODULE 3

INPUT/OUTPUT ORGANIZATION



ACCESSING I/O-DEVICES

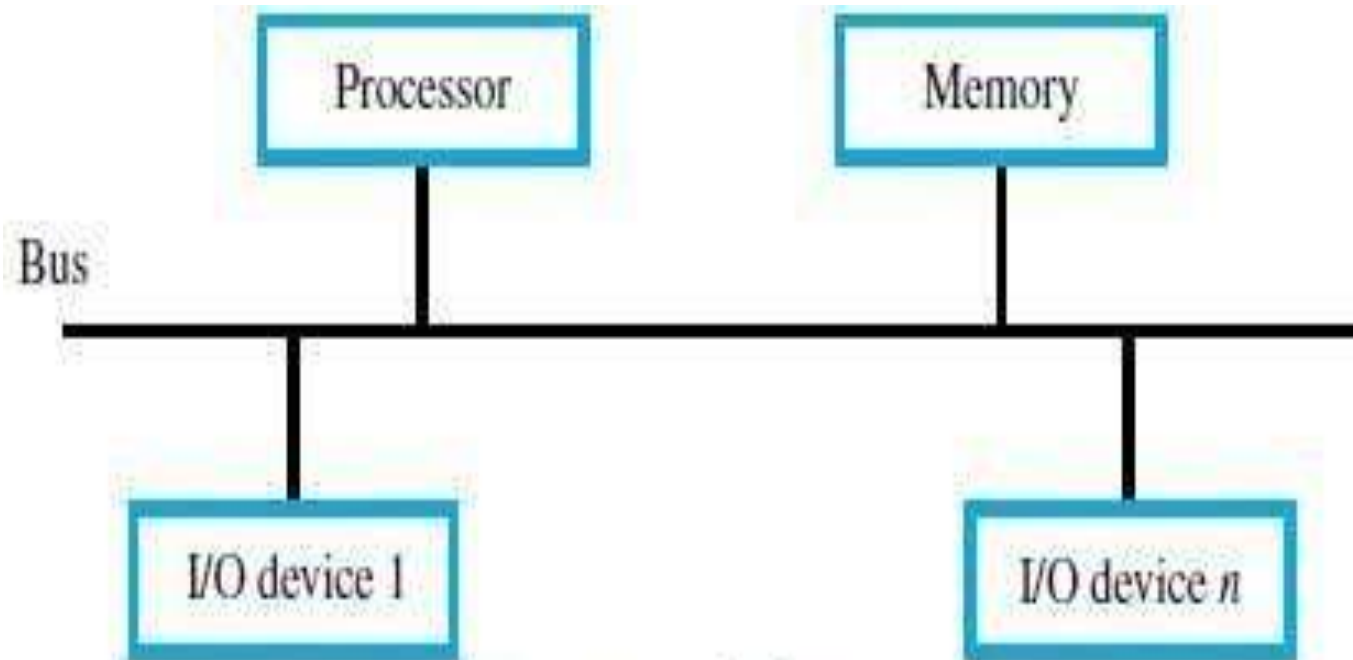


Figure 7.1

A single-bus structure.



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

1.Memory mapped I/O

2.I/O mapped I/O

Memory Mapped I/O

- Memory and I/O-devices share a common address-space.

For example,

Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0.

Here, DATAIN à address of the input-buffer of the keyboard.



I/O Mapped I/O

- Memory and I/O address-spaces are different.
- A special instructions named **IN** and **OUT** are used for data-transfer



I/O INTERFACE FOR INPUT DEVICES

- Address Register
- Status Register
- Data Register
 - >DataIN
 - >DataOUT

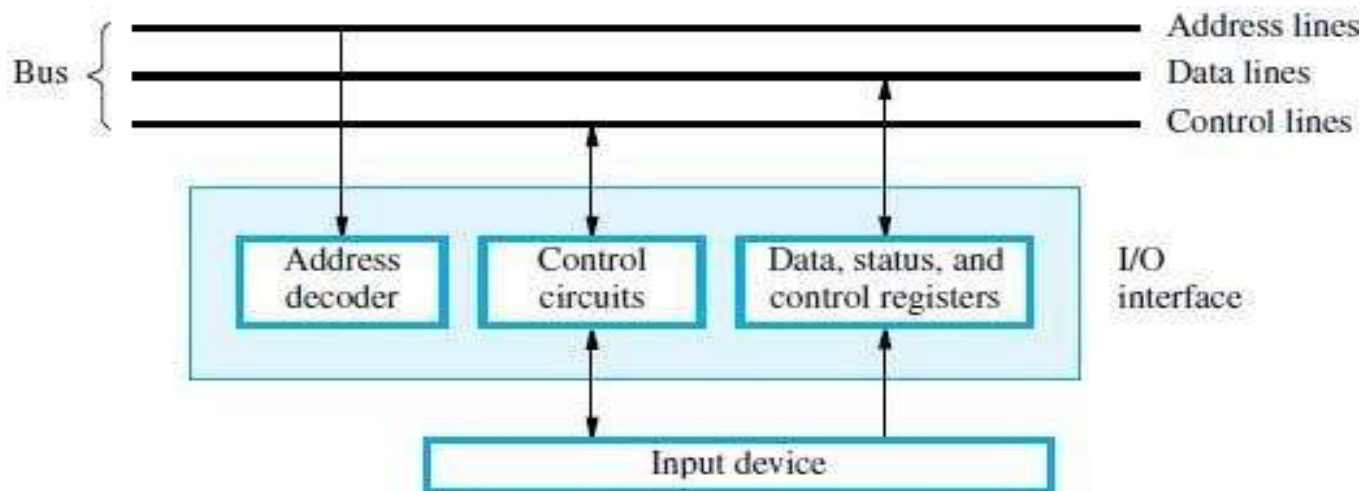


Figure 7.2 I/O interface for an input device.



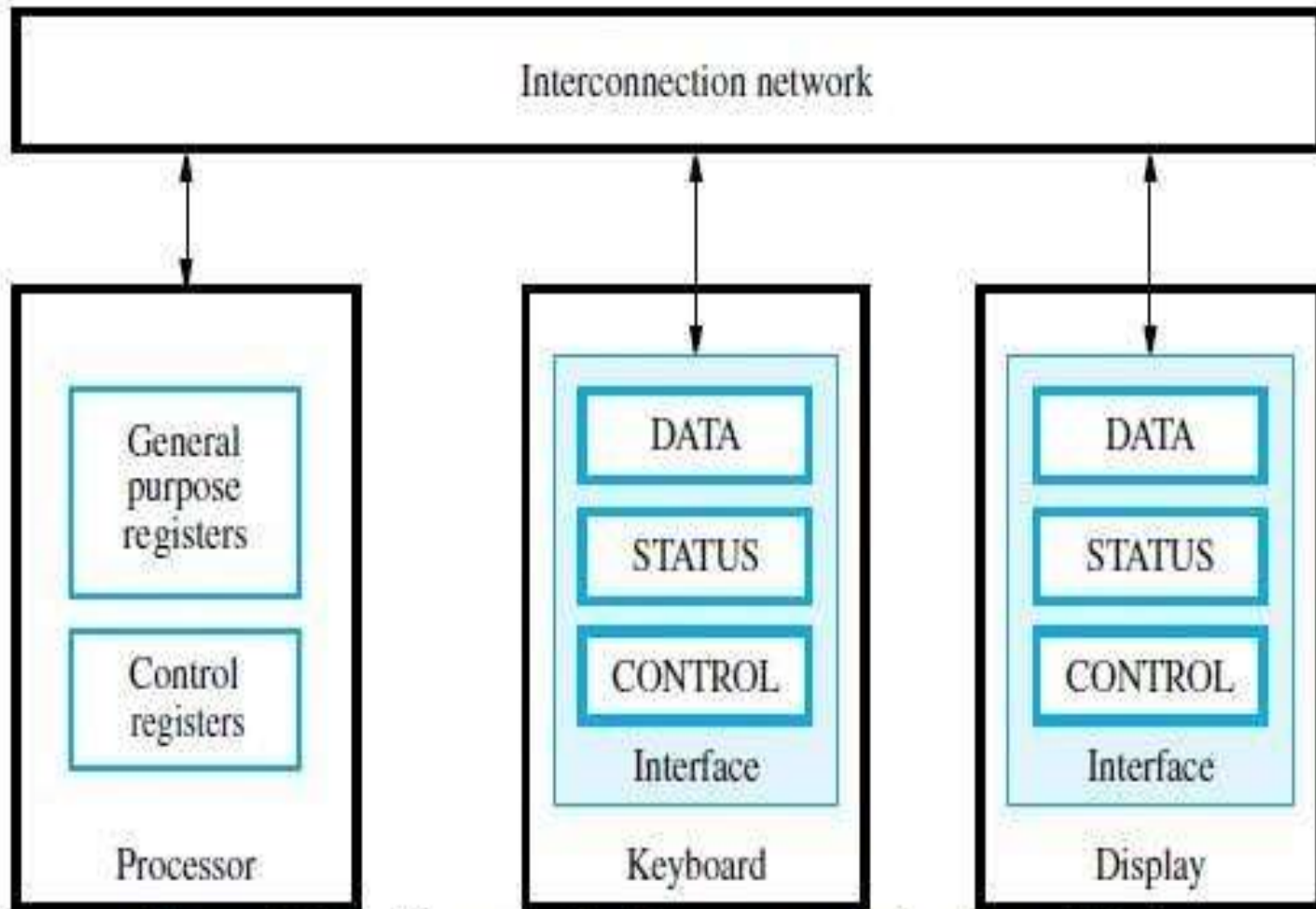


Figure 3.2

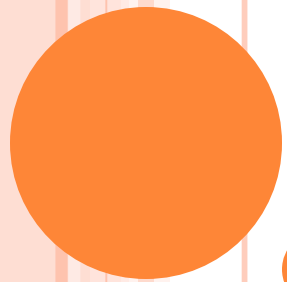
The connection for processor, keyboard, and display.



MECHANISMS USED FOR INTERFACING I/O-DEVICES

- Program Controlled I/O
- Interrupt I/O
- Direct Memory Access(DMA)





MODULE 4

Memory System



CONCEPTS

- Basic Concepts
- Semiconductor RAM Memories
 - Internal organization of memory chips
- Static memories
- Asynchronous DRAMS
- Read Only Memories
- Cache Memories
- Virtual Memories
- Secondary Storage-Magnetic Hard Disks

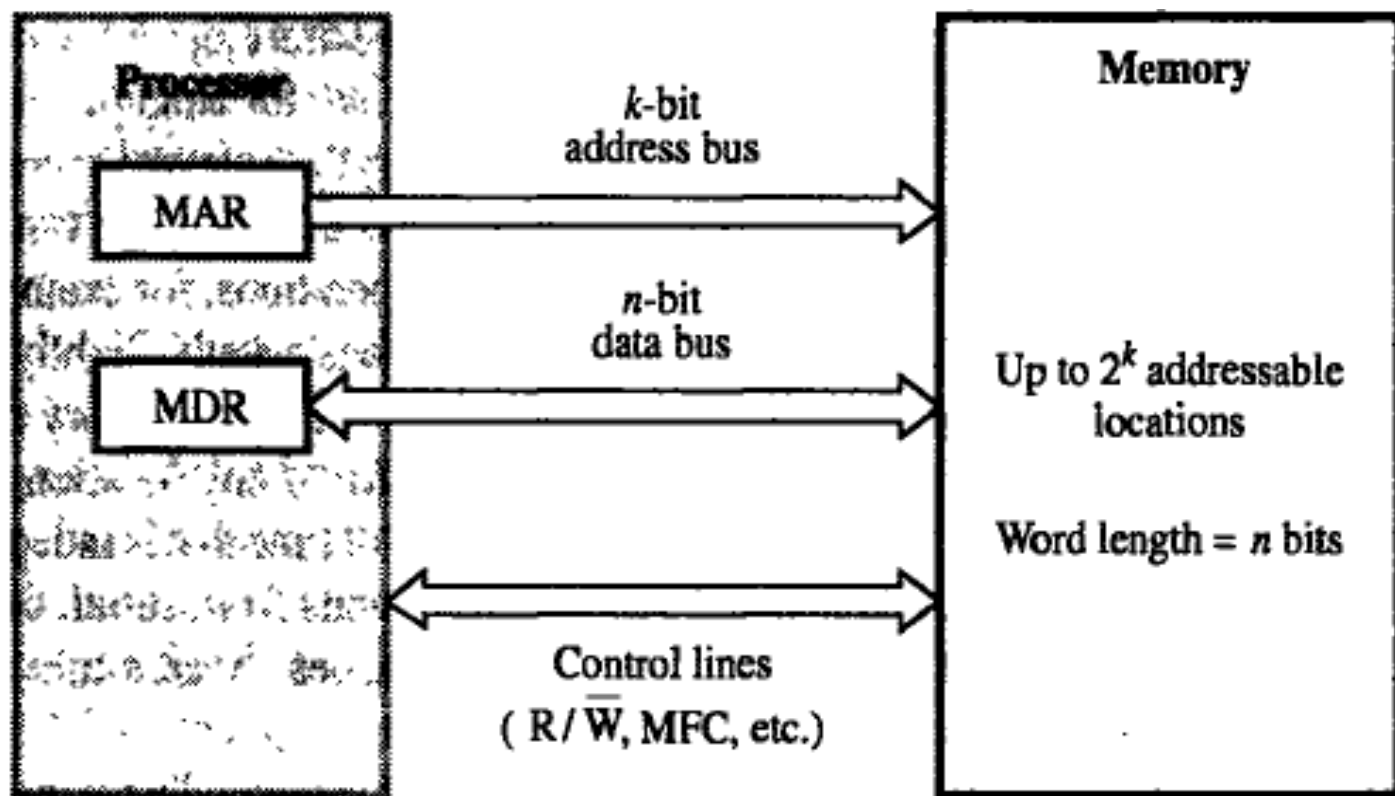


BASIC CONCEPTS

- The maximum size of the Main Memory (MM) that can be used in any computer is determined by its addressing scheme.

Word Address	Byte Address			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11





• **Figure 5.1** Connection of the memory to the processor.



SOME BASIC CONCEPTS

- Memory Access Time
- Memory Cycle Time
- RAM
- Cache Memory
- Memory Interleaving
- Virtual Memory



MEMORY ACCESS TIME

It is the time that elapses between the initiation of an operation and the completion of that operation

MEMORY CYCLE TIME

It is the minimum time delay required between the initiations of two successive memory operations



○ RAM

A memory unit is called a Random Access Memory if any location can be accessed for a READ or WRITE operation in some fixed amount of time that is independent of the location's address

CACHE MEMORY

The CPU of a computer can usually process instructions and data faster than they can be fetched from compatibly priced main memory unit.



○ MEMORY INTERLEAVING

This technique divides the memory system into a number of memory modules and arranges addressing so that successive words in the address space are placed in different modules.

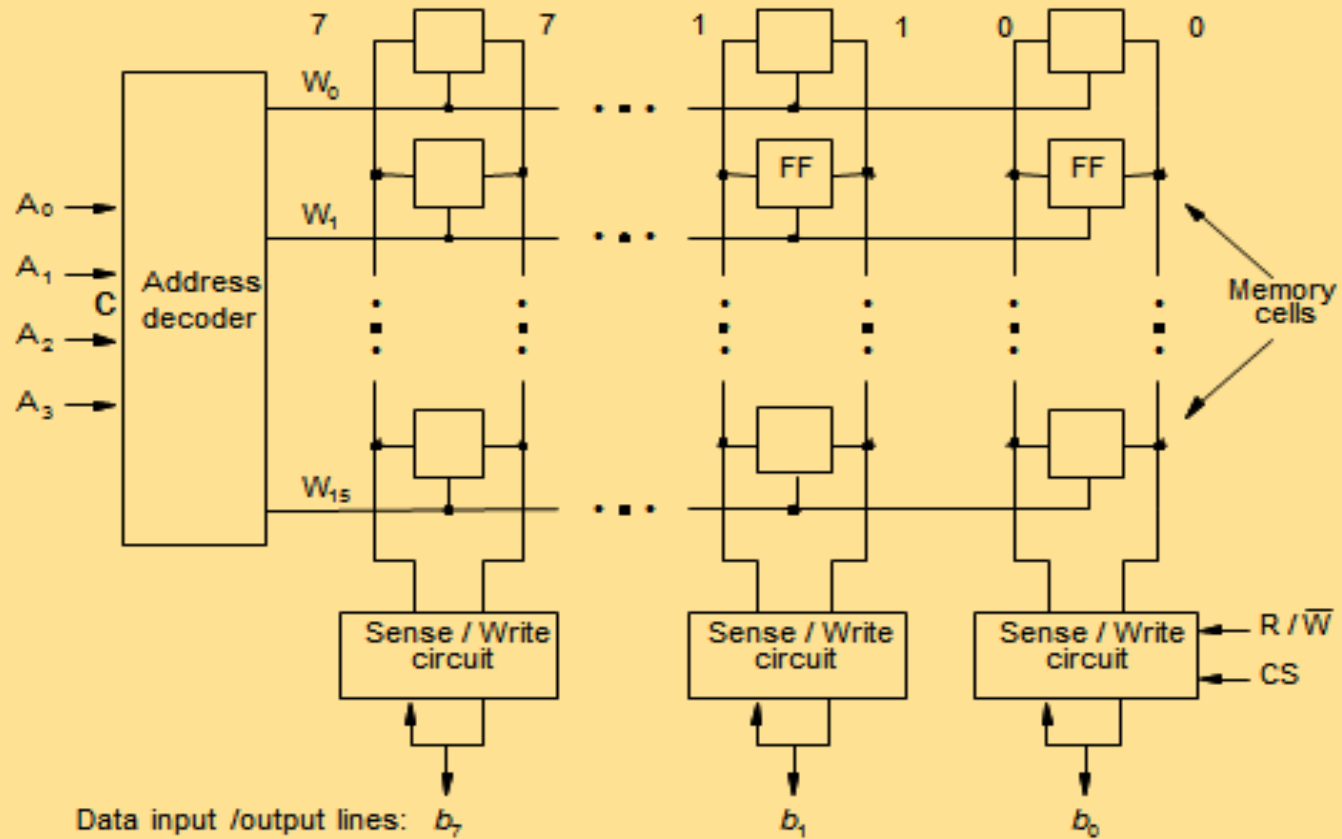


○ VIRTUAL MEMORY

- In a virtual memory System, the address generated by the CPU is referred to as a virtual or logical address.
- The corresponding physical address can be different and the required mapping is implemented by a special memory control unit, often called the memory management unit.



Internal Organization of Memory Chips



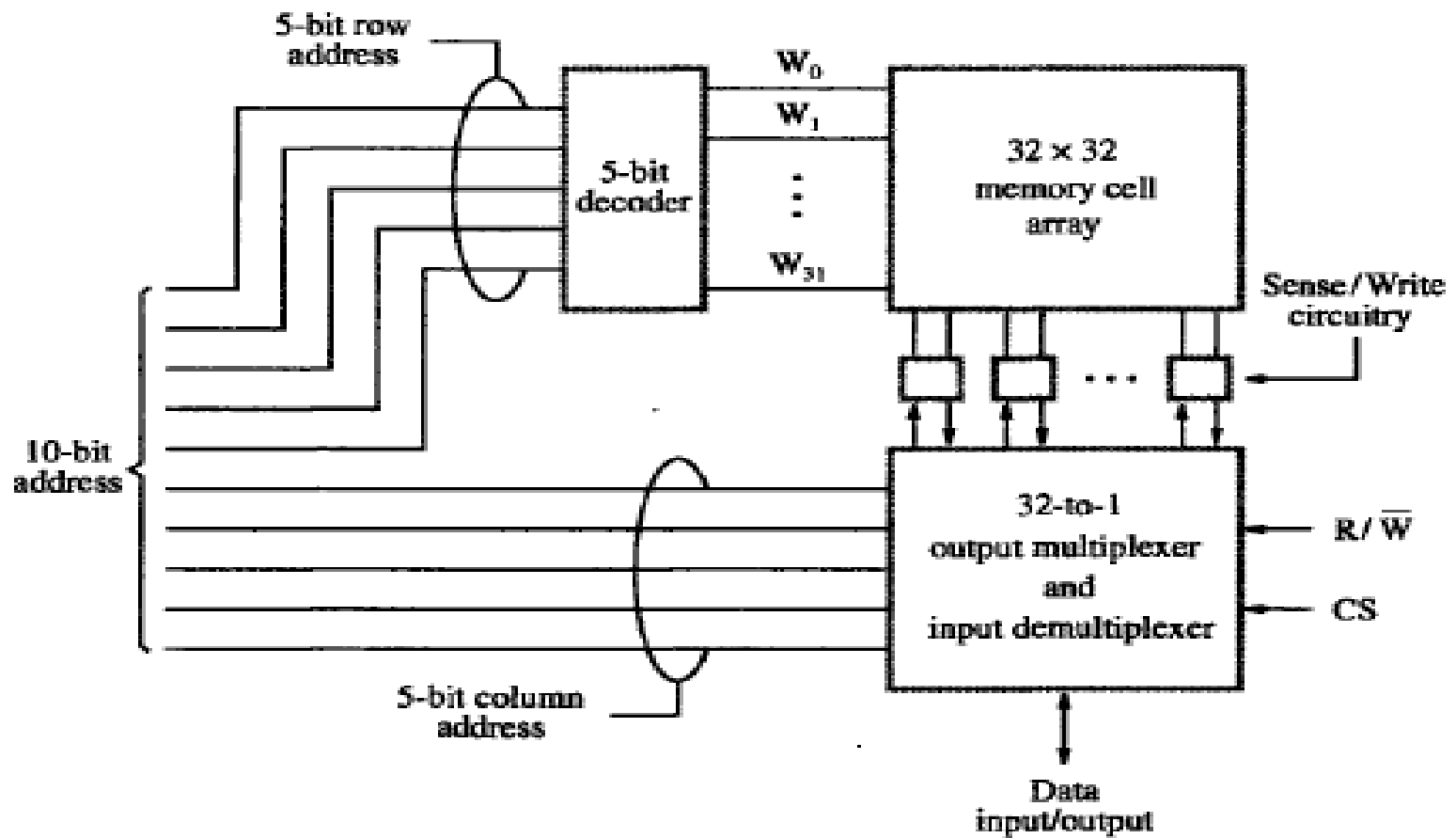


Figure 5.3 Organization of a 1K x 1 memory chip.



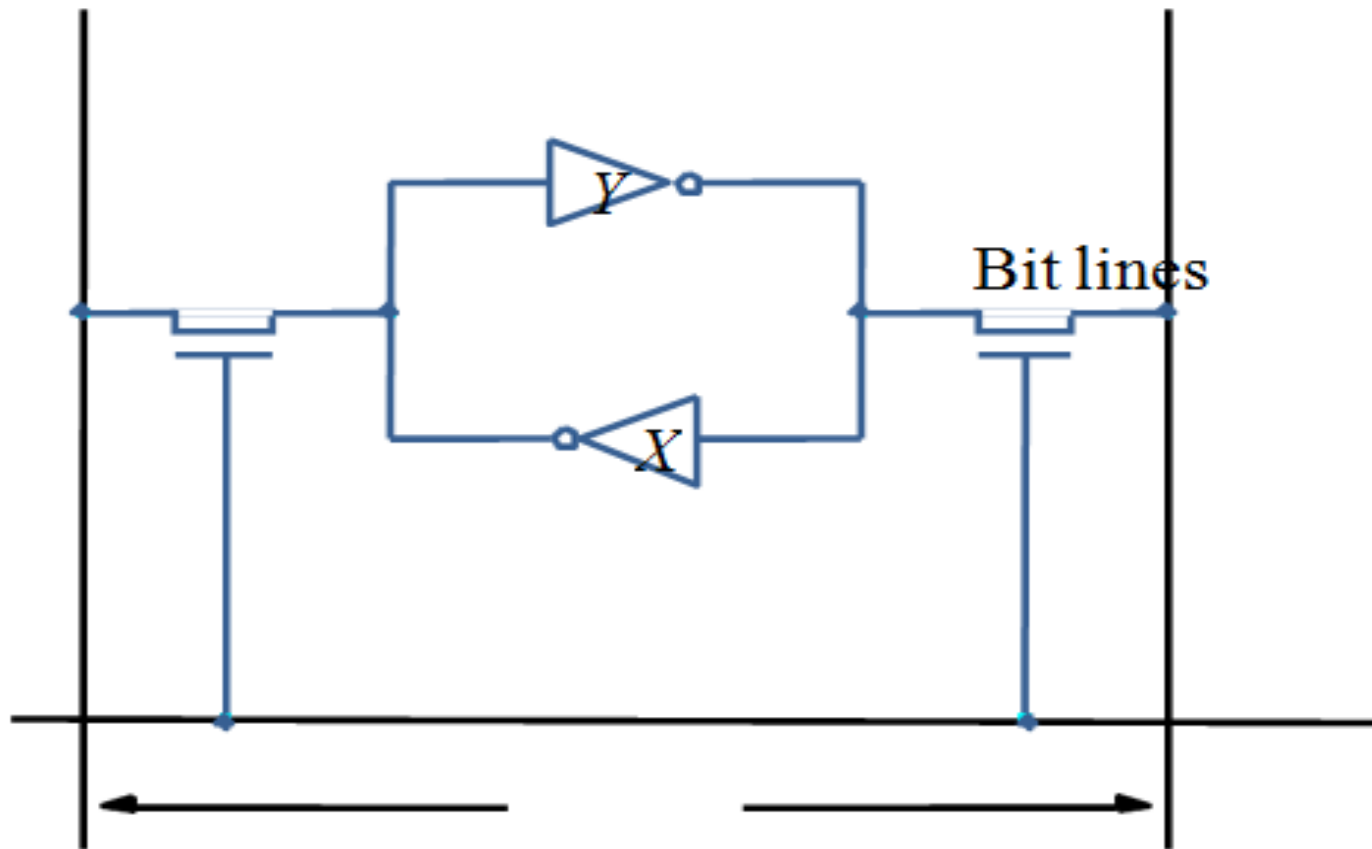
STATIC MEMORIES

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

1. Read Operation

2. Write Operation





CMOS CELL

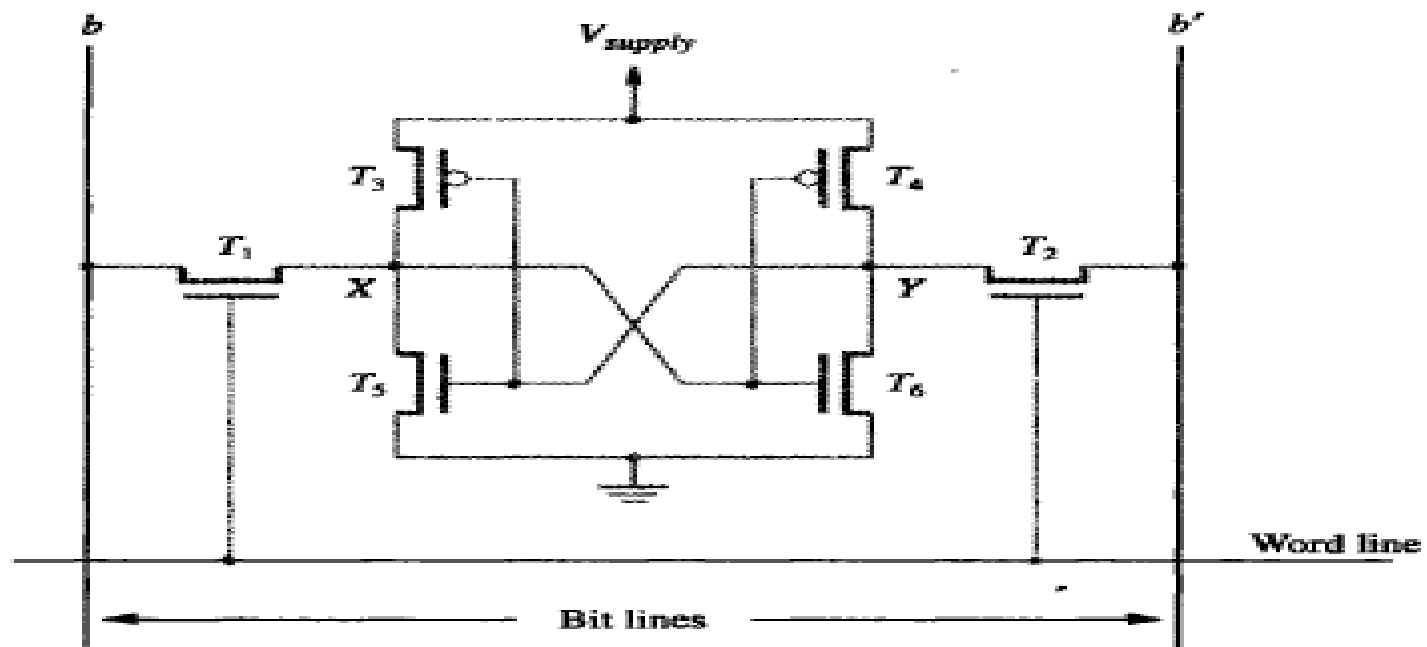


Figure 5.5 An example of a CMOS memory cell.



ASYNCHRONOUS DRAMS

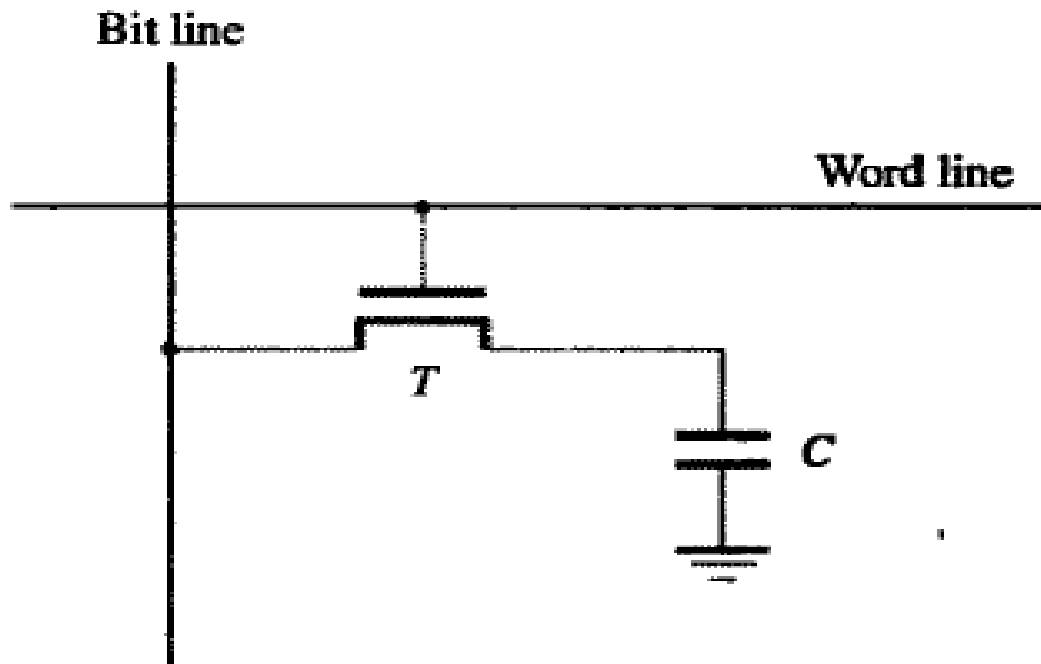
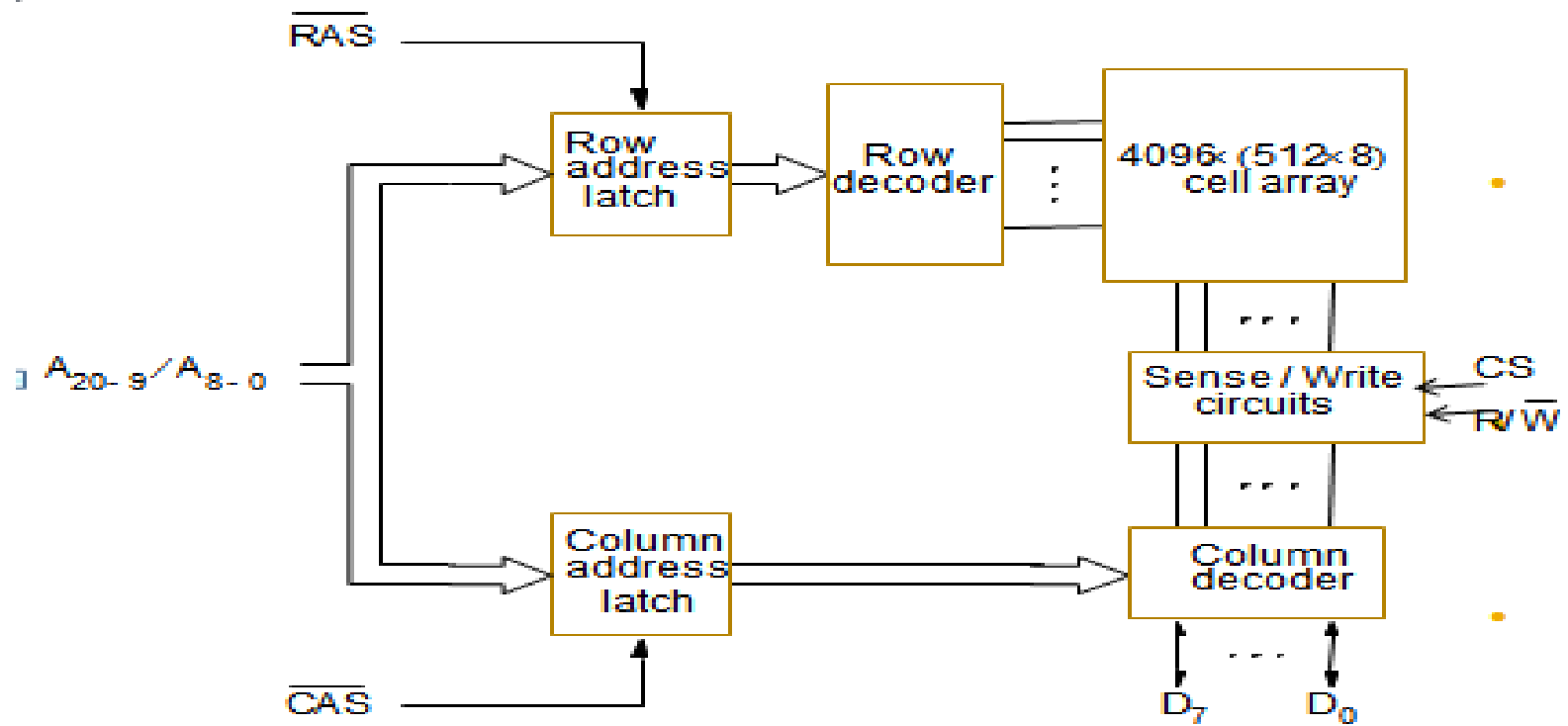


Figure 5.6 A single-transistor dynamic memory cell.



INTERNAL ORGANIZATION OF A 2Mx8 DYNAMIC MEMORY CHIPS



READ ONLY MEMORIES

- ROM
- PROM
- EPROM
- EEPROM
- Flash Memories



ROM(READ ONLY MEMORY

- Semiconductor read-only memory (ROM) units are well suited as the control store components in micro programmed processors and also as the parts of the main memory that contain fixed programs or data.

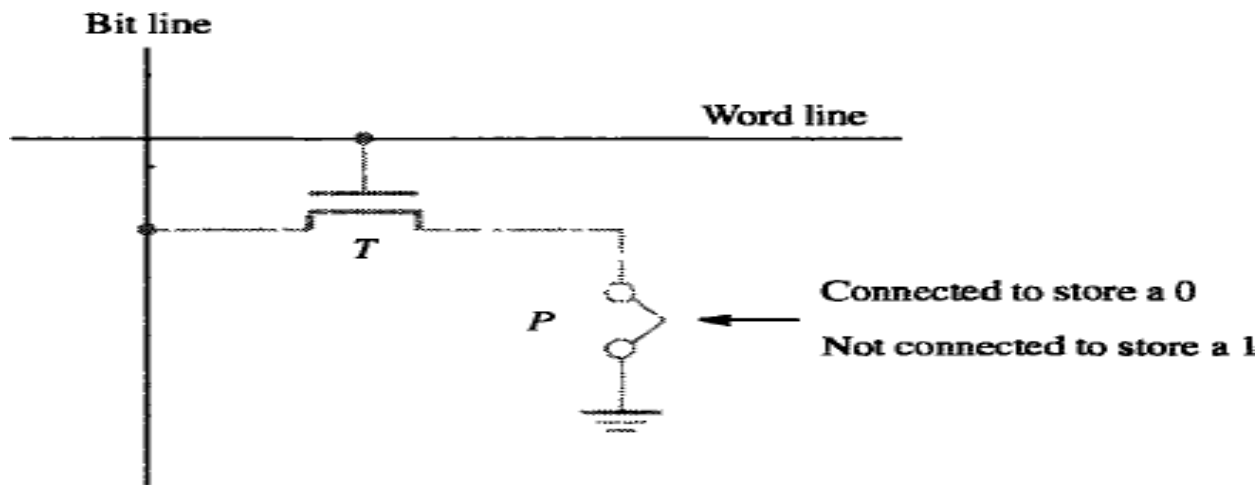


Figure 5.12 A ROM cell.



PROM & EPROM

- Data are written into a ROM at the time of manufacture programmable ROM (PROM) devices a PROMs provide a faster and considerably less expensive approach.
- Chips which allow the stored data to be erased and new data to be loaded. Such a chip is an erasable, programmable ROM, usually called an EPROM allow the data to be loaded by the user.



EEPROM

- The contents of EPROM cells can be erased by increasing the discharge rate of the storage capacity or by several orders of magnitude.
- When electrical erasure is used, however, the process can be made selective. An electrically erasable EPROM, often referred to as EEPROM.



FLASH MEMORY

- Has similar approach to EEPROM.
- Read the contents of a single cell, but write the contents of an entire block of cells.
- Flash devices have greater density.
- Higher capacity and low storage cost per bit.



- Power consumption of flash memory is very low, making it attractive for use in equipment that is battery-driven.
- Single flash chips are not sufficiently large, so larger memory modules are implemented using flash cards and flash drives.



CACHE MEMORIES

- Cache memory is based on the property of computer programs known as “locality of reference”.
- Instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly. This is called “locality of reference”.
Its types are:

Temporal locality of reference

Spatial locality of reference



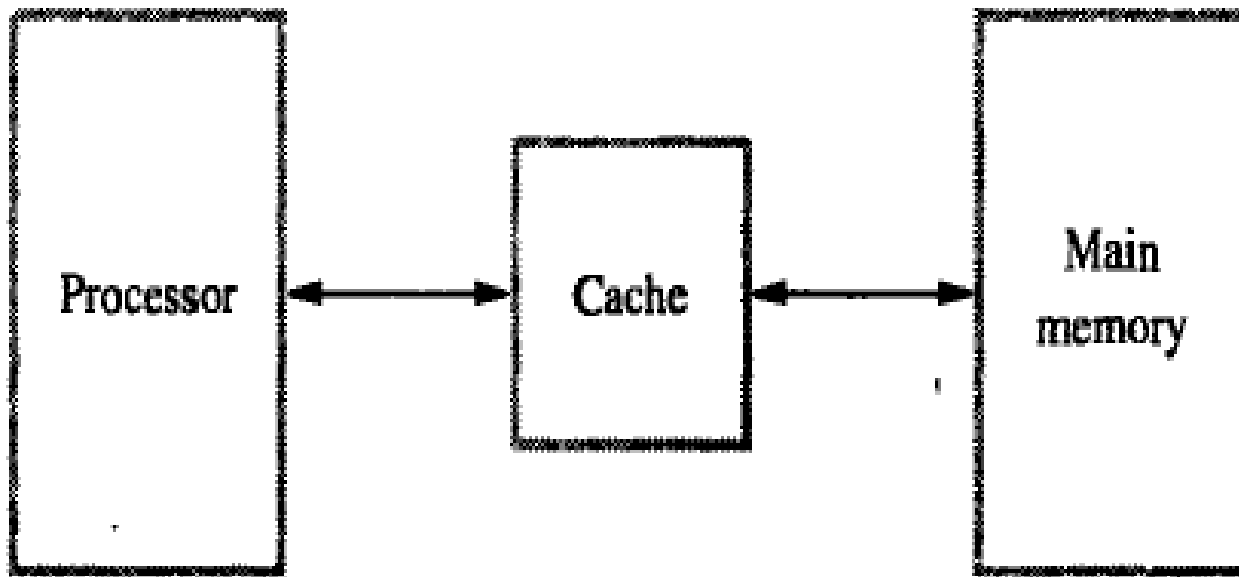


Figure 5.14 Use of a cache memory.



- Cache Hit
- Read Hit
- Write Hit
- Cache Coherence Problem
- Mapping Function
- Cache Miss



VIRTUAL MEMORY

- Virtual memory is an architectural solution to increase the effective size of the memory system.
- Virtual-memory techniques.
- Virtual address



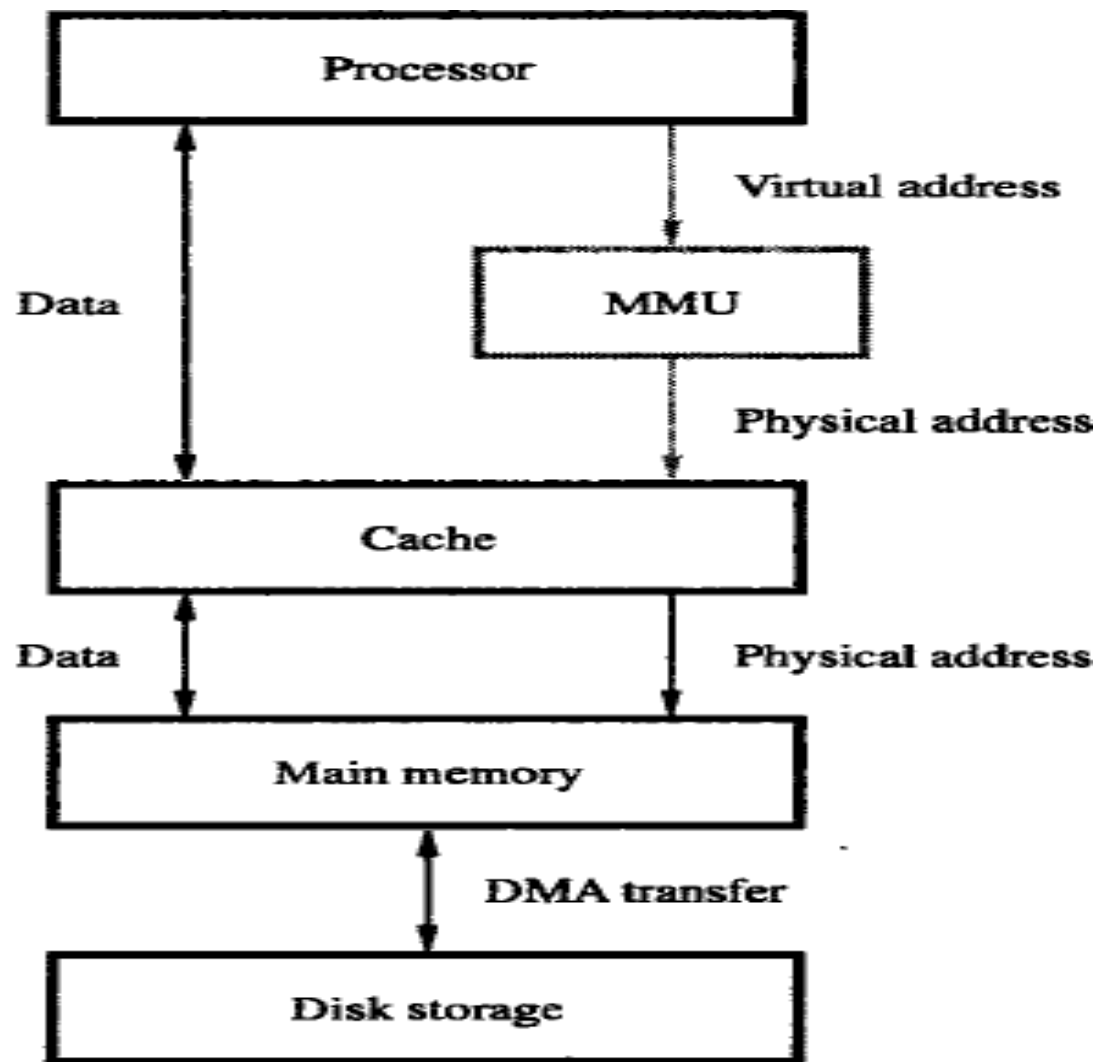


Figure 5.26 Virtual memory organization.



VIRTUAL ADDRESS TRANSLATION

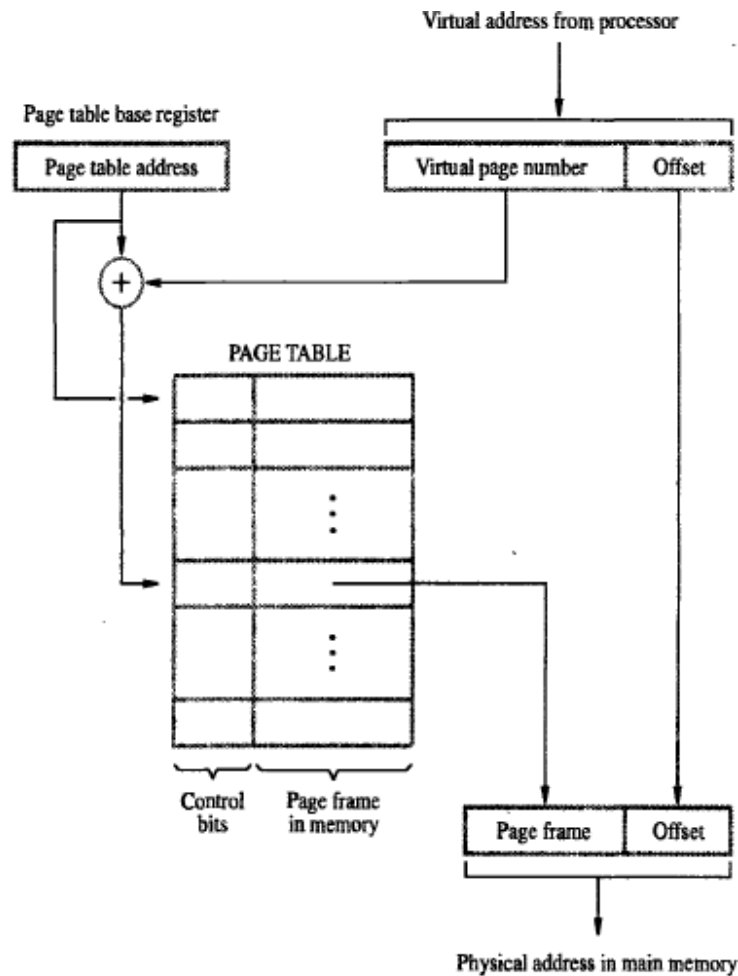


Figure 5.27 Virtual-memory address translation.



VIRTUAL MEMORY ADDRESS TRANSLATION

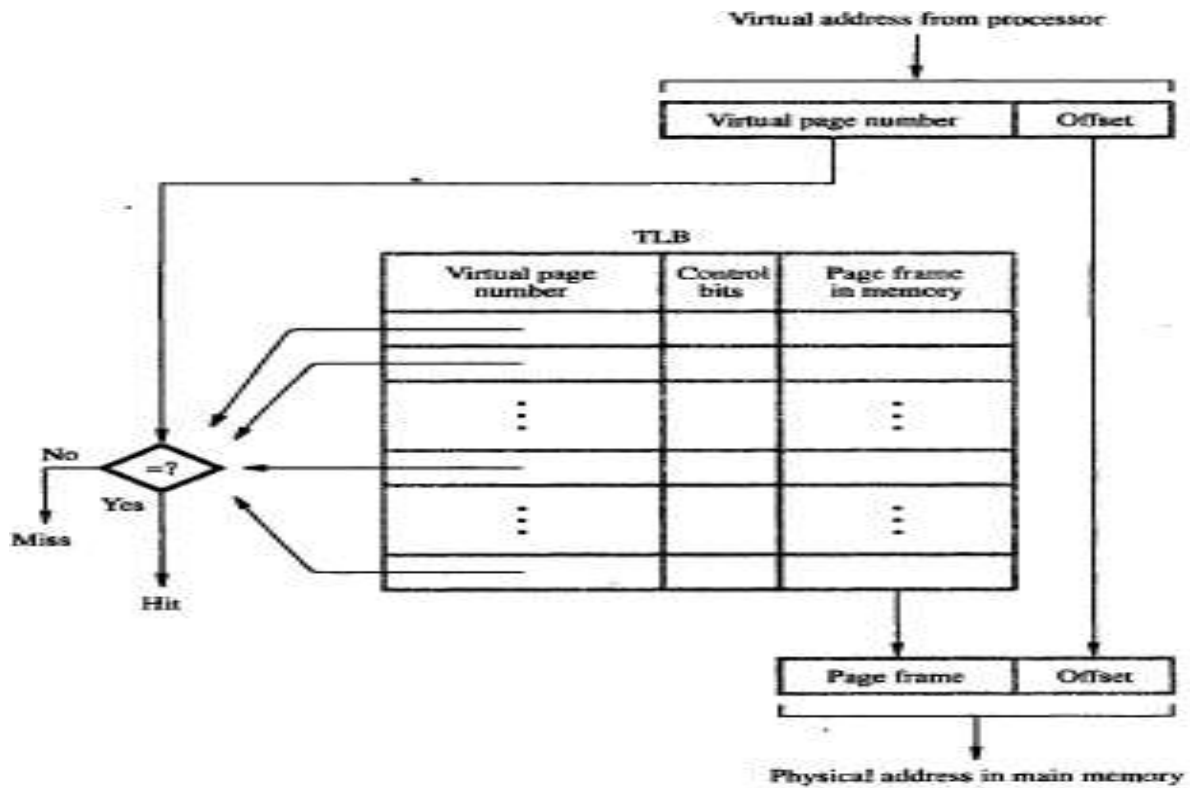


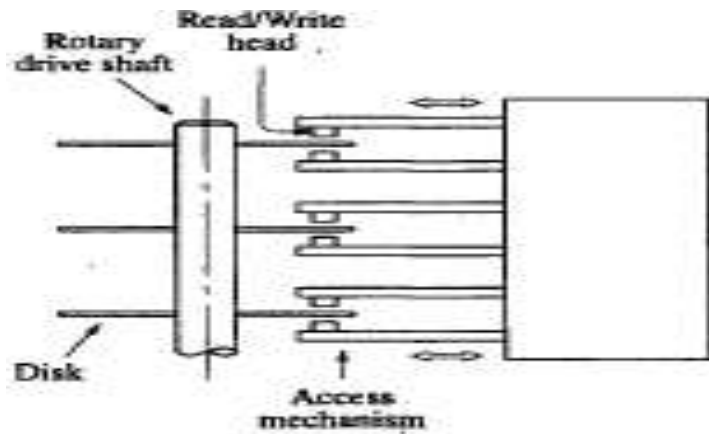
Figure 5.28 Use of an associative-mapped TLB.



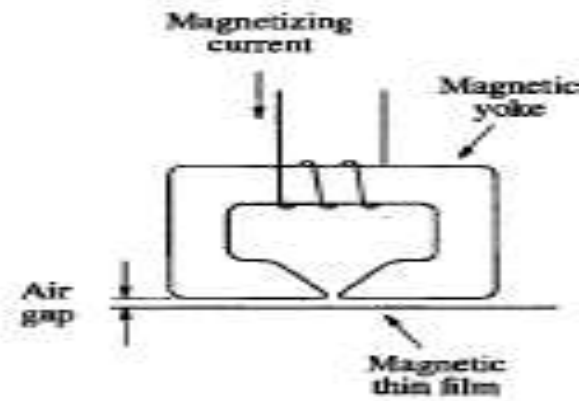
SECONDARY STORAGE DEVICES

1. Magnetic Disk Drives: Hard disk Drive organization
2. **Platters and Read/Write Heads**
3. **Drive Electronics**
4. **Data organization on the Disk**

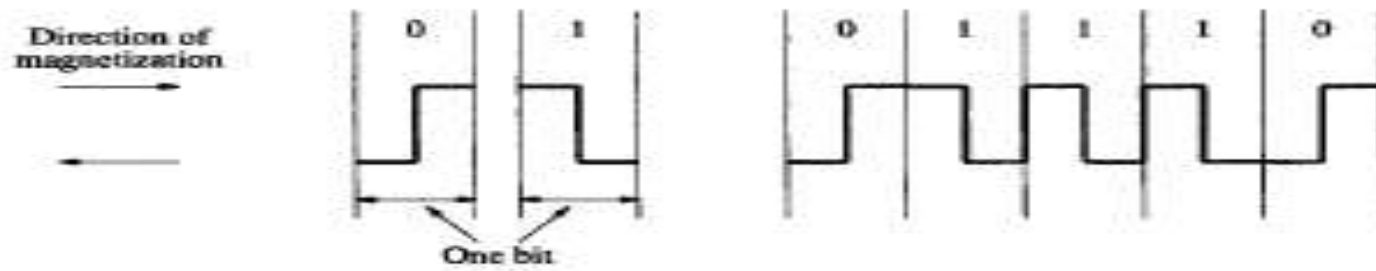




(a) Mechanical structure



(b) Read/Write head detail



(c) Bit representation by phase encoding

Figure 5.29 Magnetic disk principles.



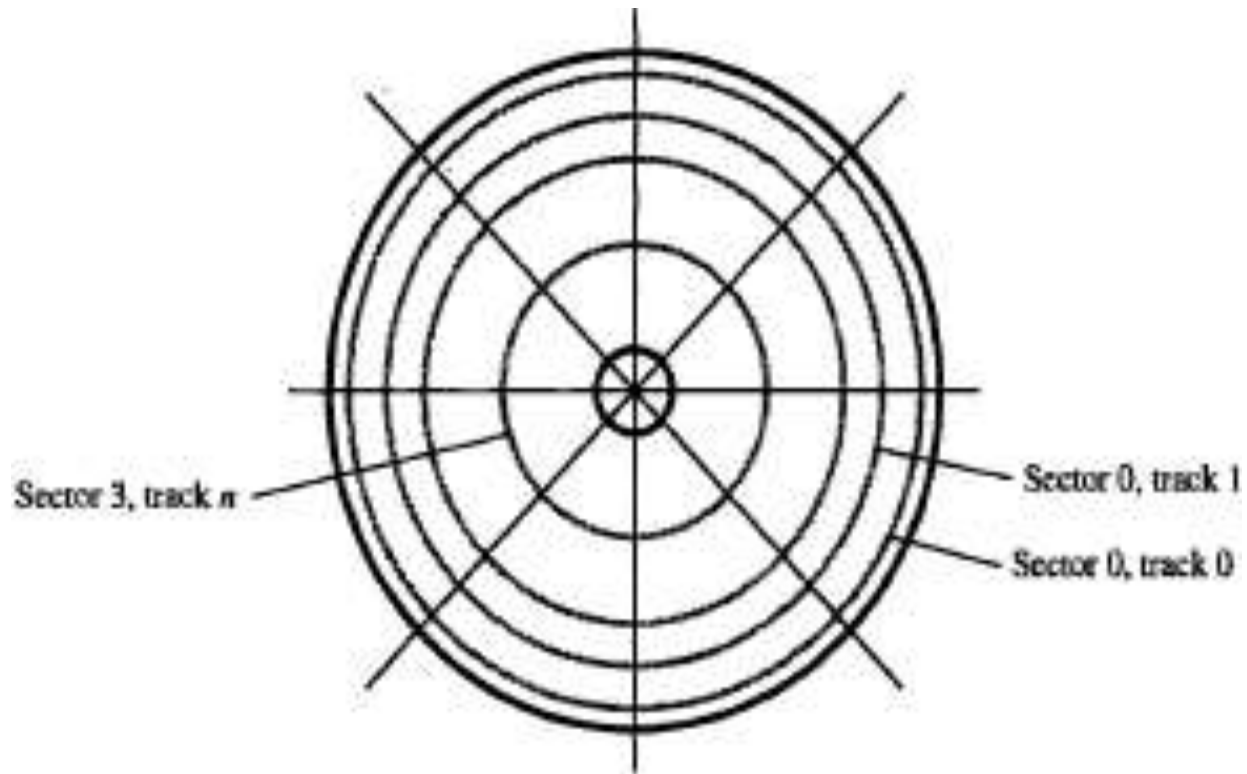


Figure 5.30 Organization of one surface of a disk.



DISK DRIVE DYNAMIC PROPERTIES

- Seek time
- Track to track access time
- Rotational latency
- Average Access time
- Burst rate
- Sustained data rate



OPTICAL DISK

Compact Disk (CD) Technology:- The optical technology that is used for CD system is based on laser light source. A laser beam is directed onto the surface of the spinning disk. Physical indentations in the surface are arranged along the tracks of the disk.

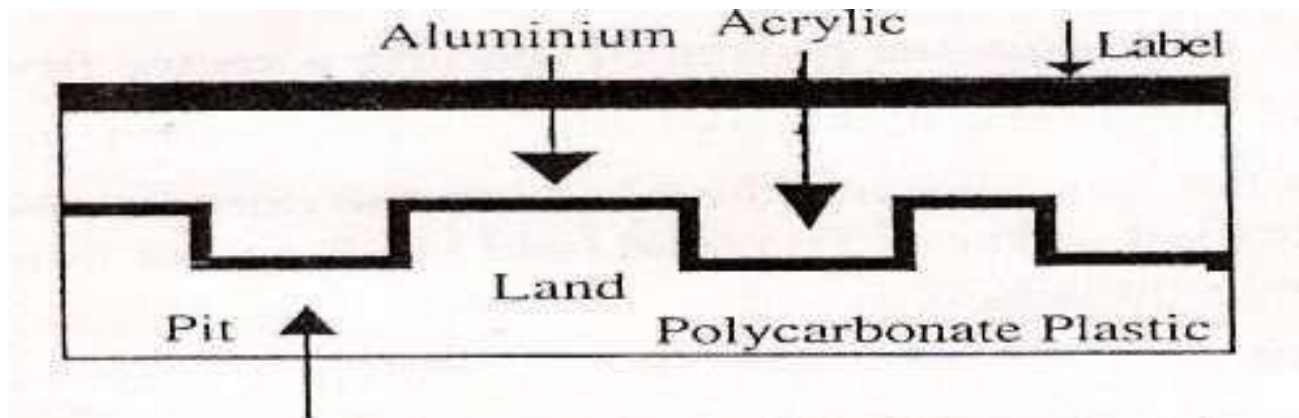


Fig. 16 Optical disk (Cross Section)



Thank you

