Introduction (DBMS)

- Purpose of Database Systems
- View of Data
- Database Languages
- Relational Databases
- Database Design
- Object•based and semistructured databases
- Data Storage and Querying
- Transaction Management
- Database Architecture
- Database Users and Administrators
- Overall Structure
- History of Database Systems

Database Management System

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases touch all aspects of our lives

Purpose of Database Systems

- In the early days, database applications were built directly on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become —buried‖ in program code rather than being stated explicitly
    - Hard to add new constraints or change existing ones

Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent accessed needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
      - Example: Two people reading a balance and updating it at the same time
  - Security problems
    - Hard to provide user access to some, but not all, data
- Database systems offer solutions to all the above problems

Levels of Abstraction

- **Physical level:** describes how a record (e.g., customer) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

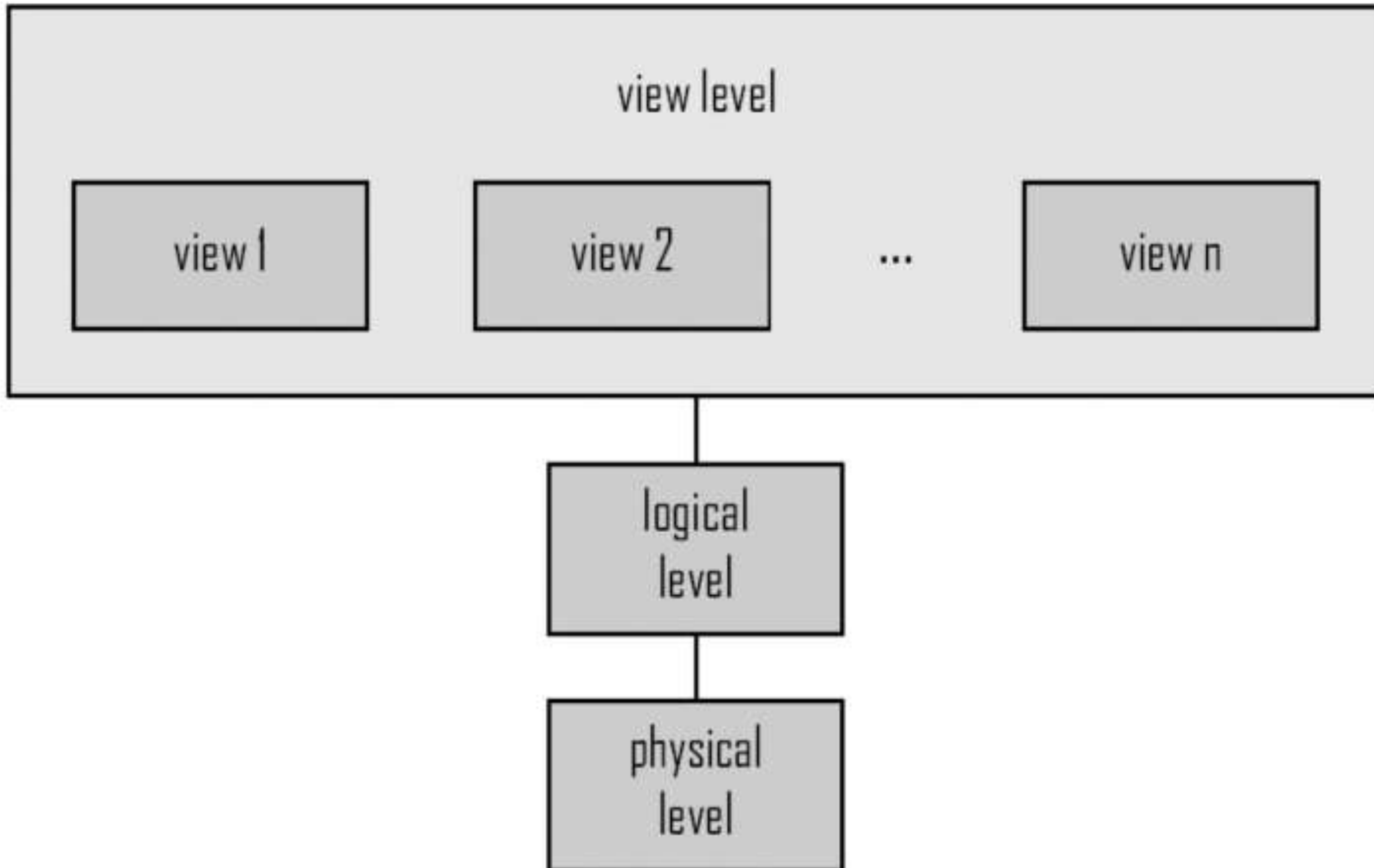
  type *customer* = **record**

  customer_id : string;
  customer_name : string;
  customer_street : string;
  customer_city : integer;

  end;

- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

View of Data



Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - Example: The database consists of information about a set of customers and accounts and the relationship between them)
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity•Relationship data model (mainly for database design)
- Object•based data models (Object•oriented and Object•relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Data Manipulation Language (DML)

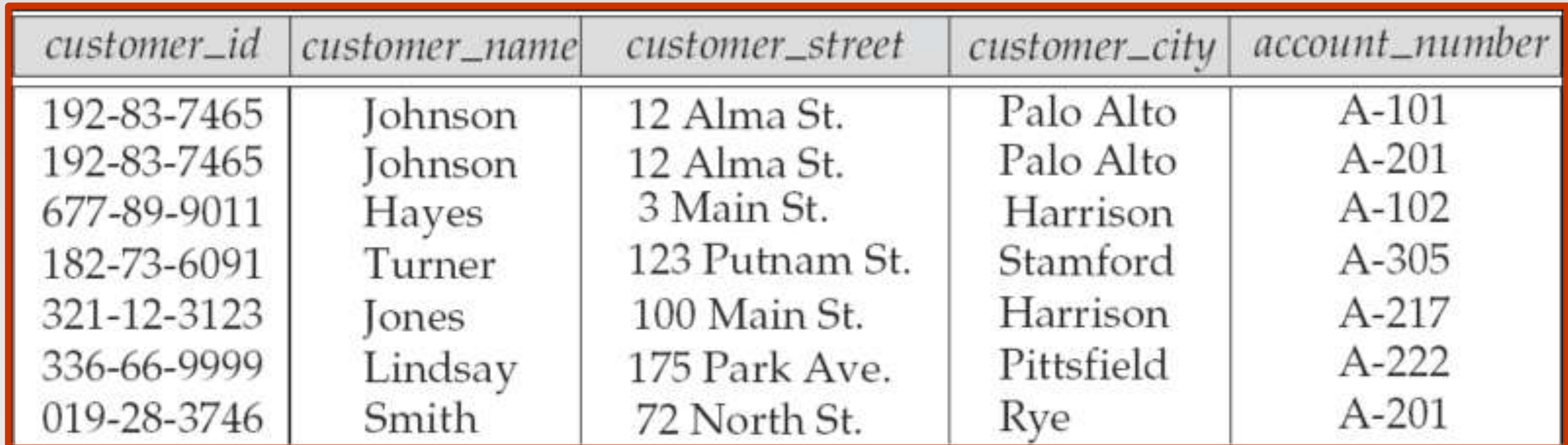
- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
- Two classes of languages
  - **Procedural** – user specifies what data is required and how to get those data
  - **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data
- SQL is the most widely used query language

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *account* (
  account•number **char**(10),
  balance **integer**)
- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Data *storage and definition* language
    - Specifies the storage structure and access methods used
  - Integrity constraints
    - Domain constraints
    - Referential integrity (**references** constraint in SQL)
    - Assertions
  - Authorization

s



customer_id	*customer_name*	*customer_street*	*customer_city*	*account_number*
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

A Sample Relational Database

customer_id	customer_name	customer_street	customer_city
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	balance
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer_id	account_number
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

SQL

- SQL: widely used non•procedural language
  - Example: Find the name of the customer with customer•id 192•83•7465

    select *customer.customer_name*
    from *customer*
    where *customer.customer_id* = ‗192•83•7465‘

  - Example: Find the balances of all accounts held by the customer with customer•id 192•83•7465

    select *account.balance*
    from *depositor*, *account*
    where *depositor.customer_id* = ‗192•83•7465‘ **and**
    depositor.account_number = *account.account_number*

- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

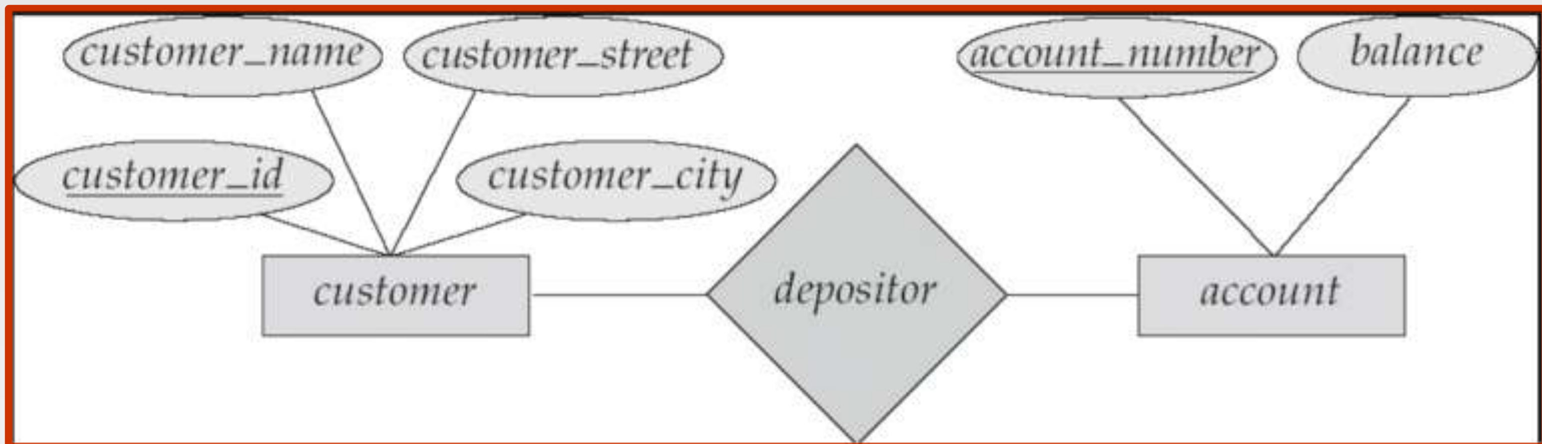
Database Design

The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a ―good‖ collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

The Entity•Relationship Model

- Models an enterprise as a collection of *entities* and *relationships*
  - Entity: a —thing‖ or —object‖ in the enterprise that is distinguishable



Database Architecture

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client•server
- Parallel (multi•processor)
- Distributed

Database Users

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naïve users** – invoke one of the permanent application programs that have been written previously
  - Examples, people accessing database over the web, bank tellers, clerical staff

Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

History of Database Systems

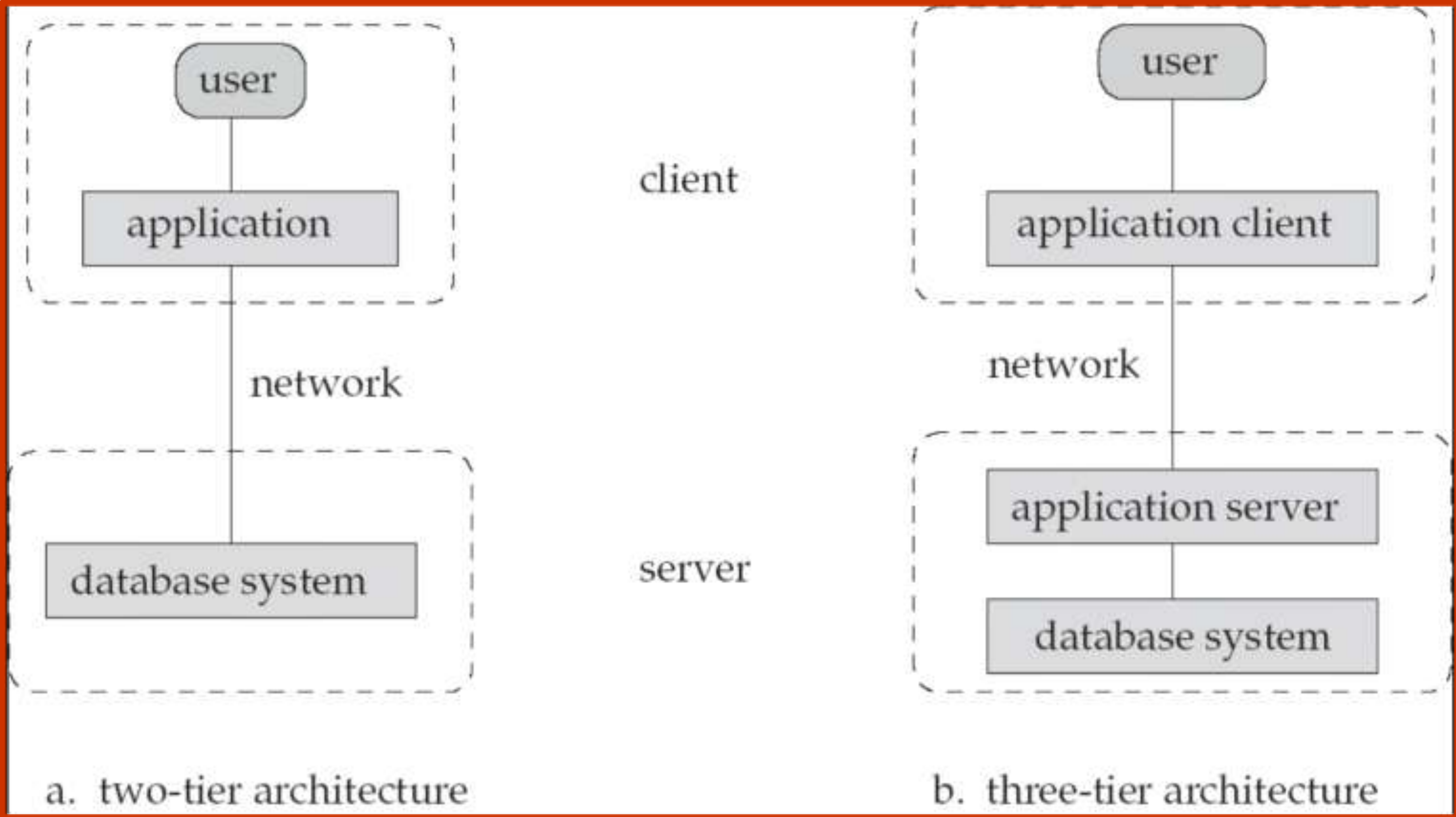
- 1950s and early 1960s:
  - Data processing using magnetic tapes for storage
    - Tapes provide only sequential access
  - Punched cards for input
- Late 1960s and 1970s:
  - Hard disks allow direct access to data
  - Network and hierarchical data models in widespread use
  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for this work
    - IBM Research begins System R prototype
    - UC Berkeley begins Ingres prototype
  - High•performance (for the era) transaction processing

History (cont.)

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
  - Object•oriented database systems
- 1990s:
  - Large decision support and data•mining applications
  - Large multi•terabyte data warehouses
  - Emergence of Web commerce
- 2000s:
  - XML and XQuery standards
  - Automated database administration

Figure

customer_id	*account_number*	*balance*
192-83-7465	A-101	500
192-83-7465	A-201	900
019-28-3746	A-215	700
677-89-9011	A-102	400
182-73-6091	A-305	350
321-12-3123	A-217	750
336-66-9999	A-222	700
019-28-3746	A-201	900


user
application
client
network
database system
server
a. two-tier architecture
user
application client
network
application server
database system
b. three-tier architecture

Data Definition Language

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- The set of indices to be maintained for each relations.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Domain Types in SQL

- **char(n).** Fixed length character string, with user•specified length *n.*
- **varchar(n).** Variable length character strings, with user•specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine•dependent).
- **smallint.** Small integer (a machine•dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user•specified precision of *p* digits, with *n* digits to the right of decimal point.
- **real, double precision.** Floating point and double•precision floating point numbers, with machine•dependent precision.
- **float(n).** Floating point number, with user•specified precision of at least *n* digits.
- More are covered in Chapter 4.

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity•constraint$_1$),
  ...,
  (integrity•constraint$_k$))

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i

- Example:

```
create table branch
    (branch_name    char(15) not null,
    branch_city     char(30),
    assets          integer)
```

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)

Example: Declare *branch_name* as the primary key for *branch*
.

create table *branch*
(*branch_name* char(15),
branch_city char(30),
assets integer,
primary key (*branch_name*))

primary key declaration on an attribute automatically ensures **not null** in SQL•92 onwards, needs to be explicitly stated in SQL•89

Drop and Alter Table Constructs

- The **drop table** command deletes all information about the dropped relation from the database.
- The **alter table** command is used to add attributes to an existing relation:

  alter table *r* **add** *A D*

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
  - All tuples in the relation are assigned *null* as the value for the new attribute.
- The **alter table** command can also be used to drop attributes of a relation:

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*
  - Dropping of attributes not supported by many databases

The select Clause

- The **select** clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all branches in the *loan* relation:

  select *branch_name*
  from *loan*

- In the relational algebra, the query would be:

$$_{branch_name}(loan)$$

- NOTE: SQL names are case insensitive (i.e., you may use upper• or lower•case letters.)
  - E.g. *Branch_Name* *BRANCH_NAME* *branch_name*
  - Some people use upper case wherever we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select**.**
- Find the names of all branches in the *loan* relations, and remove duplicates

  select distinct *branch_name*
  from *loan*

- The keyword **all** specifies that duplicates not be removed.

  select all *branch_name*
  from *loan*

The select Clause (Cont.)

- An asterisk in the select clause denotes "all attributes"

```
select *
from loan
```

- The **select** clause can contain arithmetic expressions involving the operation, +, –, , and /, and operating on constants or attributes of tuples.
- The query:

```
select loan_number, branch_name, amount 100
from loan
```

would return a relation that is the same as the *loan* relation, except that the value of the attribute *amount* is multiplied by 100.

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

  select *loan_number*
  from *loan*
  where *branch_name* = 'Perryridge' **and** *amount* > 1200

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

The where Clause (Cont.)

- SQL includes a **between** comparison operator
- Example: Find the loan number of those loans with loan amounts between $90,000 and $100,000 (that is, $90,000 and $100,000)

select *loan_number*
from *loan*
where *amount* **between** 90000 **and** 100000

The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *borrower X loan*

select
from *borrower, loan*

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

select *customer_name, borrower.loan_number, amount*
from *borrower, loan*
where *borrower.loan_number = loan.loan_number* **and**
branch_name = 'Perryridge'

String Operations

- SQL includes a string•matching operator for comparisons on character strings. The operator —like‖ uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all customers whose street includes the substring —Main‖
.

```
select customer_name
from customer
where customer_street like '% Main%'
```

- Match the name —Main%‖

```
like 'Main\%' escape '\'
```

- SQL supports a variety of string operations such as
  - concatenation (using —||‖)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all customers having a loan in Perryridge branch

  select distinct *customer_name*
  from *borrower, loan*
  where *borrower loan_number* = *loan.loan_number* **and**
  branch_name = 'Perryridge'
  order by *customer_name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *customer_name* **desc**

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

Aggregate Functions (Cont.)

- Find the average account balance at the Perryridge branch.

  select avg *(balance)*
  from *account*
  where *branch_name* = 'Perryridge'

- Find the number of tuples in the *customer* relation.

  select count (*)
  from *customer*

- Find the number of depositors in the bank.

  select count (distinct *customer_name)*
  from *depositor*

Aggregate Functions – Group By

- Find the number of depositors for each branch.

select *branch_name*, **count (distinct** *customer_name*)
from *depositor*, *account*
where *depositor.account_number* = *account.account_number*
group by *branch_name*

Note: Attributes in **select** clause outside of aggregate functions must appear in **group by** list

Aggregate Functions – Having Clause

- Find the names of all branches where the average account balance is more than $1,200.

```
select branch_name, avg (balance)
       from account
       group by branch_name
       having avg (balance) > 1200
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.
  - Example: Find all loan number which appear in the *loan* relation with null values for *amount.*

    select *loan_number*
    from *loan*
    where *amount* **is null**
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- However, aggregate functions simply ignore nulls
  - More on next slide

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - Example*: 5 < null or null <> null or null = null*
- Three•valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown,*
    (false **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
  - **—***P* **is unknown**‖ evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Null Values and Aggregates

- Total all loan amounts

  select sum (*amount*)
  from *loan*

  - Above statement ignores null amounts
  - Result is *null* if there is no non•null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

Modification of the Database – Updates

- Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.
  - Write two **update** statements:

    update *account*
    set *balance* = *balance* 1.06
    where *balance* > 10000

    update *account*
    set *balance* = *balance* 1.05
    where *balance* 10000

  - The order is important
  - Can be done better using the **case** statement (next slide)

Chapter Topics

- The Basics of a C++ Program
- Data Types
- Arithmetic Operators and Operator Precedence
- Expressions
- Input
- Increment and Decrement Operators
- Output
- Preprocessor Directives
- Program Style and Form
- More on Assignment Statements

The Basics of a C++ Program

- A C++ program is a collection of one or more subprograms (functions)
- Function
  - Collection of statements
  - Statements accomplish a task
- Every C++ program has a function called **`main`**

Example Program

```
#include <iostream>
using namespace std;
int main()
{
  cout<<"Welcome to C++ Programming"<<endl;
  return 0;
}
```

```
Welcome to C++ Programming
```

Program Output

The Basics of a C++ Program

- Programming language
  - a set of rules, symbols, special words
- Rules
  - syntax – specifies legal instructions
- Symbols
  - special symbols (**+ - * !** …)
- Word symbols
  - reserved words
  - (**int, float, double, char** …)

Identifiers

- Rules for identifiers
  - must begin with letter or the underscore _
  - followed by any combination of numerals or letters
  - recommend meaningful identifiers
- Evaluate the following

```
ElectricCharge
23Skidoo
snarFbLat
```

C++'s Data Types

Simple Structured Pointers

Data Types

- Simple data types include
  - Integers
  - Floating point
  - Enumeration
- Integer data types include

char

short

int

long

bool

Numerals, symbols,

Numbers without decimals

Values *true* and *false* only

75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-S
-1.482	-1.482000EO

Floating-Point
Data Type

float double long double

-2147483648 to 2147483647 4

bool true and false 1

Data Types

- The `string` Type
  - a programmer-defined type
  - requires `#include <string>`
- A string is a sequence of characters

```
"Hi Mom"
"We're Number 1!"
"75607"
```

Arithmetic Operators and Operator Precedence

- Common operators for calculations
  + - * / %
- Precedence same as in algebraic usage
  - Inside parentheses done first
  - Next * / % from left to right
  - Then + and - from left to right
- Note operator precedence chart, page 1035

Expressions

- An expression includes
  - constants
  - variables
  - function calls
  - combined with operators

```
3 / 2 + 5.0
sin(x) + sqrt(y)
```

Expressions

- Expressions can include
  - values all of the same type
    `3 + 5 * 12 – 7`
  - values of different (compatible) types
    `1.23 * 18 / 9.5`
- An operation is evaluated according to the types of the operands
  - if they are the same, the result is the type of the operands
  - if the operands are different (int and float) then the result is float

Type Casting

- Implicit change of type can occur
  - when operands are of different type
- It is possible to explicitly specify that an expression be converted to a different type

```
static_cast < type > (expression)

static_cast <int> (3.5 * 6.9 / x)
```

Input

- Storing data in the computer's memory requires two steps

1. Allocate the memory by declaring a variable
2. Have the program fetch a value from the input device and place it in the allocated memory location

123

x

```
cin >> x
```

Allocating Memory

- Variable
  - A memory location whose content may change during program execution
- Declaration:
  - Syntax:

```
type identifier;
```

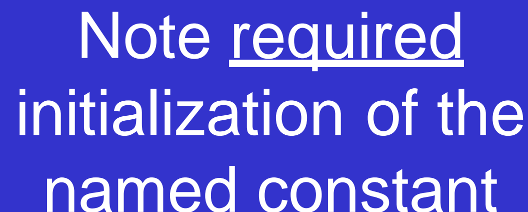
  - Example:

```
double x;
int y = 45;
```

Note <u>optional</u> initialization of the variable

Allocating Memory

- Named Constant
  - A memory location whose content cannot be changed
- Declaration
  - Syntax:
    ```
    const type identifier = value;
    ```
  - Example
    ```
    const  double PI = 3.14159;
    ```



Putting Data Into Variables

- At initialization time
- Assignment statement
  - Syntax:
    ```
    variable = expression;
    ```
  - Example
    ```
    x = 1.234;
    volume = sqr (base) * height;
    ```
- Input (read) statement
  - Syntax:
    ```
    cin >> variable ;
    ```
  - Example
    ```
    cin >> height;
    ```

Program Example

Increment and Decrement Operators

- Pre-increment `++x;`
  equivalent to `x = x + 1;`
  - Pre-decrement `--x;`
  - Changes the value before execution of a statement `y = ++x;`
- Post-increment `intVal++;`
  - Post-decrement `intVal--;`
  - Changes the value after execution of the statement `y = x++;`

Output

- Values sent to an output device
  - Usually the screen
  - Can also be a file or some device
- Syntax for screen output:

```
cout << expression << …
```

- Example

```
cout << "The total  is "<< sum << endl;
```

Output command

Insertion operator

Sample
ogram

Values
be prin

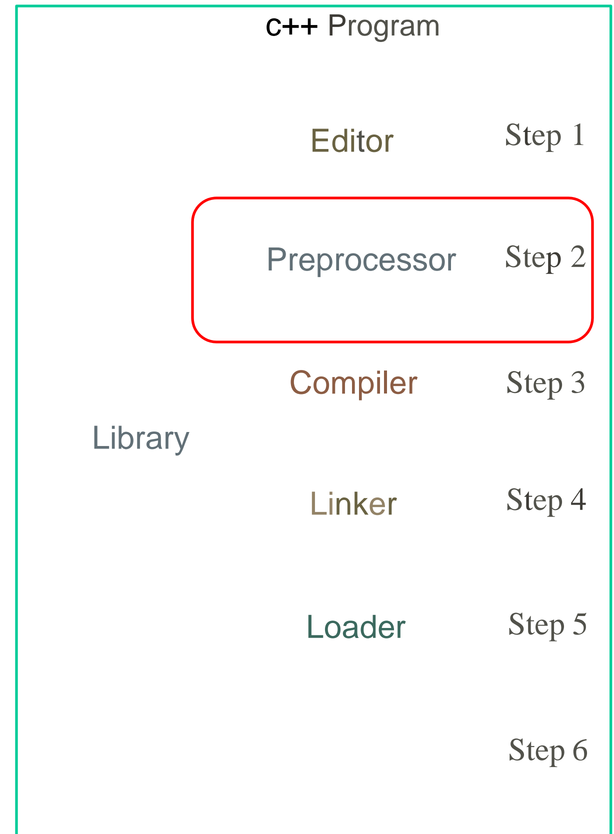
Manipulator for carriage return

```
cout << "The total is\t "<< sum << endl;
```

\n		Cursor moves to the beginning of the next line
\t	Tab	Cursor moves to the next tab stop
\b	Backspace	Cursor moves one space to the left
\r	Return	Cursor moves to the beginning of the current line (not the next line)
\\	Backslash	Backslashis printed
\'	Single quotation	Single quotation mark is printed
	Double quotation	Double quotation mark is printed

Preprocessor Directives

- Commands supplied to the preprocessor
  - Runs before the compiler
  - Modifies the text of the source code before the compiler starts
- Syntax
  - start with **`#`** symbol
  - **`#include <headerFileName>`**
- Example **`#include <iostream>`**



C++ Program
Editor
Step 1
Preprocessor
Step 2
Compiler
Step 3
Library
Linker
Step 4
Loader
Step 5
Step 6

Namespace

- The `#include <iostream>` command is where `cin` and `cout` are declared
- They are declared within a namespace called `std`
- When we specify
  `using namespace std;`
  - Then we need not preface the `cin` and `cout` commands with `std::cin` and `std::cout`

Program Style and Form

- Every program must contain a function called main

```
int main (void)
{   …  }
```

- The `int` specifies that it returns an integer value
- The `void` specifies there will be no arguments
- Also can say

```
void main( )
{  …  }
```

Program Style and Form

- Variables usually declared
  - inside main
  - at beginning of program
- Use blanks and space to make the program easy for humans to read
- Semicolons ; required to end a statement
- Commas used to separate things in a list

Program Style and Form

- Documentation
  - Comments specified between
    `/*   this is a comment   */`
    and following `//   also a comment`
  - Always put at beginning of program

```
/* name,
date,
cpo,
purpose of program
*/
```

Program Style and Form

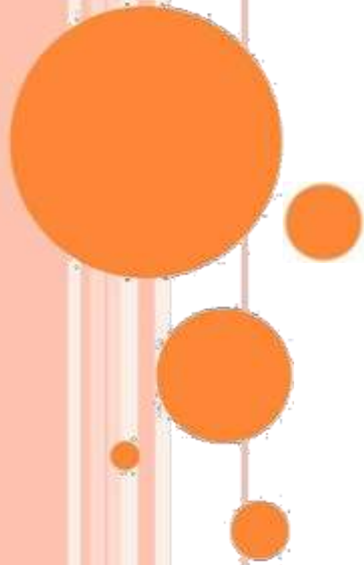
- Names of identifiers should help document program

```
double  electricCharge;
// instead of ec
```

- Prompt keyboard entry

```
cout << "Enter the value for x -> ";
cin >> x;
```

DISTRIBUTED COMPUTING



CONTENTS:-

- Overview
- History
- Introduction
- Working of Distributed system
- Types
- Motivation
- goals
- characteristics
- architecture
- example
- Advantages
- Disadvantages
- Conclusion
- Reference

OVERVIEW DISTRIBUTED COMPUTING

- A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing.
- In the term distributed computing, the word distributed means spread out across space. Thus, distributed computing is an activity performed on a distributed system.
- These networked computers may be in the same room, same campus, same country, or in different country.

History

- The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s.
- The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s.
- The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its European counterpart International Symposium on Distributed Computing (DISC) was first held in 1985.

INTRODUCTION

(a)

(b)

In distributed system each processor have its own memory. The computational entities are called computers or nodes.

In distributed computing a program is split up into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing.

WORKING OF DISTRIBUTED SYSTEM:

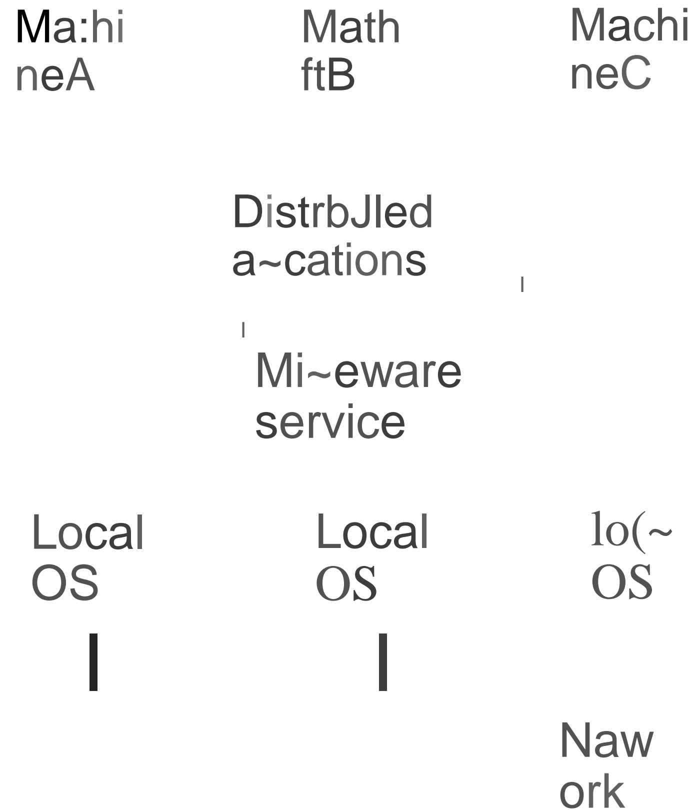


Fig. A Distributed System

TYPES OF DISTRIBUTED COMPUTING:

- **Grid computing**

  Multiple independent computing clusters which act like a "grid" because they are composed of resource nodes not located within a single administrative domain. (formal)

  The creation of a "virtual supercomputer" by using spare computing resources within an organization.

- **Cloud computing**

  Cloud computing is a computing paradigm shift where computing is moved away from personal computers or an individual application server to a "cloud" of computers. Users of the cloud only need to be concerned with the computing service being asked for, as the underlying details of how it is achieved are hidden. This method of distributed computing is done through pooling all computer resources together and being managed by software rather than a human.

MOTIVATION

The main motivations in moving to a distributed system are the following:

- Inherently distributed applications.
- Performance/cost.
- Resource sharing.
- Flexibility and extensibility.
- Availability and fault tolerance.
- Scalability.

GOALS

- **Making Resources Accessible**

  The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.

- **Distribution Transparency**

  An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers.

- **Openness**

  An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.

- **scalability**

  Scalability of a system can be measured along at least three different dimensions.

CHARACTERISTICS

- **Resource Sharing:-** Resource sharing is the ability to use any hardware, software or data anywhere in the system.
- **Openness:-** Openness is concerned with extensions and improvements of distributed systems.
- **Concurrency:-** Concurrency arises naturally in distributed systems from the separate activities of users, the independence of resources and the location of server processes in separate computers.
- **Scalability:-** Scalability concerns the ease of the increasing the scale of the system (e.g. the number of processor) so as to accommodate more users and/or to improve the corresponding responsiveness of the system.
- **Fault tolerance:-** Fault tolerance cares the reliability of the system so that in case of failure of hardware, software or network, the system continues to operate properly, without significantly degrading the performance of the system.
- **Transparency:-** Transparency hides the complexity of the distributed systems to the users and application programmers.

ARCHITECTURE

Examples of distributed systems

Examples of distributed systems and applications of distributed computing include the following:

- Telecommunication networks:
  - Telephone networks and cellular networks
  - Computer networks such as the Internet
- Network applications:
  - World wide web and peer-to-peer networks
  - Massively multiplayer online games and virtual reality communities
- Real-time process control:
  - Aircraft control systems
  - Industrial control systems
- Parallel computation:
  - Scientific computing, including cluster computing and grid computing and various volunteer computing projects
  - Distributed rendering in computer graphics

ADVANTAGES

- Economics
- Speed
- Inherent distribution of applications
- Reliability
- Extensibility and Incremental Growth
- Distributed custodianship
- Data integration
- Missed opportunities

DISADVANTAGES

- Complexity
- Network problem
- Security

CONCLUSION

In this age of optimization everybody is trying to get optimized output from their limited resources. The concept of distributed computing is the most efficient way to achieve the optimization. In case of distributed computing the actual task is modularized and is distributed among various computer system. It not only increases the efficiency of the task but also reduce the total time required to complete the task.

DATA STRUCTURES

UNIT-1

Learning Outcomes:

- Understand the properties of various data structures.
- Identify the strengths and weaknesses of different data structures.
- Various Complexity notations of an algorithm.
- Logic behind various Searching and sorting Algorithms.

Data Type and Data Structure

Data type

- Set of possible values for variables
- Operations on those values

Ex : int, float, char ……….

Data Structure

A data structure is an arrangement of data in a computer's memory or even disk storage.

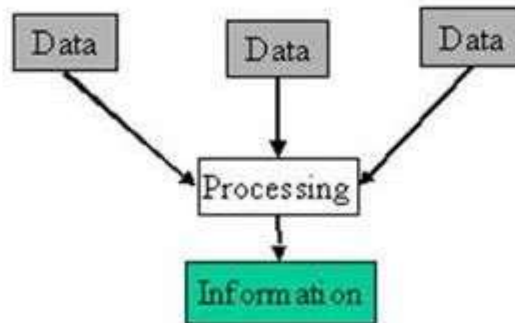
The logical and mathematical model of a particular organization of data is called a data structure.

A **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Data & Information

Data - Data usually refers to raw data, or unprocessed data. It is the basic form of data, data that hasn't been analyzed or processed in any manner.

Information - Once the data is analyzed, it is considered as information.



Linear and Non-Linear Data structures

- **Linear data structure**: A data structure is said to be linear if the elements form a sequence, for example Array, Linked list, stack ,queue etc.

- **Non-Linear data structure**: Elements in a nonlinear data structure do not form a sequence, for example Tree, graph.

Array

Linear array (One dimensional array) : A list of finite number n of similar data elements referenced respectively by a set of n consecutive numbers, usually 1, 2, 3,.....n. That is a specific element is accessed by an index.

Let, Array name is A then the elements of A is : $a_1, a_2 \ldots\ldots A_n$ or by the bracket notation A[1], A[2], A[3],............., A[n].The number k in A[k] is called a subscript and A[k] is called a subscripted variable.

Fundamental data structure

- HOMOGENEOUS collection of values (all the same type)
- store values sequentially in memory
- associate INDEX with each value
- use array name and index to quickly access kth **element.**

Example

A linear array STUDENT consisting of the name of six students

STUDENT

1	Atul Kumar
2	Sumona
3	Richa
4	Kamaljeet
5	Avleen
6	Jatin

Here, STUDENT[4] denote Kamaljeet

Array (con…)

Linear arrays are called one dimensional arrays because each element in such an array is referenced by one subscript.

(Two dimensional array) : Two dimensional array is a collection of similar data elements where each element is referenced by two subscripts.

Such arrays are called matrices in mathematics.

Multidimensional arrays are defined analogously

MATRICES

	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Here, MATRICES[3,3]=11

Array (con…)

Array Data Structure

1. It can hold multiple values of a single type.
2. Elements are referenced by the array name and an *ordinal* index.
3. Each element is a *value*
4. Indexing begins at zero in C.
5. The array forms a contiguous list in memory.
6. The name of the array holds the address of the first array element.
7. We specify the array size at compile time, often with a named constant or at run time using calloc(), malloc() & realloc() in C and new in C++.

Arrays – Advantage & disadvantage

1. Fast element access.
2. Impossible to resize. Many applications require resizing so linked list was introduced.

Linked lists

- A linked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.

- Dynamically allocate space for each element as needed.

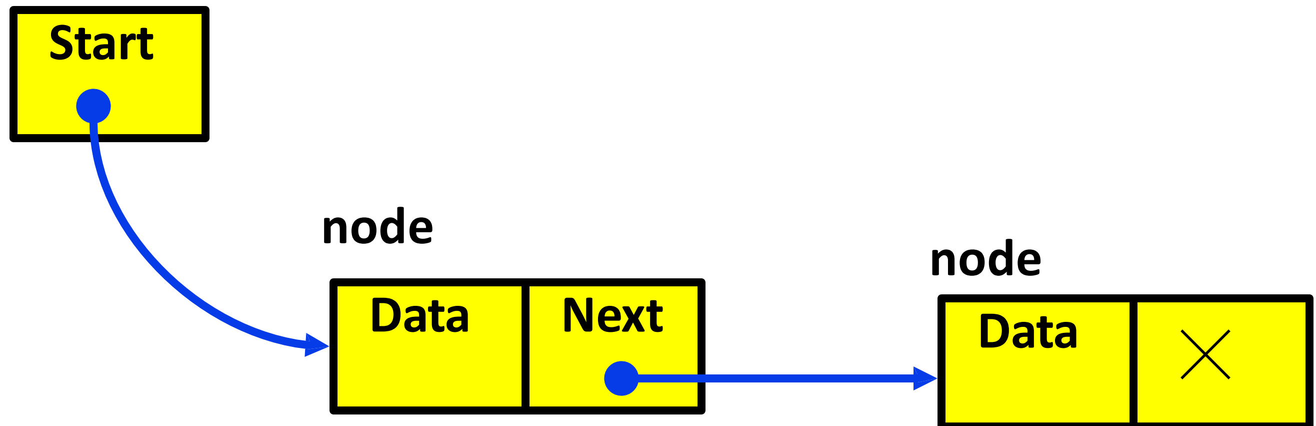
Node

In linked list –

1. Each node of the list contains the data item and a pointer to the next node.

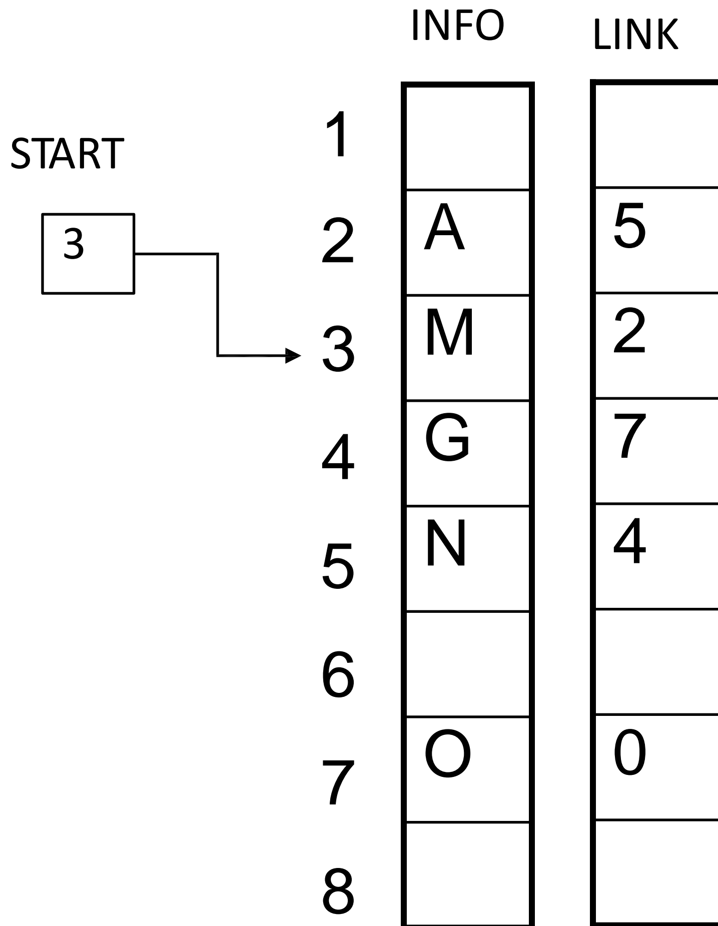
2. There is a pointer to the list Start which contains address of first node.
   Initially NULL

Linked lists (con…)



Linked list with 2 nodes

Linked lists (con…)



START=3, INFO[3]=M

LINK[3]=2, INFO[2]=A

LINK[2]=5, INFO[5]=N

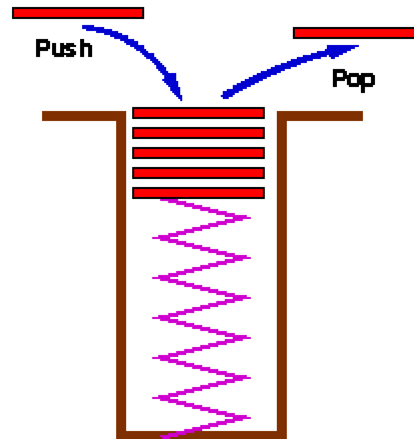
LINK[5]=4, INFO[4]=G

LINK[4]=7, INFO[7]=O

LINK[7]=0, NULL value, So the list has ended

Stacks

- Stacks are a special form of data structures where insertion & deletion of items takes place at one end called TOP of the stack.
- Two methods
  - add item to the top of the stack
  - remove an item from the top of the stack
- Like a plate stacker
- Follows LIFO (Last in First Out) principle.

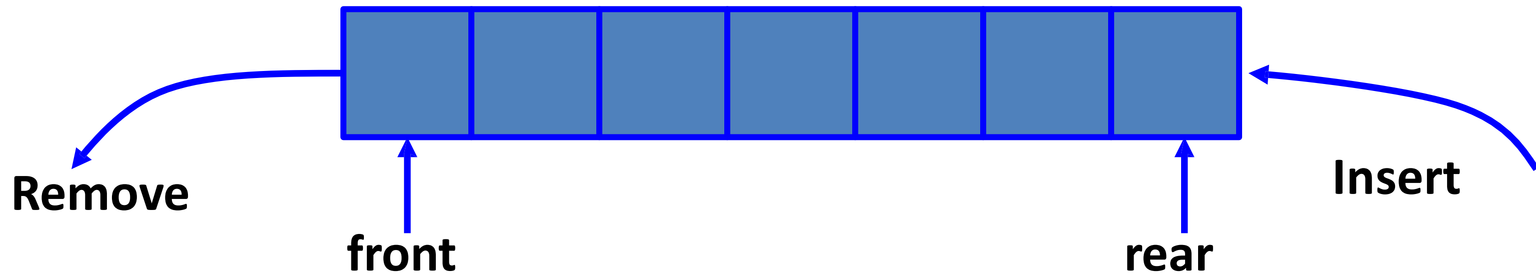


Queues

- Like a stack, a queue is also a list. However, with a queue, insertion is done at one end, while deletion is performed at the other end.

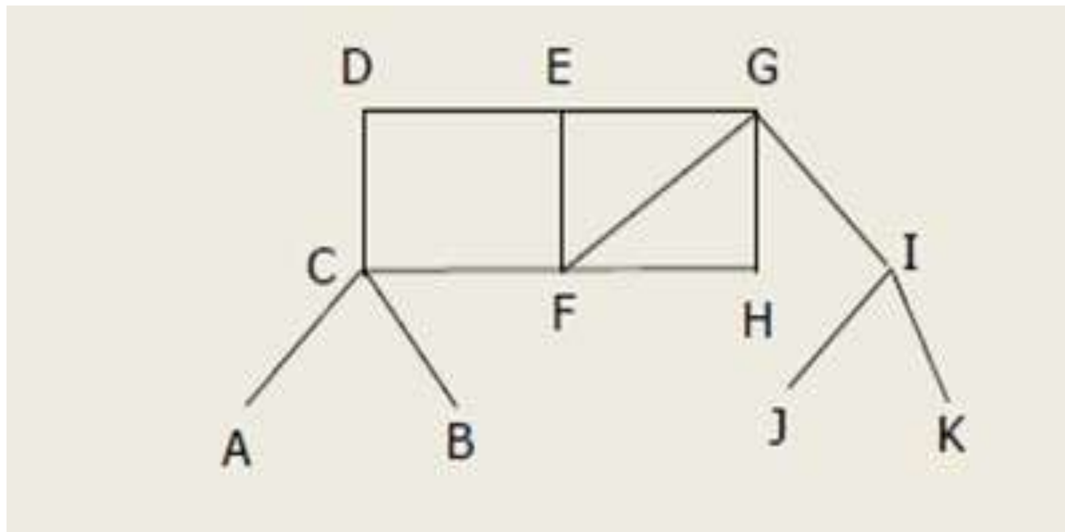
  -- The insertion end is called rear

  -- The deletion end is called front



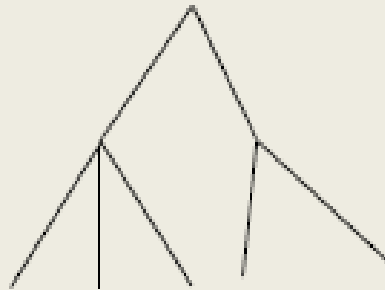
Graph

- A graph data structure consists of vertices and edges.
- For Ex, A network of computers is represented by a graph.

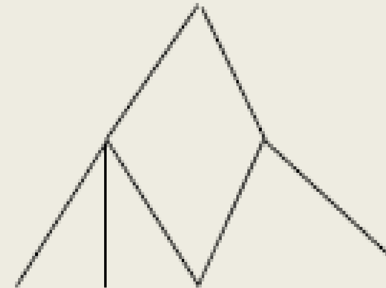


Tree

- A tree is a graph that does not contain any cycles.



A tree



Not a tree

Data structure operations

The data appearing in our data structure is processed by means of certain operations. In fact, the particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed. The following four operations play a major role:

Traversing
Accessing each record exactly once so that certain items in the record may be processed. (This accessing or processing is sometimes called 'visiting" the records.)

Searching
Finding the location of the record with a given key value, or finding the locations of all records, which satisfy one or more conditions.

Inserting
Adding new records to the structure.

Deleting
Removing a record from the structure.

Data structure operations (Con..)

The following two operations, which are used in special situations, will also be considered:

Sorting:

Arranging the records in some logical order

Merging:

Combining the records in two different sorted files into a single sorted file.

Linear Arrays

A linear array is a list of finite number n of homogeneous data elements(i.e. data elements of same type)

a) The elements of the array are referenced respectively by an index set

b) The elements of the array are stored respectively in successive memory locations.

Let, Array name is A then the elements of A is : a1,a2….. an
Or by the bracket notation A[1], A[2], A[3],…………., A[n]

1	247
2	56
3	429
4	135
5	87
6	156

DATA[1] = 247

DATA[2] = 56

DATA[3] = 429

DATA[4] = 135

DATA[5] = 87

DATA[6] = 156

Representation of linear array in memory

Let LA be a linear array in the memory of the computer. The memory of the computer is a sequence of addressed locations.

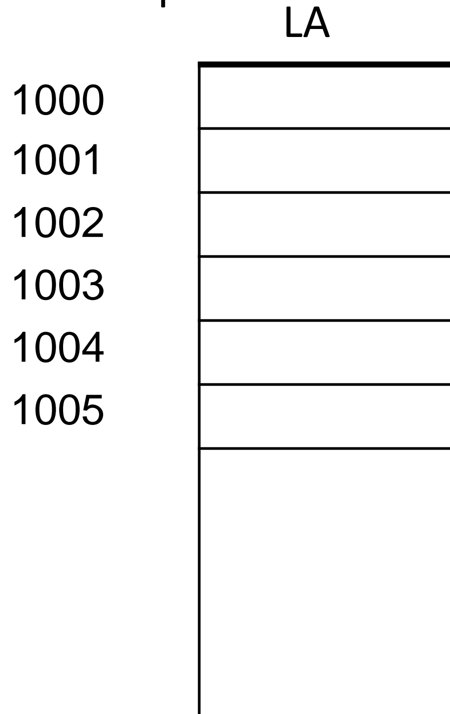


Fig : Computer memory

The computer does not need to keep track of the address of every element of LA, but needs to keep track only of the first element of LA, denoted by

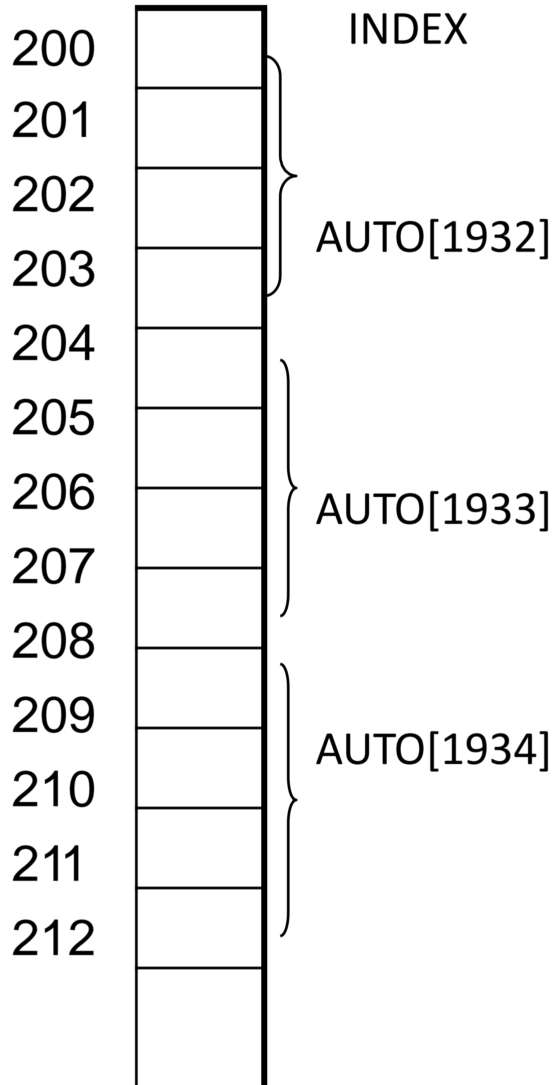
Base(LA)

called the base address of LA. Using this address Base(LA), the computer calculates the address of any element of LA by the following formula :

LOC(LA[k]) = Base(LA) + w(K – lower bound)

Where w is the number of words per memory cell for the array LA

Representation of linear array in memory



Example :
An automobile company uses an array AUTO to record the number of auto mobile sold each year from 1932 through 1984. Suppose AUTO appears in memory as pictured in fig A . That is Base(AUTO) = 200, and w = 4 words per memory cell for AUTO. Then,
LOC(AUTO[1932]) = 200, LOC(AUTO[1933]) =204
LOC(AUTO[1934]) = 208
the address of the array element for the year K = 1965 can be obtained by using :
LOC(AUTO[1965]) = Base(AUTO) + w(1965 – lower bound)
=200+4(1965-1932)=332

Representation of linear array in memory(contd.)

200	202	204	206	208	210	212	214	216	218	220	222	224	226	228	230
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Example:

LOC(LA[6])=200+2(6-0) = 200+12 = 212

LOC(LA[9])=200+2(9-0) = 200+18 = 218

LOC(LA[15])=200+2(15-0) = 200+30 = 230

Traversing linear arrays

Print the contents of each element of DATA or Count the number of elements of DATA with a given property. This can be accomplished by traversing DATA, That is, by accessing and processing (visiting) each element of DATA exactly once.

Algorithm: Given DATA is a linear array with lower bound LB and upper bound UB . This algorithm traverses DATA applying an operation PROCESS to each element of DATA.

1. Set K : = LB.
2. Repeat steps 3 and 4 while K<=UB:
3. Apply PROCESS to DATA[k]
4. Set K : = K+1.
5. Exit.

Traversing linear arrays

Example :
An automobile company uses an array AUTO to record the number of auto mobile sold each year from 1932 through 1984.
a) Find the number NUM of years during which more than 300 automobiles were sold.
b) Print each year and the number of automobiles sold in that year

1. Set NUM : = 0.
2. Repeat for K = 1932 to 1984:

   if AUTO[K]> 300, then : set NUM : = NUM+1
3. Exit.

1. Repeat for K = 1932 to 1984:

   Write : K, AUTO[K]
2. Exit.

Inserting and Deleting

- Inserting refers to the operation of adding another element to the Array

  Inserting an element somewhere in the middle of the array require that each subsequent element be moved downward to new locations to accommodate the new element and keep the order of the other elements.

- Deleting refers to the operation of removing one element from the Array

  Deleting an element somewhere in the middle of the array require that each subsequent element be moved one location upward in order to “fill up” the array. Fig shows Milanjit Inserted, Sumona deleted.

STUDENT

1	Amit Saraswat
2	Sumona
3	Sahil
4	Tammana
5	
6	

STUDENT

1	Amit Saraswat
2	Sumona
3	Milanjit
4	Sahil
5	Tammana
6	

STUDENT

1	Amit Saraswat
2	Milanjit
3	Sahil
4	Tammana
5	
6	

Insertion in an array

INSERTING AN ELEMENT INTO AN ARRAY:

Insert (LA, N, K, ITEM)

Here LA is linear array with N elements and K(position) is a positive integer such that K<=N.This algorithm inserts an element ITEM into the Kth position in LA.

ALGORITHM

Step 1. [Initialize counter] Set J:=N

Step 2. Repeat Steps 3 and 4 while J>=K

Step 3. [Move Jth element downward] Set LA [J+1]: =LA [J]

Step 4. [Decrease counter] Set J:=J-1

[End of step 2 loop]

Step 5 [Insert element] Set LA [K]: =ITEM

Step 6. [Reset N] Set N:=N+1

Step 7. Exit

Deletion from an array

DELETING AN ELEMENT FROM A LINEAR ARRAY
Delete (LA, N, K, ITEM)

Here LA is a linear array with N elements and k is a positive integer such that K<=N. This algorithm deletes the Kth element from LA.

ALGORITHM

Step 1. Set ITEM: = LA [K]
Step 2. Repeat for steps 3&4 for J=K to N-1:
Step 3. [Move J+1st element upward] Set LA [J]: =LA [J+1]
Step4. [Increment counter] J=J+1
[End of step2 loop]
Step 5. [Reset the number N of elements in LA] Set N:=N-1
Step 6. Exit

Searching – Linear search

Linear Search :

Algorithm : A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

1. Set K : = 1, LOC : =0.
2. Repeat steps 3 and 4 while LOC = 0 and K<=N:
3. If ITEM = DATA[K], then : Set LOC : =K .
4. Set K : = K+1.

[End of step 2 loop]

5. [Successful?]

If LOC = 0, then :

Write : ITEM is not in the array DATA.

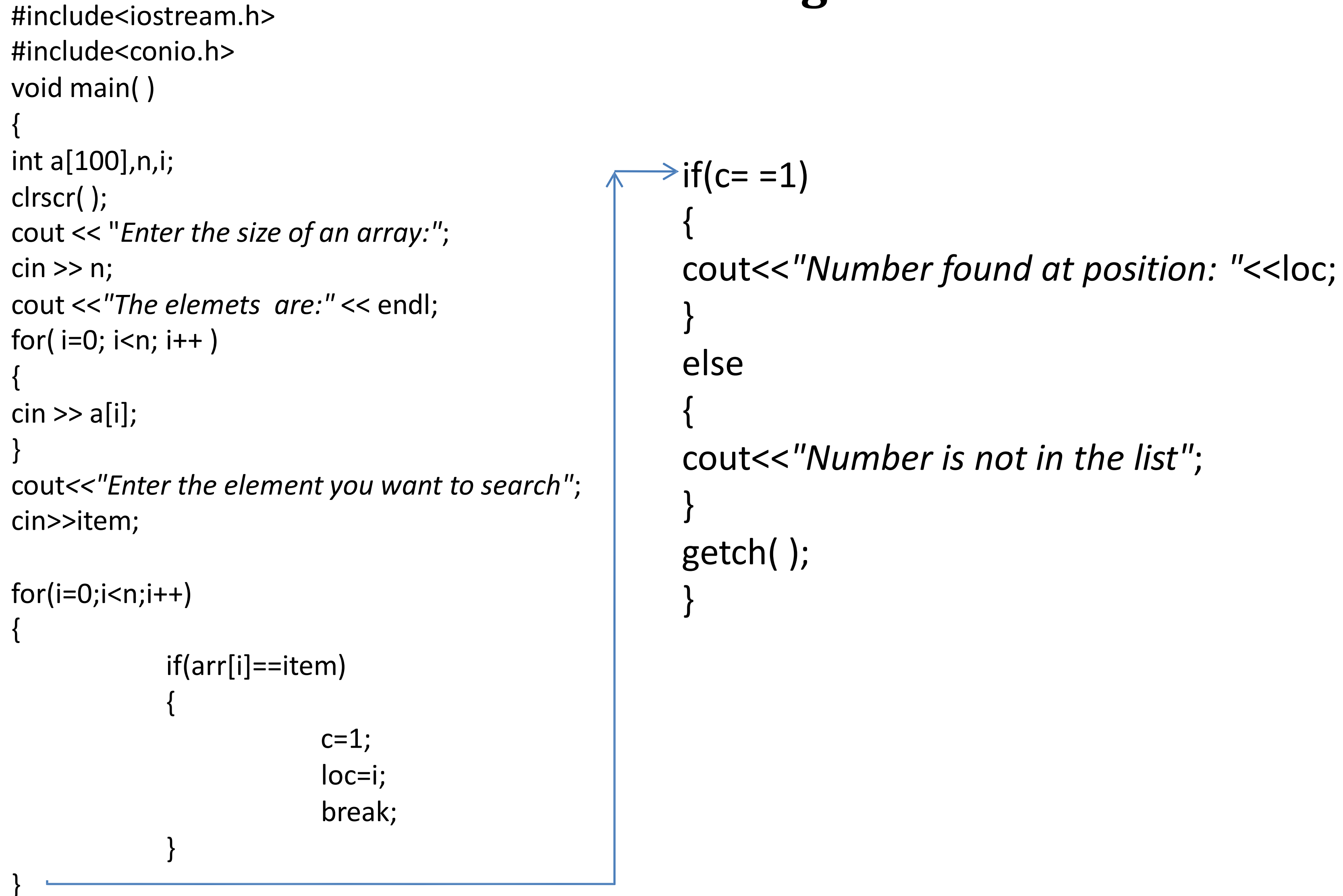
Else :

Write : LOC is the location of ITEM.

[End of if structure]

6. Exit.

Linear Search Program



Searching –Binary Search

- **Binary Search:**
- **Algorithm**: Binary(DATA,LB,UB,ITEM): Here data is a sorted array with lower bound LB and upper bound UB and ITEM is an element to be searched. The variables BEG,END and MID denote, resp, the beginning, end and middle locations of a segment of elements of DATA. This algorithm find the location LOC of ITEM in DATA or sets LOC=NULL.

Binary Search Algorithm

BINARY(DATA, LB, UB, ITEM, LOC)

1. **Set BEG=LB; END=UB; and MID=INT((BEG+END)/2).**
2. **Repeat step 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM**
3. **If ITEM < DATA[MID] then**

   Set END= MID - 1

   Else:

   Set BEG= MID+1

   [end of if structure]
4. **Set MID= INT((BEG+END)/2)**

   [End of step 2 loop]
5. **If ITEM = DATA[MID] then**

   Set LOC= MID

   Else:

   Set LOC= NULL

   [end of if structure]
6. **Exit.**

Binary Search example (Seek for 123)

Seek the value 123

2 6 7 34 76 123 234 567 677 986

first (1) mid (5) (10) last

2 6 7 34 76 123 234 567 677 986

(6) first (8) mid (10) last

2 6 7 34 76 123 234 567 677 986

(6) first last (7)
mid

Binary Search Program

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<abstarry.h>
void main()
{
int a[20],n,beg,end,mid,f=0,element;
cout<<"\nEnter number of elements:";
        cin>>n;
cout<<"\nEnter elements in sorted order:\n";
for(int i=0;i<n;i++)
        cin>>a[i];
cout<<"\n You have entered:";
for(i=0;i<n;i++)
        cout<<a[i]<<" ";
cout<<"\nEnter element you want to search:";
        cin>>element;
beg=0;
end=n-1;
while(beg<=end)
{
mid=(beg+end)/2;
if(element<a[mid])
        beg=mid-1;
else if (element>a[mid])
        beg=mid+1;
else if(element==a[mid])
        {
        cout<<"!..................!";
        cout<<"\nposition is:"<<mid;
        cout<<"\n!..................!";
        f=1;
        break;
        }
}
if(f==0)
cout<<"\nElement does not exist";
        getch();
}
```

Sorting – Bubble sort

- Bubble sorting is an algorithm in which we are comparing first 2 values and put the larger on at higher index.
- This process do iteratively until the largest value is not reached at last index. Then start again from 0 index up to n-1 index.
- Bubble sort is also known as exchange sort.
- Bubble sort is the simplest sorting algorithm.

Bubble sort(contd..)

- **Sorting takes an unordered collection and makes it an ordered one.**

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

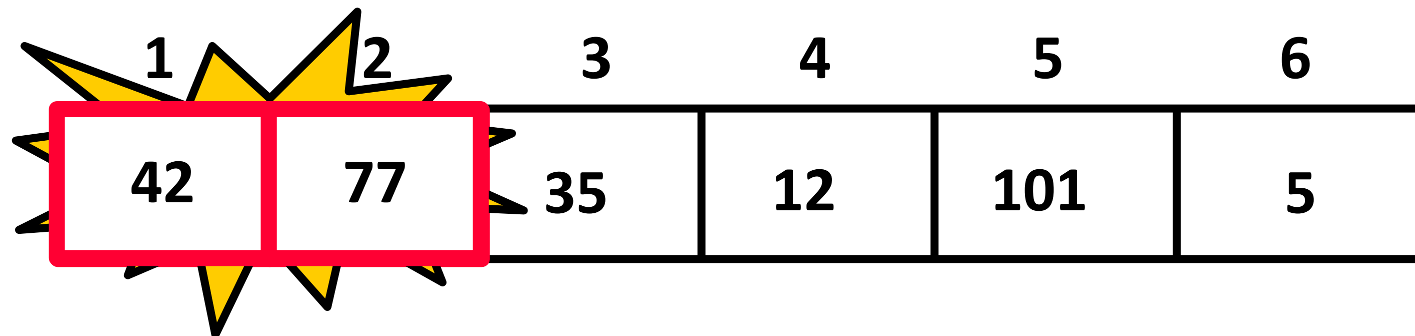
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

1	2	3	4	5	6
77	42	35	12	101	5

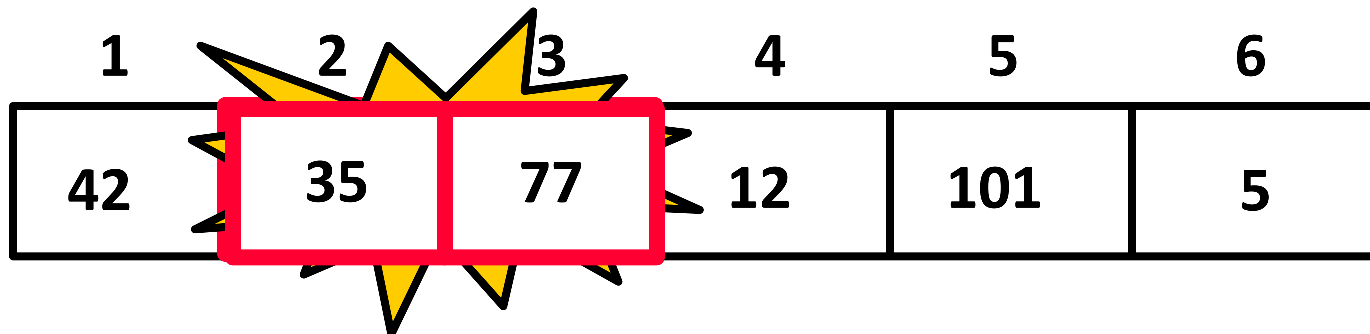
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



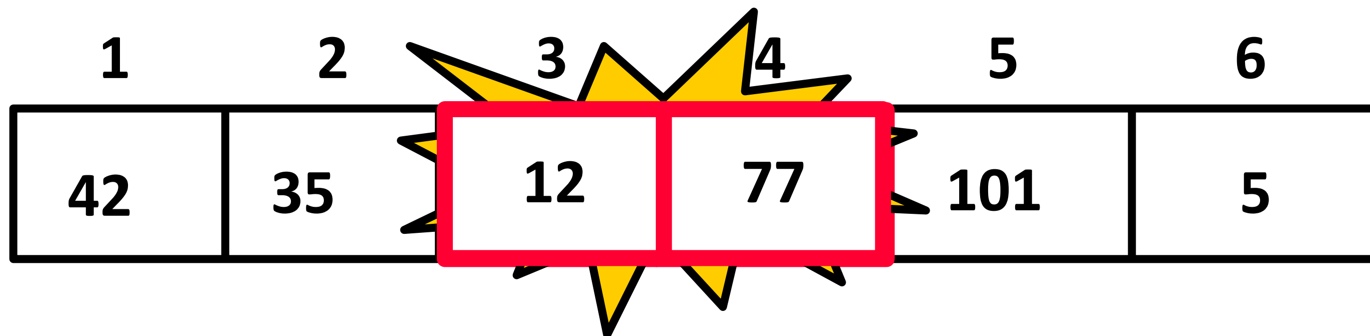
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



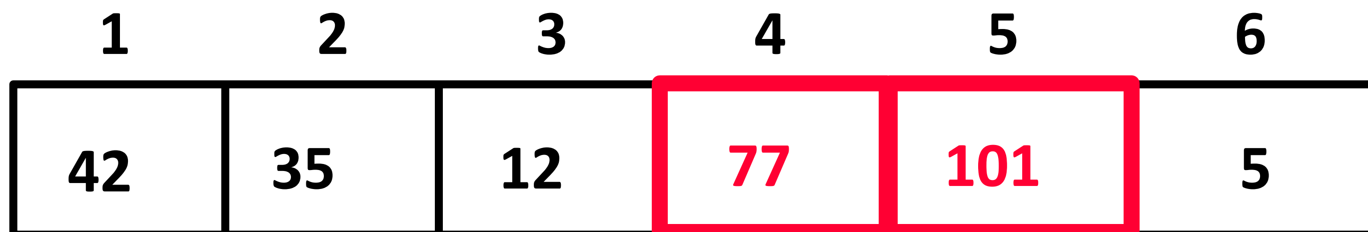
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



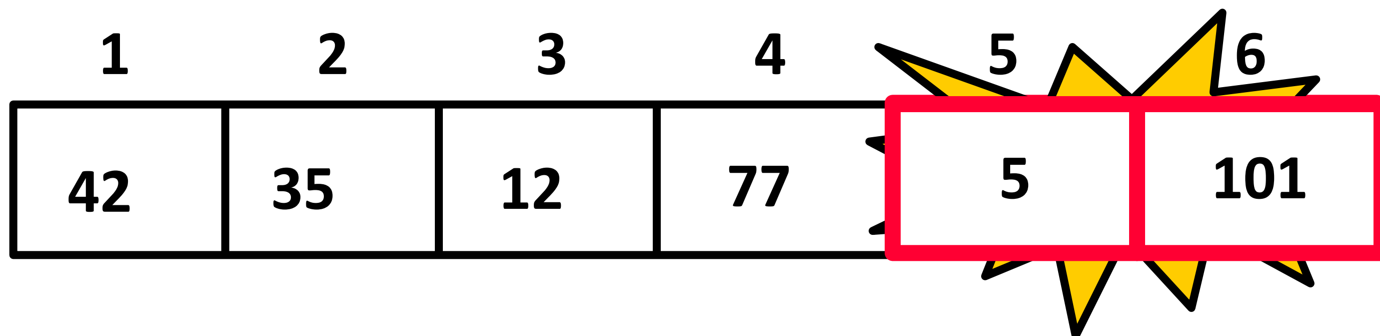
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Bubble sort(contd.)

- **Notice that only the largest value is correctly placed**
- **All other values are still out of order**
- **So we need to repeat this process**

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Repeat "Bubble Up" How Many Times?

- **If we have N elements…**
- **And if each time we bubble an element, we place it in its correct location…**
- **Then we repeat the "bubble up" process N – 1 times.**
- **This guarantees we'll correctly place all N elements.**

Reducing the Number of Comparisons

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

"Bubbling" All the Elements

1	2	3	4	5	6
42	35	12	77	5	101

1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

1	2	3	4	5	6
5	12	35	42	77	101

N - 1

Bubble sort(contd..)

Algorithm Bubble_Sort (DATA, N):

1. Repeat steps 2 and 3 for K = 1 to N-1.
2. Set PTR: =1.[Initializes pass pointer PTR]
3. Repeat while PTR<=N-K: [Executes pass]

a) If DATA[PTR]>DATA[PTR+1],then:
   TEMP := DATA[PTR], DATA[PTR] := DATA[PTR+1], DATA[PTR+1] := temp
   [End of if structure]

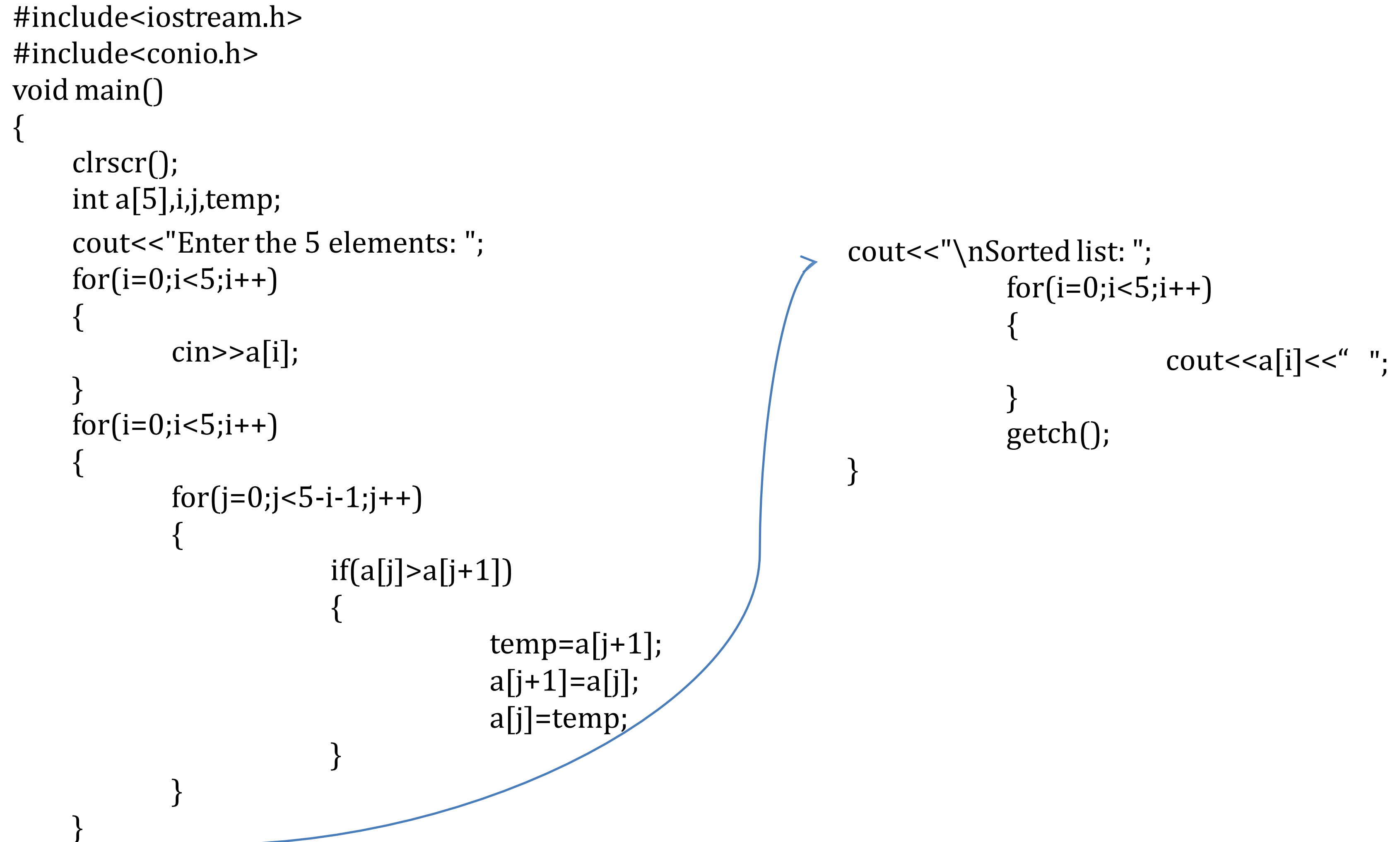
a) Set PTR: =PTR+1
   [End of inner loop]
   [End of step 1 Outer loop]

4. Exit

1	2	3	4	5	6
77	42	35	12	101	5

Program for Bubble sort

Program to sort an array of integers in ascending order using bubble sort.



```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	int a[5],i,j,temp;
	cout<<"Enter the 5 elements: ";
	for(i=0;i<5;i++)
	{
		cin>>a[i];
	}
	for(i=0;i<5;i++)
	{
		for(j=0;j<5-i-1;j++)
		{
			if(a[j]>a[j+1])
			{
				temp=a[j+1];
				a[j+1]=a[j];
				a[j]=temp;
			}
		}
	}
cout<<"\nSorted list: ";
		for(i=0;i<5;i++)
		{
			cout<<a[i]<<"   ";
		}
		getch();
}
```

Algorithm for Insertion sort

Algorithm Insertion_Sort (DATA, N):

1. Repeat steps 2 to 4 for K = 2,3…N-1:
2. Set TEMP:= a[K] and J=k-1
3. Repeat while TEMP<a[J] and J>=1
    a) Set a[J+1]:= a[J]. [Moves element forward]
    b) Set J:=J-1
    [End of loop]

4. Set a[J+1]:=TEMP. [Inserts element in proper place]
    [End of step 1 Outer loop]

5. Exit

Example: Insertion Sort

Pass 1: 1 comparisons

1	2	3	4	5	6	7
35	20	40	100	3	10	15

1	2	3	4	5	6	7
35	35	40	100	3	10	15

1	2	3	4	5	6	7
20	35	40	100	3	10	15

Pass 2: 2 comparisons

1	2	3	4	5	6	7
20	35	40	100	3	10	15

1	2	3	4	5	6	7
20	35	40	100	3	10	15

Insertion Sort(contd..)

Pass 3: 3 comparisons

1	2	3	4	5	6	7
20	35	40	100	3	10	15
20	35	40	100	3	10	15

Pass 4: 4 comparisons

1	2	3	4	5	6	7
20	35	40	100	3	10	15
20	35	40	100	100	10	15
20	35	40	40	100	10	15
20	35	35	40	100	10	15
20	20	35	40	100	10	15
3	20	35	40	100	10	15

Insertion Sort(contd..)

Pass 5: 5 comparisons

1	2	3	4	5	6	7
3	20	35	40	100	10	15
3	20	35	40	100	100	15
3	20	35	40	40	100	15
3	20	20	35	40	100	15
3	10	20	35	40	100	15

Insertion Sort(contd..)

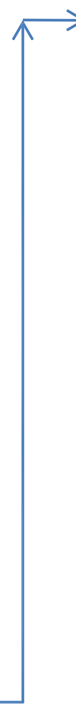
Pass 6: 6 comparisons

1	2	3	4	5	6	7
3	10	20	35	40	100	15
3	10	20	35	40	100	100
3	10	20	35	40	40	100
3	10	20	35	35	40	100
3	10	20	20	35	40	100
3	10	15	20	35	40	100

Program for Insertion Sort

Program to sort an array of integers in ascending order using insertion sort.

```
#include<iostream.h>
#include<conio.h>
void main()
{
        clrscr();
        int a[5],i,j,k,temp;
        cout<<"Enter the 5 elements: ";
        for(i=0;i<5;i++)
        {
                cin>>a[i];
        }
        for(k=1;k<5;k++)
        {
                temp=a[k];
                J=k-1;
                while((temp<a[j]) && (j>=0))
                {
                        a[j+1]=a[j];
                        j=j-1;
                }
                a[j+1]=temp;
        }
```



```
cout<<"\nSorted list: ";
        for(i=0;i<5;i++)
        {
                cout<<a[i]<<"  ";
        }
        getch();
}
```

Algorithm for Selection Sort

Algorithm : (selection_sort) SELECTION(A,N)

This algorithm sorts the array A with N elements.

1. Repeat steps 2 & 3 for K = 1,2…N-1:
2. Call MIN(A,K,N,LOC)
3. Interchange A[K] & A[LOC].]
   Set TEMP=A[K], A[K]=A[LOC] & A[LOC]=TEMP
   [End of step1 loop]
4. Exit.

Algorithm: MIN(A,K,N,LOC)

This algorithm finds the location LOC of the smallest element among A[K], A[K+1], …+A[N]

1. Set MIN=A[k] & LOC=K. [Initializes pointers]
2. Repeat for J=K+1,K+2,..N:
   If MIN>A[J], then: Set MIN=A[J] & LOC=J
   [End of loop]
3. Return

Example: Selection Sort

1	2	3	4	5	6	7	8
77	33	44	11	88	22	66	55

k=1

1	2	3	4	5	6	7	8	
77	33	44	11	88	22	66	55	min=77, loc=1
77	33	44	11	88	22	66	55	min=33, loc=2
77	33	44	11	88	22	66	55	min=33,loc=2
77	33	44	11	88	22	66	55	min=11,loc=4
77	33	44	11	88	22	66	55	min=11,loc=4
77	33	44	11	88	22	66	55	min=11,loc=4
77	33	44	11	88	22	66	55	min=11,loc=4
77	33	44	11	88	22	66	55	min=11,loc=4
								swap(a[k],a[loc])
11	33	44	77	88	22	66	55	swap(a[1],a[4])

k=2

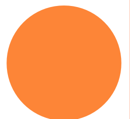
1	2	3	4	5	6	7	8	
11	33	44	77	88	22	66	55	min=33,loc=2
11	33	44	77	88	22	66	55	min=33,loc=2
11	33	44	77	88	22	66	55	min=33,loc=2
11	33	44	77	88	22	66	55	min=33,loc=2
11	33	44	77	88	22	66	55	min=22,loc=6
11	33	44	77	88	22	66	55	min=22,loc=6
11	33	44	77	88	22	66	55	min=22,loc=6
								swap(a[k],a[loc])
11	22	44	77	88	33	66	55	swap(a[2],a[6])

k=3

1	2	3	4	5	6	7	8	
11	22	44	77	88	33	66	55	min=44,loc=3
11	22	44	77	88	33	66	55	min=44,loc=3
11	22	44	77	88	33	66	55	min=44,loc=3
11	22	44	77	88	33	66	55	min=33,loc=6
11	22	44	77	88	33	66	55	min=33,loc=6
11	22	44	77	88	33	66	55	min=33,loc=6
								swap(a[k],a[loc])
11	22	33	77	88	44	66	55	swap(a[3],a[6])

INTRODUCTION

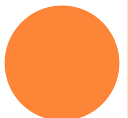
- The Microprocessor is the 'brain of the microcomputer'
- It is a single chip which is capable of
  - processing data,
  - controlling all of the components which make up the microcomputer system
- Microprocessor used to sequence executions of instructions that is in memory.
- Microprocessor Fetch , Decode , and Execute the instruction.
- The internal architecture of the microprocessor is complex.



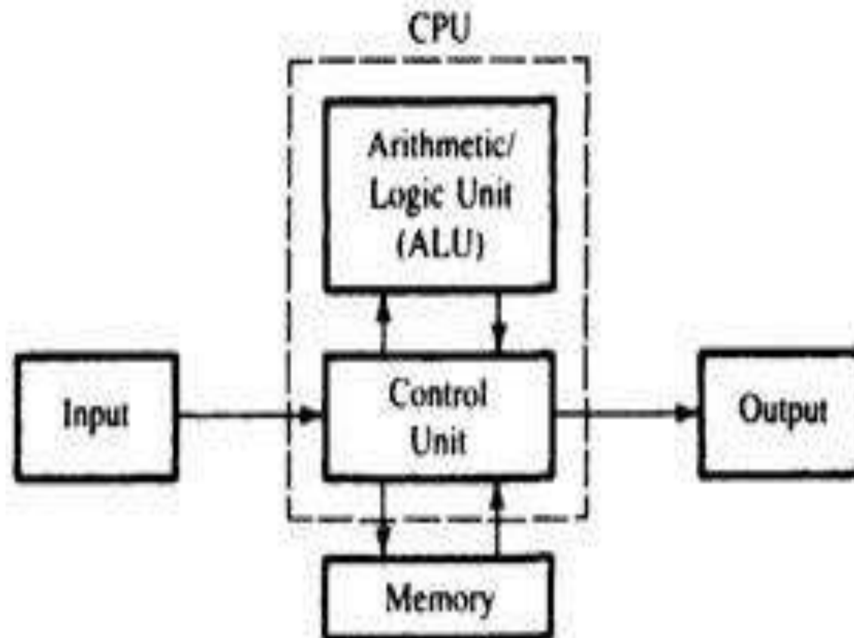
INTRODUCTION

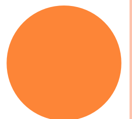
microprocessor (MPU) typically contains:

- Registers: Temporary storage locations for programming instruction or data.
- The Arithmetic Logic unit (ALU):This part of the MPU performs both arithmetic and logical operations
- Timing and Control Circuits: That keep all of the other parts of system (Registers , ALU, memory & I/O) working together in the right time sequence.

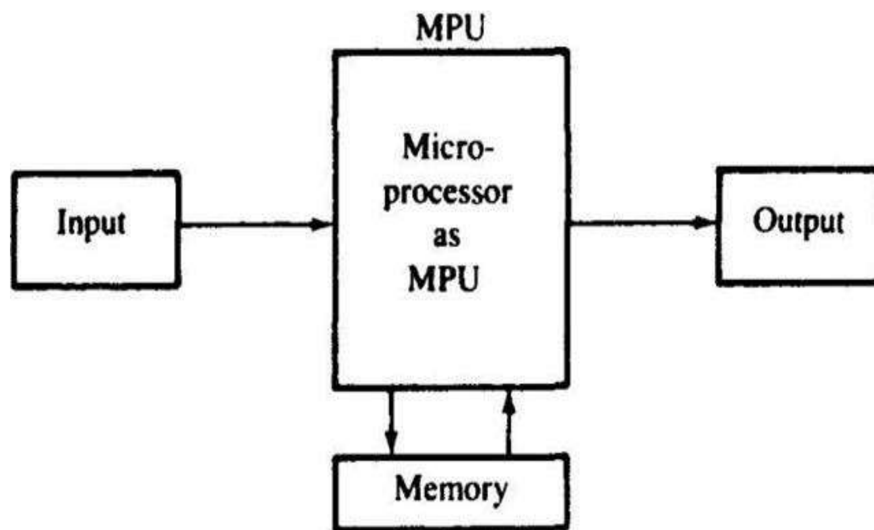


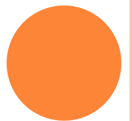
INTRODUCTION





INTRODUCTION

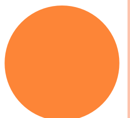




INTRODUCTION

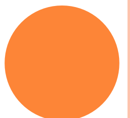
Microprocessor is connected with:

- Input: It is used to give the input data to the microprocessor,
- Output: It is used to provide the result of calculation,
- Memory: It is used to store the data.



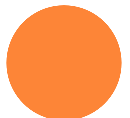
INTRODUCTION

- Microprocessor (MPU)
- A Microprocessor is a CPU on a single chip, it contains:
  - ¾ ALU,
  - ¾ Instruction decoder,
  - ¾ Registers,
  - ¾ Bus control etc.
- Micro-computer (u-Computer) contains:
  - ¾ small computer
  - ¾ peripheral I/O
  - ¾ memory
- Microcontroller (uC) contains:
  - ¾ Computer on a single chip of silicon

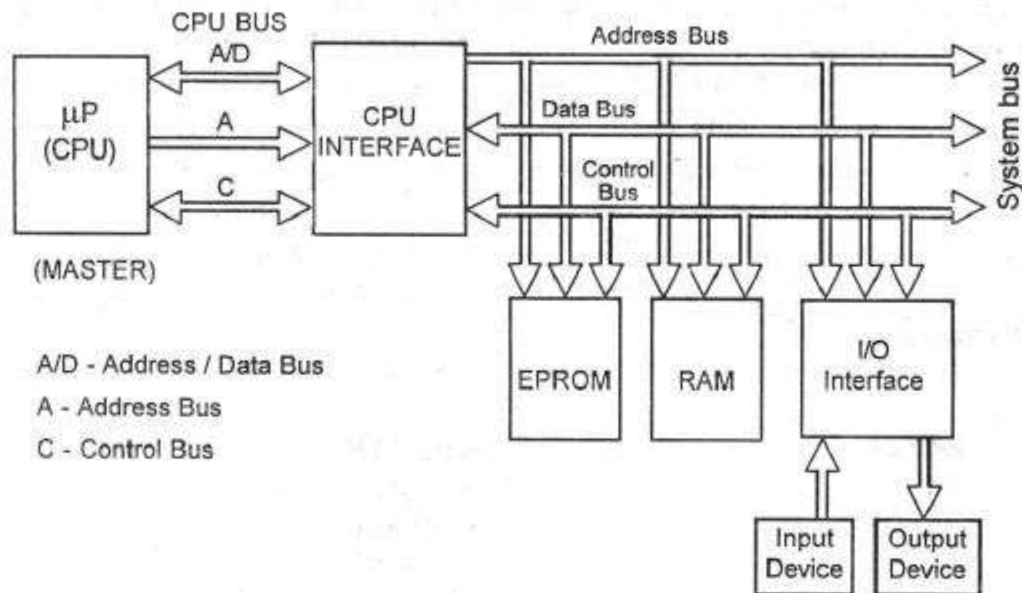


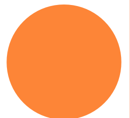
MICROPROCESSOR VS MICROCONTROLLER

- A Microprocessor:
  - Only is a single-chip CPU
  - Bus is available bus is available
  - RAM capacity,
  - No. of port is selectable
  - RAM is larger than ROM (usually)
- A Microcontroller:
  - contains a CPU and RAM,ROM ,Prepherals, I/O port in a single IC,
  - Internal hardware is fixed,
  - Communicate by port,
  - ROM is larger than RAM (usually),
  - Small power consumption,
  - Single chip, small board,
  - Implementation is easy,
  - Low cost.



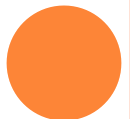
Architecture of Microprocessor





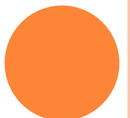
ARCHITECTURE OF MICROPROCESSOR

- BUSES: The buses are group of lines that carries data, address or control signals.
- The CPU Bus has multiplexed lines, i.e., same line is used to carry different signals.
- The CPU interface is provided to demultiplex the lines, to generate chip select signals and additional control signals.
- The system bus has separate lines for each signal
  - Address bus: carries the address of a unique memory or input/output (I/O) device.
  - Data bus: carries data stored in memory (or an I/O device) to the CPU or from the CPU to the memory (or I/O device).
  - Control bus: is a collection of control signals that coordinate and synchronize. the whole system



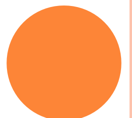
MEMORY

- The memory in a computer system stores the data and instructions of the programs.
- Main memory types:
  - ¾ ROM (read-only memory): programmed permanently at the factory, cannot be altered.
  - ¾ RAM (random-access memory): read and write memory.
  - ¾ EPROM (erasable programmable ROM): nonvolatile, written electrically but erased optically
  - ¾ EEPROM (electrically ROM): nonvolatile, both written and erased electrically.



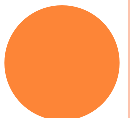
Introduction to 8085

- ThefeaturesofINTEL8085are:
- It is an 8 bit processor.
- It is as single chip N MOS device with 40 pins
- It has multiplexed address and data bus.(AD0-AD7).
- It works on 5-Volt dc power supply.
- The maximum clock frequency is 3-MHz while minimum frequency is 500-kHz.
- It provides 74 instructions with 5 different addressing mode.
- It provides 16 address lines.



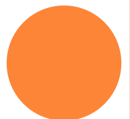
Introduction to 8085

- It generates 8 bit I/O address so it can access 2^8=256input ports.
- It provides 5 hardware interrupts: TRAP, RST 5.5, RST 6.5, RST 7.5, INTR.
- It provides Acc, one flag register, 6 general purpose registers and two special purpose registers (SP,PC).



8085 PIN DIAGRAM

- It is a 40-pin DIP chip designed using NMOS.
- ALE: It is used to separate the multiplexed Address/Data lines into lower order address line(A7A0) and data lineD7D0.
- RD: Active low, Indicates that memory device is ready to be read.
- WR: Active low, Indicates that data on the data bus are ready too be written.
- IO/M: Differentiate between I/O operation and memory operations.
- HIGH :I/O operation is carried on. LOW:
- Memory operation is carried on.
- s1,s0:Similar to IO/M but are rarely used.
- Timing Circuit: Crystal oscillator is connected to these pins X1,X2 they need to be at 3MHz frequency for the operation, so input of 6MHz is to be given.



8085 PIN DIAGRAM

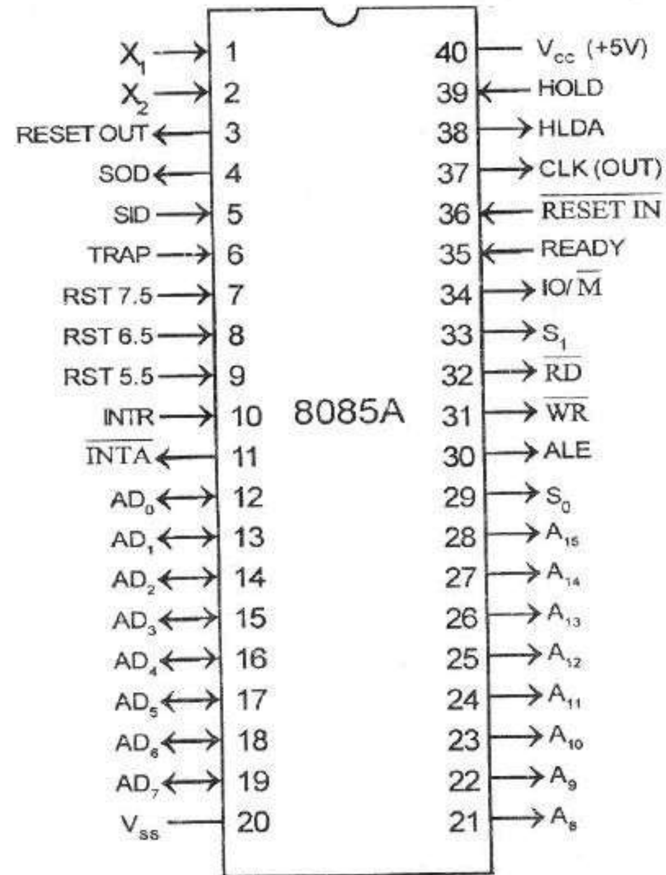
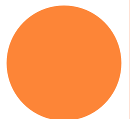


Fig 1.3 : 8085 Microprocessor Signals and Pin Assignment

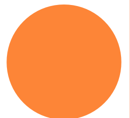
8085 PIN DIAGRAM

- CLK OUT :The output from this port can be used as system clock for other devices.
- READY: When High, Microprocessor start to proceed the process provided by the Input/output devices.
- HOLD: When it is high. It means that some process is already in progress and it suspends all other process.
- HOLDA: The above status is indicated by providing HOLDA(HOLD Acknowledge) as High.
- RESETIN: Reset the microprocessor. It is active low.
- RESETOUT: Output is obtained when High.
- INTR: Interrupt signal to microprocessor. It is required to provide the location the location of ISR(Interrupt service routine).
- INTA: when microprocessor receives an interrupt request on INTR, this is acknowledged by sending a low signal to INTA.



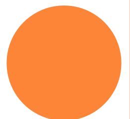
ARCHITECTURE OF 8085

INTA
RST 6.5
TRAP
INTR
RST 5.5
RST 7.5
SID
SOD
Interrupt Control
Serial I/O Control
8-Bit Internal Data Bus
Accumulator (8)
Temp. Reg. (8)
Instruction Register (8)
Multiplexer
W Temp. Reg. (8)
Z Temp. Reg. (8)
B Reg. (8)
C Reg. (8)
D Reg. (8)
E Reg. (8)
H Reg. (8)
L Reg. (8)
Reg. Select
Stack Pointer (16)
Program Counter (16)
Incrementer/Decrementer Address Latch (16)
Register Array
Flag Flip-Flops (5)
Arithmetic Logic Unit (ALU) (8)
Instruction Decoder and Machine Cycle Encoding
Power Supply
+5 V
GND
Timing and Control
X_1
X_2
CLK GEN
Control
Status
DMA
Reset
CLK OUT
READY
$\overline{RD}$
$\overline{WR}$
ALE
S_0
S_1
IO/$\overline{M}$
HOLD
HLDA
$\overline{RESET\ IN}$
RESET OUT
Address Buffer (8)
Data/Address Buffer (8)
A_{15}–A_8 Address Bus
AD_7–AD_0 Address/Data Bus



ARCHITECTURE OF 8085

- The architecture of.8085 is shown in figure.
- The internal architecture of 8085 includes the ALU, timing and control unit, instruction register and decoder, register array, interrupt control and serial I/O control.
- ALU: performs the arithmetic and logical operations. The operations performed by ALU of 8085 are: addition, subtraction, increment, decrement, logical AND, OR, EXCLUSIVE -OR, compare, complement.
- The accumulator and temporary register are used to hold the data during an arithmetic / logical operation.

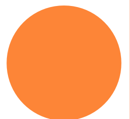


ARCHITECTURE OF 8085

- After an operation the result is stored in the accumulator and the flags are set or reset according to the result of the operation.
- FLAG REGISTER: There are five flags in 8085, which are:

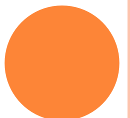
S	Z	X	Ac	X	P	X	Cy

- sign flag (8), zero flag (Z), auxiliary carry flag (AC),parity flag (P) and carry flag (CY).
- Apart from Accumulator (A-register), there are six general-purpose programmable registers B, C, D, E, H and L.
- The temporary registers W and Z are intended for internal use of the processor and it cannot be used by the programmer.



Architecture of 8085

- They can be used as 8-bit registers or paired to store 16-bit data. The allowed pairs are B-C,D-E and H-L.
- STACK POINTER (SP): The stack pointer SP, holds the address of the stack top.
- PROGRAM COUNTER (PC):The program counter (PC) keeps track of program execution. To execute a program the starting address of the program is loaded in program counter.



ADDRESSING MODES OF 8085

- There are five addressing modes in 8085.
- Immediate Addressing Mode: An immediate is transferred directly to the register.

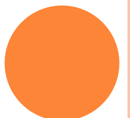
  Eg: - MVI A, 30H (30H is copied into the register A)

  MVI B,40H(40H is copied into the register B).
- Register Addressing Mode: Data is copied from one register to another register.

  Eg: - MOV B, A (the content of A is copied into the register B)

  MOV A, C (the content of C is copied into the register A).
- Direct Addressing Mode: Data is directly copied from the given address to the register.

  Eg: - LDA 3000H (The content at the location 3000H is copied to the register A).

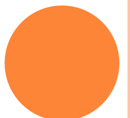


- Indirect Addressing Mode: The data is transferred from the address pointed by the data in a register to other register.

  Eg: MOV A, M (data is transferred from the memory location pointed by the register to the accumulator).

- Implied Addressing Mode: - This mode doesn't require any operand. The data is specified by opcode itself.

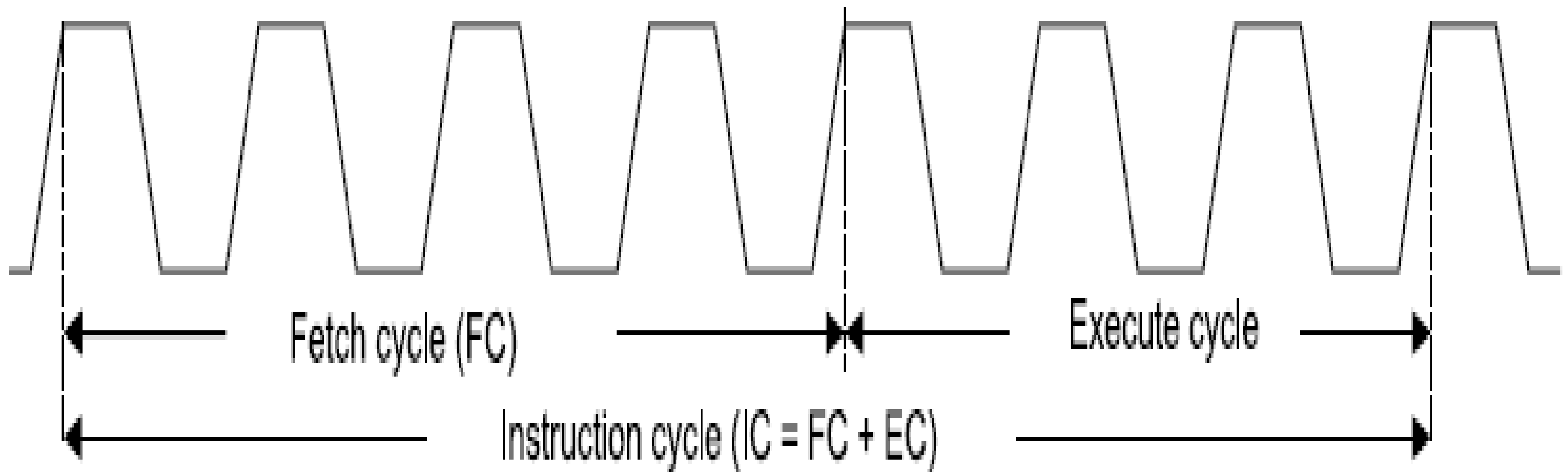
  Eg: - RAL, CMP.



Timing Diagrams, Interrupts and Addressing Modes of 8085

Timing Diagram

Instruction Cycle:

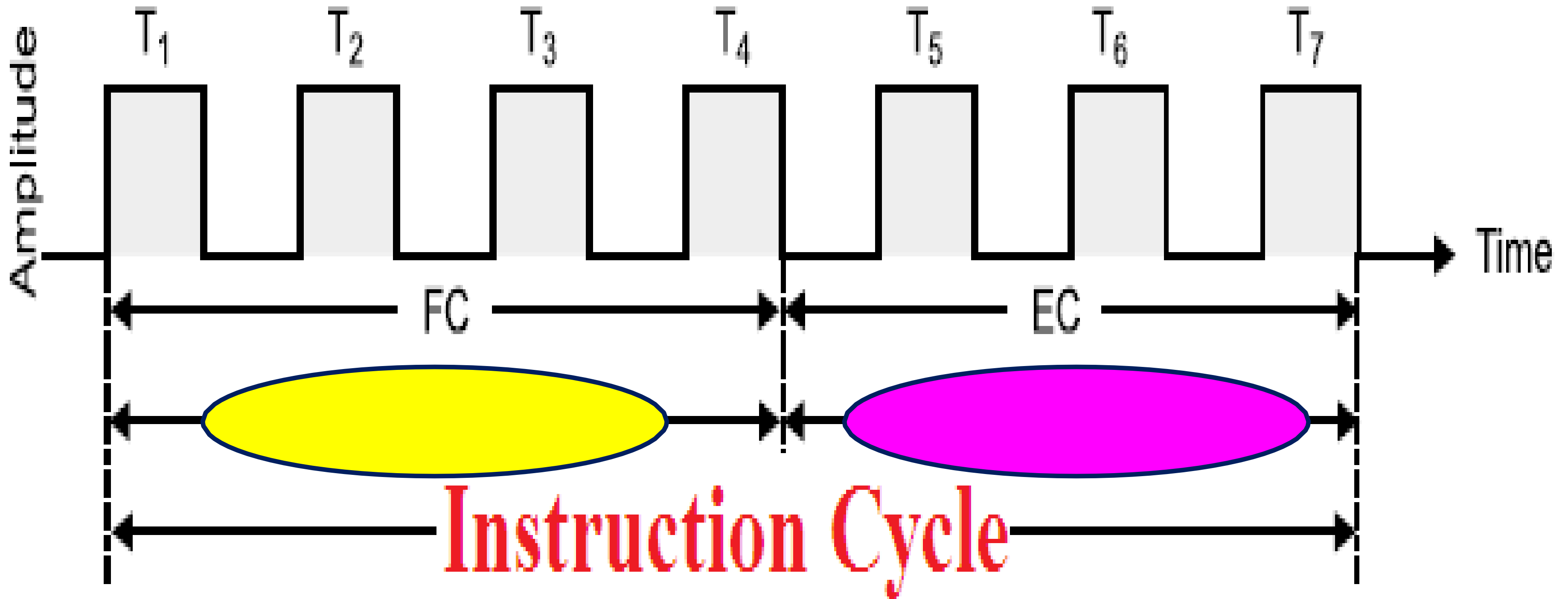


Fetch

Decode

Execute

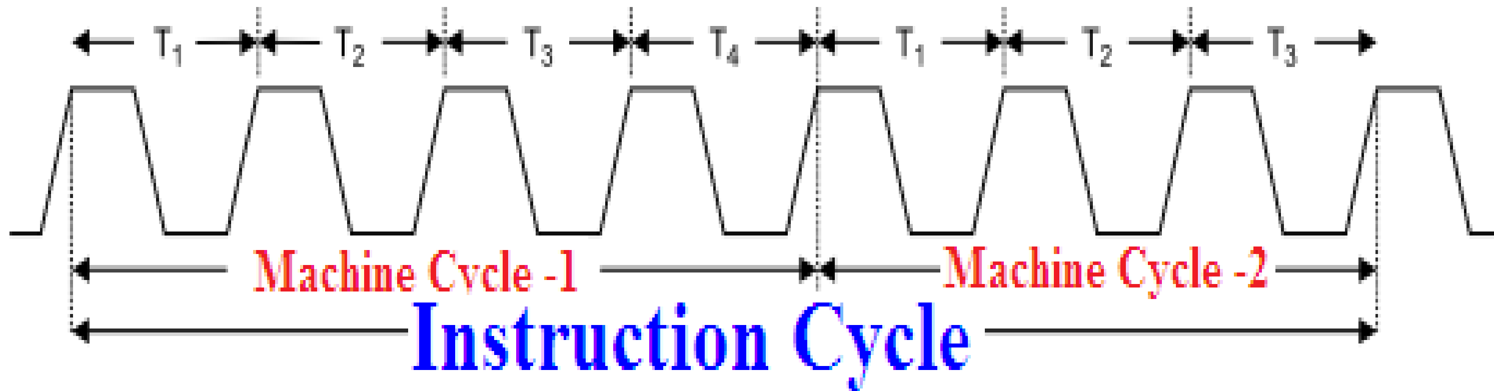
Machine Cycle:



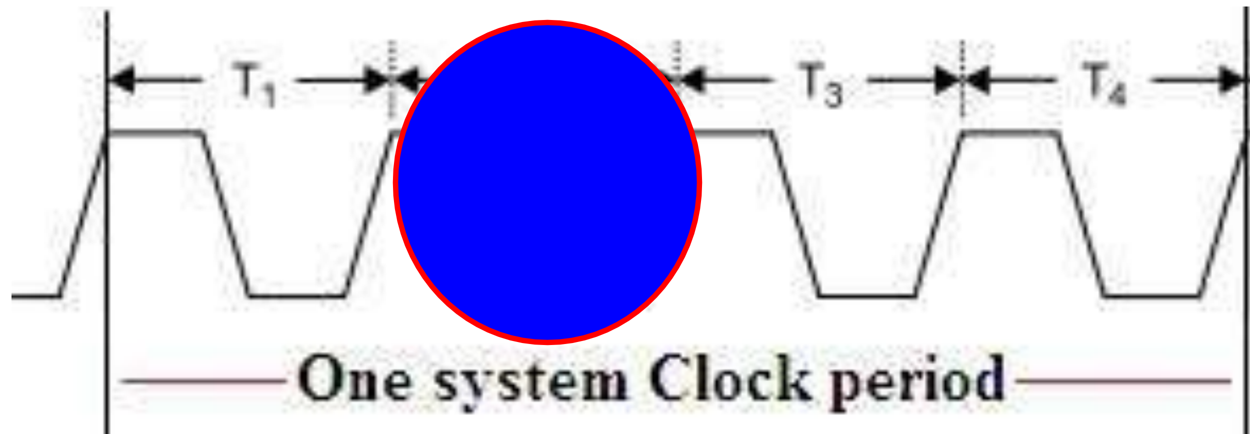
time required to access the memory or input/output devices is called machine cycle

T- States:

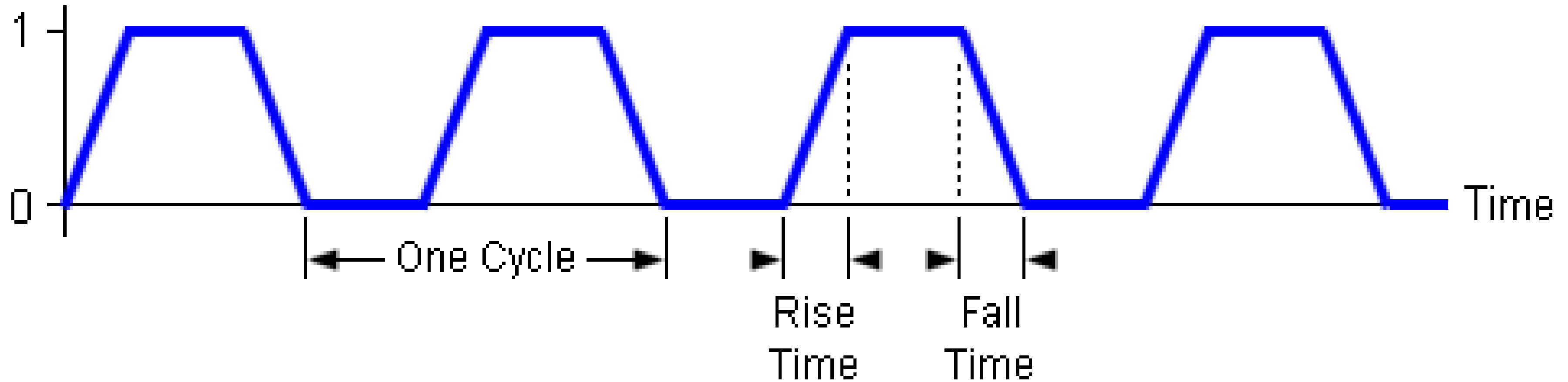
The machine cycle and instruction cycle takes **multiple clock periods**.



A portion of an operation carried out in one system clock period is called as T-state

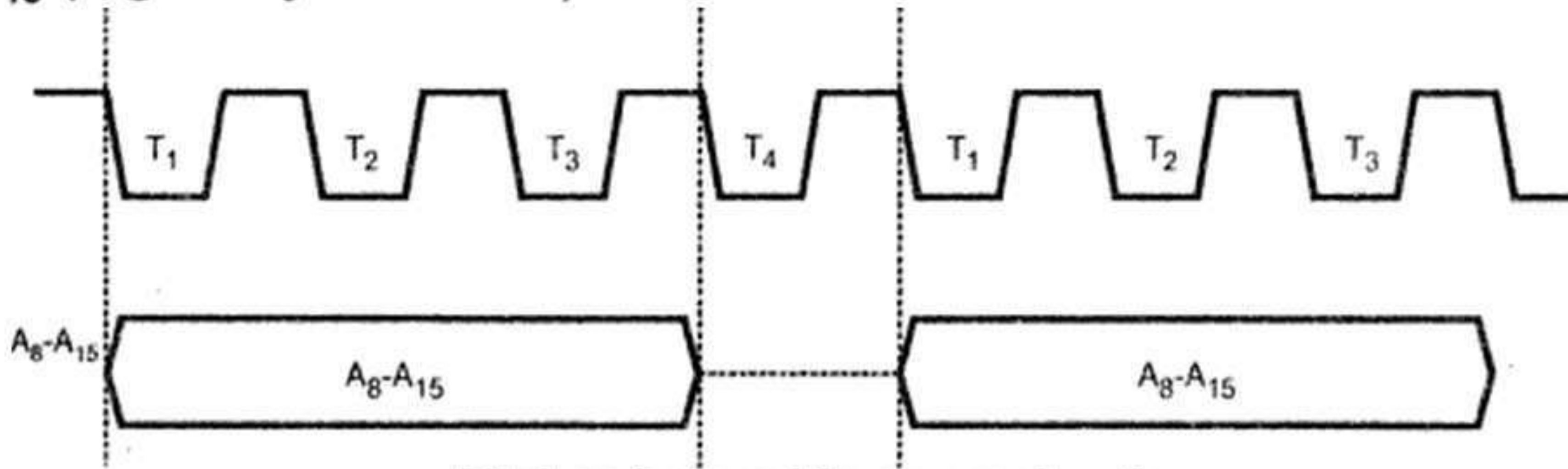


Clock Signal:



- Microprocessor operates with reference to clock signals.
- X1 and X2 we provide clock signals and this frequency is divided by two.
- This frequency is called as the operating frequency.

A_8 - A_{15} (Higher Byte Address) :



Higher byte address on A_8 - A_{15}

D_0 - D_7 (Data Bus) :

T1 T2 T3 T1 T2 T3

Address Data Address Data

(a) read machine cycle

(b) write cycle

Data bus

$IO/\overline{M}$, S_0, S_1 :

Machine cycle 1 — Machine cycle 2

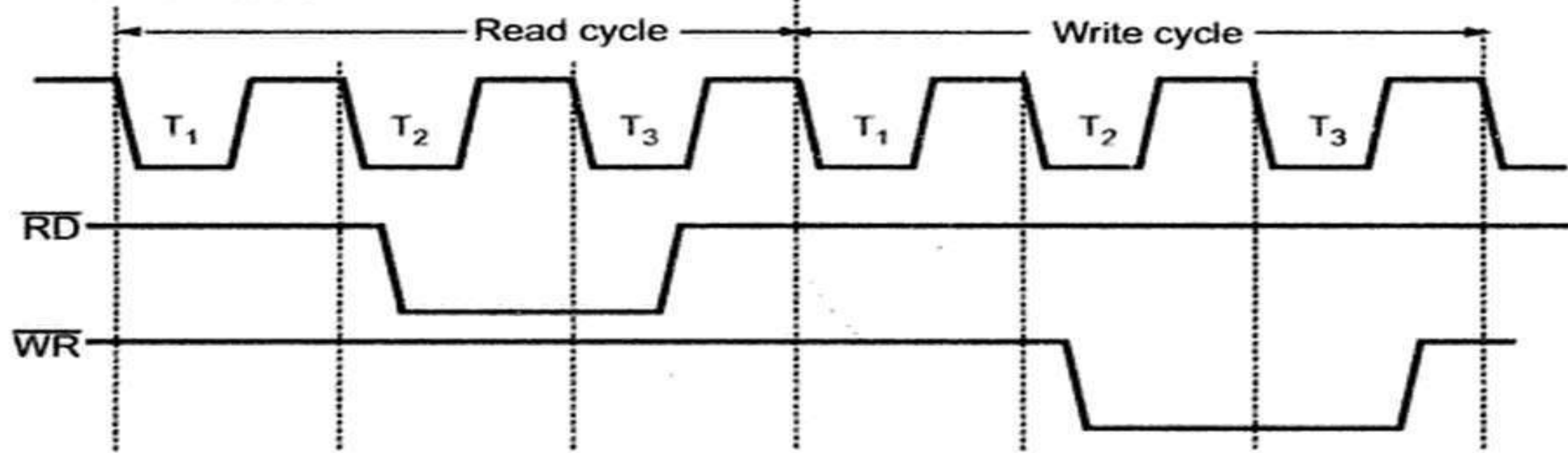
T_1 T_2 T_3 T_4 T_1 T_2 T_3

$IO/\overline{M} = 0$, $S_0 = 1$, $S_1 = 1$ — $IO/\overline{M} = 0$, $S_0 = 0$, $S_1 = 1$

Opcode fetch — Memory read

Status signals

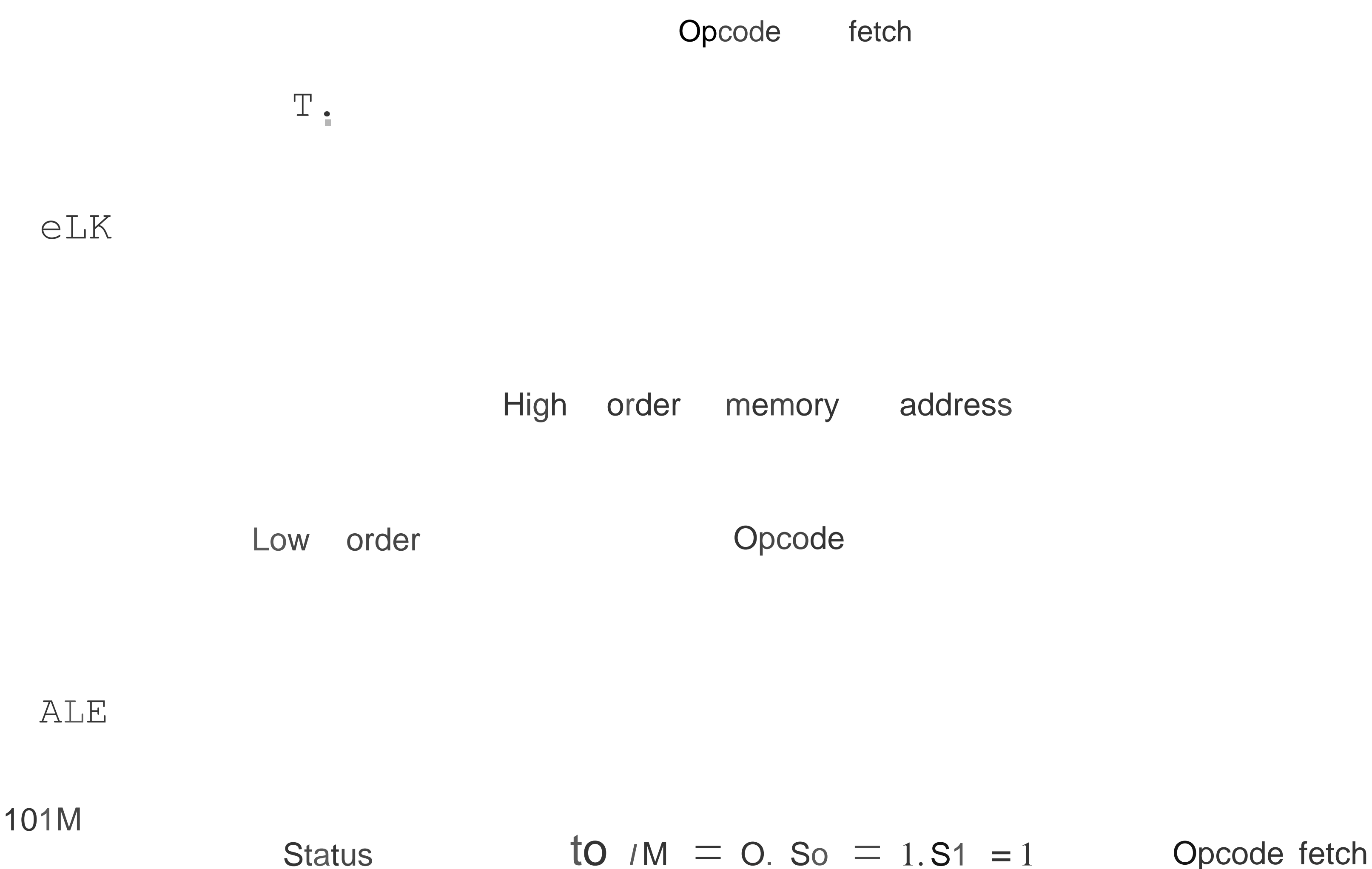
$\overline{RD}$ and $\overline{WR}$:



$\overline{RD}$ and $\overline{WR}$ signals

Rule for timing diagram

1. ALE=I Tl States.
2. AO-A7=Group Signal. Tl States.
3. A8-A15 = Group Signal. Tl,T2,T3 States.
4. DO-D7=Group Signal.
   - Read Cycle: TI- Address, T3 - Data
   - Write Cycle: TI- Address, T2,T3 - Data
5. 101M SO,SI = Group Signal. Tl,T2,T3,T4 States.
6. RD,WR= O. T2,T3 States



(b) Opcode fetch machine cycle

Memory Read

eLK

Memory address

ALE

Data from memory ---I

10 / M = O. 51 - '. So = 0

RD

(b) Memory read machine cycle

MefTIOryWrite

T~ T2 T3

eLK

, " "

Memory address

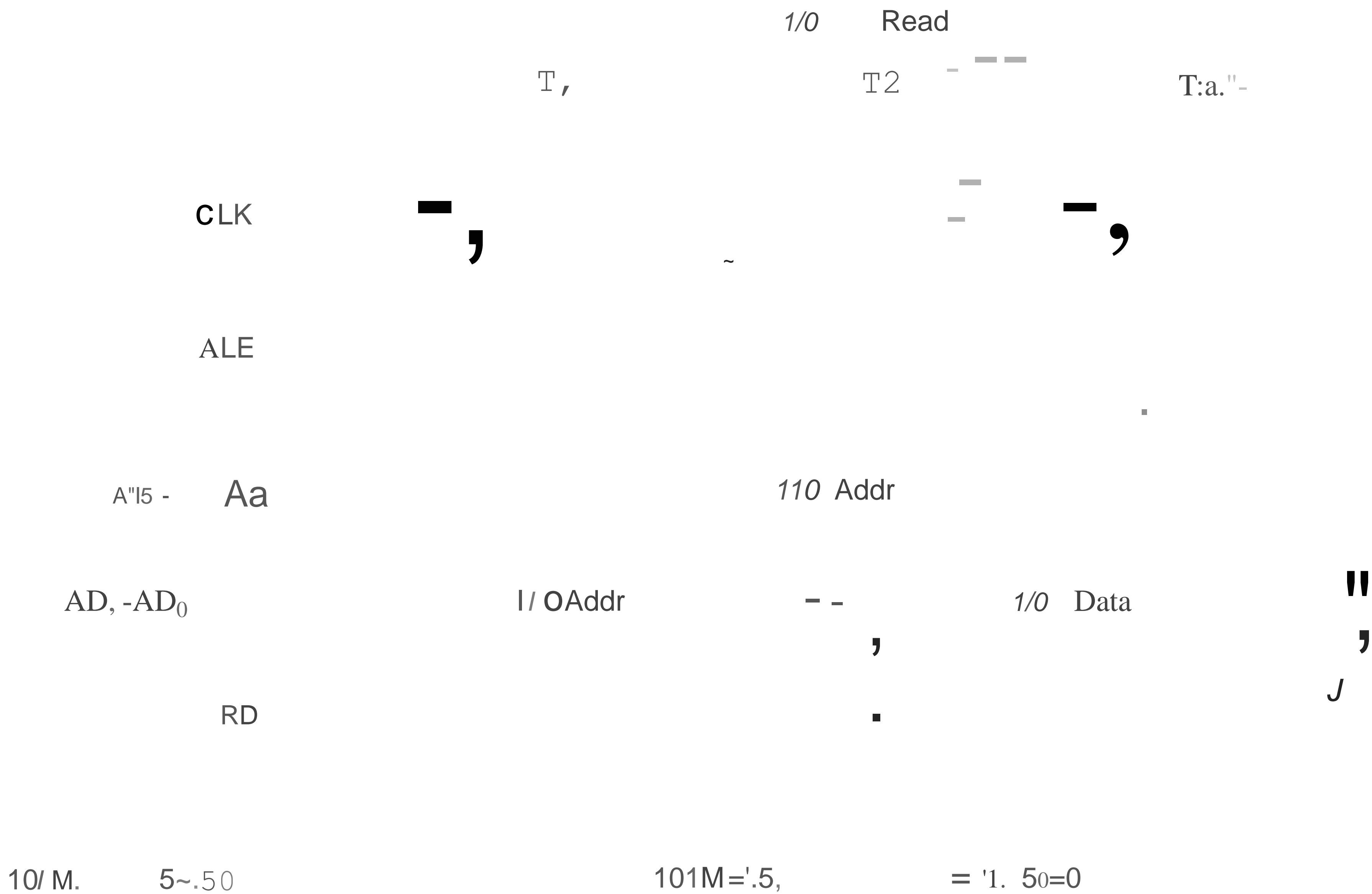
ALE V

A7 -A0 Data from CPU

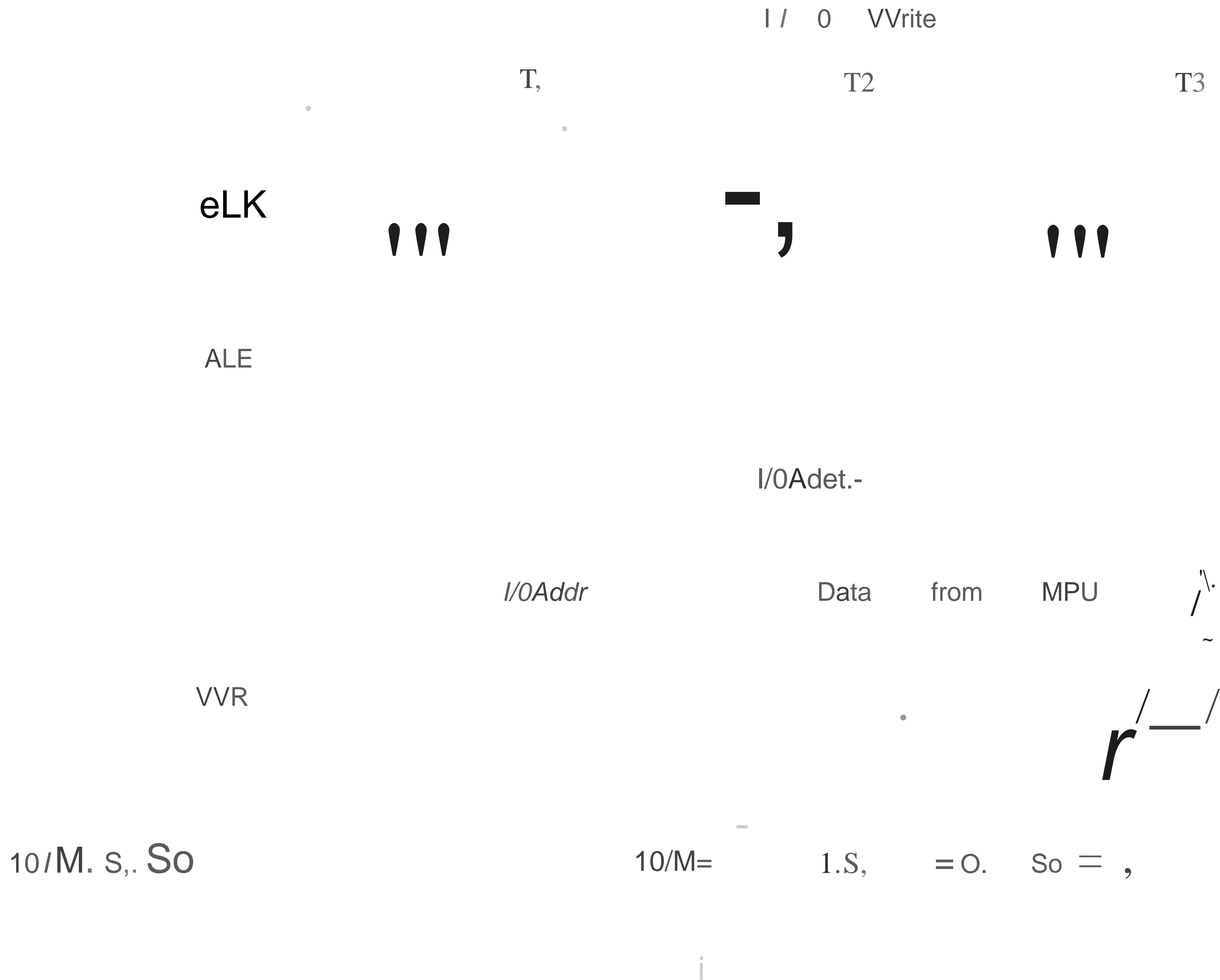
10 / fVI fO /"Q = O.$, = O. 50= ~

(

WiR



(b) *1/0* read memory cycle



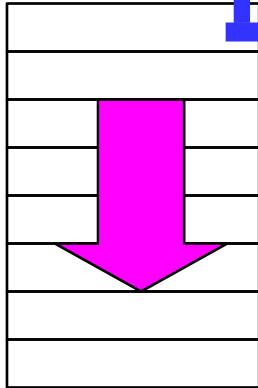
(b) vawrite machine cycle

Interrupts

- An interrupt is considered to be an emergency signal that may be serviced.
  - The Microprocessor may respond to it as soon as possible.
- What happens when MP is interrupted ?
  - When the Microprocessor receives an interrupt signal, it **suspends the currently executing program** and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.
  - Each interrupt will most probably have its own ISR.

Interrupts in 8085

TYP ES O F INTERRUPT



Interrupt

Save program counter

Disable interrupts

OF HARDWARE

Send out interrupt acknowledge

INTA

Go to service routine

Service routine

Get original program counter

Go back

Main routine

VECTORED AND NON "ECTORED

lVIASKABLE AND NO N-lVIASKi-\BLE

Addressed

Delayed

- Classification of Interrupts

Interrupts can be classified into two types:

- Maskable Interrupts (Can be delayed or Rejected)
- Enable Or Disable By EI And DI Instruction

- Non-Maskable Interrupts (Can not be delayed or Rejected)

Interrupts can also be classified into two types:

- Vectored (the address of the service routine is hard-wired)

- Non-vectored (the address of the service routine needs to be supplied externally by the device)

Responding to Interrupts

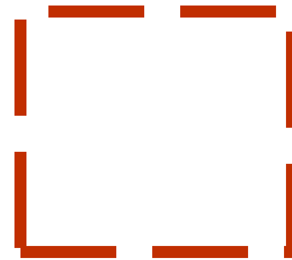
- Responding → "delayed or immediate"
  " Maskable or Non-maskable "
- Redirecting the execution to the ISR
  " Pre Defined Address or Address to be Defined"
  " Vectored or Non-vectored "

  - Vectored: The address of the subroutine is already known to the Microprocessor

  - Non Vectored: The device will have to supply the address of the subroutine to the Microprocessor

5 - Interrupts in 8085

- There are 5 interrupt inputs:
  - TRAP (non maskable)
  - RST7.5
  - RST6.5
  - RST5.5
  - INTR



The 8085 Interrupts

- The 8085 has 5 interrupt inputs.
  - The INTR input.
    - The INTR input is the only non-vectored interrupt.
    - INTR is maskable using the EI/DI instruction pair.
  - RST 5.5, RST 6.5, RST 7.5 are all automatically vectored.
    - RST 5.5, RST 6.5, and RST 7.5 are all maskable.
  - TRAP is the only non-maskable interrupt in the 8085
    - TRAP is also automatically vectored

The 8085 Interrupts

Interrupt name	Maskable	Vectored	VECTOR ADDRESS
TRAP	No	Yes	0024H
RST 7.5	Yes	Yes	003CH
RST 6.5	Yes	Yes	0034H
RST 5.5	Yes	Yes	002CH
INTR	Yes	No	--

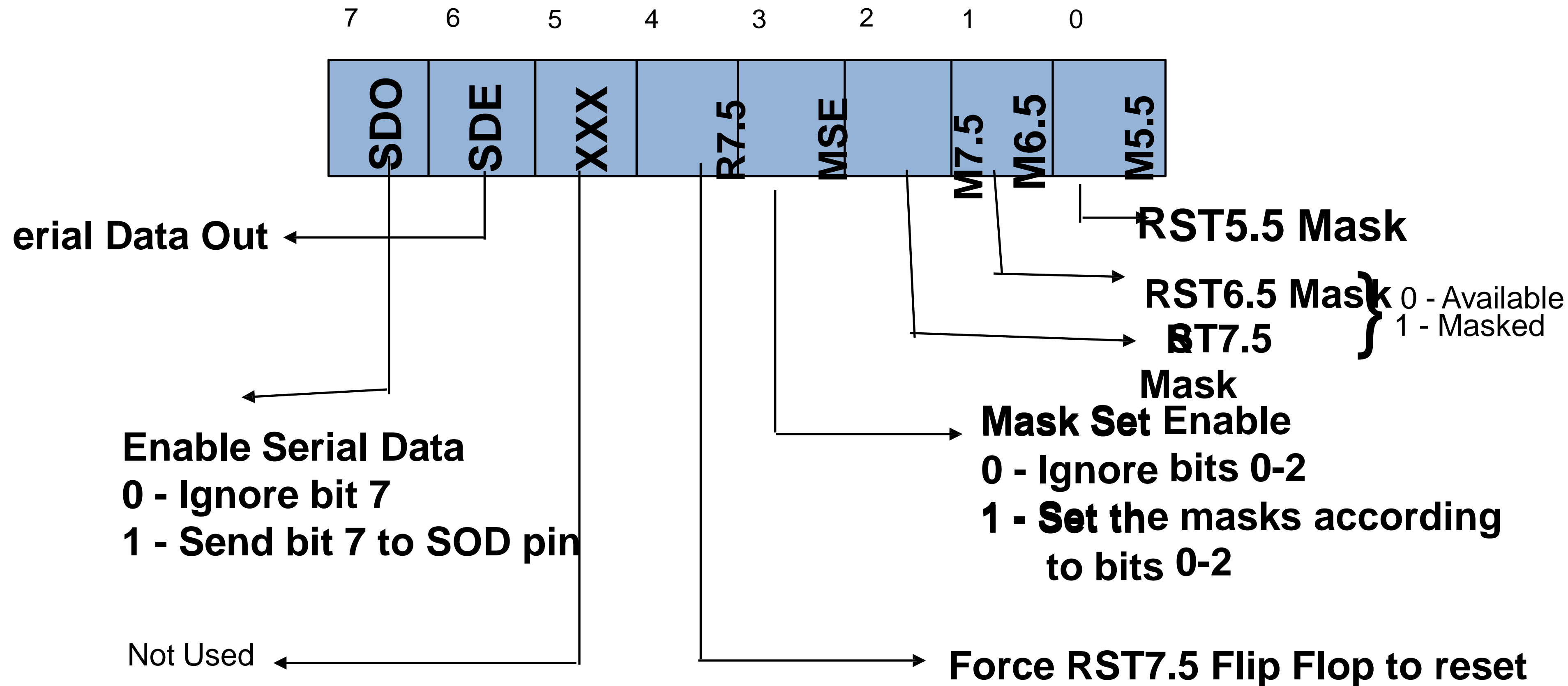
8085 INTERRUPTS

- The ‘**EI**’ instruction is a one byte instruction and is used to **Enable the maskable interrupts.**
- The ‘**DI**’ instruction is a one byte instruction and is used to **Disable the maskable interrupts.**
- The 8085 has a single Non-Maskable interrupt. **“TRAP”**

8085 Interrupts

Interrupt type	Trigger	Priority	Maskable	Vector address
TRAP	Edge and Level	1st	No	0024H
RST 7.5	Edge	2nd	Yes	003CH
RST 6.5	Level	3rd	Yes	0034H
RST 5.5	Level	4th	Yes	002CH
INTR	Level	5th	Yes	-

How SIM Interprets the Accumulator



SIM and the Interrupt Mask

- Bit 0 is the mask for RST 5.5
- Bit 1 is the mask for RST 6.5
- Bit 2 is the mask for RST 7.5.
    - If the mask bit is 0, the interrupt is available.
    - If the mask bit is 1, the interrupt is masked.

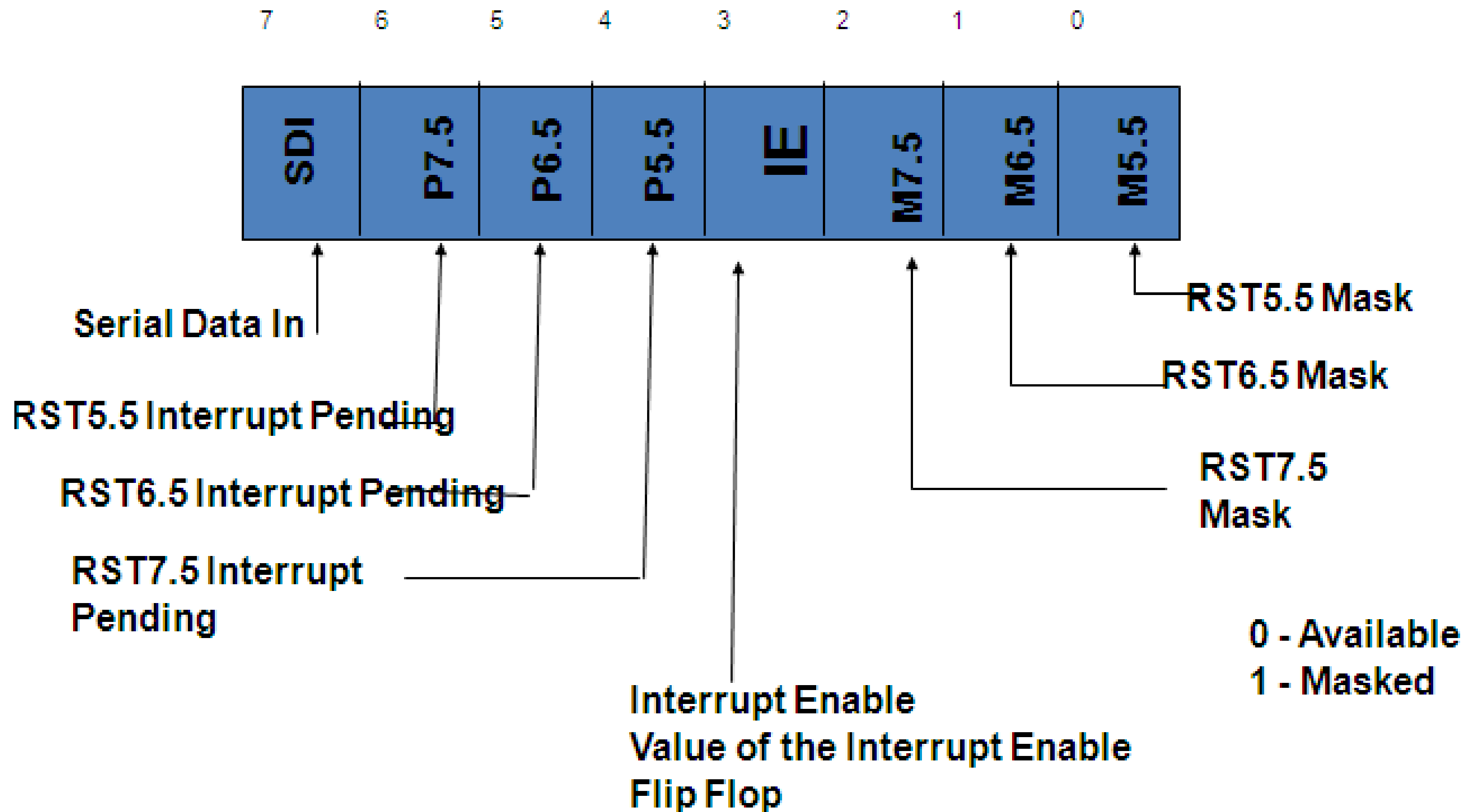
- Bit 3 (Mask Set Enable - MSE) is an enable for setting the mask.
  - If it is set to 0 the mask is ignored and the old settings remain.
  - If it is set to 1, the new setting are applied.
  - The SIM instruction is used for multiple purposes and not only for setting interrupt masks.
    - It is also used to control functionality such as Serial Data Transmission.
    - Therefore, bit 3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified

SIM and the Interrupt Mask

- The RST 7.5 interrupt is the only 8085 interrupt that has memory.
  - If a signal on RST7.5 arrives while it is masked, a flip flop will remember the signal.
  - When RST7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal.
  - This flip flop will be automatically reset when the microprocessor responds to an RST 7.5 interrupt.

- Bit 4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.
- Bit 5 is not used by the SIM instruction

RIM sets the Accumulator's different bits



TRAP

- TRAP is the only non-maskable interrupt.
  - It does not need to be enabled because it cannot be disabled.
- It has the highest priority amongst interrupts.
- It is edge and level sensitive.
  - It needs to be high and stay high to be recognized.
  - Once it is recognized, it won't be recognized again until it goes low, then high again.
- TRAP is usually used for power failure and emergency shutoff.

Types of Addressing Modes

- Intel 8085 uses the following addressing modes:
  1. Direct Addressing Mode
  2. Register Addressing Mode
  3. Register Indirect Addressing Mode
  4. Immediate Addressing Mode
  5. Implicit Addressing Mode

Direct Addressing Mode

- In this mode, the address of the operand is given in the instruction itself.

LDA 2500 H	Load the contents of memory location 2500 H in accumulator.

- LDA is the operation.
- 2500 H is the address of source.
- Accumulator is the destination.

Register Addressing Mode

- In this mode, the operand is in general purpose register.

MOV A, B	Move the contents of register B to A.

- MOV is the operation.
- B is the source of data.
- A is the destination.

Register Indirect Addressing Mode

- In this mode, the address of operand is specified by a register pair.

MOV A, M	Move data from memory location specified by H-L pair to accumulator.

- MOV is the operation.
- M is the memory location specified by H-L register pair.
- A is the destination.

Immediate Addressing Mode

- In this mode, the operand is specified within the instruction itself.

MVI A, 05 H Move 05 H in accumulator.

- MVI is the operation.
- 05 H is the immediate data (source).
- A is the destination.

Implicit Addressing Mode

- If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction.

CMA	Complement accumulator.

- CMA is the operation.
- A is the source.
- A is the destination.

Introduction to Java

What Is Java?

- History
- Characteristics of Java

History

- James Gosling and Sun Microsystems
- Oak
- Java, May 20, 1995, Sun World
- HotJava
  - The first Java-enabled Web browser
- JDK Evolutions
- J2SE, J2ME, and J2EE (not mentioned in the book, but could discuss here optionally)

Characteristics of Java

- Java is simple
- Java is object-oriented
- Java is distributed
- Java is interpreted
- Java is robust
- Java is secure
- Java is architecture-neutral
- Java is portable
- Java's performance
- Java is multithreaded
- Java is dynamic

JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- Java 2 SDK v 1.2 (a.k.a JDK 1.2, 1998)
- Java 2 SDK v 1.3 (a.k.a JDK 1.3, 2000)
- Java 2 SDK v 1.4 (a.k.a JDK 1.4, 2002)

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

Java IDE Tools

- Forte by Sun MicroSystems
- Borland JBuilder
- Microsoft Visual J++
- WebGain Café
- IBM Visual Age for Java

Getting Started with Java Programming

- A Simple Java Application
- Compiling Programs
- Executing Applications

A Simple Application

Example 1.1

```
//This application program prints Welcome
//to Java!
package chapter1;

public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

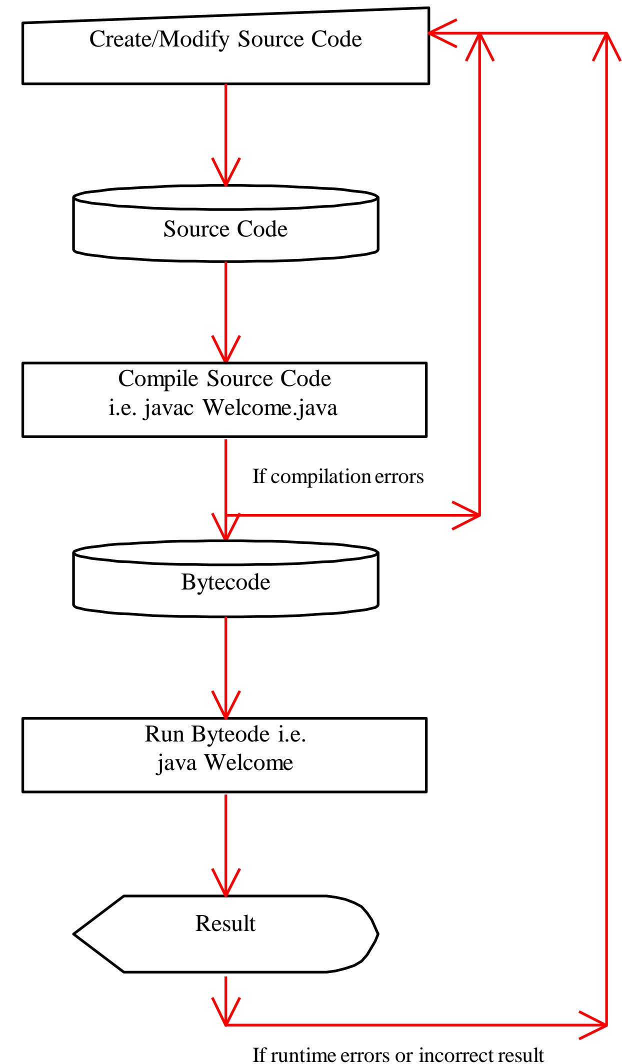
Source

Run

NOTE: To run the program, install slide files on hard disk.

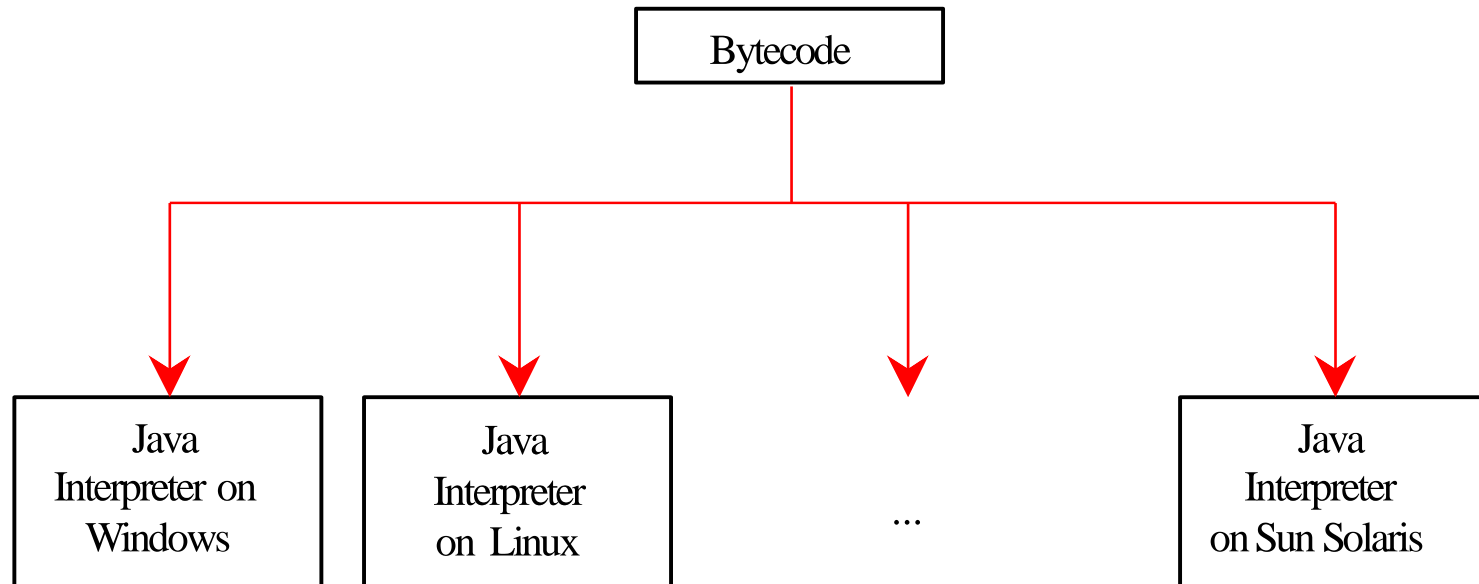
Creating and Compiling Programs

- On command line
  - `javac file.java`



Executing Applications

- On command line
  - `java classname`



Example

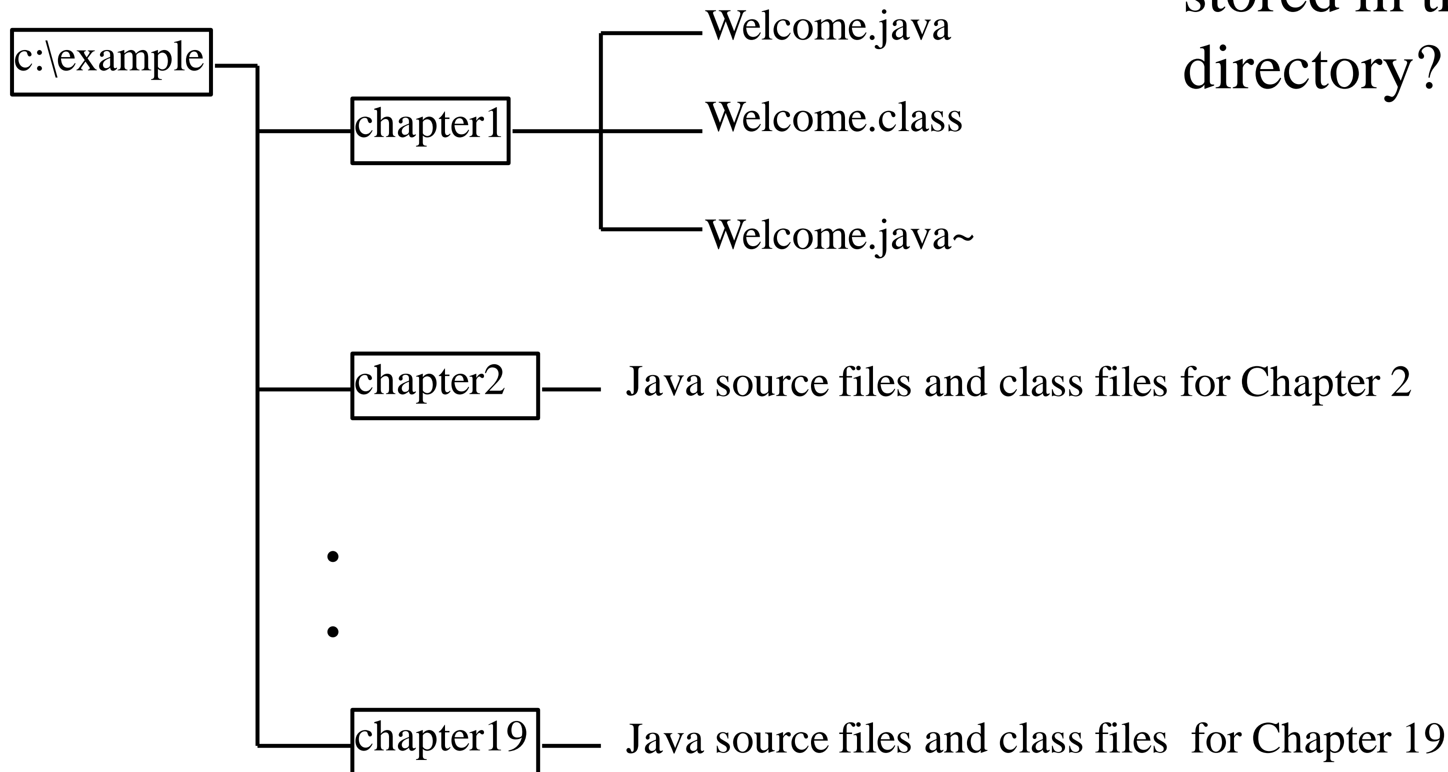
```
javac Welcome.java

java Welcome

output:...
```

Compiling and Running a Program

Where are the files stored in the directory?



Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

In Java, comments are preceded by two slashes (//) in a line, or enclosed between /* and */ in one or multiple lines. When the compiler sees //, it ignores all text after // in the same line. When it sees /*, it scans for the next */ and ignores any text between /* and */.

Reserved Words

Reserved words or *keywords* are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class. Other reserved words in Example 1.1 are public, static, and void.

Modifiers

Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected. A public datum, method, or class can be accessed by other programs. A private datum or method cannot be accessed by other programs.

Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Example 1.1 is a statement to display the greeting "Welcome to Java!" Every statement in Java ends with a semicolon (;).

Blocks

A pair of braces in a program forms a block that groups components of a program.

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Class block

Method block

Classes

The *class* is the essential Java construct. A class is a template or blueprint for objects. To program in Java, you must understand classes and be able to write and use them. The mystery of the class will continue to be unveiled throughout this book. For now, though, understand that a program is defined by using one or more classes.

Methods

What is System.out.println? It is a *method*: a collection of statements that performs a sequence of operations to display a message on the console. It can be used even without fully understanding the details of how it works. It is used by invoking a statement with a string argument. The string argument is enclosed within parentheses. In this case, the argument is "Welcome to Java!" You can call the same println method with a different argument to print a different message.

main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[] args) {
  // Statements;
}
```

Displaying Text in a Message Dialog Box

you can use the showMessageDialog method in the JOptionPane class. JOptionPane is one of the many predefined classes in the Java system, which can be reused rather than "reinventing the wheel."

Source   Run

The showMessageDialog Method

```
JOptionPane.showMessageDialog(null, "Welcome to
Java!",
  "Example 1.2",
JOptionPane.INFORMATION_MESSAGE));
```

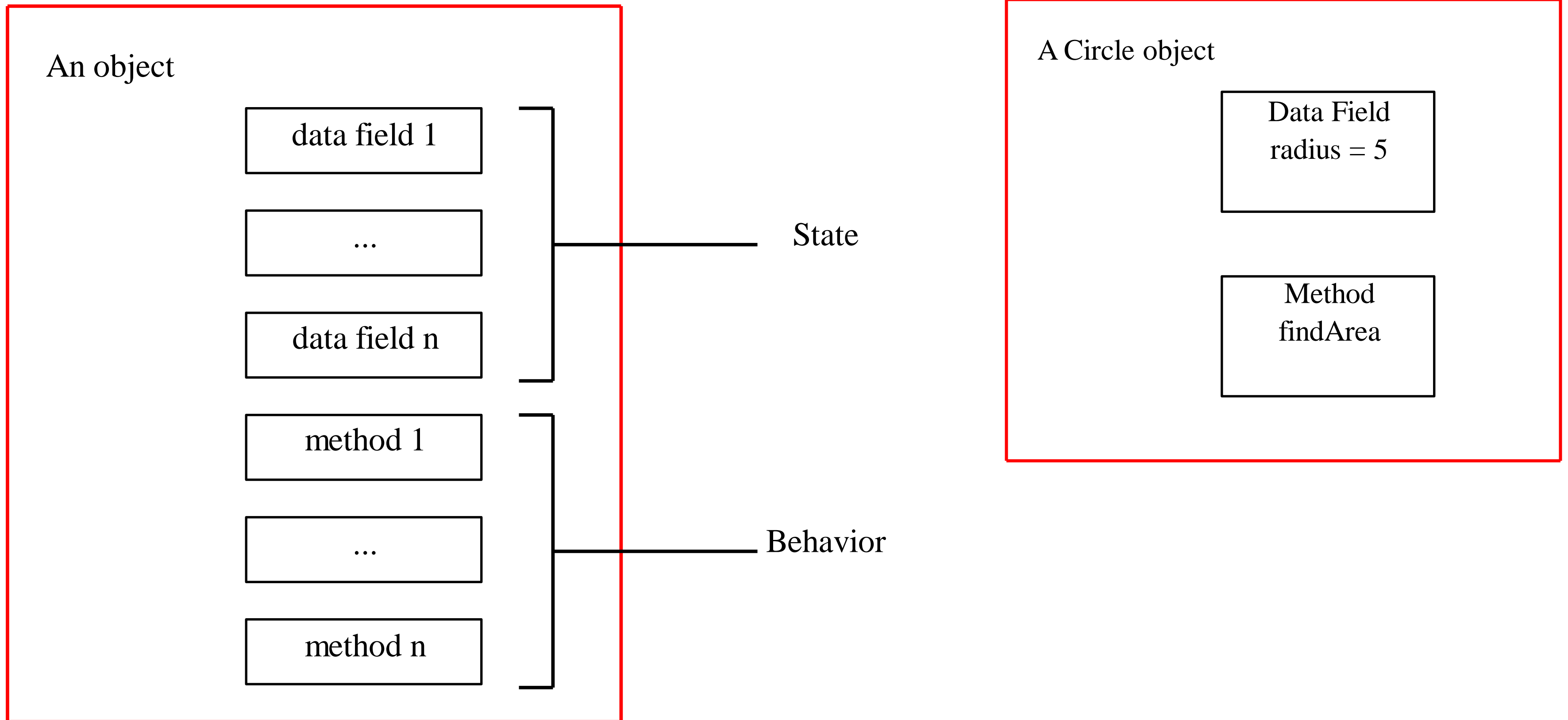
The exit Method

Use Exit to terminate the program and stop all threads.

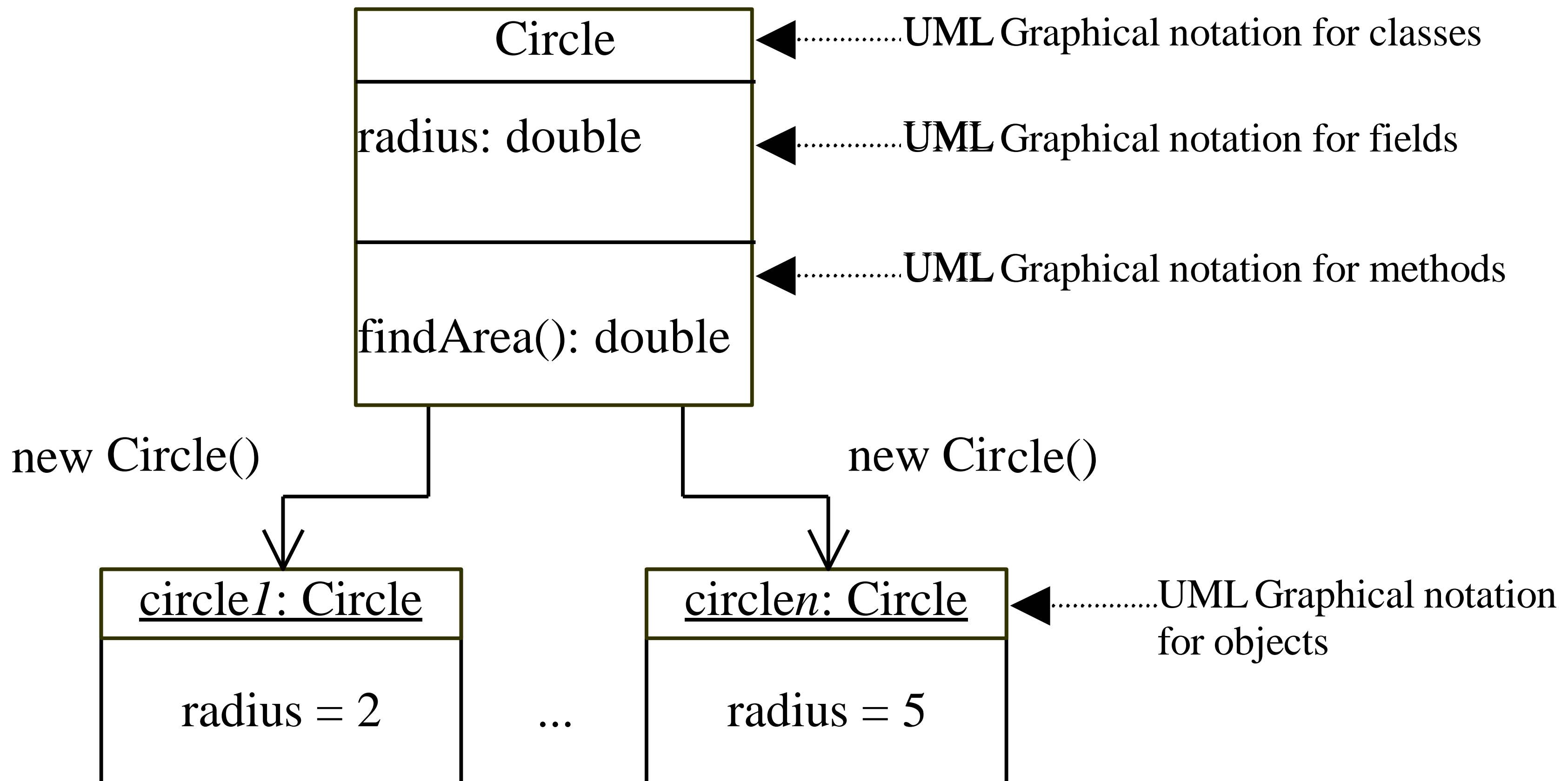
NOTE: When your program starts, a thread is spawned to run the program. When the showMessageDialog is invoked, a separate thread is spawned to run this method. The thread is not terminated even you close the dialog box. To terminate the thread, you have to invoke the exit method.

Java Classes

OO Programming Concepts



Class and Objects



Class Declaration

```
class Circle {
  double radius = 1.0;

  double findArea(){
    return radius * radius * 3.14159;
  }
}
```

Declaring Object Reference Variables

```
ClassName objectReference;
```

Example:

```
Circle myCircle;
```

Creating Objects

```
objectReference = new ClassName();
```

Example:

```
myCircle = new Circle();
```

The object reference is assigned to the object reference variable.

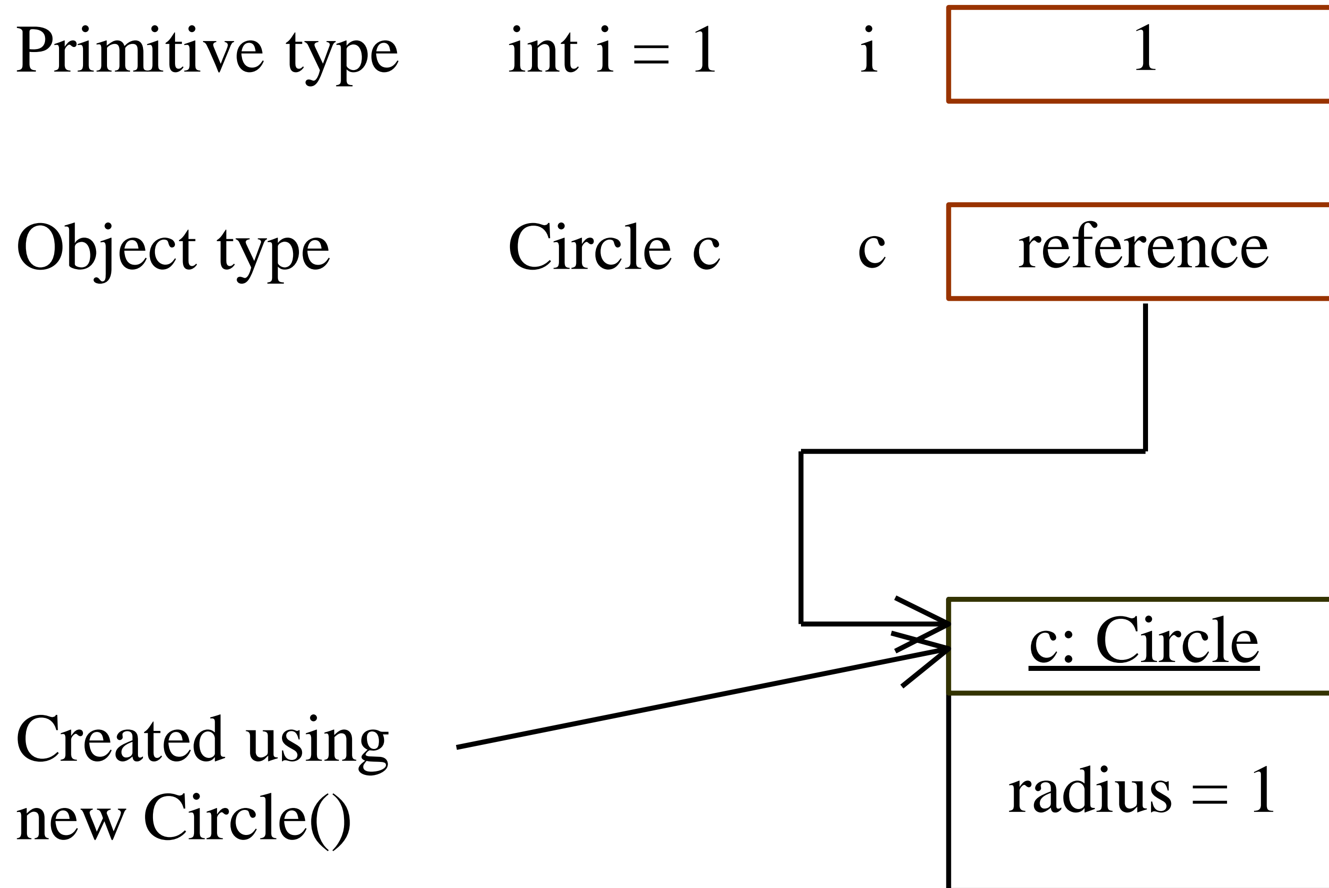
Declaring/Creating Objects in a Single Step

```
ClassName objectReference = new ClassName();
```

Example:

```
Circle myCircle = new Circle();
```

Differences between variables of primitive Data types and object types

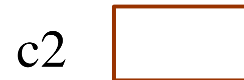


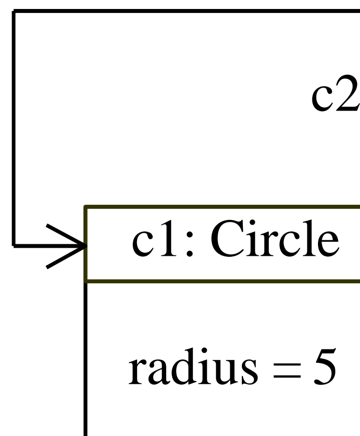
Copying Variables of Primitive Data Types and Object Types

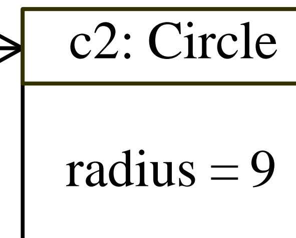
Primitive type assignment
i = j

Object type assignment
c1 = c2

Before: After: Before: After:







Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer useful. This object is known as garbage. Garbage is automatically collected by JVM.

Garbage Collection, cont

TIP: If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The Java VM will automatically collect the space if the object is not referenced by any variable.

Accessing Objects

- Referencing the object's data:

```
objectReference.data
myCircle.radius
```

- Invoking the object's method:

```
objectReference.method
myCircle.findArea()
```

Using Objects

- Objective: Demonstrate creating objects, accessing data, and using methods.

TestCircle

Run

Constructors

```
Circle(double r) {
  radius = r;
}

Circle() {
  radius = 1.0;
}
```

Constructors are a special kind of methods that are invoked to construct objects.

```
myCircle = new Circle(5.0);
```

Constructors, cont.

A constructor with no parameters is referred to as a *default constructor*.

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

Using Classes from the Java Library

- Objective: Demonstrate using classes from the Java library. Use the JFrame class in the javax.swing package to create two frames; use the methods in the JFrame class to set the title, size and location of the frames and to display the frames.

TestFrame

Run

Using Constructors

- Objective: Demonstrate the role of constructors and use them to create objects.

TestCircleWithConstructors

Run

Visibility Modifiers and Accessor Methods

By default, the class, variable, or data can be accessed by any class in the same package.

- `public`

  The class, data, or method is visible to any class in any package.

- `private`

  The data or methods can be accessed only by the declaring class.

The get and set methods are used to read and modify private properties.

Using the `private` Modifier and Accessor Methods

In this example, private data are used for the radius and the accessor methods getRadius and setRadius are provided for the clients to retrieve and modify the radius.

TestCircleWithAccessors

Run

Passing Objects to Methods

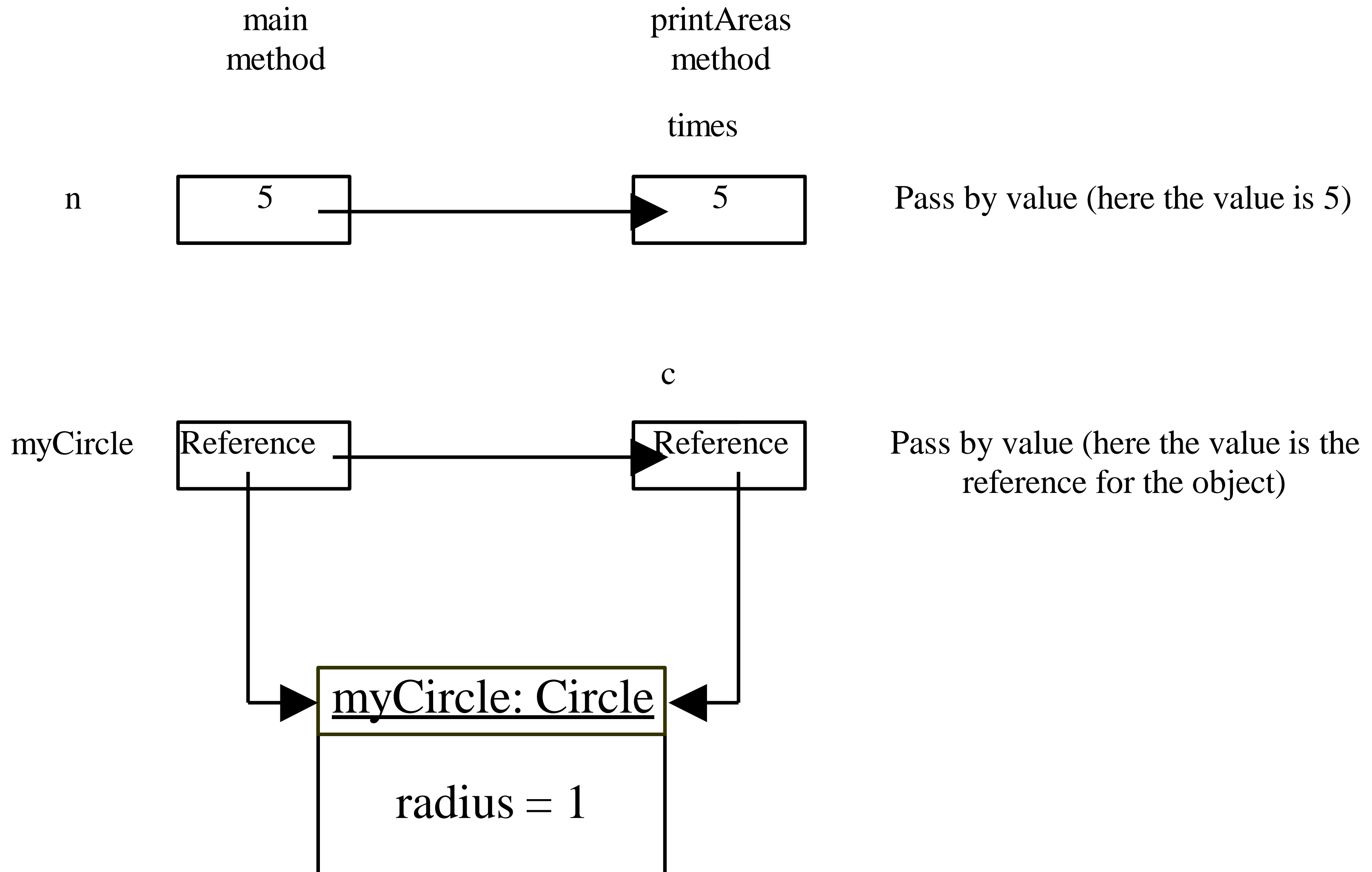
- Passing by value (the value is the reference to the object)

Example 6.5 Passing Objects as Arguments

TestPassingObject

Run

Passing Objects to Methods, cont.



Instance
Variables and Methods

Instance variables belong to a specific instance.

Instance methods are invoked by an instance of the class.

Class Variables, Constants, and Methods

Class variables are shared by all the instances of the class.

Class methods are not tied to a specific object.

Class constants are final variables shared by all the instances of the class.

Class Variables, Constants, and Methods, cont.

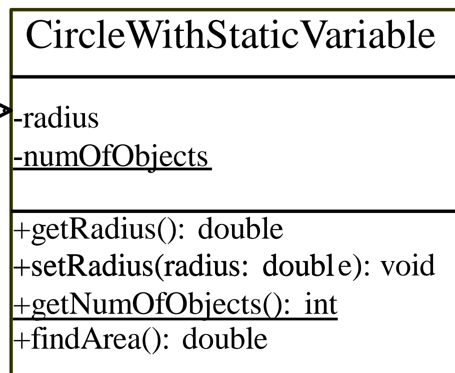
To declare class variables, constants, and methods, use the static modifier.

Class Variables, Constants, and Methods, cont.

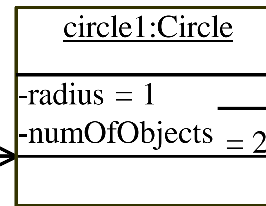
UML Notation:
+: public variables or methods
-: private variables or methods
underline: static variables or metods

Memory

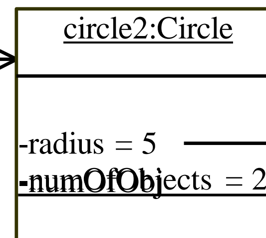
radius is an instance variable, and numOfObjects is a class variable



instantiate



instantiate



radius

numOfObjects

radius

Using Instance and Class Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable numOfObjects to track the number of Circle objects created.

TestCircleWithStaticVariable

Run

Scope of Variables

- The scope of instance and class variables is the entire class. They can be declared anywhere inside a class.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

The Keyword this

- Use this to refer to the current object.
- Use this to invoke other constructors of the object.

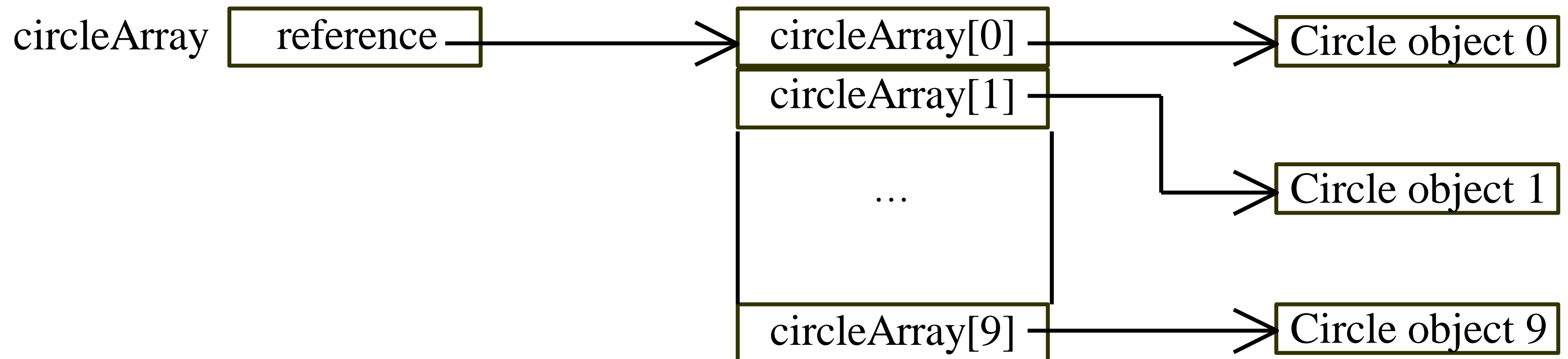
Array of Objects

```
Circle[] circleArray = new Circle[10];
```

An array of objects is actually an *array of reference variables*. So invoking circleArray[1].findArea() involves two levels of referencing as shown in the next figure. circleArray references to the entire array. circleArray[1] references to a Circle object.

Array of Objects, cont.

```
Circle[] circleArray = new Circle[10];
```



Array of Objects, cont.

Summarizing the areas of the circles

TotalArea

Run

Class Abstraction

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.

Java API and Core Java classes

- `java.lang`

  Contains core Java classes, such as numeric classes, strings, and objects. This package is implicitly imported to every Java program.

- `java.awt`

  Contains classes for graphics.

- `java.applet`

  Contains classes for supporting applets.

Java API and Core Java classes, cont.

- `java.io`

  Contains classes for input and output streams and files.

- `java.util`

  Contains many utilities, such as date.

- `java.net`

  Contains classes for supporting network communications.

Java API and Core Java classes, cont.

- `java.awt.image`

  Contains classes for managing bitmap images.

- `java.awt.peer`

  Platform-specific GUI implementation.

- Others:

  `java.sql`

  `java.rmi`

Thank You