COMPUTER ARCHITECTURE

UNIT STANDARDS

14921:	Describe the types of computer systems and associated hardware configurations
14917:	Explain computer architecture concepts
14944:	Explain how data is stored on computers

Visit us on our website or E-mail us at:
mscho@msccollege.co.za
www.msccollege.co.za

MSC EDUCATION HOLDINGS (PTY) LTD
POSTNET 262
PRIVATE BAG X9063
EAST LONDON
5200

COMARC11V1

TABLE OF CONTENTS

MODULE UNIT STANDARD ALIGNMENT

This module is aligned to the following unit standards:

US Number	Title	NQF	Credits
14921	Describe the types of computer systems and associated hardware configurations	4	6
14917	Explain computer architecture concepts	4	7
14944	Explain how data is stored on computers	4	7

PURPOSE, OUTCOMES AND ASSESSMENT CRITERIA

14921 - Describe the types of computer systems and associated hardware configurations

This unit standard is intended:

- to provide conceptual knowledge of the areas covered
- for those working in, or entering the workplace in the area of Information Systems & Technology Management
- as additional knowledge for those wanting to understand the areas covered

People credited with this unit standard are able to:

- Describe past, present and future computer hardware configurations.
- Describe categories of computer system applications.

The performance of all elements is to a standard that allows for further learning in this area.

Outcome 1

- Describe past, present and future computer hardware configurations.

Assessment Criteria

- The description lists the characteristics of the configurations.
- The description justifies categorisation of examples.
- The description explains performance characteristics of the configurations.
- The description explains the environmental requirements of the configurations.

Outcome 2

- Describe categories of computer system applications.

Assessment Criteria

- The description identifies categories of computer system applications.
- The description justifies categorisation of examples.
- The description explains the performance characteristics of the categories.

14917 – Explain computer architecture concepts.

Purpose: This unit standard is intended:

- to provide a fundamental knowledge of the areas covered
- for those working in, or entering the workplace in the area of Information Systems & Technology Management.
- as additional knowledge for those wanting to understand the areas covered

People credited with this unit standard are able to:

- Explain computer architecture elements
- Explain the organisation of a computer
- Describe the design constraints in the design of instruction sets for computers

The performance of all elements is to a standard that allows for further learning in this area

Outcome 1

- Explain computer architecture elements.

Assessment Criteria

- ○ The explanation identifies the functions of elements which make up computer architecture.
- ○ The explanation outlines the functions of elements which make up computer architecture.
- ○ The explanation distinguishes categories of each element and outlines their features.
- ○ The explanation identifies examples of the application of architecture elements.

Outcome 2

- Explain the organisation of a computer.

Assessment Criteria

- ○ The explanation identifies the purpose of computer components.
- ○ The explanation outlines how components achieve their outcomes in terms of their relationships, and the structure of the computer.

Outcome 3

- Describe the design constraints in the design of instruction sets for computers.

Assessment Criteria

- ○ The description identifies the constraints, and outlines the issues involved.
- ○ The description outlines how the constraints have been accommodated, by using examples.

14944 – Explain how data is stored on computers.

Purpose: This unit standard is intended:

- to provide conceptual knowledge of the areas covered
- for those working in, or entering the workplace in the area of Information Systems and Technology Management

- as additional knowledge for those wanting to understand the areas covered.

People credited with this unit standard are able to:

- Describe the roles of IS departments in organisations
- Describe the structures of IS departments in organisations

The performance of all elements is to a standard that allows for further learning in this area.

Outcome 1

- Demonstrate an understanding of computer data types.

Assessment Criteria

- ○ The demonstration distinguishes between data types and includes examples (Range: Bits (0, 1) , Bytes, Numbers , Integers (+- values, whole items), Floating point (temperature, voltage, etc), Boolean (0/1 on/off)).
- ○ The description of the use of coding systems in a business environment distinguishes categories of coding systems and includes examples(Range: BCD, EBCDIC (IBM mainframes), ASCII (micro & mini computers), UNICODE).
- ○ The demonstration illustrates how data manipulation operations are performed on data types.

Outcome 2

- Describe computer data structures.

Assessment Criteria

- ○ The description distinguishes types of computer data structures and identifies examples (Range: Bits, Bytes, Characters, Fields, Records, Files, Databases).
- ○ The description distinguishes types of computer files and identifies examples (Range: Master, Transaction, Temporary, Document , Serial, Sequential, Indexed, Direct).
- ○ The description distinguishes types of computer databases and identifies examples (Range: Hierarchical, Network, Relational, SQL (Completeness, Non-redundancy, Structure).

ALIGNMENT MATRIX

The above outcomes can be found in the following learning units:

Unit Std Number and Name	Specific Outcomes	Learning Unit Number and Name
14921: Describe the types of computer systems and associated hardware configurations	Specific Outcome 1 Specific Outcome 2 Specific Outcome 3 Specific Outcome 4	Learning Unit 1: Types of computer hardware configurations and associated application systems
14917: Explain computer architecture concepts	Specific Outcome 1 Specific Outcome 2 Specific Outcome 3	Learning Unit 2: Computer Architecture
14944: Explain how data is stored on computers	Specific Outcome 1 Specific Outcome 2	Learning Unit 3: Storage of Data

KEY TO ICONS USED IN THIS MODULE

	Group work, but remember that you need to report back.		Individual activity
	Questions		Role-play
	Class discussion		Research
	Case Study		Notes and observations
	Formative Assessment Activity		Summative Assessment Preparation

1

LEARNING UNIT 1: TYPES OF COMPUTER HARDWARE CONFIGURATIONS AND ASSOCIATED APPLICATION SYSTEMS

Learning outcomes to be achieved

- Describe past, present and future computer hardware configurations.
- Describe categories of computer systems applications.

1. INTRODUCTION

1.1 COMPUTER HARDWARE CONFIGURATIONS

The four major groups of computers are micro-computer, mini-computer mainframe and supercomputer. This classification of hardware configurations is largely historical based and an outgrowth of the different requirements that each computer generation was created for.

1.1.1 Microcomputers

The term "Microcomputer" came into popular use after the introduction of the minicomputer which simply means a small computer. Microcomputer is a desktop sized digital computer that can serve one user at a time.

The microcomputer is also known as the "Personal Computer ", stand-alone computer or desktop computer. The name "Personal Computer" is largely related to IBM who introduced and marketed the first widely available, commercialized microcomputer called "IBM PC" where "PC" stood for Personal Computer. As the most dominant microcomputer in the introduction of microcomputers the IBM PC became the standard against which all microcomputers were compared and the branding that IBM created, the "PC" helped provide a name for the industry as it grew.

The category is termed "micro" because the fundamental component that allowed the category's development was the micro-processor, a revolutionary design in processors that brought down the prices of computers and also pushed the envelope on performance.

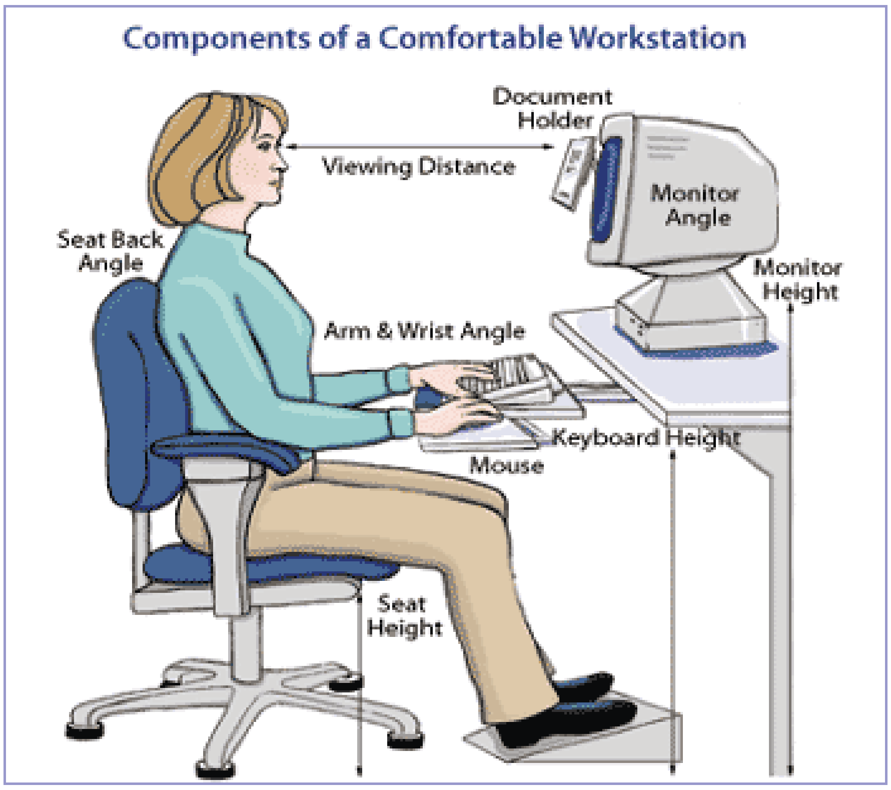
The lower and affordable costs in the development and production of the microprocessor and various components of the microcomputer allowed microcomputers to become a broad market general purpose computer. The general appeal or focus of microcomputers on the individual's enjoyment and productivity has been a key factor in the microcomputer's development and enhancement over the years. Where other computers were used by a broad group of people within an Organisation, the microcomputer was mostly used by the individuals for single activities that required more processing power than other technologies of the time had to offer.

Where larger systems were focused on the productivity of the whole department or company, the microcomputer was often only effective for a single person's work.

One example of how Microcomputers differ is the use of Games. In many cases, people buy microcomputers that are capable or running high-end games. Not only does this provide entertainment value but it also ensures high performance capabilities. If you can play the latest games, you can probably run the most stringent of applications.

Microcomputers can be grouped into four smaller groups. They are workstations (desktops), laptops, notebooks and PDAs. Each refers to the physical usage of the machines and not necessarily the power or the size. Workstations generally refer to machines used for intensive calculations or designs such as architecture and buildings.

Figure - Workstations



Workstations also refer to machines used by business people for word processing, spread sheeting etc., and are physically placed onto a desk. The laptop and the notebook are portable machines.

Laptops generally are bigger and clumsier due to the power supply being encased in the box holding the CPU. The notebook is normally smaller. Notebooks and laptops can work on batteries for hours allowing the user to work with it in aeroplanes, in the car and places computers are not normally available. Initially designed as stand-alone machines the PC and the operating systems popular on the PCs do not integrate as well into networked environments as the other computer systems. Personal Digital Assistants (PDAs) are a very recent enhancement of electronic productivity tools based on microprocessor technology. Personal Digital Assistants such as Apple's Message pad 2000 and later The Apple Message Pad 2100, with a vastly improved handwriting recognition system, 162 MHz Strong ARM SA-110 RISC processor, Newton OS 2.1, and a better, clearer, backlit screen, attracted critical plaudits. This new technology has provided business people, and technocrats, with common daily Organisation tools (such as a diary, telephone book, and notepad) in a small packaging usually no larger than a stenographer/short-hand notepad and many other.

1.1.2 Minicomputers

Minicomputers are a mid-sized computer that is usually placed out of sight in an air conditioned room. We normally refer to these computers as servers. They usually store information that's relevant to a host of people and they have enough processing power to cope with high amounts of processing. They allow for simultaneous access from multiple users. They are smaller than mainframe computers, slower and more affordable.

As a rule, these computers are untouched and mostly unseen. There's no need for any user interface or input and output devices. In general, the only requirement is a network connection.

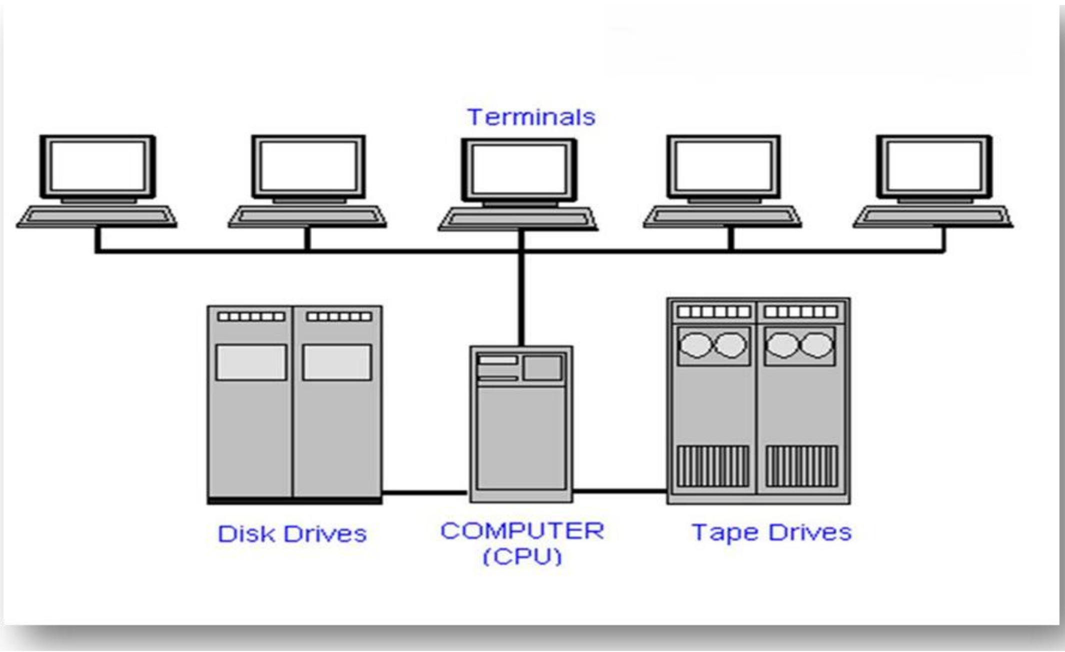
The technology advancement from vacuum tubes to transistors leads to the ability of manufacturers and designers being able to create computers smaller than mainframes and consequently more affordable for a broader range of buyers. The advancement of technology in microchips and microprocessors have allowed minicomputers, although more expensive, to maintain a greater processing power and then the more commonly used microcomputer.

For example	Large supermarkets around the world need to have their cash register send sales information to the same computer (so that the data is collected in one place). This work requires more input devices (cash registers) and output devices (Screens) than the microcomputer was designed to have access to. Some Organisations, like TCF and Morris Hedstrom, operate the local supermarket using local area networks and microcomputers, but tie the various (100's) supermarkets together to the central office to a minicomputer. The minicomputer is also required to

	work when the entire sales people key in sales information at the same time, this requires a powerful machine that can work with all the input devices at the same time.

Minicomputers are used by medium sized business and small sites of large organisations such as Nissan South Africa. They are also used in factories to control automated assembly lines for manufacturing or large process control. Where numerous equipment have to be coordinated and operated in time-critical sequence, such as a car manufacturing plant, or chemical plant, minicomputers are found to coordinate the many peripheral devices, collate and assess input from many other peripheral devices.

Figure - Minicomputers



The microcomputer is inadequate in processing power and peripheral connections to complete the work and mainframe computers are too expensive for the job to be cost-effective.

An important measure for minicomputers and mainframes is the reliability of the machine as it generally has to operate 24 hours where every minute of operation is important to the company. A minute of downtime, where the computer is not functioning correctly, is calculated in lost money for organisations. Microcomputers have not been manufactured nor warranted for critical operations and many microcomputer manufacturers explicitly state in their promotional material that the microcomputers are not designed nor intended for critical operations.

	Terminals used by computer operators to enter and review data do not have local hard disks nor floppy disks. A typical minicomputer may have 20 to 100 terminals connected to the mini allowing as many people to review data on screen, as well as from printouts.

Popular makers of minicomputers include DEC - Digital Equipment Corporation who built the popular VAX minicomputer used in universities, banks and engineering firms.

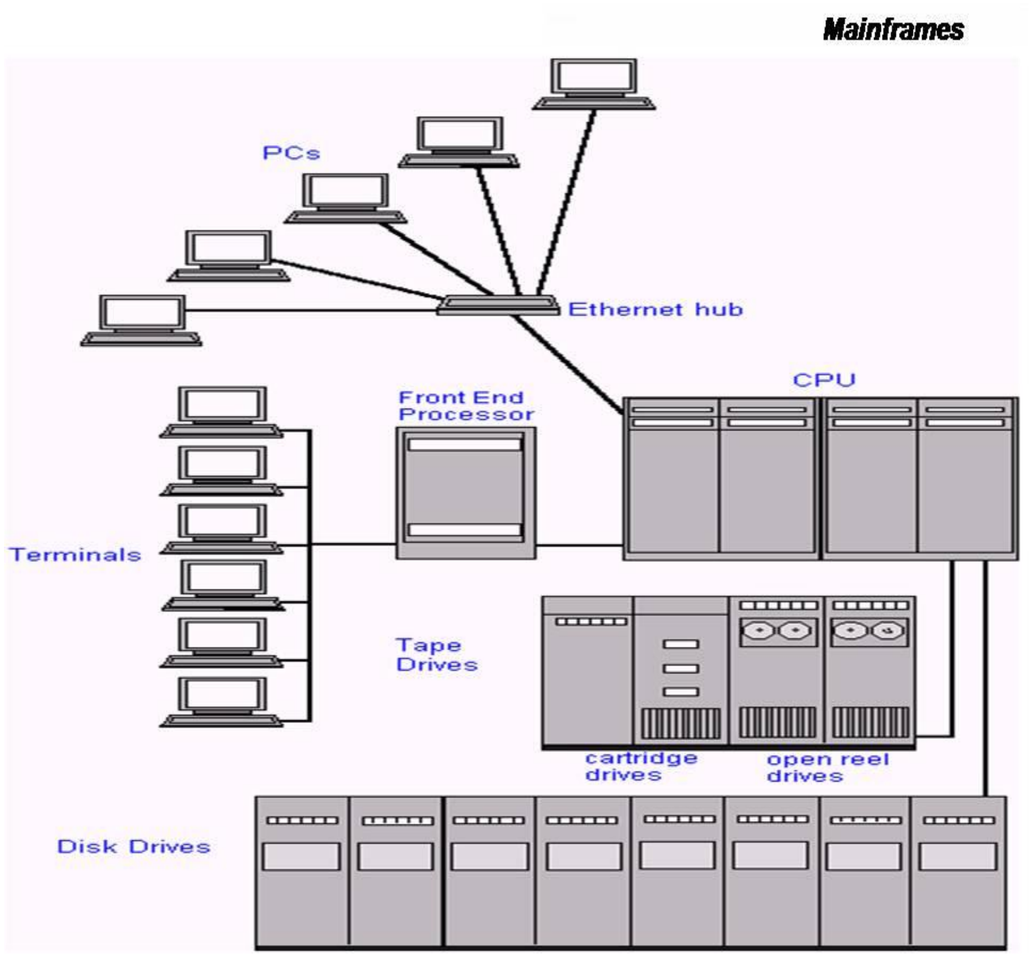
IBM also creates a very popular minicomputer range with a branding of the AS400.

Hewlett Packard has a popular minicomputer range branded the HP9000.

1.1.3 Mainframes

Mainframes (often colloquially referred to as Big Iron are powerful computers used mainly by large Organisations for critical applications, typically bulk data processing such as census, industry and consumer statistics, enterprise resource planning, and financial transaction processing.

Figure - The Mainframe



The term originally referred to the large cabinets that housed the central processing unit and main memory of early computers. Later the term was used to distinguish high-end commercial machines from less powerful units.

Mainframes have even more access to storage space and to input/output devices. To work with these extra devices mainframes also have more powerful 'processors'. This power is useful and required by large corporations who have large amounts of data to process.

For example	Large overseas banks that have millions of customer accounts to update regular will need very powerful machines to process this data. These large banks would have a mainframe maintain their customer account records (like their bank book account balance, how much they withdrew, deposited) so that the customer can turn-up at any of the branches (local bank building) to withdraw/deposit money. So Thabo who has a bank account in ABSA can take a trip from Pretoria to Cape Town, turn up to the bank in Khayalitsha, Cape Town to withdraw some money.

Another power of mainframes is that they are designed to connect input/output devices that span vast distances. Like the above example, to connect devices that can be as far away from each other as Pretoria and Cape Town, Khayalitsha. The powerful hardware and CPU chips that can support the above work can also be used for making complex calculations, and designs.

Mainframes are often used by corporations who design jets or studying aerodynamics effects etc. One major inducement for the development of these computers originally was the increasing volumes of data collected by the US Bureau of Statistics during the US Census. The volume of data was growing rapidly such that the data required over 9 years collating, compiling, and publishing. As a consequence the US Census (which is carried out every 10 years) would re-occur before the data from the previous census was available for the public and the published information was already significantly inaccurate on the date of release.

Due to the limits of existing technologies of the time, the components of the “Mainframe” was much larger than they are today and required vast spaces to store them. Hence one of the terms synonymous with mainframe computers is “Big Iron” for the literally BIG space the machines took up and the BIG amount of metal it required to put a machine together.

The most popular maker of mainframes is I.B.M. International Business Machines, a large Multi-national Corporation headquartered in the U.S.A.

1.1.4 Supercomputers

Supercomputers are known as a class of extremely powerful digital computers. The term is commonly applied to the fastest high-performance systems available at a given time; current personal computers are more powerful than the supercomputers of just a few years ago. Supercomputers are used primarily for scientific

and engineering work. Unlike conventional computers, they usually have more than one CPU, often functioning in parallel (simultaneously); even higher-performance supercomputers are now being developed through use of massively parallel processing, incorporating thousands of individual processors. Supercomputers have huge storage capacity and very fast input/output capability, and can operate in parallel on corresponding elements of arrays of numbers rather than on one pair of elements at a time Supercomputers have been developed from the processing requirements of advanced research projects by engineers, scientists and more recently by other research fields. The supercomputer is a title generally given to computers with processing power well exceeding those of the fastest mainframes and its focus is processing data and manipulation of that data. A significant difference between mainframes and supercomputers is the primary focus of super computers on processing capabilities independent of the many varied input/output devices expected on mainframes.

The major cause for the development of super-class computers has been the continuing need in the scientific community for faster and faster calculations. Scientists working on atomic physics, computer intensive calculations such as natural speech recognition continue to require more and more computational speed to test their theorems. An example of the use of supercomputers is the

South African Weather Services for national weather analysis and prediction. The Bureau collects weather information from around the world and makes instant analysis of this data. This information is critical for airlines who cannot afford to be late with their knowledge of what the weather conditions are over their flight paths. The incredibly large amount of data that needs to be collected and the requirement for the answers to become available immediately is why a supercomputer is used. Large oil companies such as Sasol use supercomputers to analyse their geological findings and weapons developers use supercomputers to test out new theories and strategies.
Again, the incredible amounts of data that geologists collect, and the complexity of matching data with what can be potentially oil or minerals requires very powerful computing facilities.

Car manufacturers simulate in three dimensions a car design and the number of probable accidents that can occur to manage the design safety requirements of vehicles without having to solder any metal or build an engine. Supercomputers have been required for this simulation process due to the complex calculations evaluating a simulated accident and the display in three dimensions. Most people do not currently need the ability to chart their own world weather map so the supercomputer is not for everyone. Supercomputers usually require specially built facilities and a large staff of computer technicians and computer engineers to maintain the hardware and software.

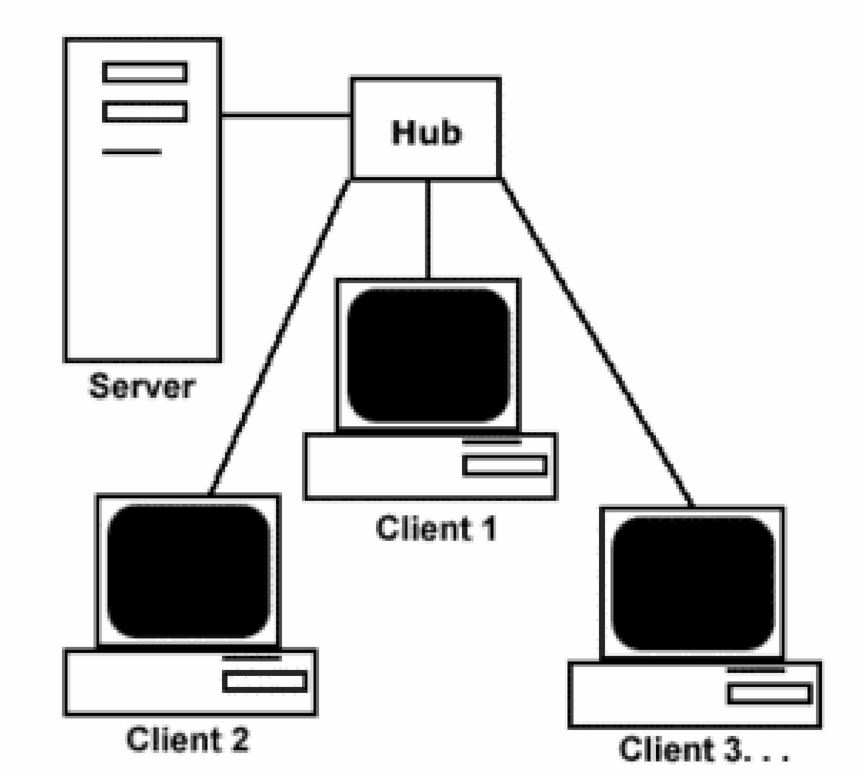
The most commonly known Supercomputer brand is the "Cray", from Cray Research the first company to develop a "Super" computer. The distinctive, good looks of the Cray Super Computer housed elaborate electronics in a look good box that was poured through with liquid coolant to keep the system cool while it works.

1.1.5 Client-server

In an environment with more than around 6-10 computers, a peer to peer network begins to become more trouble than it is worth. Your computers start to slow down. You can never find the file you are looking for, and security is non-existent.

If this is the case, companies switch to a client-server network by bringing in a dedicated server to handle the load. The server is "dedicated" because it is optimized to quickly serve requests from the clients (other computers on the network) and it can be well maintained.
It's the purpose of this unit standard to explain how the "Client/Server" architecture is really a fundamental enabling approach that provides the most flexible framework for using new technologies like the World Wide Web, as they come along. The old paradigm of host centric, time shared computing has given way to a new client/server approach, which is message based and modular. The examples below show how most new
Technologies can be viewed as simply different implementation strategies built on a client/server foundation.

Figure - The diagram below shows a simple client-server network:



What is a server? A server is a computer that is running software that enables it to serve specific requests from client computers For example, you can have a file server that acts as a central storage place for your network, a print server that routes print requests and status information between computers and printers connected on the network, as well as a multitude of other servers and server functions.

Even though most people use the term "client/server" when talking about group computing with PC's on networks, PC network computing evolved before the client/server model started gaining acceptance in the late 1980's. These first PC networks were based on the file sharing metaphor illustrated in the figure entitled FILE SERVER. In file sharing, the server simply downloads or transfers files from the shared location to your desktop where the logic and data for the job run in their entirety. This approach was popularized mostly by Xbase style products (dBASE, FoxPro and Clipper). File sharing is simple and works as long as shared usage is low, update contention is very low, and the volume of data to be transferred is low compared with LAN capacity.

As personal computer (PC) LAN computing moved into the 90's two megatrends provided the impetus for client/server computing. The first was that as first generation PC LAN applications and their users both grew, the capacity of file sharing was strained. Multi-user Xbase technology can provide satisfactory performance for a few up to maybe a dozen simultaneous users of a shared file, but it's very rare to find a successful implementation of this approach beyond that point. The second change was the emergence and then dominance of the GUI metaphor on the desktop. Very soon GUI presentation formats, led by Windows and Mac, and became mandatory for presenting information. The requirement for GUI displays meant that traditional mini or mainframe applications with their terminal displays soon looked hopelessly out of date.

The architecture and technology that evolved to answer this demand was client/server, in the guise of a two-tiered approach. By replacing the file server with a true database server, the network could respond to client requests with just the answer to a query against a relational DBMS (rather than the entire file). One benefit to this approach, then, is to
significantly reduce network traffic. Also, with a real DBMS, true multi-user updating is now easily available to users on the PC LAN. By now, the idea of using Windows or Mac style PC's to front end a shared database server is familiar and widely implemented.

A server provides many benefits including:

- Optimisation
- Centralisation
- Security
- Redundancy and Back-up
- Server Hardware
- Server Software

Optimisation

Server hardware (the physical, touchable, material parts of a computer or other system) is designed to quickly serve requests from clients.

Centralisation

Files are in one location for easy administration.

Security

Multiple levels of permissions - access privileges associated with a file or directory can prevent users from doing damage to files.

Redundancy and Back-up

Data can be stored in redundant ways (for example copied onto another hard disk on the server using special technology called RAID), or stored on external media such as tapes, so it can be restored quickly in case of problems. A server, like any computer, consists of two parts, the hardware and the software.

Server Hardware

Any normal desktop computer could act as a server, but typically you want something much more robust. Standard server hardware includes:

Hot swappable drives to speed up adding or replacing hard disks (used in RAID) - drives can be changed without having to shut down the server.

The ability to support multiple processors – in actual fact this is the brains behind a computer. Processors are responsible for performing calculations and tasks that make computer programs work. Multiple processors can be an advantage if you need to run applications that are processor intensive such as a very large database. Support for larger amounts of RAM - the more memory you have the faster your network can run. Faster input and output - information can travel around the network more quickly. Fast network cards.

Server Software

Server software comes in two categories, operating systems and applications:

Network Operating Systems. A network operating system is an operating system which includes software to communicate with other computers via a network. There are many different operating systems for servers just as there are many different operating systems for desktop computers. Windows Server 2003, Linux and Novell Netware are the three main network operating system competitors, but they are not the only ones.

A Network Operating System has many features built-in. All include file serving, print serving, backup and some way to secure those resources. Some Network Operating Systems also include a web server (to allow you to host your website or intranet yourself) or an email server (for email distribution around your network), while others require you to buy these items separately. Research all the options before making a decision on the Network Operating System for your server.

Figure out precisely what you want by browsing through websites and sales pamphlets. Then, try to find a Computer Guru who knows both your network and your organisation, and ask this person what they think would work best for your organisation.

If you can, try to make sure that this person is not the person who would be doing the work or selling you the product, otherwise they may have some conflicts of interest. This is a big decision, and it will dramatically affect all of your future computer relationships.

Server Applications

Server applications can be designed for nearly every purpose imaginable, from fax servers to remote access servers. Every application will have specific server requirements, and will be typically designed to run on either Windows Server 2003 or Netware and increasingly Linux. Many servers run multiple applications (e.g. email and faxing) to serve a variety of needs.

1.1.6 CPU and RAM

Since from the beginning of Electronic Numerical Integration and Calculator (ENIAC), new ideas revolving computer technology where implemented. Computer technology changes quickly, this statement does not seem to adequately describe what sometimes seems to be the breakneck pace of improvements in the heart of any electronic computing engine, the central processing unit (CPU). The transistor, invented at Bell Labs in 1947, is the fundamental electronic component of the CPU chip. Higher performance CPUs requires more logic circuitry, and this is reflected in steadily rising transistor densities. Simply put, the number of transistors in a CPU is a rough measure of its computational power which is usually measured in floating point mathematical operations per second (FLOPS). The more transistors there are in the CPU, or silicon engine, the more work it can do.

Trends in transistor density over time, reveal that density typically doubles approximately every year and a half according to a well know axiom known as Moore’s Law. This proposition, suggested by Intel co-founder Gordon Moore (Moore 1965), was part observation and part marketing prophesy. In 1965 Moore, then director of R&D at Fairchild Semiconductor, the first large-scale producer of commercial integrated circuits, wrote an internal paper in which he drew a line though five points representing the number of components per integrated circuit for minimum cost for the components developed between

1959 and 1964 (Source):
http://www.computerhistory.org/semiconductor/ timeline/1965-Moore.html, accessed 12 January 2008). The prediction arising from this observation became a self-fulfilling prophecy that emerged as one of the driving principals of the semiconductor industry. As is related to computer CPUs (one type of integrated circuit), Moore’s Law states that the number of transistors packed into a CPU doubles every 18–24 months.

In 1997, the Pentium II (2) had 7.5 million transistors, in 2000 the Pentium 4 had 420 million, and the trend continues so that in 2007, the Dual-Core Itanium 2 processor has 1.7 billion transistors.

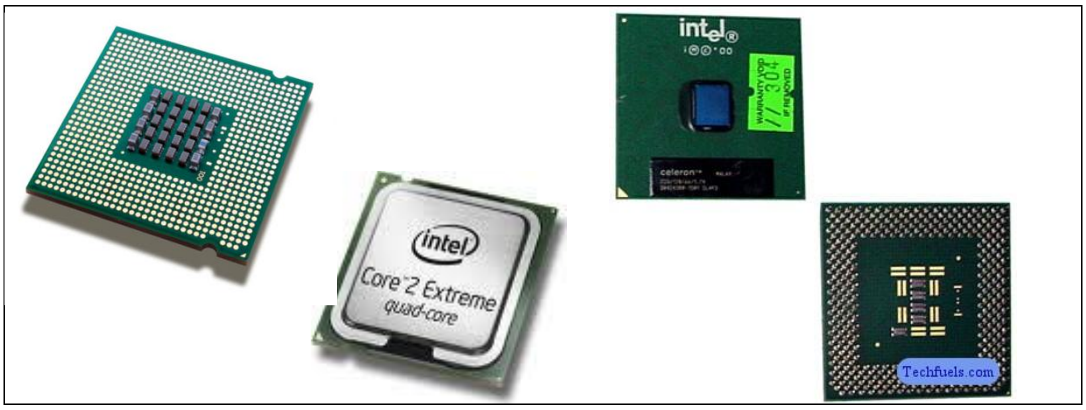
In addition to transistor density, data handling capabilities (i.e. progressing from manipulating 8, to 16, to 32, to 64 bits of information per instruction), ever increasing clock speeds (Fig. 1.2), and the number of instructions executed per second, continue to improve. The remarkable thing is that while the number of transistors per CPU has increased more than 1,000 times over the past 26 years, and another 1,000 times since 1996, performance (measured with millions of instructions per second, MIPS) has increased more than 10,000 times since the introduction of the

8088 (Source: http://www.jcmit.com/cpu-performance.htm, accessed 12 January 2008).

Scientific analysts, who use large databases, scientific visualization applications, statistics, and simulation modelling, need as many MIPS as they can get. The more powerful computing platforms described above will enable us to perform analyses that we could not perform earlier.

In the original edition we predicted that ‘‘Three years from now CPU’s will be four times faster than they are today and multi-processor designs should be commonplace.’’ This prediction has generally proven to be true.

Figure - CPUs



CPU performance has continued to increase according to Moore’s Law for the last 40 years, but this trend may not hold up in the near future. To achieve higher transistor densities requires the manufacturing technology (photolithography) to build the transistor in smaller and smaller physical spaces. The process architecture of CPUs in the early 1970s used a 10 micrometer (mm, 10 _6m) photolithography mask. The newest chips use a 45 nanometer (nm, 10_9m) mask. As a consequence of these advances, the cost per unit of performance as measured in gigaflops has dramatically declined

It appears that manufacturing technology seems to reach its limits in terms of how dense silicon chips can be manufactured – in other words, how many transistors can fit onto CPU chips and how fast their internal clocks can be run. As stated recently in the BBC News, ‘‘the industry now believes that we are approaching the limits of what classical technology – classical being as refined over the last 40 years – can do.’’ (Source: http://news.bbc.co.uk/2/hi/science/nature/4449711.stm, accessed 12 January 2008). There is a problem with making microprocessor circuitry smaller. Power leaks, the unwanted leakage of electricity or electrons between circuits packed ever closer together, take place. Overheating becomes a problem as processor architecture gets ever smaller and clock speeds increase.

Traditional processors have one processing engine on a chip. One method used to increase performance through higher transistor densities, without increasing clock speed, is to put more than one CPU on a chip and to allow them to independently operate on different tasks (called threads). These advanced chips are called multiple-core processors.

Difference between Dual-core processor and Quad-core processor

Dual-core processor	Quad-core processor
Squeezes two CPU engines onto a single chip. A dual-core processor theoretically doubles your computing power since a dual-core processor can handle two threads of data simultaneously. The Result is there is less waiting for tasks to complete.	Have four engines. Multiple-core chips are all 64-bit meaning that they can work through 64 bits of data per instruction. That is twice rate of the current standard 32-bit processor. A quad-core chip can handle four threads of data. Progress marches on. Intel announced in February 2007 that it had a Prototype CPU that contains 80 processor cores and is capable of 1 teraflop (1012 floating point operations per second) of processing capacity.

The potential uses of a desktop fingernail-sized 80-core chip with supercomputer-like performance will open unimaginable opportunities (Source: http://www.intel.com/ pressroom/archive/releases/20070204comp.htm, accessed 12 January 2008).

As if multiple core CPUs were not powerful enough, new products being
developed will feature ‘‘dynamically scalable’’ architecture, meaning that virtually every part of the processor – including cores, cache, threads, interfaces, and power – can be dynamically allocated based on performance, power and thermal requirements

(Source: http://www.hardwarecentral.com/hardwarecentral/reports/article.php/3668756, accessed 12 January 2008). Supercomputers may soon be the same size as a laptop if IBM brings to the market silicon nanophotonics. In this new technology, wires on a chip are replaced with pulses of light on tiny optical fibbers for quicker and more power-efficient data transfers between processor cores on a chip. This new technology is about 100 times faster, consumes one-tenth as much power, and generates less heat (Source: http://www.infoworld.com/article/07/12/06/IBM-researchers-build-supercomputeron-a-chip_1.html, accessed 12 January 2008).

Multi-core processors pack a lot of power. There is just one problem: most software programs are lagging behind hardware improvements. To get the most out of a 64-bit processor, you need an operating system and application programs that support it.

1.1.7 Emerging systems

The settings on a computer are called ***computer configuration***. Through different settings different things are achieved (i.e., configurations to connect to certain wireless networks; firewall configuration to deny all unsolicited network requests; graphics settings that enable high-performance graphics, etc.)
Currently there are 64-bit versions of Linux, Solaris, and Windows XP, Vista and Windows 7. However, 64-bit versions of most device drivers are not available, so for today's uses, a 64-bit operating system can become frustrating due to a lack of available drivers.

Another current developing trend is building high performance computing environments using computer clusters, which are groups of loosely coupled computers, typically connected together through fast local area networks. A cluster works together so that multiple processors can be used as though they are a single computer. Clusters are usually deployed to improve performance over that provided by a single computer, while typically being much less expensive than single computers of comparable speed or availability. Beowulf is a design for high-performance parallel computing clusters using inexpensive personal computer hardware. It was originally developed by NASA's Thomas Sterling and Donald Becker. The name comes from the main character in the Old English epic poem Beowulf.

A Beowulf cluster of workstations is a group of usually identical PC computers, Configured into a multi-computer architecture, running an Open Source Unix-like operating system, such as BSD. They are joined into a small network and have libraries and programs installed that allow processing to be shared among them. The server node controls the whole cluster and serves files to the client nodes. It is also the cluster's console and gateway to the outside world. Large Beowulf machines might have more than one server node, and possibly other nodes dedicated to particular tasks, for example consoles or monitoring stations. Nodes are configured and controlled by the server node and do only what they are told to do in a disk-less client configuration. There is no particular piece of software that defines a cluster as a Beowulf.

Commonly used parallel processing libraries include Message Passing Interface; and Parallel Virtual Machine. Both of these permit the programmer to divide a task among a group of networked computers, and recollect the results of processing. Software must be revised to take advantage of the cluster. Specifically, it must be capable of performing multiple independent parallel operations that can be distributed among the available processors. Microsoft also distributes a Windows Compute Cluster Server 2003 to facilitate building a high-performance computing resource based on Microsoft's Windows platforms.

One of the main differences between Beowulf and a cluster of workstations is that Beowulf behaves more like a single machine rather than many workstations.

In most cases client nodes do not have keyboards or monitors, and are accessed only via remote login or through remote terminals. Beowulf nodes can be thought of as a CPU + memory package which can be plugged into the cluster, just like a CPU or memory module can be plugged into a motherboard.
Beowulf systems are now deployed worldwide as they are simply to use. Typical Computer configurations consist of multiple machines built on AMD's Opteron 64-bit and/or Athlon X2 64-bit processors. Memory is the most readily accessible large-volume storage available to the CPU. We expect that standard RAM configurations will continue to increase as operating systems and application software become more full-featured and demanding of RAM.

For example	The "recommended" configuration for Windows Vista Home Premium Edition and Apple's new Leopard operating systems is 2 GB of RAM, 1 GB to hold the operating system leaving 1 GB for data and application code. Years (1999–2001) 64–256 megabytes (MB) of Dynamic RAM where available and machines with 64 MB of RAM will be typical.

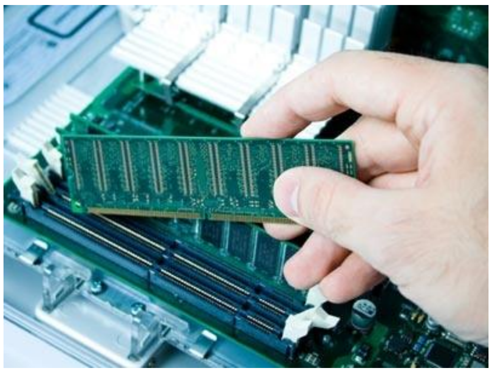
Over the years, advances in semiconductor fabrication technology have made gigabyte memory configurations not only a reality, but commonplace. Not all RAM performs equally. Newer types, called double data rate RAM (DDR) decrease the time it takes for the CPU to communicate with memory, thus speeding up computer execution. DDR comes in several flavours. DDR has been around since 2000 and is sometimes called DDR1. DDR2 was introduced

In 2003 It took a while forDDR2 to reach widespread use, but you can find it in most new computers today. DDR3 began appearing in mid-2007. RAM simply holds data for the processor.

✎	However, there is a cache between the processor and the RAM: the L2 cache. The processor sends data to this cache. When the cache overflows, data are sent to the RAM. The RAM sends data back to the L2 Cache when the processor needs it. DDR RAM transfers data twice per clock cycle. The clock rate, measured in cycles per second, or hertz, is the rate at which operations are performed.

DDR clock speeds range between 200 MHz (DDR-200) and 400 MHz (DDR-400). DDR-200 transfers 1,600 megabits per second (Mb s_1:106 bits s_1), while DDR-400 transfers 3,200 MB s_1. DDR2 RAM is twice as fast as DDR RAM. The bus carrying data to DDR2 memory is twice as fast. That means twice as much data are carried to the module for each clock cycle. DDR2 RAM also consumes less power than DDR RAM. DDR2 speeds range between 400 MHz (DDR2-400) and 800 MHz (DDR2-800). DDR2-400 transfers 3,200 MB s_1. DDR2-800 transfers 6,400 MB s_1. DDR3 RAM is twice as fast as DDR2 RAM, at least in theory. DDR3 RAM is more power efficient than DDR2RAM. DDR3 speeds range between 800MHz (DDR3-800) and 1,600 MHz (DDR3-1600). DDR3-800 transfers 6,400 MB s_1; DDR3-1600 transfers 12,800 MB s_1.

Figure - RAM



As processors increased in performance, the addressable memory space also increased as the chips evolved from 8-bit to 64-bit. Bytes of data readily 8 B.A. Megrey and E. Moksness accessible to the processor are identified by a memory address, which by convention starts at zero and ranges to the upper limit addressable by the processor. A 32-bit processor typically uses memory addresses that are 32 bits wide. The 32-bit wide address allows the processor to address 232 bytes (B) of memory, which is exactly 4,294,967,296 B, or 4 GB. Desktop machines with a gigabyte of memory are common, and boxes configured with 4 GB of physical memory are easily available. While 4 GB may seem like a lot of memory, many scientific databases have indices that are larger. A 64-bit wide address theoretically allows 18 million terabytes of addressable memory (1.8 1019 B). Realistically 64-bit systems will typically access approximately 64 GB of memory in the next 5 years.

1.1.8 Portable Computing or standalone

Another recent technology is the appearance of powerful portable computer systems. The first portable computer systems (i.e. ‘‘luggables’’)
Were large, heavy, and often portability came at a cost of reduced performance.
Current laptop, notebook, and subnotebook designs are often comparable to desktop systems in terms of their processing power, hard disk, and RAM storage and graphic display capabilities. Earlier in 1996 and years after it was not unusual, when attending a scientific or working group meeting, to see most participants arrive with their own portable computers loaded with data and scientific software applications. But today, it is unusual to see scientists attending technical meetings arrive without a portable computer like one of those below.

Figure - Portable Computers

Since 1996, the performance and cost gap between notebooks and desktops capable of performing scientific calculations has continued to narrow, so much so, that the unit growth rate of notebook computers is now faster than for desktops. With the performance gap between notebooks and desktop systems narrowing, commercial users and consumers alike are beginning to use the notebooks more and more as a desktop replacement since the distinction between the two as far as what work can be accomplished is becoming more and more blurred. Moreover, the emergence of notebook ‘‘docking stations’’ allows the opportunity to plug notebooks into workplace or institution’s network and internet cafe resources when one is in need to do so and then unplug the notebook at the end of the day to take it home or on the road, all the while maintaining one primary location for important data, software, working documents, literature references, email archives, and internet bookmarks. We have seen that miniaturization of large capacity hard disk storage, memory sticks, printers, and universal access to email made available via a cell phone and ubiquitous Internet connectivity (see below) all contribute to a portable computing environment, making the virtual office a reality.

Figure - Portable Technology

1.1.9 Hardware Advances

It is difficult not to appreciate at how quickly computer technology advances. The current typical desktop or laptop computer, compared to the original monochrome 8 KB random access memory (RAM), 4 MHz 8088 microcomputer or the original Apple II, has improved several orders of magnitude in many areas. The most notable of these hardware advances are processing capability or we can say the speed, colour graphics resolution and display technology, hard disk storage, and the amount of RAM. The most remarkable thing is that since 1982, the cost of a high-end microcomputer system was fairly demanding in rand's terms compared to now.

ACTIVITY 1 – REVIEWED - US 14921 SO1

Divide into groups. Explain what is meant by Computer Configuration. Complete this activity in your Portfolio of Evidence Workbooks.

1.2 CATEGORIES OF COMPUTER SYSTEM APPLICATIONS

Here we will have a look at the different categories of computer system applications. You have come to know computer system applications as software programs. These are the software packages we install on our systems in order to have certain functionalities that we require. The four categories are; Business, Scientific, Education and Home.

Remember, computer system applications are not the same as system software. System software includes operating systems, which govern computing resources. Today large applications running on remote machines such as Websites are considered to be system software, because the end-user interface is generally

You will also learn how the different types of applications perform as well as the types of processing methods they can use.

1.2.1 Application Categories

Business applications

Business applications are a broad category that includes most commercially available software packages available today. The most common applications are spread sheet (MS Excel), word processing (MS Word), database (MS Access), presentation (MS Power Point) and project management (MS Project). Some more examples are web browsers (Internet Explorer, Firefox), E-mail clients (MS Outlook, Thunderbird), computer aided design (Turbo CAD, Auto CAD), business management (SAP) and point of sales software (Quickbooks and Pastel).

Scientific applications

Scientific applications are applications used in the fields of medicine, scientific research, engineering and chemistry. These applications are highly specialised. They sometimes perform complex calculations and can therefore not be used on desktop computers. They are also difficult to operate and cannot be easily understood by anyone.

Engineers make use of sophisticated simulators to test in a virtual environment how their constructions will react under certain circumstances before they start building or constructing.

Scientists make use of equation editors and flowchart software. Examples of such applications are Microsoft Visio (Flowchart tool) and Microsoft Equation Editor.

Home applications

Most of the time, our computers at home and at work have much the same software setup. However, many companies do not allow their employees certain privileges such as music, videos, pictures, web browsing that is non-work related and gaming so we have these things on our computers at home. For music and movies we may have Windows Media Player, for pictures we may have ACDsee and for gaming there may be a variety of gaming application (games) that we may have installed at any one time. We may also have an office package installed with Word processing, spread sheets and so on.

Educational applications

Educational applications are software packages such as digital encyclopaedias, language learning tools, typing tutors and many more. Some encyclopaedic software may include Encarta, Britannica and Wikipedia which is an internet based encyclopaedia that is free to be accessed by anyone and contributed to by anyone.

1.2.2 Processing methods

Different processing methods allow users to obtain different levels of functionality from a computer. For instance, the sales person at the sales desk of a company needs to find a specific customer in order to create an invoice for some goods the customer wants to buy. The company have literally thousands of customers that all have accounts there. The sales clerk cannot possibly go through the whole list of customers in search one single person and what if there were two people with the same name and surname? So the company assign customer numbers to all their customers and the sales clerk only need to enter the customer number into a search field after which the computer does the searching and calls up the customer’s account in seconds.

Batch processing

Let’s say you are a professional photographer and you need to convert 800 images all from RAW format into JPG, resize them and add a border and a watermark. This will take you forever to do. You would have to go through every image several times. Some photo editing applications have a batch processing function that will allow you to do all of your changes on one image and the program will apply them to all of the images in the batch you specify.

In the mainframe world, batch processing can help user's process large amounts of data automatically. Running a batch job on a mainframe is far more effective than feeding it commands one by one. This way, the mainframe will know exactly what to do and complete the task all at once. Batch jobs can also be scheduled to run at certain times of the day so doing automating your workflow and saving you time.

Interactive processing

Interactive processing is where user input is required for an application to perform certain tasks. Compute games are a good example of this as you need to give the game input for anything to happen. Another such example may be point of sales systems. The POS system awaits input from the clerk. Let's say scanning of a barcode. Once the system received a barcode it immediately checks for the product that was scanned and presents the clerk with all the relevant information regarding the product such as description, price, stock levels etc. Thus interactive processing requires input in order to generate output for the user.

Real-time processing

Interactive processing is also real-time processing because your input immediately results in an output. If you are in your car and you step on the brake or accelerator you will immediately either speed up or slow down. This is a real-time reaction. Similarly, if you are chatting to your friends on MXiT, they immediately see the text you entered. You also immediately see the text they entered. Thus the conversation is happening in real-time.

Process control

“Process control is a statistics and engineering discipline that deals with architectures, mechanisms, and algorithms for controlling the output of a specific process. See also control theory.” - Wikipedia

Accelerating your car is a process that has a specific and desired outcome to reach and maintain a specific speed in accordance to the national traffic law e.g. 60KM/h and is kept constant for as long as needed. The speed is referred to here as the controlled variable but it is at the same time the input variable that you read from your speedometer in order to decide whether to accelerate or decelerate. The desired speed is the set point. The state of the accelerator (pressed in or released) is called the manipulated variable since it is subject to control actions

“A commonly used control device called a programmable logic controller, or a PLC, is used to read a set of digital and analog inputs, apply a set of logic statements, and generate a set of analog and digital outputs. Using the example in the previous paragraph, the room temperature would be an input to the PLC. The logical statements would compare the set point to the input temperature and determine whether more or less heating was necessary to keep the temperature constant. A PLC output would then either open or close the hot water valve, an incremental amount, depending on whether more or less hot water was needed. Larger more complex systems can be controlled by a Distributed Control System (DCS) or SCADA system.” – Wikipedia.

INFORMAL ACTIVITY – VIEWED – US 14921 SO1

Discuss the question: Where did computers come from, and where is it going?

Background information

<u>History in the Making</u>

Prior to electronic computers, mechanical devices used for calculations tabulations used mechanical gearing to tabulate occurrences of events and were problematic due to the wear (malfunctions) that occur with moving pieces of machinery and the problems of creating accurate finishes on the mechanical devices to fit the designs.

The First Electric Calculating Machines – Herman Hollerith

To solve the problem, Herman Hollerith invented a calculating, tabulating machine that used electricity rather than mechanical gears. Holes representing information to be tabulated were punched in cards with the location of each hole representing a specific piece of information (male, female, age, etc.) The cards were then inserted into the machine and metal pins used to open and close electrical circuits. Hollerith's machine was immensely successful and based on its success and together with some friends they formed a company that eventually became known as

International Business Machines (IBM). The first *computer-like* machine is generally thought to be the Mark I, which was built by a team from IBM and Harvard University. The Mark I used mechanical telephone relay switches to store information and accepted data on punched cards, processed it and then output the new data. Because it could not make decisions about the data it processed, the Mark I was not, however, a real computer but was instead a highly sophisticated calculator. It was, nevertheless, impressive and has now been dismantled (decommissioned) with parts of it on display at the Undergraduate Science building at Harvard University.

The First Electronic Computer – Eniac.

In 1943 work began on the *Electronic Numerical Integration and Calculator*, or ENIAC. It was originally a secret military project which was to be used to calculate the trajectory of artillery shells. In one of its first demonstrations it was given a problem that would have taken a team of mathematicians three days to solve. It solved the problem in twenty seconds.

ENIAC was different from the Mark I in several important ways. First, it occupied 63m^2 and it weighed 30 tons. Second, it used 17,000 vacuum tubes instead of relay switches. To be the first true computer. ENIAC had two major shortcomings. First, it was difficult to change its instructions to allow the computer to solve different problems. It had originally been designed only to compute artillery trajectory tables, but when it needed to work on another problem it could take up to three days of wire pulling, re-plugging and switch flipping to change instructions. Second, because the tubes it contained were constantly burning out, the ENIAC was unreliable.

The mainframe grew out of vacuum tubes and as technology improved the mainframe became smaller and less expensive. Correspondingly, the Organisations who could afford the original expensive machines had more money to spend on more features, capabilities so although the mainframe decreased significantly in size, it still remained a
large creature using up a lot of electricity and space.

The Stored Program Computer – John von Neumann

In the late 1940's, John von Neumann considered the idea of storing computer instructions and data in memory, which was accessed by a central processing unit, or CPU.

The CPU would control all the functions of the computer electronically so that it would not be necessary to flip switches or pull wires to change the instructions. Now it would be possible to solve many different problems by simply typing in new instructions at the keyboard. Together with other computer scientists, von Neumann designed and built the EDVAC (Electronic Discrete Variable Automatic Computer) and the EDSAC (Electronic Discrete Storage Automatic Computer).

With the development of the concept of stored instructions or “programs”, the modern computer age was ready to begin. Since then the development of new computers has progressed rapidly, but von Newman’s concept has remained, for most part, unchanged.
The next computer to employ von Neumann’s concepts was the Universal Automatic Computer, called UNIVAC, developed in 1951.

Computers at this time continued to use many vacuum tubes which made them large and expensive. UNIVAC weighed 35 tons. These computers were so expensive to purchase and run that only the largest corporations and the US government could afford them.

Their ability to perform up to 1000 calculations per second, however, made them popular.

The Transistor – BELL Laboratories.

It was BELL Laboratories’ invention of the transistor that made smaller and less expensive computers possible, with increased calculating speeds of up to 10,000 calculations per second. Although the size of the computers shrank, they were still large and

	expensive. In 1963, IBM, using ideas it had learned while working on projects for the military, introduced the first medium-sized computer named the “model 650.” It was still expensive, but it was capable of handling the flood of paperwork produced by the many government agencies and businesses. These new computers also saw a change in the way data was stored. Punched cards were replaced by magnetic tape and high speed reel-to-reel tape machines. Using magnetic tape gave computers the ability to read (access) and write (store) data quickly and more reliably. Another important advance occurring at the time was the development of programming languages. Previously, computers had to be programmed by setting different switches to their On or Off positions. The first programming languages were very similar, being strings of 1’s and 0’s representing the status of the switches (1 for On, and 0 for Off). These were called “low-level” languages. Languages such as FORTRAN (Formula Translator), which was the first popular “high-level” language, allowed programmers to use English-like instructions such as READ and WRITE. With them, it was possible to type instructions directly into the computer or on punched cards, eliminating the time consuming task of re-writing.

A number of high-level languages have been developed since that time including COBOL (Common Business Oriented Language), BASIC (Beginner's All-purpose Symbolic Instruction Code), Ada, C, and Pascal. Cobol was commissioned by the US Department of Defence in 1959 to provide a common language for use on all computers and the development committee was chaired by Commodore Grace Murray Hopper of the US Navy.

The Integrated Circuits and the Microprocessor

The next major technological advancement was the replacement of the transistor by tiny integrated circuits or "chips." Chips are blocks of silicon with logic circuits etched onto their surface. They are smaller and cheaper than transistors and can contain thousands of circuits on a single chip. Integrated circuits also give computers tremendous speed allowing them to process information at a rate of 1,000,000 calculations per second. One of the most important benefits of using integrated circuits is to *decrease the cost* and size of computers. The IBM System 360 was one of the first computers to use integrated circuits and was so popular with businesses that IBM had difficulty keeping up with the demand. Computers had come down in size and price to such a point that smaller organisations such as universities and hospitals could now afford them.

A very important advance to occur in the early 70's was the invention of the microprocessor, an entire CPU on a single chip. In 1970, Marcian Hoff, an engineer at Intel Corporation, designed the first of these chips. As a result, in 1975 the ALTAIR microcomputer was born which led to the creation of small software companies such as Microsoft and in 1977 Stephen Wozniak and Steven Jobs designed and built the first mass market microcomputer, the Apple. Microcomputers were inexpensive and engineers, hobbyists were able to take their computers home.

The computer revolution had finally come home for many.

ACTIVITY 2 – REVIEWED – US 14921 SO1,2

Complete this activity in your Portfolio of Evidence Workbooks.

2

LEARNING UNIT 2: COMPUTER ARCHITECTURE

Learning outcomes to be achieved

- Explain computer architecture elements.
- Explain the organisation of a computer.
- Describe the design constraints in the design of instruction sets for computers.

2. INTRODUCTION

In our modern time of living, we rely on computer systems and networks all the time, whether you like or know about it or not. From a simple device such as a calculator, TV remote control, the TV itself and an electronic parking meter, they are all computerised to a certain extent. They help automate our sophisticated world to help free up tedious man hours and save companies money. This is both a good and a bad thing as people may lose their jobs because an automated system is taking over. Then again, it may also create new jobs as specialists are required to operate the new automated machinery.

Automation is not the only application for computer systems. We use them to store all our documents and customer data, project data and any other forms of record you can think of. They help us to manage huge quantities of data, sort information, solve crimes and do our general day to day business.

Computer systems consist of Hardware, Software and Networks. Put together they are referred to as information technology or IT. This learning unit will focus on the explanation of those concepts and where they fit into day-to-day life and business infrastructures.

2.1 COMPUTER ARCHITECTURE ELEMENTS

2.1.1 Hardware

Hardware is the physical components of a computer. That means we can see, touch and hear them. Hardware provides a platform for software to work. These are components such as the mouse, keyboard, processor (CPU), monitor and so on. They are there for us to interact with the computer or in the case of components such as the processor and motherboard, they are there to manipulate data and perform specific functions to our liking. Devices that stand separate from a computer but are connected to it by means of cables are called peripheral devices. These are devices such as printers, keyboards, display units, etc.

Hardware is mechanical in nature. This means that a hardware device can only perform the task it was designed for. A display unit cannot be used as a webcam (unless it has one built in) and neither can a mouse be used to print documents.

2.1.2 Software

Software is intangible (untouchable) and refers to the collection of programs such as office packages, operating systems, drawing tools etc. that we use to perform certain tasks. We refer to computer programs as software because even though the disks they come on are tangible (hardware) we still cannot touch the software code itself. We need the hardware to interpret the code and also to manipulate it and its operation or to create digital content by use of the programs. Even saved files like spread sheets can be referred to as software or programs because they contain data manipulating functions such as formulae.

Software comes in various forms. They are user installable programs and firmware.

Software – System

System software is transparent, low level software that provides interface with the computer’s hardware. All software communicates with hardware but not as directly as system software. You will learn about the Kernel later.

An everyday example of system software is an operating system (OS). A computer cannot function without an operating system One can make use of operating systems such as Microsoft’s Windows, One of the Linux flavours or if you are using an Apple Macintosh, Mac OS.

Software - Application

After you have installed your operating system, your system may not be very useful. You may have the need to do word processing, create spread sheets, make videos or play games. Application software is the type of software that provides you the functionality you require.

2.1.3 Firmware

Firmware controls hardware devices. Some hardware devices are designed to perform complex tasks such as play back audio or video. In order for these devices to perform their tasks they need what is called firmware. The device itself may be able to output audio and video because it has all the right circuitry to do that. Firmware provides an interface to the hardware and allows you to browse and select the items you would like to play and sends the output information to the relevant hardware components.

Applications of firmware:

- Computer BIOS (basic input output system)
- Mobile device software (e.g. cell phones, MP3 players, GPS, etc.)
- Vehicular control systems (A car's engine control unit / ECU)
- Game consoles
- Television sets and satellite decoders
- Computer display units

2.1.4 Virtual Machines

Machine virtualisation is a revolutionary technology which saves companies tonnes of money on hardware, in turn saving a lot of energy and time consumed through management.

Virtual machines allow for multiple virtual computers to be operated from one single physical host. It allows for multiple virtual guests on a single physical host without affecting the performance of the physical host too much. A virtual machine behaves exactly like a physical machine would. Each virtual machine is assigned an amount of memory that it is allowed to use from the physical host as well as a fixed amount of disk space that is pre allocated. Memory usage is dynamic and virtual machine operational data is cached to the hard drive of the host machine when the virtual machine is idle for long enough. This frees up resources for other virtual machines to use.

There are two types of virtual machine. They are a) System virtual machines and b) Process virtual machines:

a) System virtual machines

The most common function of system virtual machines is server virtualisation. This application saves companies lots of money by doing away with bulky hardware. Instead of having one physical machine for every server application, one can now have several virtual machines on one physical machine. If you would’ve had (for example) seven physical machines, you will now only have one with seven virtual machines. This saves power, cooling costs and hardware and maintenance cost. Companies can now spend more money on one good machine rather than multiple machines that all require upgrades too.

In short, the advantages are:

- System virtual machines requires less hardware
- They cost less to manage
- Can easily be deployed on another machine should its host fail

- Virtual machines are scalable
- They are friendlier toward the environment because,
- They consume less energy
- Virtual machines consume less space

b) Process virtual machines

These are also sometime referred to as application virtual machines. They are used to create a platform independent programming environment. This allows an application to start and stop in exactly the same way irrespective of the operating system. Java is a very good example of application virtual machines. You will have the same code for an application and only your run-time environment will be different. This means that you will have a different version of the Java Runtime Environment (JRE) installed depending on the operating system you are working on. This means that an application you wrote in Java will be able to run on virtually any operating system and even hardware platform (e.g. cell phones, PDAs etc.) because it runs in an application virtual machine.

2.1.5 Functional levels within a computer

Functional levels are also referred to as abstraction levels or layers. See them as the organisation of different levels of operation like that of a business or a university as in the example taken from Wikipedia.

Figure: Example of abstraction (Functional levels)

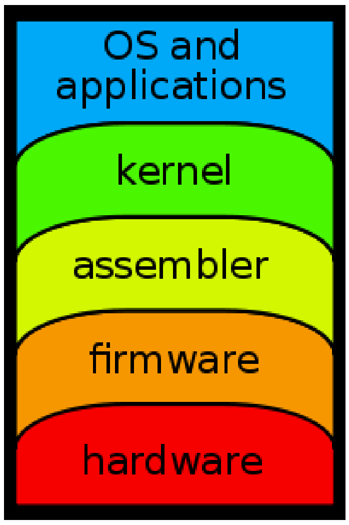
Extract from Wikipedia – Principle of abstraction

University

- Faculty of Science
  - Department of Physics
    - Subject - Physics 101
      - Topic - Fluid dynamics
  - Department of Earth Sciences
  - Department of Biology
- Faculty of Arts
  - Department of History
    - Australian History
      - 1850-1854 Victorian Gold rush
  - Department of Philosophy

This example may seem somewhat out of context but it's a good way of understanding how abstraction works. Similarly, a computer also has abstraction levels or to get back into context, functional levels. There are 5 main functional levels demonstrated by this illustration.

Figure: Functional Levels



The figure shows the abstraction levels from the least abstract (bottom) to the most abstract (top). We say that hardware is the least abstract (or concrete) because it is tangible. We can touch hardware and we know for a fact that it is there. Software on the other hand is the most abstract element of computer architecture.

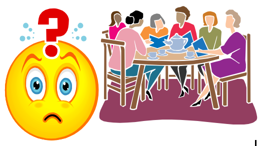
Because hardware and software are so different in their levels of abstraction, they need a little help to communicate or to work together and therefore we have the Kernel, the assembler and firmware. The firmware is part of the hardware helps the hardware communicate with the assembler and vice versa.

Assembler language is easy for hardware to understand and therefore it is necessary for an assembler to translate user written programs into assembly language. The type of assembly language used depends on the type of architecture used by the system.

The Kernel is considered the common component of an operating system because all programs need the kernel in order to work. The kernel controls hardware resources as well as provide an inter process communication facility or IPC. This allows programs to communicate with both the hardware and other programs or processes.

ACTIVITY 3 – REVIEWED – US14917 SO2

Complete this activity in your Portfolio of Evidence Workbooks.

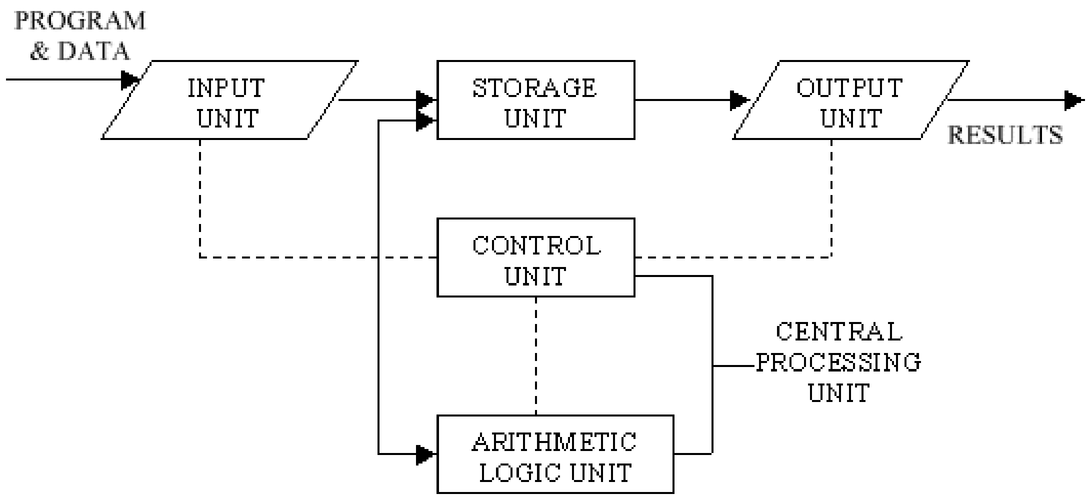


2.2 THE ORGANISATION OF A COMPUTER

Computers don’t just work. They need to be highly organised in order to function correctly. Computers are very complex machines with lots of things going on and huge amounts of technology incorporated into very small spaces. For instance a typical processor such as an AMD Athlon dual core processor can have as many as 233 million silicon transistors. There’s also a specific way for information to flow through the hardware.

At input you could have a keyboard or CD drive. From there the info can either be stored directly onto the hard drive or sent to the Central Processing Unit that consists of the control unit and arithmetic logic unit (ALU) for processing before it is either stored again to the hard drive or sent for output to one of the peripheral devices such as the display unit or printer. The diagram below demonstrates the process.

Figure: Data flow process



The most common hardware devices you will encounter are discussed below.

2.2.1 Computer box

The computer box can be referred to as the skin of the computer. It holds all of the computer’s essential components such as the motherboard, processor, storage devices, power supply unit (PSU), etc.

2.2.2 Motherboard (Houses the Central Processing Unit)

Whenever you have a number of components that have to work together, you need something to keep them together and help them communicate. A motherboard does just this. It is the component that connects everything together. Processor, memory, hard drives, display cards, sound cards, network adapter, keyboard, mouse and all other protocol adapters plug in to the motherboard. Nowadays, a lot of these components are actually built into the motherboard. It also incorporates what is known as the north and south bridges. The north bridge is also known as the Memory Controller Hub (MCH) or Integrated Memory Controller. The collective of the north and south bridge is referred to as the chipset. The chipset also performs the duty of the system bus which you will learn about later in this unit.

2.2.3 Central Processing Unit (CPU)

In order to work at all, a computer needs a CPU (also referred to as a processor). This is the component responsible for taking decisions, manipulating data and creating output for us to use. A processor chip contains a processor core that consists of millions of silicon transistors. It is an extremely complex device beyond the comprehension of a normal human being. Processors are one of the main deciding factors for computer speed and are normally the first component to be replaced if more speed is required. They are normally rated in MHz (megahertz) and nowadays GHz (gigahertz). There are also other measuring units such as MIPS (Million instructions per second), Cycles per second and FLOPS (Floating point Operations per Second). These are normally used to rate the performance of mainframe processors but can be used for other processors as well.

2.2.4 Memory – RAM (Storage Unit)

RAM or random access memory are there for the processor to temporarily store data while it is processing instructions. RAM is the first line of storage and is referred to as primary storage. For instance while converting or compiling video files, the computer stores frame by frame information in the memory while it is being converted. Only after a frame is converted will the conversion software instruct the processor to write the information to the hard drive (secondary storage). Once a program or process is terminated the memory it used is freed up for other processes to use. This is why we are always asked whether we would like to save our documents or any changes we’ve made to them. If we don’t save them the information will be removed from memory never to be seen again. RAM is primary memory since all information the computer works with is located in the RAM. Only non-essential information or information that won’t be needed for some time is stored on the hard drive.

There are different types of memory which is beyond the scope of this course for discussion. I will mention the names so that you can at least recognise it when you have to do with it.
They are:

- DRAM (Dynamic RAM)
- SDRAM (Synchronous Dynamic RAM)
- DDR (Double Data Rate Synchronous Dynamic RAM)
- RDRAM (Rambus Dynamic RAM)

2.2.5 Memory (ROM)

ROM or Read Only Memory is memory that can only be read but not changed. The contents are fixed. The most common form of ROM is CDs and DVDs with the latest addition being Blu-Ray. These are disks you can write to and maybe add info to it later but it cannot be erased with the exception of rewritable disks (CD-RW/DVD-RW).They have a dissolvable chemical film inside that can be smoothed out in order to erase the data so doing creating a once again clean writable disk. This cannot be done on the fly though. The disks need to be erased and formatted first.

2.2.6 BIOS and Battery

Another occurrence of ROM is computer BIOS. The first motherboards' BIOS chips were ROM. It was possible to change the settings but it was necessary to have an on-board battery in order to power the BIOS while the computer was switched off to prevent the settings from being lost. Then a clever guy invented flash memory. This meant that BIOS settings could be stored without the need for a battery. Nowadays the battery you see on a motherboard is merely to keep the system clock running. Making sure that the time is always correct.

2.2.7 Hard disk

Hard disks are an essential component. Before they existed people had loads of tapes and floppy diskettes to make their computers work. In the era of the ZX spectrums one would have a small device with only a keyboard built into it and a tape recorder connected via an audio cable to the ZX spectrum. Then one would have to play back the data stored on an audio cassette. It sounded much like a fax machine or dialup modem. This would then load the program into the 64KB of memory the machine had and display it on your television set.

Thanks to the hard disk, we can now easily access all of our applications from one place in an instant. And what's more, you can store your documents and other information on it too.

Hard disks are sensitive devices. A fall or bump can easily damage the device. They have another limitation. Due to its mechanical nature, the laws of physics are its greatest enemy. Its data seek times are greatly affected because of inertia. Spindle speeds affect the performance of a hard drive since the faster they spin, the more durable the material it is made of need to be. Nevertheless, the technology came a long way and today we can see speeds of up to 300MB/s for normal desktop computer hard disks and spindle speeds of up to 15 000RPM.

We use hard disk drives as a permanent mode of storage. We store files that we will need in the long term. The computer itself uses the hard drive for this very same purpose. Any information that resides in the RAM that the computer won’t need soon is written to the hard drive and then read only when the computer really needs the information. Any information in the RAM is lost as soon as power to the system is removed. We can thus say the hard drive is a long term storage device.

There are different hard drive technologies. We won’t be going into much detail now but you should at least know about them. The types are classified by type of drive and then the type of interface.

Types of drive:

- Mechanical
- Solid State

Mechanical hard drives have motors that drive spindles or platters on which data is magnetically stored, quite similar to floppy diskettes and tapes. Every hard drive's worst nightmare is exposure to magnets or being dropped or bumped too hard. Newer mechanical drives can resist up to 320 G-Forces when parked (when the needle is put away safely). Some vendors like Seagate come up with ingenious ways of protecting hard drives but truth be told, mechanics always fail sooner or later.

Solid state drives have no moving parts thus promises to be less fragile than its counterpart. It is designed to mimic normal hard drives using SRAM or DRAM rather than flash memory. These drives are still very expensive but will definitely become more affordable as its popularity increase over the next few years.

Types of drive interfaces:

- **IDE / Parallel ATA** - ATA is acronym for Advanced Technology Attachment, Parallel refers to the way devices are connected.
- **SCSI** - Pronounced as Scuzzy and used in servers for its improved performance. SCSI is an acronym for Small Computer Systems Interface
- **S-ATA (Serial ATA)** - Replaces IDE and has improved transfer rates of up to 300MB/s. Serial refers to the way devices are connected.
- **SAS Serial Attached SCSI** – SAS is the Serial counterpart of normal SCSI. It makes use of the S-ATA principles to improve performance.
- **Fibre channel** – Used in data centres for its high data transfer rates. You won't see these drives inside a compute

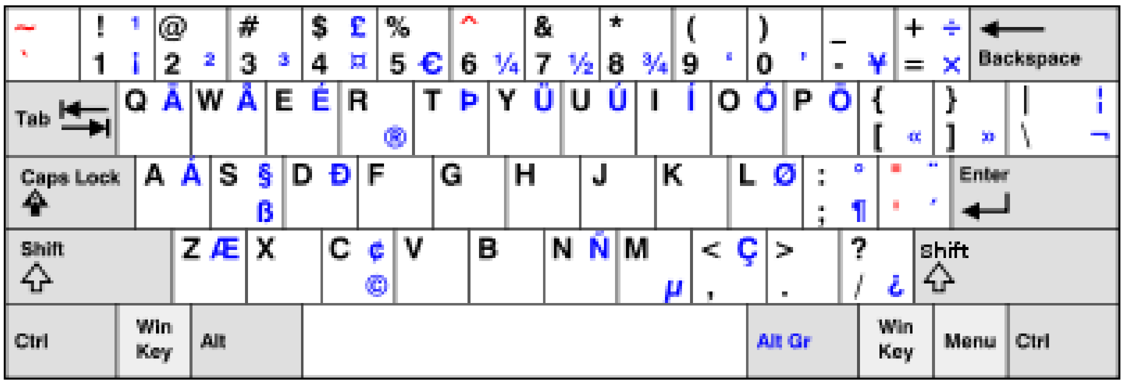
or server but in storage racks and they are remotely accessed via the FCP protocol (Fibre Channel Protocol)

2.2.8 Keyboard

Most computers have one of these. It is one of the primary methods of inputting information into a computer. It's also one of the primary ways of interacting with your computer and making it do things.

Keyboards have different key layouts depending on the region of the world they are used in. In South Africa, we mostly use U.S. International keyboards. These keyboards have what we call a QWERTY (pronounced kwertie) layout. This refers to the first 5 alphabetic keys found in the top left corner.

Figure: QWERTY Keyboard



2.2.9 Mouse

Another way of interacting with a computer is by means of a mouse. There are several types of mice or pointing devices. The first of these were the mechanical mice that had a ball inside which role over two wheels connected to motion sensors. This mouse's are now nearly obsolete.

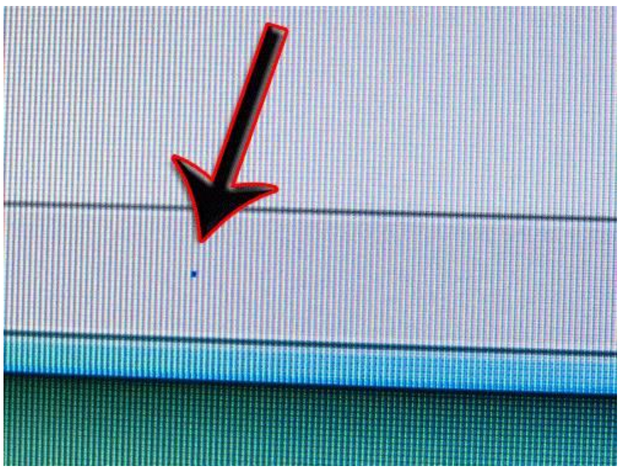
Figure: Trackball Mouse

The second is the optical or laser mouse. Optical mice have any of several types of lights. They are either a red, blue or infra-red laser. There's a photo sensor that takes hundreds of pictures every second using the light from these lasers. The computer then compares every picture and moves the mouse pointer according to the displacement in the imagery. Lastly, one of the other common devices is called a trackball and they are normally used for graphics designs. A trackball is almost the upside down of a mechanical mouse. The figure shows one such mouse.

2.2.10 Monitor / Display Unit

There are two types of display units. The first is a cathode ray tube display or CRT for short (also known as rear projection display). Though it's quite interesting to know, we are not going into the detail of how these units work. For many years they have been the main display type but now there are two new successors, LCD and LED with LCD being the older of the two technologies. Once again

Figure: A dead pixel



we won’t be going into the detail of the working of these units. The basic benefits of LCD and LED are clarity and contrast ratios also being the inspiration for HD technology. They use far less power than CRT and may even exceed the expected lifetime. They are also radiation free and LED displays are environmentally friendly because they use no harmful chemicals to achieve their display, unlike LCD.

LCD is an acronym for Liquid Crystal Display and LED an acronym for Light Emitting Diode.

They are extremely complicated to manufacture and therefore one can sometimes expect to find dead pixels on a brand new screen.

2.2.11 Power Supply Unit (PSU)

An electronic device cannot operate without a proper power input or supply of power. The power supply unit or PSU is responsible for this. They come in several power configurations and the type you buy may depend on the type of hardware you are using. High end PCs may sometimes have PSUs as strong as 750Watt or more. Not everyone needs that much power, though you cannot determine the type of PSU required by the type of workload you plan on placing your computer under. You’d much rather consider the type of Processor, Motherboard and Graphics adapter because those are the components that consume power. Also keep in mind the amount of hard drives you will be using.

2.2.12 PC Cards

Also known as PCMCIA cards, PC cards were originally designed for laptop computer storage expansion but soon after its invention all kinds of devices such as modems and network cards was available in this form factor. Some early digital cameras also made use of these devices for storage. PCMCIA is an acronym for Personal Computer Memory Card International Association and was to provide a competing standard for the Japanese JEIDA memory card. Later the faster express card slot was introduced and provided better performance. PCMCIA dissolved because of this therefore no modern computer makes use of the original PCMCIA slots.

Today common uses for the express card are 3G cards, network adapters (wired and wireless), USB controllers and many more.

2.2.13 Cables

There are so many different types of cables used in computers. Luckily someone had the bright idea of giving them uniquely shaped plugs so we cannot get all confused when setting up our hardware.

Figure: IDE Strap

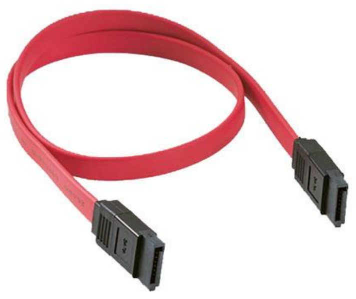
IDE (P-ATA)

You need cables to connect your hard drives to the motherboard. There are a few types of cable but your PC will only use either an IDE or S-ATA cable. An IDE drive (really known as P-ATA or parallel ATA) uses a flat, belt like cable with 40 connectors on each end of the cable.

S-ATA

The faster replacement for IDE is S-ATA and they use a much more compact cable to carry data from the drive to the motherboard. It has two small flat plugs on each end.

Figure: S-ATA Cable



USB 1.0, 2.0 and 3.0

USB or Universal Serial Bus is really what the name says it is, universal. Whe connect our Keyboards, Mice, Printers, Cameras, Scanners, Cellphones and who knows what else via this protocol. USB have come a long way sinve its insvention.

Figure: USB 1.1, 2.0 and 3.0 cables

USB1.1	USB2.0	USB3.0

VGA Cables

Computer displays have come a long way since the monochrome display and so has the cables they use. The most common now is the 15-pin S-VGA cable though there are two new standards, DVI and HDMI.

S-VGA (Super Video Graphics Array) is more of an analogue interface used for CRT displays and for some LCDs though hampering the quality of the display. Therefore the DVI interface was designed to provide a better visual experience. For high definition technology there's the HDMI interface which does pretty much what DVI does except for incorporating audio into the interface providing an all-round mind numbing audio visual experience.

Figure: Display cables

S-VGA	DVI	HDMI

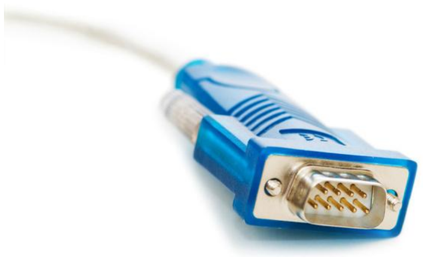
PS2

Prior to the invention of USB we used to have PS2 ports into which keyboards and mice plugged in to. They are normally coloured green for mice and purple for keyboards.

Figure: PS2 Cable

COM / Serial

Figure: Serial Cable

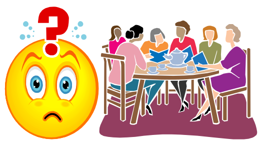


Before PS2 we connected our mice via COM or Communications ports better known as serial ports. You will still find these ports on most computers today since they are still used for dialup modems, shell access to servers or hardware routers and switches.

LPT

Figure: LPT Cable

LPT, an abbreviation for Line Print Terminal and also known as a Parallel port, are commonly used for printing.



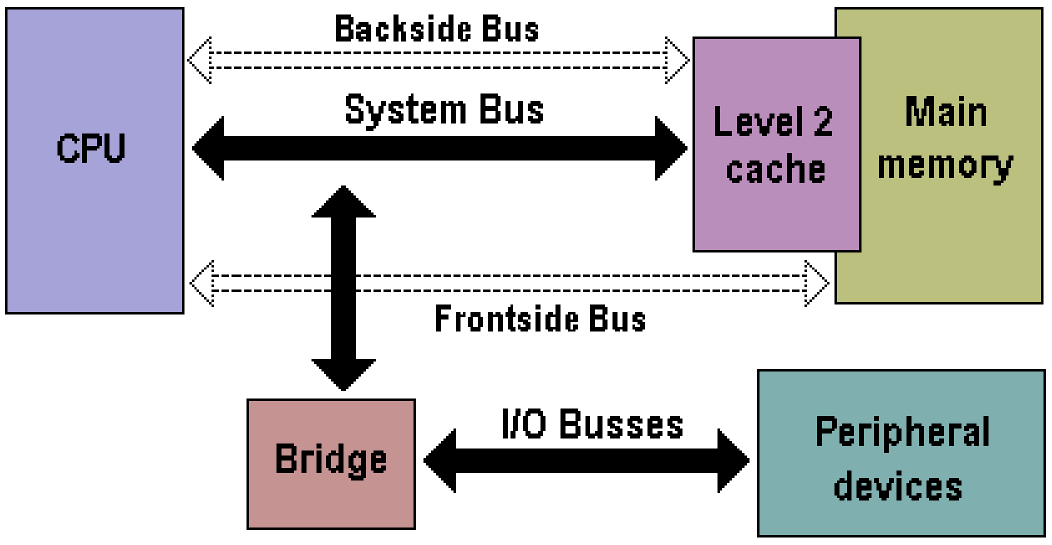
ACTIVITY 4 – REVIEWED – US14917 SO2

Complete this activity in your Portfolio of Evidence Workbooks.

2.2.14 How hardware components relate to each other

The figure below shows the basic interconnectivity of devices in a computer by means of busses.

Figure: Bus Diagram



In the early days, processors were responsible for controlling devices and reading and writing to and from them. This took up precious CPU time. Another problem was that all devices were required to talk at the same speed which in turn affected to growth of CPU speed. This was a huge limitation. Another limitation was that devices were not able to communicate with each other directly. All communications had to be handled by the processor itself. The introduction of busses meant that there was now a “post office” to which devices could send their data. Devices could then send data to other devices on either the same or other busses without the necessity of the processor.

Busses lead to the increase of performance for both processors and memory because they were now isolated from other devices. There was only one similarity between non-bus and bus systems. All of the devices on a bus had to talk at the same speed. This was a problem for high performance devices and lead to motherboard manufacturers incorporating AGP (accelerated graphics port) ports on their boards. This was the case for many other bus types such as P-ATA (Parallel Advanced Technology Attachment) which was replaced by S-ATA (Serial Advanced Technology Attachment). AGP was replaced again in 2004 by PCI-Express.

Devices on the same bus would communicate with each other via that bus. Though when they need to communicate with other devices on other busses they would need to make use of the bridge which would send the communications to the relevant busses e.g. from the USB to the CPU on the System bus.

ACTIVITY 5 – REVIEWED – US14917 SO2

Complete this activity in your Portfolio of Evidence Workbooks.

2.3 DESIGN CONSTRAINTS IN INSTRUCTION SETS

A computer needs a heart just like a human does. The only difference is that these hearts need to do more than just shove blood around the system. In order for a processor to do its job right, software engineers need to design an instruction set architecture and they do so by use of Register Transfer Language or RTL for short. RTL is a programming language used to tell the processor how every operation will work for every instruction in the instruction set architecture. In short, it describes what the processor is capable of doing. By establishing the capability of the processor one also establishes its limitations or the constraints.

A constraint is something that keeps something else from happening or puts a limit on how that something happens. A design constraint of an instruction set determines how instructions in the particular set are constrained.

The six typed of design constraints are:

- Instruction length
- Memory transfer
- Instruction format
- Operand specification
- Instruction fetch
- Word length

2.3.1 Design constraints in detail

Instruction length

Depending on design, processors can only handle certain lengths of instructions. This is obviously determined by the system architects. If an instruction is longer or more complex than what the processor supports, it won't be able to execute it. Instruction length is determined by the amount of operands in the instruction. The longer the instruction the more complex it is. For instance, an instruction with 3 operands will be more complex than an instruction with 1 operand. An operand is the part of an instruction that the processor will be working on or that will be changed.

Memory transfer

As we discussed earlier, memory is the workbench of the processor. The processor needs to be able to move data in and out of memory in the correct places. This ability is a very essential part of a processor's instruction set architecture.

Instruction format

Computer instructions in the smallest level are made up of binary digits or bits. A bit is a "1" or a "0" value that is represented by a specific point on a storage medium (such as a hard drive or DVD) by the presence ("1") or absence ("0") of an electronic charge. A bit also represents the presence ("1") or absence ("0") of electronic charges in all digital devices. This includes bit is not limited to the processor, mice, keyboards, printers, etc.

Bits are put together in specific ways to make up instructions. Different types of instructions are obviously formatted differently and if an instruction is formatted incorrectly the processor won’t be able to understand the instruction and therefore it won’t be able to perform the instruction.

Operand specification

An operand is that part of an instruction that tells the processor what it needs to do in order to correctly complete the instruction. The more operands in an instruction the longer and more complex the instruction become. Instruction set architecture or ISA for short makes use of memory addresses and registers. A memory address is the address where the processor can put information about a process it is working on. A register is one single piece or area of memory that the processor uses to store an instruction while it is being processed. It is important that you know the difference between the memory address and a register because they are not located in the same place. Processors have what we call cache memory and the cache memory contains the registers we are talking about. The operands in the instructions tell the computer what areas of memory it should use while the instruction is processed. Operands also indicate to the computer where processed information should be stored in order for it to be easily accessed later.

An instruction should therefore indicate to the processor what operands are to be used in order for the instruction to be processed. We call this operand specification.

Instruction fetch

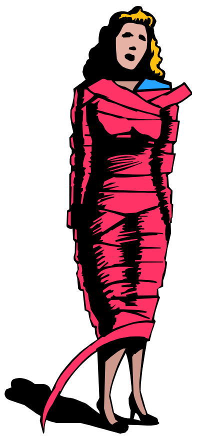
For a computer to process an instruction it needs to transfer the instruction from wherever it is stored into the correct areas of memory inside the memory’s work space. Along with the instruction there’s other information about how the instruction is to be executed and this too needs to be transferred. This is referred to as the fetch cycle so when a computer placed the instruction and its data into the correct places in memory a fetch cycle is completed. Instruction fetch can be defined as a machine cycle used by the processor to obtain instructions from memory.

Word length

Processors are normally rated in bits. For instance modern day computers all have 64bit processors. This refers to the word length. Word length is the amount of bits that are strung together when sent to the processor for processing. For instance a 32bit processor cannot handle 64bit strings. The more bits are grouped together the more complicated the instruction becomes. Though, a 64bit processor is much faster than a 32bit processor. 64bit processors can process more instructions because it is capable of processing larger quantities of data.

2.3.2 How constraints can be overcome

Make use of a RISC processor

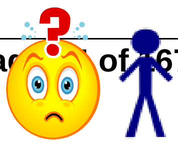


Using a processor built based on RISC principles means that the processor spends less time trying to perform one complex operation but instead performs a sequence of simple operations that do the same thing, only much more efficiently. RISC is an acronym for Reduced Instruction Set Computing. The principles of RISC determine that a processor be built not with a reduced amount of instructions but with instruction sets that is reduced in their complexity. An example of a processor based on RISC principles would be Intel's x86 and x64 ranges of processors. These processors are in our everyday desktop and laptop computers. Using RISC will ensure that work is done more efficiently and promptly.

Increase processor speed

In order to compensate for the performance deficiencies a certain instruction set may have, engineers may increase the speed of the processor in order to help speed up the processing of more complex instructions and so doing overcoming constraints.

Here's a practical real world example which isn't computer related. If a car accelerates from 0 – 100KM/h in 6 seconds and another car does so in 5.5 seconds, the manufacturer of the first car may try to increase the top speed of the car in order to compensate for the 0.5 seconds lost at acceleration.



ACTIVITY 6 – REVIEWED – US 14917 SO 3

Complete this activity in your Portfolio of Evidence Workbooks.

LEARNING UNIT 3: STORAGE OF DATA

Learning outcomes to be achieved

- Demonstrate an understanding of computer data types.
- Describe computer data structures.

3. INTRODUCTION

Data types are important to all computer programming languages. It would be difficult to maintain information within a computer program. Since the major duty of computer programming is to take information, process it, and deliver the information in a different form to the user, data types play a significant part in determining how this goal is achieved.

Different programming languages have different constraints upon the data types that they provide.

It should also be noted that some languages are strongly typed, meaning that the data type of a piece of information has to be declared before that variable (or slot) can be used. Weakly typed languages, on the other hand (like many BASIC variants) do not require that a variable's data type is declared before use, but it should always maintain the same data type throughout its lifespan.

Languages usually allow the possibility to cast (convert) between compatible types.

3.1 COMPUTER DATA TYPES

3.1.1 The binary numbering system

The *binary numbering system* consists of two digits called *binary digits* (bits): zero and one. A switch in the off state represents zero, and a switch in the on state represents one. Today's computers are built from transistors (Electronic Switches). This means that a transistor can only represent one of the two digital switches.

However, two digits don't provide sufficient data to do anything but store the number zero or one in memory. We store more data in memory by logically grouping together switches. For example, two switches enable us to store two binary digits, which give us four combinations, as shown below and these combinations can store numbers 0 through 3. Digits are zero-based, meaning that the first digit in the binary numbering system is zero, not 1. Memory is organized into groups of eight bits called a *byte*, enabling 256 combinations of zeros and ones that can store numbers from 0 through 255.

Table: Combination of Switches

Switch 1	Switch 2	Decimal Value
0	0	0
0	1	1
1	0	2
1	1	3

Binary, Octal and Hexadecimal numbers

- Computers can input and output decimal numbers but they convert them to internal binary representation.

- Binary is good for computers, but is hard for people to read
    - Use numbers easily computed from binary
- Binary numbers use only two different digits: {0,1}
    - Example: 120010 = 00000100101100002
- Octal numbers use 8 digits: {0 - 7}
    - Example: 120010 = 042608
- Hexadecimal numbers use 16 digits: {0-9, A-F}
    - Example: 120010 = 04B016 = 0x04B0
    - One does not distinguish between upper and lower case

3.1.2 Data types

Abstract data types are divided into two categories namely primitive data types and user-defined data types.

3.1.2.1 Primitive data types

Bit

A **bit** or **binary digit** is the basic unit of information in computing and telecommunications; it is the amount of information that can be stored by a digital device or other physical system that usually exist in only two distinct states.

In computer science, a bit is also defined as a variable or computed quantity that can have only two possible values which are often interpreted as binary digits and are usually denoted by the Arabic numerical digits 0 and 1. The term "bit" is a contraction of *binary digit*. The two values may also be understood as logical values (*true*/*false*, *yes*/*no*), algebraic signs (+/−), activation states (*on*/*off*), or any other two-valued attributes. In some programming languages, numeral 0 is equivalent (or convertible) to logical *false* and 1 equals *true*. The correspondence between these values and the physical states of the underlying storage or device is a matter of convention, and different assignments may be used even within the same device or program.

The symbol for bit, as a unit of information, is either "bit" (recommended by the ISO/IEC standard 80000-13 (2008)) or lowercase "b" (recommended by the IEEE 1541 Standard (2002)).

Byte

The byte is a unit of digital information in computing and telecommunications. It is an ordered collection of bits, in which each bit denotes the binary value of 1 or 0. Historically, a byte was the number of bits (typically 5, 6, 7, 8, 9, or 16) used to encode a single character of text in a computer and it is for this reason the basic addressable element in many computer architectures. The size of a byte is typically hardware dependent, but the modern de facto standard is 8 bits, as this is a convenient power of 2. Most of the numeric values used by many applications are representable in 8 bits and processor designers optimize for this common usage. Signal processing applications tend to operate on larger values and some digital signal processors have 16 or 40 bits as the smallest unit of addressable storage (on such processors a byte may be defined to contain this number of bits).

Word

In computing, **word** is a term for the natural unit of data used by a particular computer design. A word is simply a fixed sized group of bits that are handled together by the system. The number of bits in a word (the **word size** or **word length**) is an important characteristic of computer architecture.

The size of a word is reflected in many aspects of a computer's structure and operation; the majority of the registers in the computer are usually word sized and the amount of data transferred between the processing part computer and the memory system, in a single operation, is most often a word. The largest possible address size, used to designate a location in memory, is typically a hardware word (in other words, the full sized natural word of the processor, as opposed to any other definition used on the platform).

Modern computers usually have a word size of 16, 32 or 64 bits but many other sizes have been used, including 8, 9, 12, 18, 24, 36, 39, 40, 48 and 60 bits. The slab is an example of a system with an earlier word size. Several of the earliest computers used the decimal base rather than binary, typically having a word size of 10 or 12 decimal digits and some early computers had no fixed word length at all.

Integer (int), Short integer, Long integer

Integers are whole numbers, and integer variables are used when you know there will not be anything after the decimal point, e.g. if you're writing a lottery ball generator, all the balls have whole numbers on them. The difference between short integers, integers and long integers is the number of *bytes* used to store them. This will vary according to the operating system and hardware you're using, but these days you can assume that an integer will be at least 16 bits, and a long integer is probably at least 32. In a 32-bit environment, it is more efficient to use long integers (i.e. a whole word), and so many compilers will automatically use long integers unless you specify a short one.

Important Integer information:

- sbyte (-128 to 127): signed 8-bit
- byte (0 to 255): unsigned 8-bit
- short (-32,768 to 32,767): signed 16-bit
- ushort (0 to 65,535): unsigned 16-bit
- int (-2,147,483,648 to 2,147,483,647): signed 32-bit
- uint (0 to 4,294,967,295): unsigned 32-bit
- long (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807): signed 64-bit
- ulong (0 to 18,446,744,073,709,551,615): unsigned 64-bit

Float, Single, Double

Floating point numbers contain fractional parts - i.e. they are not whole numbers. The single and double quantifiers are similar to the short and long quantifiers used with integers. They indicate how many bits are being used to store a variable. Floating point arithmetic can lead to problems with rounding and accuracy, so when dealing with a limited number of decimal places, it is better to use integers and multiply all your values by a power of 10. For example, if you're dealing with money, it's better to work in cents and use integers than to work in rand and use floating point variables.

Important Floating point information:

- Floating-point types are:

  float (±1.5 × 10^{-45} to ±3.4 × 10^{38}): 32-bits, precision of 7 digits

  double (±5.0 × 10^{-324} to ±1.7 × 10^{308}): 64-bits, precision of 15-16 digits
- The default value of floating-point types:

  Is 0.0F for the float type

  Is 0.0D for the double type

Fixed-Point number

In computing, a fixed-point number representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (*e.g.*, after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating point number representation.

Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

Characters

In computer terminology, a character is a unit of information that roughly corresponds to a grapheme, grapheme-like unit, or symbol, such as in an alphabet or syllable in the written form of a natural language.

Examples of characters include letters, numerical digits, and common punctuation marks (such as '.' or '-'). The concept also includes control characters, which do not correspond to symbols in a particular natural language, but rather to other bits of information used to process text in one or more languages. Examples of control characters include carriage return or tab, as well as instructions to printers or other devices that display or otherwise process text.

Characters are typically combined into strings.

Strings

- Fixed-length string

Strings are variables that contain text, and they come in two sorts. With a fixed-length string, you declare how many characters the string is going to hold. Certain API calls in Windows require the use of fixed-length strings, but generally they are not used in BASIC. In C they are implemented as an array (or vector) of *char*s.

- Variable-length string

A variable-length string is one where you don't define the length. This is the default type in BASIC, and is useful for taking user input where you don't know what the response will be. The maximum length of the string will depend on your environment, but it should be at least 255 characters.

Rational

A Rational number is a number that can be written in the form *a*/*b*, where *a* and *b* are integers, with *b* ≠ 0 (*b is not equal to 0)*. The set of all rational numbers is usually denoted by **Q**. Rational numbers cannot be represented natively in a computer.

Some programming languages provide a built-in (primitive) **rational** data type to represent rational numbers like 1/3 and -11/17 without rounding, and to do arithmetic on them. Examples are the ratio type of Common Lisp, and analogous types provided by most languages for algebraic computation, such as Mathematica and Maple. Many languages that do not have a built-in rational type still provide it as a library-defined type.

Boolean

A Boolean variable can store one of two values - either TRUE or FALSE. Like *char*, this is usually an integer - in Visual BASIC, for example, FALSE is 0 and TRUE is -1, and the TRUE and FALSE values themselves are constants.

In computer science, the Boolean or logical data type is a primitive data type having one of two values: true or false, intended to represent the truth values of logic and Boolean algebra.

In programming languages with a built-in Boolean data type, like Pascal and Java, the comparison operators such as '>' and '≠' are usually defined to return a Boolean value. Also, conditional and iterative commands may be defined to test Boolean-valued expressions.

Languages without an explicit Boolean data type, like C may still represent truth values by some other data type. C uses an integer type, with false represented as the zero value, and true as any non-zero value (such as 1 or -1). Indeed, a Boolean variable may be regarded (and be implemented) as a numerical variable with a single binary digit (bit), which can store only two values.

Variables

A variable is a placeholder of information that can usually be changed at run-time. Variables allow you to:

- Store information
- Retrieve the stored information
- Manipulate the stored information

Variable Characteristics

A variable has:

- Name
- Type (of stored data)
- Value

Example:

- int count = 5;
- Name: counter
- Type: int
- Value: 5

3.1.2.1 User defined data types

This is a group of primitive data types defined by the programmer. For example, if a programmer wanted to store students' grades in memory. She will need to store 4 data elements: the student's ID, first name, last name, and grade. She could use primitive data types for each data element, but primitive data types are not grouped together; each exists as separate data elements.

Another approach is to group primitive data types into a user-defined data type to form a record. Remember that a database consists of one or more tables. A table is similar to a spread sheet consisting of columns and rows. A row is also known as a record. A user-defined data type defines columns (primitive data types) that comprise a row (a user-defined data type).

The format used to define a user-defined data type varies depending on the programming language used to write the program. Some programming languages, like Java, do not support user-defined data types but instead, attributes of a class are used to group together primitive data types.

In the C and C++ programming languages, you define a user-defined data type by defining a structure. Think of a structure as a stencil of the letter *A*. The stencil isn't the letter *A*, but it defines what the letter *A* looks like. If you want a letter *A*, you place the stencil on a piece of paper and trace the letter *A*. If you want to make another letter *A*, you use the same stencil and repeat the process. You can make as many letter *A*'s as you wish by using the stencil.

ACTIVITY 7 – REVIEWED – US14944 SO1

Complete this activity in your Portfolio of Evidence Workbooks.

3.2 CHARACTER SETS AND CHARACTER ENCODING

3.2.1 Character encoding

Computer systems process characters using numeric codes not the graphical representation of the character. For example, when a database stores the letter **X**, a numeric code that is interpreted by software as the letter is stored. These numeric codes are useful in a global environment because of the potential need to convert data between different character sets.

You specify an encoded character set when you create a computer program or a database. Choosing a character set determines languages that can be represented in the program or the database. It will affect:

- How you develop applications that process character data
- How you create the database schema
- How the database works with the operating system

A group of characters (for example, alphabetic characters, ideographs, symbols, punctuation marks, and control characters) can be encoded as a character set. An **encoded character set** assigns unique numeric codes to each character in the character repertoire (available characters). The numeric codes are called **code points** or **encoded values**.

The computer industry has various encoded character sets. Character sets differ in the following ways:

- The number of characters available
- The available characters (the **character repertoire**)
- The scripts used for writing and the languages they represent
- The code values assigned to each character
- The encoding scheme used to represent a character

A **character encoding** system consists of a code that pairs each character from a given repertoire with something else, such as a sequence of natural numbers, octets or electrical pulses, in order to facilitate the transmission of data (generally numbers and/or text) through telecommunication networks or storage of text in computers.

Other terms like **character encoding**, **character set***(char set)*, and sometimes **character map** or **code page** are used almost interchangeably, but these terms now have related but distinct meanings

To represent numeric, alphabetic, and special characters in a computer's internal storage and on magnetic media, we must use some sort of coding system. In computers, the code is made up of fixed size groups of binary positions. Each binary position in a group is assigned a specific value; for example 8, 4, 2, or 1. In this way, every character can be represented by a combination of bits that is different from any other combination.

In this section you will learn how the selected coding systems are used to represent data.

3.2.2 Binary coded decimal

In computer science and electronic systems, **binary-coded decimal** (**BCD**) (sometimes called **natural binary-coded decimal**, **NBCD**) or, in its most common modern implementation, packed decimal, is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display, and allows faster decimal calculations. Its drawbacks are a small increase in the complexity of circuits needed to implement mathematical operations. Uncompressed BCD is also a relatively inefficient encoding—it occupies more space than a purely binary representation.

In **BCD**, a digit is usually represented by four bits which, in general, represent the decimal digits 0 through 9. Other bit combinations are sometimes used for a sign or for other indications (e.g., error or overflow).
Although uncompressed BCD is not as widely used as it once was, decimal fixed-point and floating-point are still important and continue to be used in financial, commercial, and industrial computing.

Recent decimal floating-point representations use base-10 exponents, but not BCD encodings. Current hardware implementations, however, convert the compressed decimal encodings to BCD internally before carrying out computations. Software implementations of decimal arithmetic typically use BCD or some other 10^n base, depending on the operation.

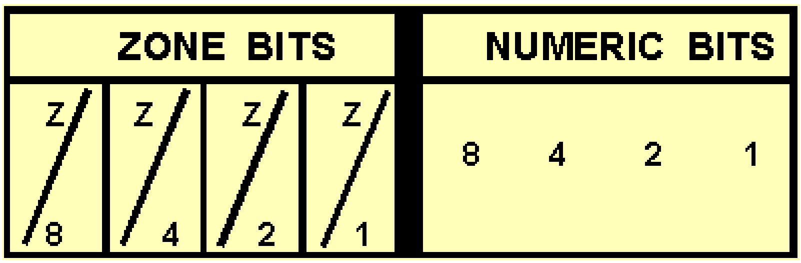
3.2.3 Extended binary coded decimal interchange code (EBCDIC)

Using an 8-bit code, it is possible to represent 256 different characters or bit combinations. This provides a unique code for each decimal value 0 through 9 (for a total of 10), each **uppercase** and **lowercase** letter (for a total of 52), and for a variety of special characters. In addition to four numeric bits, **four** zone bit positions are used in 8-bit code as illustrated in figure below. Each group of the eight bits makes up one alphabetic, numeric, or special

Format for EBCDIC and ASCII codes

When you look at the table, you will notice that the four rightmost bits in EBCDIC are assigned values of 8, 4, 2, and 1. The next four bits to the left are called the zone bits.

Table: Format for EBCDIC Codes

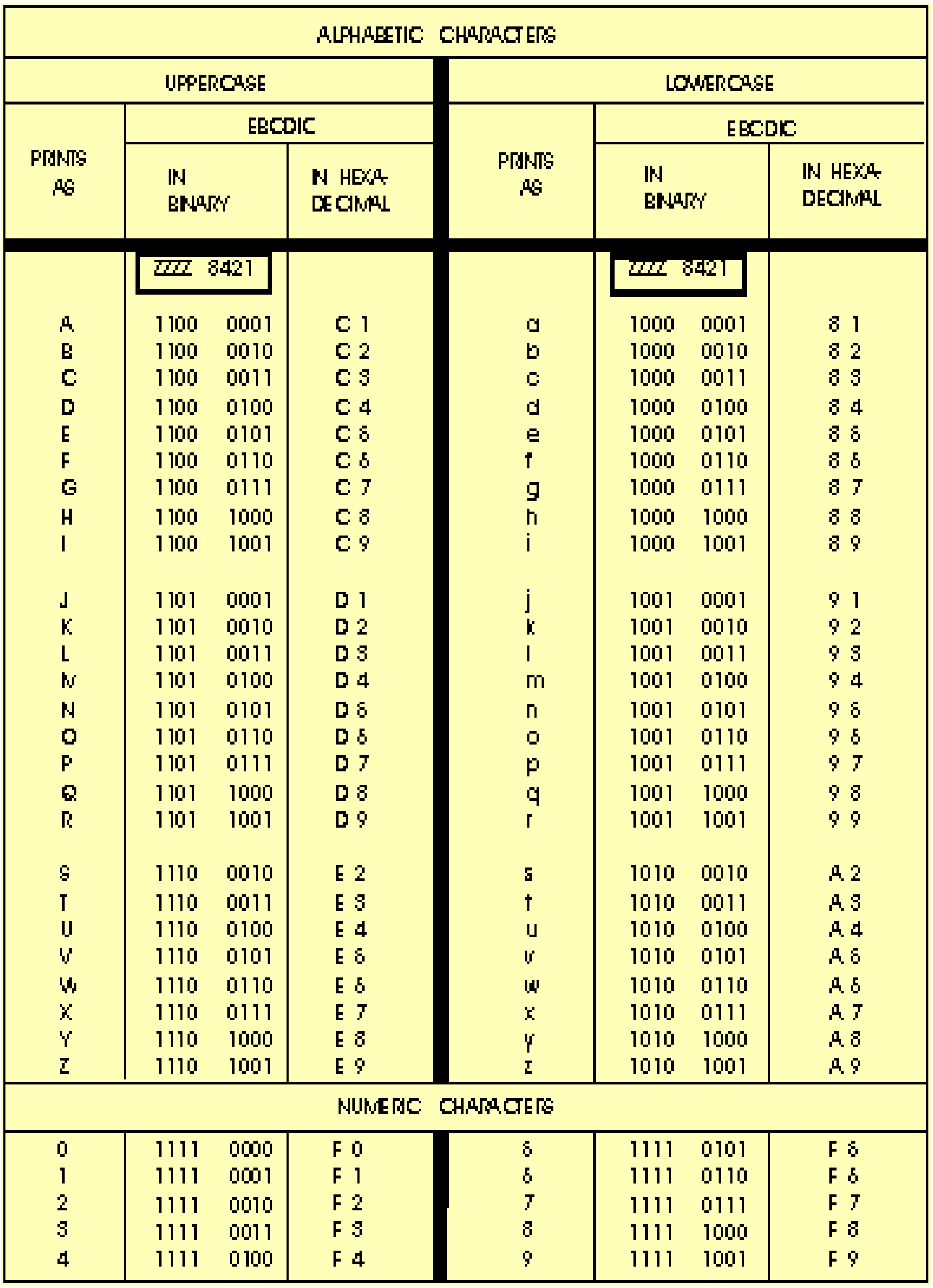


ZONE BITS				NUMERIC BITS			
Z/8	Z/4	Z/2	Z/1	8	4	2	1

The EBCDIC coding chart for uppercase and lowercase alphabetic characters and for the numeric digits 0 through 9 is shown in the next table, with their hexadecimal equivalents. Hexadecimal is a number system used with some computer systems. It has a base of 16 (0-9 and A-F). A represents 10; B represents 11; C represents 12; D represents 13; E represents 14; and F represents 15. In EBCDIC, the bit pattern 1100 is the zone combination used for the alphabetic characters A through I, 1101 is used for the characters J through R, and 1110 is the zone combination used for characters S through Z. The bit pattern 1111 is the zone combination used when representing decimal digits. For example, the code 11000001 is equivalent to the letter **A**; the code 11110001 is equivalent to the decimal digit 1. Other zone combinations are used when forming special characters. Not all of the 256 combinations of 8-bit code have been assigned characters.

Table: Eight-bit EBCDIC coding chart

(Including hexadecimal equivalents)



ALPHABETIC CHARACTERS					
UPPERCASE			LOWERCASE		
PRINTS AS	EBCDIC		PRINTS AS	EBCDIC	
	IN BINARY	IN HEXADECIMAL		IN BINARY	IN HEXADECIMAL
	ZZZZ 8421			ZZZZ 8421	
A	1100 0001	C 1	a	1000 0001	8 1
B	1100 0010	C 2	b	1000 0010	8 2
C	1100 0011	C 3	c	1000 0011	8 3
D	1100 0100	C 4	d	1000 0100	8 4
E	1100 0101	C 5	e	1000 0101	8 5
F	1100 0110	C 6	f	1000 0110	8 6
G	1100 0111	C 7	g	1000 0111	8 7
H	1100 1000	C 8	h	1000 1000	8 8
I	1100 1001	C 9	i	1000 1001	8 9
J	1101 0001	D 1	j	1001 0001	9 1
K	1101 0010	D 2	k	1001 0010	9 2
L	1101 0011	D 3	l	1001 0011	9 3
M	1101 0100	D 4	m	1001 0100	9 4
N	1101 0101	D 5	n	1001 0101	9 5
O	1101 0110	D 6	o	1001 0110	9 6
P	1101 0111	D 7	p	1001 0111	9 7
Q	1101 1000	D 8	q	1001 1000	9 8
R	1101 1001	D 9	r	1001 1001	9 9
S	1110 0010	E 2	s	1010 0010	A 2
T	1110 0011	E 3	t	1010 0011	A 3
U	1110 0100	E 4	u	1010 0100	A 4
V	1110 0101	E 5	v	1010 0101	A 5
W	1110 0110	E 6	w	1010 0110	A 6
X	1110 0111	E 7	x	1010 0111	A 7
Y	1110 1000	E 8	y	1010 1000	A 8
Z	1110 1001	E 9	z	1010 1001	A 9
NUMERIC CHARACTERS					
0	1111 0000	F 0	5	1111 0101	F 5
1	1111 0001	F 1	6	1111 0110	F 6
2	1111 0010	F 2	7	1111 0111	F 7
3	1111 0011	F 3	8	1111 1000	F 8
4	1111 0100	F 4	9	1111 1001	F 9

Since one numeric character can be represented and stored using only four bits (8-4-2-1), using an 8-bit code allows the representation of two numeric characters (decimal digits) as illustrated in the table below. Representing two numeric characters in one byte (eight bits) is referred to as **packing** or **packed** data. By packing data (numeric characters only) in this way, it allows us to conserve the amount of storage space required, and at the same time, increases processing speed.

Table: Packed Data

DECIMAL VALUE	**92**	**73**
EBCDIC	10010010	01110011
BIT PLACE VALUES	84218421	8421
8421	B Y T E 1	B Y T E 2

3.2.4 American standard code for information interchange (ASCII)

Another 8-bit code, known as the American Standard Code for Information Interchange (**ASCII**) (pronounced ASS-KEY), was originally designed as a 7-bit code. Several computer manufacturers cooperated to develop this code for **transmitting** and processing data. The purpose was to **standardise** a binary code to give the computer user the capability of using several machines to process data regardless of the manufacturer: IBM, HONEYWELL, UNIVAC, BURROUGHS, and so on. However, since most computers are designed to handle (store and manipulate) 8-bit code, an 8-bit version of ASCII was developed. ASCII is commonly used in the transmission of data through data communications and is used almost exclusively to represent data internally in microcomputers.

The concepts and advantages of ASCII are **identical** to those of EBCDIC. The important difference between the two coding systems lies in the 8-bit combinations assigned to represent the various alphabetic, numeric, and special characters. When using ASCII 8-bit code, you will notice the selection of bit patterns used in the positions differs from those used in EBCDIC. For example, let's look at the characters **DP3** in both EBCDIC and ASCII to see how they compare.

Table: ASCII / EBCDIC Comparison

Character	**D**	**P**	**3**
EBCDIC	1100 0100	1101 0111	1111 0011
ASCII	0100 0100	0101 0000	0011 0011

In ASCII, rather than breaking letters into three groups, uppercase letters are assigned codes beginning with hexadecimal value **41** and continuing sequentially through hexadecimal value **5A**. Similarly, lowercase letters are assigned hexadecimal values of **61** through **7A**.

The decimal values **1** through **9** are assigned the zone code 0011 in ASCII rather that 1111 as in EBCDIC. The table below is the ASCII coding chart showing uppercase and lowercase alphabetic characters and numeric digits 0 through 9.

Table: Eight-bit ASCII coding chart

(Including hexadecimal equivalents)

ALPHABETIC CHARACTERS							
UPPERCASE				LOWERCASE			
PRINTS AS	ASCII CODE			PRINTS AS	ASCII CODE		
	IN BINARY		IN HEXA-DECIMAL		IN BINARY		IN HEXA-DECIMAL
	8421	8421			8421	8421	
A	0100	0001	4 1	a	0110	0001	6 1
B	0100	0010	4 2	b	0110	0010	6 2
C	0100	0011	4 3	c	0110	0011	6 3
D	0100	0100	4 4	d	0110	0100	6 4
E	0100	0101	4 5	e	0110	0101	6 5
F	0100	0110	4 6	f	0110	0110	6 6
G	0100	0111	4 7	g	0110	0111	6 7
H	0100	1000	4 8	h	0110	1000	6 8
I	0100	1001	4 9	i	0110	1001	6 9
J	0100	1010	4 A	j	0110	1010	6 A
K	0100	1011	4 B	k	0110	1011	6 B
L	0100	1100	4 C	l	0110	1100	6 C
M	0100	1101	4 D	m	0110	1101	6 D
N	0100	1110	4 E	n	0110	1110	6 E
O	0100	1111	4 F	o	0110	1111	6 F
P	0101	0000	5 0	p	0111	0000	7 0
Q	0101	0001	5 1	q	0111	0001	7 1
R	0101	0010	5 2	r	0111	0010	7 2
S	0101	0011	5 3	s	0111	0011	7 3
T	0101	0100	5 4	t	0111	0100	7 4
U	0101	0101	5 5	u	0111	0101	7 5
V	0101	0110	5 6	v	0111	0110	7 6
W	0101	0111	5 7	w	0111	0111	7 7
X	0101	1000	5 8	x	0111	1000	7 8
Y	0101	1001	5 9	y	0111	1001	7 9
Z	0101	1010	5 A	z	0111	1010	7 A
NUMERIC CHARACTERS							
0	0011	0000	3 0	5	0011	0101	3 5
1	0011	0001	3 1	6	0011	0110	3 6
2	0011	0010	3 2	7	0011	0111	3 7
3	0011	0011	3 3	8	0011	1000	3 8
4	0011	0100	3 4	9	0011	1001	3 9

At this point you should understand how coding systems are used to represent data in both EBCDIC and ASCII. Regardless of what coding system is used, each character will have an additional bit called a check bit or parity bit.

Parity Bit

This additional check or parity bit in each storage location is used to detect errors in the circuitry. Therefore, a computer that uses an 8-bit code, such as EBCDIC or ASCII, will have a ninth bit for parity checking.

The parity bit (also called a check bit, the C position in a code) provides an internal means for checking the validity, the correctness, of code construction. That is, the total number of bits in a character, including the parity bit, must always be **odd** or **always** be **even**, depending upon whether the particular computer system or device you are using is odd or even parity. Therefore, the coding is said to be in either **odd** or **even** parity code, and the test for bit count is called a **parity check**.

3.2.5 Unicode

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.

Design Principles. Here we will start from the proclaimed design principles of Unicode. Later there will be some critical notes and considerations. We will first consider the very general, slogan-like expressions of the goals, and then the more technical principles.

Goals of Unicode: Universality, Efficiency, Unambiguous

The Unicode standard itself says that it was designed to be universal, efficient, and unambiguous. These slogans have real meaning here, but it is important to analyze what they mean and what they do not mean. Let us first see how they are presented in the Unicode standard, and then analyze each item:

The Unicode Standard was designed to be:

- **Universal** The repertoire must be large enough to encompass all characters that are likely to be used in general text interchange, including those in major international, national, and industry character sets.
- **Efficient** Plain text is simple to parse: software does not have to maintain state or look for special escape sequences, and character synchronisation from any point in a character stream is quick and unambiguous. A fixed character code allows for efficient sorting, searching, display, and editing of text.
- **Unambiguous** Any given Unicode code point always represents the same character.

Universality means much more than just creating a superset of sets of characters. Practically all other character codes are limited to the needs of one language or a collection of languages that are similar in their use of characters, such as Western European languages. Unicode needs to encompass a variety of essentially different collections of characters and writing systems. For example, it cannot postulate that all text is written left to right, or that all letters have uppercase and lowercase forms, or that text can be divided into words separated by spaces or other whitespace.

Moreover, Unicode has been designed to be universal among character codes. That is, it assigns code points to the characters included in other codes, even if the characters could be treated as variants or combinations of other characters. The reason is that Unicode was also designed for use as an intermediate code. You can take character data in any character code and convert it to Unicode without losing information. If you convert it back, you get the exact original data. You can also convert it to a third character code, provided that it is capable of representing all the characters. If the source and destination codes treat, say, £ (pound sign) and ₤ (lira sign) as different, they will appear as different after the conversion that used Unicode as an intermediate code.

Thus, universality implies complexity rather than simplicity. Unicode needs to define properties of characters in a manner that makes explicit many things that we might take for granted because they are not evident at all across writing systems.

Efficiency refers here to efficient processing of data. When all characters have unique identifying numbers, and they are internally represented by those numbers, it is much easier to work with character data than in a system where the same number may mean different characters, depending on encoding or font or other issues. However, efficiency is relative. In particular:

- Efficiency of processing often requires presentation that is wasteful in terms of storage needed (e.g., using four octets for each character). This in turn causes inefficiency in data transfer.
- The representation forms of Unicode are not always efficient in processing. In particular, the common UTF-8 format requires linear processing of the data stream in order to identify characters; it is not possible to jump to the *n*th character in a UTF-8 encoded string.
- Unicode contains a large amount of characters and features that have been included only for compatibility with other standards. This may require pre-processing that deals with compatibility characters and with different Unicode representations of the same character (e.g., letter é as a single character or as two characters).
- For a specific data-processing task, Unicode can be less efficient than other codes. The efficiency goal needs to be understood with the implicit reservation "to the extent possible, given the universality goal."

Unambiguity may look like a self-evident principle, but not all character codes are unambiguous in the Unicode sense. For example, ISO 646 permits variation in some code points, allowing the use of a single code point for either # or £ by special agreement. Moreover, in Unicode, unambiguity also means unambiguity across time and versions: a code point, once assigned, will never be re-assigned in a future version.

Sometimes a fourth fundamental principle, uniformity, is mentioned. It has been described as a principle of using a fixed-length character code, to allow efficient processing of text. However, as currently defined, Unicode does not use a fixed-length code in a simple sense. In some Unicode encodings, all characters are represented using the same number of octets (or bits), but in many important encodings, such as UTF-8, the lengths may vary.

The 10 Design Principles

The Unicode standard describes "The 10 Unicode Design Principles," where the first two are the same as those quoted in the previous section, universality and efficiency. The unambiguity principle is not included. Obviously, the principles are meant to describe how Unicode was designed, whereas the slogan "Universality, Efficiency, Unambiguity" is meant to describe the ultimate goals.

The standard admits that there are conflicts between the principles, and it does not specify how the conflicts are resolved. As a whole, the set of principles describe ideas of varying levels (from fundamentals to technicalities), and it should be read critically. It is however important to know the underlying ideas, so we will discuss them briefly:

Universality

Unicode defines a single repertoire of characters for universal use. (See the previous section for other aspects of universality.)

Efficiency

Unicode text is simple to process. (See the previous section for the complexity of this issue.)

Characters, not glyphs

Unicode assigns code points to characters as abstractions, not to visual appearances. Although there are many borderline cases, and although the compatibility characters can be seen as violating this principle, it is still one of the fundamentals of Unicode. The relationship between characters and glyphs is rather simple for languages like English: mostly each character is presented by one glyph, taken from a font that has been chosen. For other languages, the relationship can be much more complex e.g. routinely combining several characters into one glyph.

Semantics

Characters have well-defined meanings. In fact, the meanings are often defined rather indirectly or implicitly, if at all but Unicode are generally much more explicit about meanings than other character code standards, including ISO 10646. When the Unicode standard refers to semantics, it often means (mostly) the properties of characters, such spacing, combinability, and directionality, rather than what the character really means. This is largely intentional: the ultimate meaning may vary by language, context, and usage; think about the various uses of the comma in English and other languages e.g., as thousands separator or as a decimal separator.

Plain text

Unicode deals with plain text i.e. strings of characters without formatting or structuring information (except for things like line breaks). In practice, Unicode text is mostly used along with some formatting or structuring information, such as a word processor's formatting commands or some markup; but that is treated as a separate layer in data, above the character level and outside the scope of the Unicode standard.

Logical order

The default representation of Unicode data uses logical order of data, as opposed to approaches that handle writing direction by changing the order of characters. The ordering principles also put all diacritics after the base character to which they are applied, regardless of visual placement. For example, the Greek capital letter omega with tonos has the tonos (stress mark) visually on the left of the omega, but the decomposed form of this character still consists of omega followed by combining tonos.

Unification

Unicode encodes duplicates of a character as a single code point, if they belong to the same script but different languages. For example, the letter ü denoting a particular vowel in German is treated as the same as the letter ü in Spanish, where it simply indicates that the "u" is pronounced, in a context where it would otherwise be mute.

Dynamic composition

Characters with diacritic marks can be composed dynamically, using characters designated as combining marks. You can take almost any character and combine it with any diacritic; for example, you can create ,̃ (comma with tilde) by using the normal comma character and a combining tilde. Therefore, you can write many more characters using Unicode than there are characters in Unicode (i.e., code points allocated to characters)! You can also use multiple combining marks on a character (e.g., you can just make up "a" with both a tilde and an acute accent: ã́), although good rendering of such combinations often requires advanced techniques.

Equivalent sequences

Unicode has a large number of characters that are pre-composed forms, such as é. They have decompositions that are declared as equivalent to the pre-composed form. An application may still treat the pre-composed form and the decomposition differently, since as strings of encoded characters, they are distinct. However, usually such distinctions are not made, and should not be made. The Unicode standard does not declare either the pre-composed form or the decomposed form as preferred; they are just two different forms. So-called normalisation may make either form preferred in some contexts.

Convertibility

Character data can be accurately converted between Unicode and other character standards and specifications. As explained earlier, this can be regarded as part of the universality principle.

Somewhat surprisingly, the list does not mention stability or continuity. Yet, one of the leading principles in Unicode strategy (as described in the goals as "unambiguity") is that a code point assignment once made will never be changed. When a number and a name have been given to a character, they will remain in all future versions, though the properties of the character may be changed.

Unicode Terms

Deprecated and Obsolete Characters

A *deprecated* character is a character that has been included in Unicode but declared as deprecated in the Unicode standard. This indicates a strong recommendation that the character not be used. It remains in Unicode, though, due to the stability principle. For example, a character may be declared deprecated if it turns out that it was introduced into Unicode in error.

Digraphs

A digraph is a combination of two successive characters treated as a unit in some sense, such as "ch" in many languages (e.g., when used to indicate one sound) or "ll" in Spanish, where it denotes a particular sound and might be treated in sorting as if it were a single character. Thus, a digraph is a pragmatic concept, not a formal one, and it is an example of a text element.

Text Elements

The concept of text element is informal: it means a sequence of characters (including the special case of one character) that is treated as a unit in some processing. In typical character input and output, characters are text elements. In layout processes, syllables might be treated as text elements, since line breaks are usually allowed between syllables but not within them. When you form a text concordance (a list of occurrences of words e.g., in alphabetic order), a word is a text element.

Unicode Strings

The term "Unicode string" has a more technical meaning than you might expect. It does not refer to a string (sequence) of Unicode characters (code points) but to a sequence of code units. Thus, the components of the string are of fixed size in bits (in practice, 8, 16, or 32 bits). In many programming languages, Unicode strings have a code unit size of 16 bits. This does not limit the range of characters, since such a string could be interpreted according to UTF-16.

Table: Unicode Chart

Range	Decimal	Name
0x0000-0x007F	0-127	Basic Latin
0x0080-0x00FF	128-255	Latin-1 Supplement
0x0100-0x017F	256-383	Latin Extended-A
0x0180-0x024F	384-591	Latin Extended-B
0x0250-0x02AF	592-687	IPA Extensions
0x02B0-0x02FF	688-767	Spacing Modifier Letters
0x0300-0x036F	768-879	Combining Diacritical Marks
0x0370-0x03FF	880-1023	Greek
0x0400-0x04FF	1024-1279	Cyrillic
0x0530-0x058F	1328-1423	Armenian
0x0590-0x05FF	1424-1535	Hebrew
0x0600-0x06FF	1536-1791	Arabic
0x0700-0x074F	1792-1871	Syriac
0x0780-0x07BF	1920-1983	Thaana
0x0900-0x097F	2304-2431	Devanagari
0x0980-0x09FF	2432-2559	Bengali

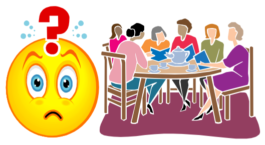
0x0A00-0x0A7F	2560-2687	Gurmukhi
0x0A80-0x0AFF	2688-2815	Gujarati
0x0B00-0x0B7F	2816-2943	Oriya
0x0B80-0x0BFF	2944-3071	Tamil
0x0C00-0x0C7F	3072-3199	Telugu
0x0C80-0x0CFF	3200-3327	Kannada
0x0D00-0x0D7F	3328-3455	Malayalam
0x0D80-0x0DFF	3456-3583	Sinhala
0x0E00-0x0E7F	3584-3711	Thai
0x0E80-0x0EFF	3712-3839	Lao
0x0F00-0x0FFF	3840-4095	Tibetan
0x1000-0x109F	4096-4255	Myanmar
0x10A0-0x10FF	4256-4351	Georgian
0x1100-0x11FF	4352-4607	Hangul Jamo
0x1200-0x137F	4608-4991	Ethiopic
0x13A0-0x13FF	5024-5119	Cherokee
0x1400-0x167F	5120-5759	Unified Canadian Aboriginal Syllabics
0x1680-0x169F	5760-5791	Ogham
0x16A0-0x16FF	5792-5887	Runic
0x1780-0x17FF	6016-6143	Khmer
0x1800-0x18AF	6144-6319	Mongolian
0x1E00-0x1EFF	7680-7935	Latin Extended Additional
0x1F00-0x1FFF	7936-8191	Greek Extended
0x2000-0x206F	8192-8303	General Punctuation
0x2070-0x209F	8304-8351	Superscripts and Subscripts
0x20A0-0x20CF	8352-8399	Currency Symbols
0x20D0-0x20FF	8400-8447	Combining Marks for Symbols
0x2100-0x214F	8448-8527	Letterlike Symbols
0x2150-0x218F	8528-8591	Number Forms
0x2190-0x21FF	8592-8703	Arrows
0x2200-0x22FF	8704-8959	Mathematical Operators
0x2300-0x23FF	8960-9215	Miscellaneous Technical

0x2400-0x243F	9216-9279	Control Pictures
0x2440-0x245F	9280-9311	Optical Character Recognition
0x2460-0x24FF	9312-9471	Enclosed Alphanumerics
0x2500-0x257F	9472-9599	Box Drawing
0x2580-0x259F	9600-9631	Block Elements
0x25A0-0x25FF	9632-9727	Geometric Shapes
0x2600-0x26FF	9728-9983	Miscellaneous Symbols
0x2700-0x27BF	9984-10175	Dingbats
0x2800-0x28FF	10240-10495	Braille Patterns
0x2E80-0x2EFF	11904-12031	CJK Radicals Supplement
0x2F00-0x2FDF	12032-12255	Kangxi Radicals
0x2FF0-0x2FFF	12272-12287	Ideographic Description Characters
0x3000-0x303F	12288-12351	CJK Symbols and Punctuation
0x3040-0x309F	12352-12447	Hiragana
0x30A0-0x30FF	12448-12543	Katakana
0x3100-0x312F	12544-12591	Bopomofo
0x3130-0x318F	12592-12687	Hangul Compatibility Jamo
0x3190-0x319F	12688-12703	Kanbun
0x31A0-0x31BF	12704-12735	Bopomofo Extended
0x3200-0x32FF	12800-13055	Enclosed CJK Letters and Months
0x3300-0x33FF	13056-13311	CJK Compatibility
0x3400-0x4DB5	13312-19893	CJK Unified Ideographs Extension A
0x4E00-0x9FFF	19968-40959	CJK Unified Ideographs
0xA000-0xA48F	40960-42127	Yi Syllables
0xA490-0xA4CF	42128-42191	Yi Radicals
0xAC00-0xD7A3	44032-55203	Hangul Syllables
0xD800-0xDB7F	55296-56191	High Surrogates
0xDB80-0xDBFF	56192-56319	High Private Use Surrogates
0xDC00-0xDFFF	56320-57343	Low Surrogates

0xE000-0xF8FF	57344-63743	Private Use
0xF900-0xFAFF	63744-64255	CJK Compatibility Ideographs
0xFB00-0xFB4F	64256-64335	Alphabetic Presentation Forms
0xFB50-0xFDFF	64336-65023	Arabic Presentation Forms-A
0xFE20-0xFE2F	65056-65071	Combining Half Marks
0xFE30-0xFE4F	65072-65103	CJK Compatibility Forms
0xFE50-0xFE6F	65104-65135	Small Form Variants
0xFE70-0xFEFE	65136-65278	Arabic Presentation Forms-B
0xFEFF-0xFEFF	65279-65279	Specials
0xFF00-0xFFEF	65280-65519	Halfwidth and Fullwidth Forms
0xFFF0-0xFFFD	65520-65533	Specials

ACTIVITY 8 – REVIEWED – US14944 SO 1

Complete this activity in your Portfolio of Evidence Workbooks.



3.3 DATA MANIPULATION

Data manipulation refers to changes that can be made onto the different data types for example manipulating integers.
For example many programming languages have several routines that are useful for manipulating data types. They may include:

Arithmetic Operators

Routines which performs arithmetic operations on *decimal*, *float*, *money*, *numeric*, and *real* data types.

These operators are likely to be the operators most familiar to you because these are math operators that you are used to using for normal math. There are seven common arithmetic operators.

Operator	Description	Example	Result
+	Addition	If x = 6 x + 4	10
-	Subtraction	If x = 6 x - 2	4
*	Multiplication	If x = 6 x * 2	12
/	Division	If x = 6 x / 3	2
++	Increment (add one)	If x = 6 x++	7
--	Decrement (subtract one)	If x = 6 x--	5
%	Modulus (division remainder)	7%2 10%4 12%6	1 2 0

Comparison Operators

Routines which compares date, time, decimal, float, money, numeric, and real data types Comparison operators are used to compare two things. You will find that when you get to working with code and its structure these will be really useful.

There are six comparison operators. Each of these returns is either true or false, depending on whether the outcome of the comparison is true or false.

Operator	Description	Example
`==`	Equal to	`5 == 8` (returns false) `8 == 8` (returns true) `"cat" == "dog"` (returns false) `"cat" == "cat"` (returns true)
`!=`	Not equal to	`5 != 8` (returns true) `8 != 8` (returns false)
`>`	Greater than	`5 > 8` (returns false) `8 > 3` (returns true)
`<`	Less than	`5 < 8` (returns true) `8 < 3` (returns false)
`>=`	Greater than or equal to	`5 >= 8` (returns false) `8 >= 3` (returns true) `8 >= 8` (returns true)
`<=`	Less than or equal to	`5 <= 8` (returns true) `8 <= 3` (returns false) `3 <= 3` (returns true)

Assignment Operators

Assignment operators are used to assign a value. These will be used extensively when using variables. There are six assignment operators. In the preceding examples, x and y are both variables. Variables need to have unique names (unless you are reusing them).

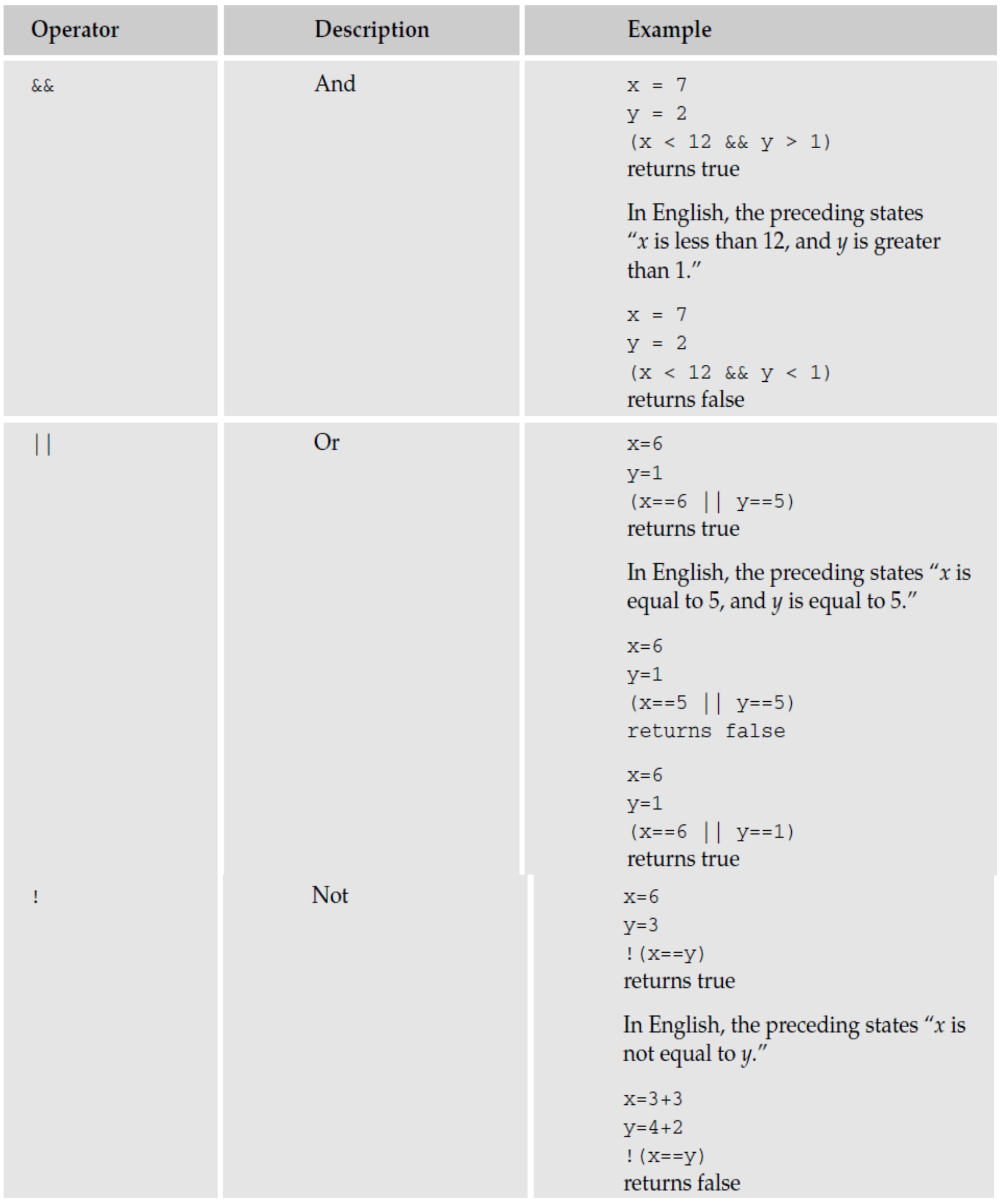
Operator	Example	Is Equivalent to . . .

`=`	`x = 6`
	`x = y`
		`x = 6`
		`x = y`

`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Logical Operators

Logical operators enable you to introduce logic to the code you write and enable you to make combinations of the preceding operators. There are three logical operators.



Operator	Description	Example

`&&`	And	`x = 7`
		`y = 2`
		`(x < 12 && y > 1)`
		returns true
		
		In English, the preceding states "*x* is less than 12, and *y* is greater than 1."
		
		`x = 7`
		`y = 2`
		`(x < 12 && y < 1)`
		returns false

`||`	Or	`x=6`
		`y=1`
		`(x==6 || y==5)`
		returns true
		
		In English, the preceding states "*x* is equal to 5, and *y* is equal to 5."
		
		`x=6`
		`y=1`
		`(x==5 || y==5)`
		`returns false`
		
		`x=6`
		`y=1`
		`(x==6 || y==1)`
		returns true

`!`	Not	`x=6`
		`y=3`
		`!(x==y)`
		returns true
		
		In English, the preceding states "*x* is not equal to *y*."
		
		`x=3+3`
		`y=4+2`
		`!(x==y)`
		returns false

String Operators

String operators are operators that work on strings. Strings are usually snippets of text. Here are two sample strings:

string1 = “Hello”

string2 = “World!”

Using string operators, you can join (or concatenate) strings together.

string3 = string1 + string2

The variable string3 will now hold the value Hello World!

One thing that newcomers to strings seem to find hard to cope with is spaces. That is, keeping track of spaces between strings and making sure that the end string makes sense.

ACTIVITY 9 – REVIEWED – US149442 SO1

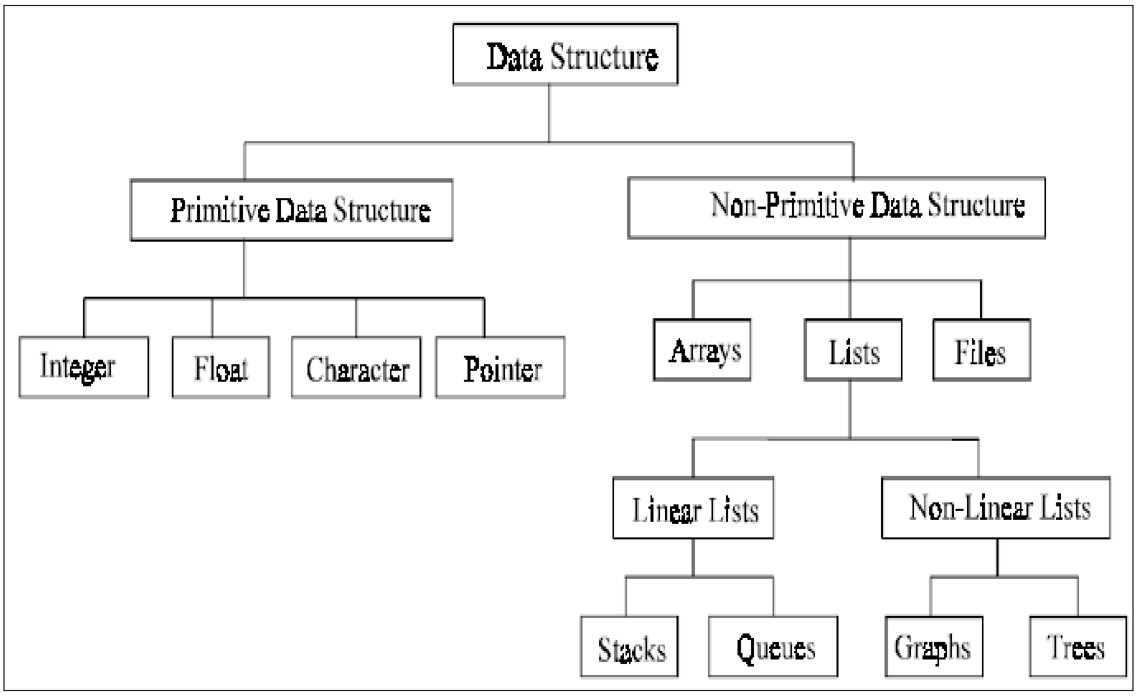
Complete this activity in your Portfolio of Evidence Workbooks.

3.4 COMPUTER DATA STRUCTURES

3.4.1 Types of data structures

Data structure is a way of organising data that considers not only the data items stored but also the relationship to each other.

Figure: Types of data structures



3.4.1.1 Primitive Data Structure

Primitive Data types are those not defined in terms of other data types. The primitive data types of a language, along with one or more type constructors provide structured types.

Numeric Types

Integer

The integer is the most common primitive data type. The integer is almost always an exact reflection of the hardware, so the mapping is trivial. There may be as many as eight different integer types in a language. For example java has four integer types i.e. byte, short, int and long. Some languages use unsigned integers which are integer values without signs and are used for binary data. Integer types are supported by the hardware.

Floating-point

These model real numbers, but are only approximations for most real values. For example TT cannot be represented in floating-point notation. On most computers, floating-point numbers are stored in binary, which exacerbates the problem. Another problem is the loss of accuracy through arithmetic operations.

Languages for scientific use support at least two floating-point types; sometimes more (e.g. **float**, and **double**.) The collection of values that can be represented by a floating-point type is defined in terms of precision and range.

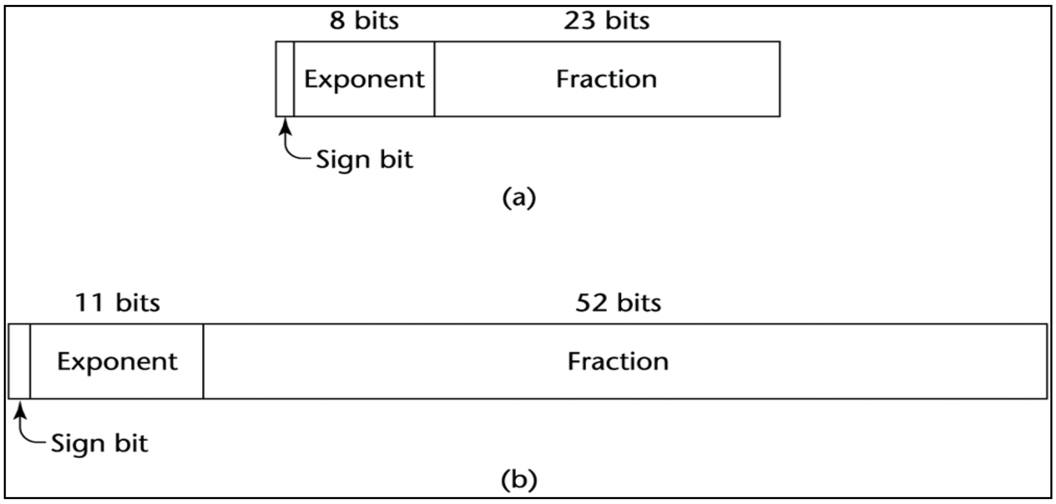
Precision: is the accuracy of the fractional part of a value, measured as the number of bits. Figure below shows single and double precision.

Range: is the range of fractions and exponents.

Decimal

The majority of larger computers that are designed to support business applications have hardware support for decimal data types. Decimal types store a fixed number of decimal digits, with the decimal point at a fixed position in the value.

Figure: Decimal types store



- **Advantage**: accuracy of decimal values.
- **Disadvantages**: limited range since no exponents are allowed, and its representation wastes memory.

Boolean Types

Introduced by ALGOL 60, Boolean types are perhaps the smallest of all data types. Their range of value is made up of only two elements: True and False.

They are used to represent switched and flags in programs, but other values like integers can be used for this purpose. Booleans can be stored in a bit but are usually stored in the smallest efficiently addressable cell of memory usually a byte. The use of Booleans enhances readability.

Character Types

Char types are stored as numeric coding (ASCII / Unicode).

3.4.1.2 Non-Primitive Data Structures

An array

An *array* is a way to reference a series of memory locations using the same name. Each memory location is represented by an array element. An *array element* is similar to one variable except it is identified by an index value instead of a name. An *index* value is a number used to identify an array element.

For example an array called grades. The first array element is called grades[0]. The zero is the index value. The square bracket tells the computer that the value inside the square bracket is an index.

```
grades[0]
grades[1]
grades[2]
```

Each array element is like a variable name. For example, the following variables are equivalent to array elements. There are similarities between array elements and variables—here are three integer variables:

```
int maryGrade;
int bobGrade;
int amberGrade;
```

You store a value into a memory location by using an assignment statement. Here are two assignment statements. The first assigns a value to a variable, and the other assigns a value to an array element. Notice that these statements are practically the same except reference is made to the index of the array element in the second statement:

```
int grades[1];
maryGrade = 90;
grades[0] = 90;
```

Suppose you want to use the value stored in a memory location. There are a number of ways to do this in a program, but a common way is to use another assignment statement like the ones shown in the next example. The first assignment statement uses two variables, the next assignment statement uses two array elements, and the last assignment statement assigns the value referenced by a variable name and assigns that value to an array element:

```
bobGrade = maryGrade;
grades[0] = grades[1];
grades[0] = bobGrade;
```

A stack

When you hear the term “stack” used outside the context of computer programming, you might envision a stack of dishes on your kitchen counter. This organisation is structured in a particular way: the newest dish is on top and the oldest is on the bottom of the stack.

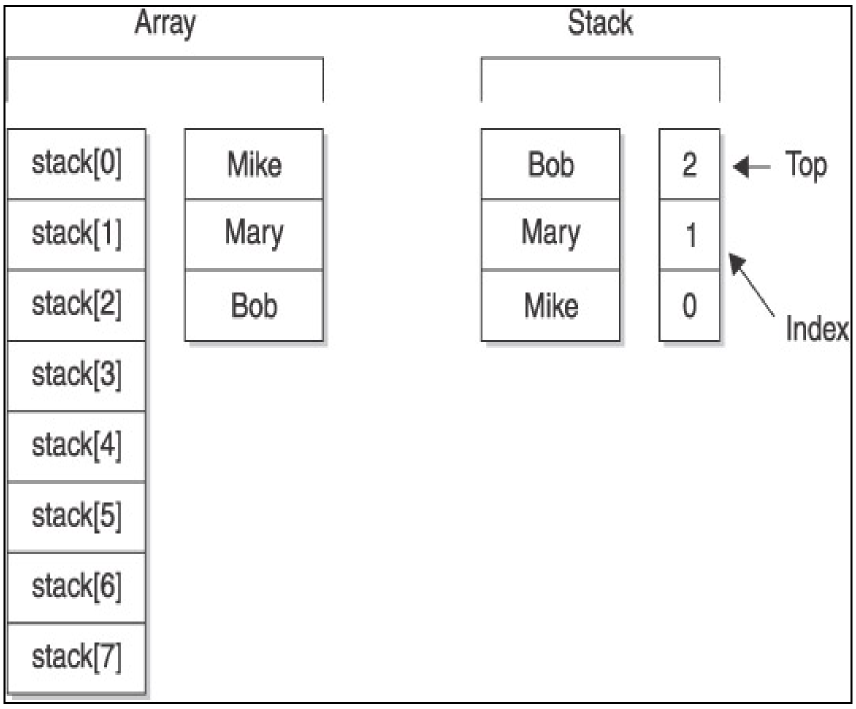
Each dish in a stack is accessed using FIFO: first in, first out. The only way to access each dish is from the top of the stack. If you want the third dish (the third oldest on the stack), then you must remove the first two dishes from the top of the stack. This places the third dish at the top of the stack making it available to be removed.

There's no way to access a dish unless the dish is at the top of the stack. You might be thinking stacks are inefficient, and you'd be correct if the objective was to randomly access things on the stack. There are other data structures that are ideal for random access, which you'll learn about throughout this book.

However, if the object is to access things in the order in which they were placed on the stack, such as computer instructions, stacks *are* efficient. In these situations, using a stack makes a lot of sense.

Note: Stacks and arrays are often bantered about in the same discussion, which can easily lead to confusion, but they are really two separate things. An array stores values in memory; a stack tracks which array element is at the top of the stack. When a value is popped off the stack, the value remains in memory because the value is still assigned to an array element. Popping it only changes the array element that is at the top of the stack.

Figure: Stacks and arrays



A queue

A queue is a sequential organization of data. Data is accessible using FIFO. That is, the first data in the queue is the first data that is accessible by your program.

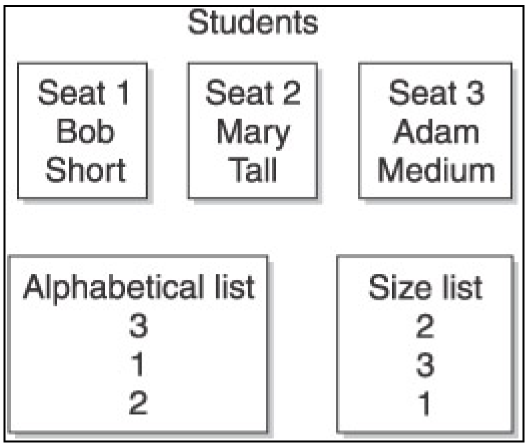
Programmers use one of two kinds of queues depending in the objective of the program, a simple queue or a priority queue. A simple queue organises items in a line where the first item is at the beginning of the line and the last item is at the back of the line. Each item is processed in the order in which it appears in the queue. The first item in line is processed first, followed by the second item and then the third until the last item on the line is processed. There isn't any way for an item to cut the line and be processed out of order.

A priority queue is similar to a simple queue in that items are organized in a line and processed sequentially. However, items on a priority queue can jump to the front of the line if they have priority. Priority is a value that is associated with each item placed in the queue. The program processes the queue by scanning the queue for items with high priority. These are processed first regardless of their position in the line. All the other items are then processed sequentially after high priority items are processed.

A linked list

A linked list is a data structure that makes it easy to rearrange the data without having to move data in memory. Sound a little confusing? If so, picture a classroom of students who are seated in no particular order. A unique number identifies each seat, as illustrated. We've also included the relative height of each student, which we'll use in the next exercise.

Figure: Linked list is a data



Let's say that a teacher needs to place students' names in alphabetical order so she can easily find a name on the list. One option is to have students change their seats so that Adam sits in seat 1, Bob sits in seat 2, and Mary in seat 3. However, this can be chaotic if there are a lot of students in the class.

Another option is to leave students seated and make a list of seat numbers that corresponds to the alphabetical order of students. The list would look something like this: 3, 1, and 2, as shown in illustration. The student in seat 3 is the first student who appears in alphabetical order, followed by the student seated in seat 1, and so on. Notice how this option doesn't disrupt the class.

Suppose you want to rearrange students in size order. There's a pretty good chance that you won't move students about the classroom. Instead, you'd probably create another list of seat numbers that reflect each student's height. Here's the list: 1, 3, and 2, which is illustrated. The list can be read from bottom to top for the shortest to tallest or vice versa for tallest to shortest.

ACTIVITY 10 – REVIEWED – US14944 SO2

Complete this activity in your Portfolio of Evidence Workbooks.

3.4.2 Computer files and file organisation

3.4.2.1 Terminology

Field Each named unit of information to which one can make reference. Examples: Person No, Name, Age etc. Field length may be either fixed (e.g. Person No) or variable (Name). If variable, one may treat it as an adequately long fixed-length field. If an actual value is shorter, the rest of the field may be padded with spaces. If it is too long, one would need to truncate the field. This is possible as long as no essential information is lost in trimming the value. There is some waste of storage space though. One may allow the field to take variable length values by ending each field with an end-of-field character. This eliminates wasted storage but complicates processing.

Record (Logical Record). Is a grouping of fields as viewed by the user. Example: Personnel Record = (Person No, Name, Age, Sex, Date appointed). The key field in a record is the field or the collection of fields which can be used to uniquely identify a record. E.g. Person No in Personnel record. Like fields, records may also have fixed or variable lengths. Fixed-length records comprise a fixed number of fixed-length fields, whereas variable-length records usually comprise a fixed number of fields some of which are of variable length.

File A file is a collection of records, all of the same type, i.e. comprising the same collection of fields.

Block (Physical Record). Is the physical unit of transfer between the backing store and the main store. This is the smallest amount of information that can be transferred. A physical record usually contains more than one whole logical record. The precise number depends on the block size and the logical record size. For example, a 2400 byte block will contain nine 256 byte records. On a magnetic disc, a sector is usually a block.

Bucket Is the logical unit of transfer between the backing storage and main memory. Bucket size is limited by the size of the main memory buffer available for input/output. If an 8k buffer is available, then a bucket may contain as many as three 2400 byte blocks. On a magnetic tape, the physical and the logical units of transfer are usually the same, i.e., a bucket contains precisely one block. On a magnetic disc, a bucket usually contains more than one sector or block.

Hit In a processing run, a record that is required is a hit. For example, if the record for Person No = A111 needs to be updated, it will be hit. The hit rate is the fraction of records which is required for processing.
Hit rate = No of records required / Total records in file.
The kind of file organisation best suited for an application is determined largely by the typical hit rate.

3.4.2.2 File organisation

File organisation is the methodology which is applied to structured computer files. Files contain computer records which can be documents or information which is stored in a certain way for later retrieval. File organisation refers primarily to the logical arrangement of data (which can itself be organised in a system of records with correlation between the fields/columns) in a file system.

The four main kinds of file organization are

i) Serial (Heap or Pile)
ii) Sequential (Ordered)
iii) Random (Direct), and
iv) Indexed sequential.

The main characteristics of these organisations as follows;

i) **Serial files**

Records in a serial file are inserted as they arrive. The file is usually read serially from beginning to end. Searching individual records is difficult because in each case the entire file must be read. On average, one half of the file must be read if the record is present and all of the file must be read if it is not present.
Serial files are usually used for temporary storage of transaction data. They can be constructed on serial or direct access media.

ii) **Sequential files**

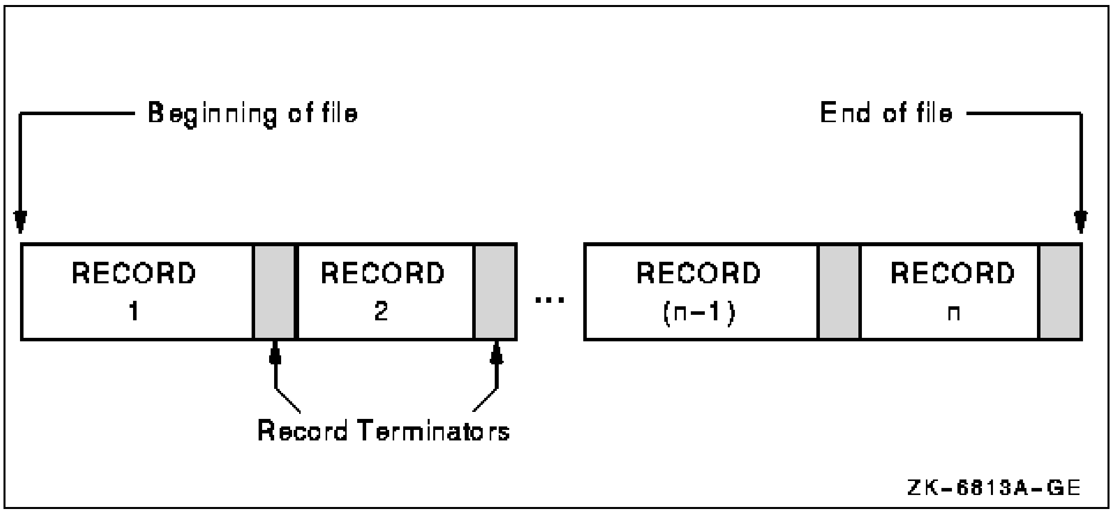
In a sequential file, records are arranged in key value order, usually ascending but it can equally well be descending. The file is usually read in the order in which it is written, from beginning to end. (Although it is possible to read serial and sequential files in other-than-serial order, this would be slow due to mechanical movement involved, irrespective of the nature of the storage medium.

Sequential file organisation is best for high hit rate applications, i.e. when in a typical run a high proportion of the records are required. Sequential files can be constructed on magnetic tape as well as magnetic disc. If constructed on a magnetic tape, searching for individual records is very slow. On average, one must look through half the file whether the record is there or not. In this sequential organisation is only marginally better than serial organisation. If the file is on magnetic disc, searching is much more efficient because one can employ binary search. This is like looking for a word in a dictionary. One probes the file a number of times and at each probe halves the search interval. This process continues till one either finds the required record, or is able to decide that the record is not in the file.

In a file with N buckets, the maximum number of binary probes required to retrieve a record is proportional to N log N, where the logarithm is to base 2. In spite of the much higher efficiency of binary searching, retrieving individual records from a sequential file is not economical because each probe requires mechanical movement and is therefore slow.

The greatest strength of sequential files is that two sequential files which share a key field and are ordered in the same way can be merged very efficiently. In a typical application, a transaction file which is initially built as a serial file is sorted into a sequential file and then used to update an old sequential master file.

Figure: Sequential files



iii) Random or Direct files

The records are not in ascending or descending key value order, but appear to be random. In fact, the storage address for a record (the bucket address) is calculated from the value of the key field by using a **hash** function. (One makes a **hash** of the key to produce a storage address).

Bucket address = Hash(Key).

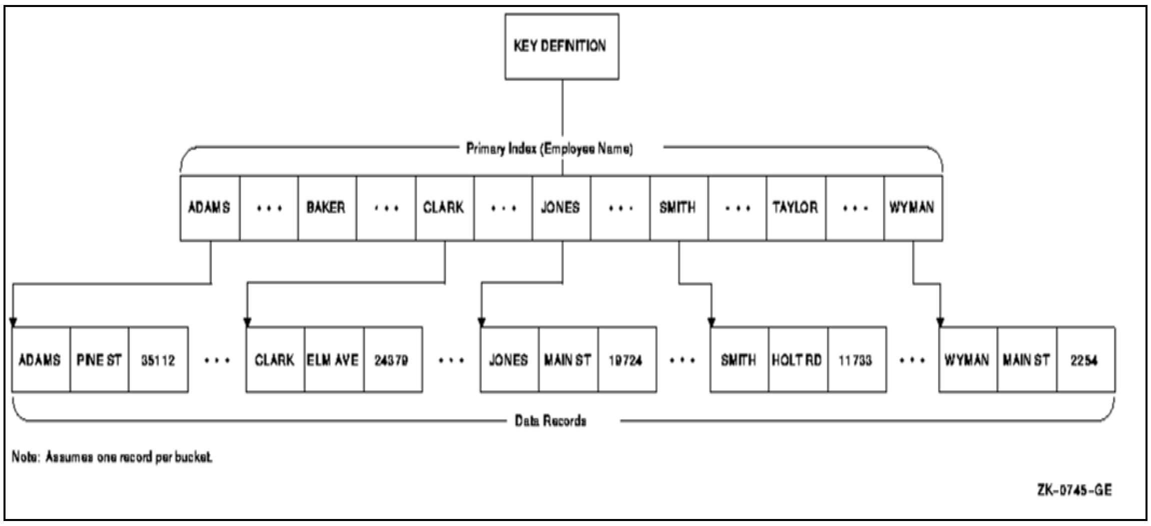
Direct files are created on direct access media. In order to retrieve a record given its key value, one applies the hash function to obtain a bucket address, retrieves the bucket and examines its contents to see if the required record is there. Retrieval of individual records is therefore quick and direct files are ideal for low hit-rate applications. For example, if the balance in a customer's account is to be updated as soon as a transaction is completed (a deposit or a withdrawal is made at the counter), one would organize the Customer Account file as a direct file on a magnetic disc.

iv) Indexed sequential files

Many applications require a mixture of low and high hit rate processing. In these cases, neither sequential nor direct organisation is adequate. Indexed sequential files are ideal for such applications. Because the records are organised sequentially, high hit rate processing is well supported. Direct access of individual records for low hit rate applications is provided by an index.

Consider a file of customer accounts held by a credit card agency. It contains three fields: Acct No, Credit Limit and Balance. Every time a customer wants to make a purchase, his credit limit and balance must be checked; this requires individual access to his record. Every month, a statement must be produced for each customer; this requires sequential access to the whole file.

Figure: Indexed sequential files



Index -- is a special file of records with two attributes, key value and the storage address of the corresponding record in the indexed file. A **dense** index contains an entry for each record in the main file. If the main file is organised sequentially, a dense index is not necessary. A **partial** index containing the highest (or the lowest) key value in each bucket is quite sufficient. The index itself would be organised sequentially.

Ideally, one would like to be able to hold the index in main memory. Then one can search the index quickly (perhaps using binary search which is fast in main memory) to obtain the storage address, and then retrieve the required record in a single disc access. However, if the main file is very large, even a partial index may be too large to be held in main memory. In this case one may consider constructing a multi-level index, i.e. an index to the index which may be small enough to be held in main memory.

The most common form of indexing is **dynamic**. This allows changes to the index as records are added to or deleted from the main file.

An index is usually constructed as a tree. The construction and use of various tree indexes will be taken up later.

3.4.2.3 Types of computer system files

Transaction files. These files represent a group of transactional data, such as sales transactions, waiting posting (updating) to the master files affected by the transaction. Examples would be *sales transaction file, payroll transaction file, collections-on-account transaction file, receiving transaction file.*

Open ***transaction files*** represent in-process transactions that do not yet represent a completed accounting transaction. Examples would be the *open purchase order file*, which represents purchases on order but have not yet been received. When the goods are received, additional data will be entered to reflect that receipt, and the transactions in the open file will move to a completed transaction file that will be posted to all master files impacted by the transactions.

Master files. Represent on-going information pertaining to an entity that is carried over from one period to the next. Examples would be the *accounts receivable master file, inventory master file*, and *general ledger master file*.

History files. Contain transactions that have been previously posted to the master files affected by the transaction. After transactions are posted, they are appended to history files, so that the firm will have an "audit trail" record of all transactions undertaken. These files are used to generate a number of reports analysing past transactions, such as sales analysis reports.

3.4.2.4 Types of computerised processing

Data Entry The process of getting transactional or master file data entered into files in the system. This could consist of keying the data from a terminal, or scanning a bar code or other scanable document. Many web-based businesses are using the customers themselves to enter data pertaining to the sales.

Editing The process of applying control procedures to data. At this point of data entry and subsequently to data entry in an attempt to identify data entry errors or other incorrect data.

Updating This process is the same as posting in a manual system, which simply represents the updating of account balances and other cumulative fields in master files to reflect transactional data.

Appending (merge)the process of adding posted transactions to history files.

Deleting - the process of deleting transactions from transaction files after they have been posted to the master files and appended to the history files. This prepares the transaction file for the next day of activity.

Sorting The process of ordering data in a file based on a specified field or fields

- Sorting of transaction files was required in older sequential access, magnetic tape supported systems, since the transaction file had to be put in the same order as the master file being updated.
- Sorting is now primarily used to arrange data in a desired order for query or report generation.

Report generation The process of generating pre-defined or ad hoc reports from files or combinations of files in the system. The reports made commonly be displayed on the screen or output to a printer.

Maintenance Processing The processing of non-transactional changes in master file (database) records that do not affect the balance in an account. This processing encompasses adding new records to a file, deleting records, making non-transactional changes such as changing the address, marital status, pay rate, etc. of an employee in the employee master file.

ACTIVITY 11 – REVIEWED – US14944 SO 2

Complete this activity in your Portfolio of Evidence Workbooks.

3.5 DATABASE SYSTEMS

A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access, or Excel. This is a collection of related data with an implicit meaning and hence is a database.

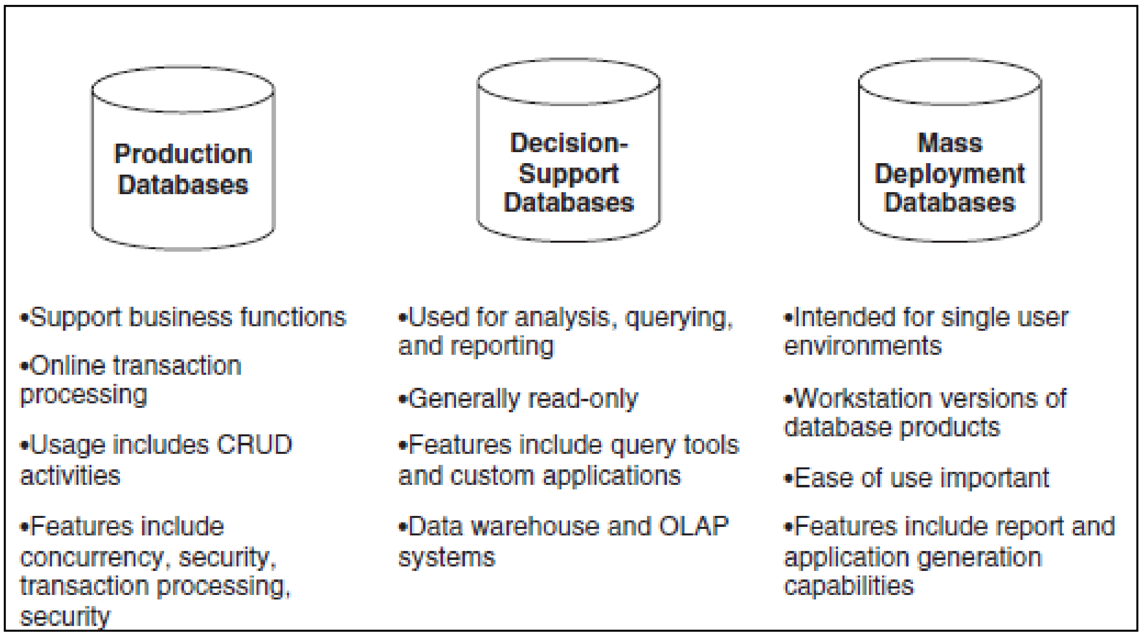
A database may be generated and maintained manually or it may be computerised. For example, a library card catalogue is a database that may be created and maintained manually. A computerised database may be created and maintained either by a group of application programs written specifically for that task or by a database management system. Of course, we are only concerned with computerised databases in this course.

3.5.1 How databases are used

You now realize the use of databases for supporting the core business of an organisation and enabling day-to-day operations. These are production databases that support the operational systems of an enterprise. More recently, with increasing demand for information, databases fulfil another important function. Databases provide support for strategic decision making in an organisation. Such decision support databases are designed and implemented separately and differently. Production databases and decision-support databases are large-scale databases for the several users within organisations.

Individuals and single departments may also use private databases. For example, a specialty department may want to send targeted mailings to specific customers and to keep these customers in a separate database. Individual business analysts may keep data and research results in a separate database just for their use. These are mass deployment individual databases. The figure below shows the separation of databases by their uses and describes some of the features.

Figure: Use of databases


Production Databases
•Support business functions
•Online transaction processing
•Usage includes CRUD activities
•Features include concurrency, security, transaction processing, security
Decision-Support Databases
•Used for analysis, querying, and reporting
•Generally read-only
•Features include query tools and custom applications
•Data warehouse and OLAP systems
Mass Deployment Databases
•Intended for single user environments
•Workstation versions of database products
•Ease of use important
•Features include report and application generation capabilities

3.5.2 Overview of data models

A data model represents the data requirements of an organisation. You can diagrammatically show a data model with symbols and figures. Data for an organisation resides in a database. Therefore, when designing a database, you first create a data model. The model would represent the real-world data requirements. It would show the arrangement of the data structures. Database software has evolved to support different types of data models. As we try to represent real-world data requirements as close as possible in a data model, we come up with a replica of the real-world information requirements. It turns out that we can look at data requirements and create data models in a few different ways. At this stage, let us survey a few leading data models. Over time, different vendors have developed commercial database management systems to support each of these common data models.

3.5.1.1 Flat file

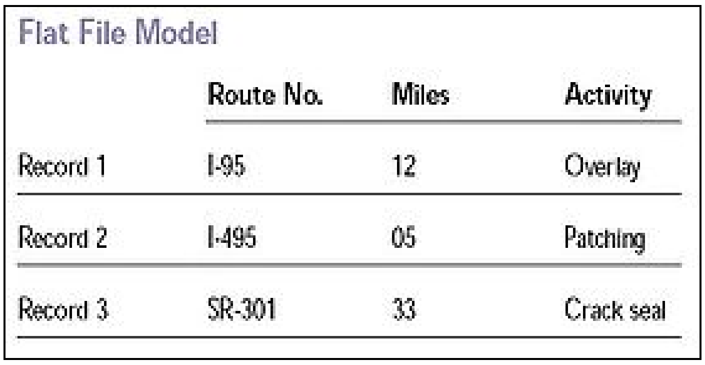
A **flat file database** describes any of various means to encode a database model (most commonly a table) as a singular file (such as .txt or .ini).

A "flat file" is a plain text or mixed text and binary file which usually contain one record per line or 'physical' record (example on disc or tape). Within such a record, the single fields can be separated by delimiters, e.g. commas, or have a fixed length. In the latter case, padding may be needed to achieve this length. Extra formatting may be needed to avoid delimiter collision. There are no structural relationships between the records.

Typical examples of flat files are /etc/password and /etc/group on Unix-like operating systems. Another example of a flat file is a name-and-address list with the fields *Name*, *Address*, and *Phone Number*.

It is possible to write out by hand, on a sheet of paper, a list of names, addresses, and phone numbers; this is a flat file database. This can also be done with any typewriter or word processor. Many pieces of computer software are designed to implement flat file databases.

Figure: Flat file



Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

3.5.1.2 Hierarchical model

Let us examine the data requirements for a typical manufacturing company. Typically in manufacturing, you have major assemblies, with each major assembly consisting of subassemblies, each subassembly consisting of parts, each part consisting of subparts, and so on. In your database for the manufacturing company, you need to keep data for the assemblies, subassemblies, parts, and subparts. And the data model for manufacturing operations must represent these data requirements.

Think about this data model. This model should show that an assembly contains subassemblies, a subassembly contains parts, and a part contains subparts. Immediately you can observe that this data model must be hierarchical in nature, diagramming the assembly at the top with subassembly, part, and subpart at successive lower levels.

In the business world, many data structures are hierarchical in nature. You can notice a hierarchy in department, product category, product subcategory, product line, and product. You can trace a hierarchy in division, subdivision, department, and employee. The next figure illustrates one such model showing the hierarchy of customer, order, and order line item. A customer may have one or more orders, and an order may have one or more line items, perhaps one line item for each product ordered.

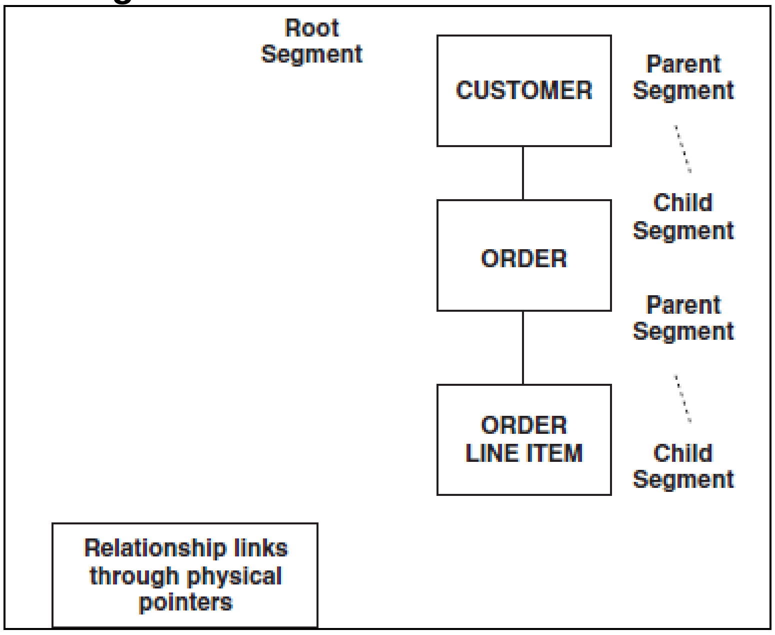
Key features of the hierarchical model

Level: Each data structure representing a business object is at one of the hierarchical levels.

Parent-Child Relationships: The relationship between each pair of data structures at levels next to each other is a parent-child relationship. CUSTOMER is a parent data segment whose child is the ORDER data segment. In this arrangement, a child segment can have only one parent segment but one parent segment may have multiple child segments. You may want to separate orders into phone orders and mail orders. In that case, CUSTOMER may have PHONE ORDER and MAIL ORDER as two child segments.

Root Segment: The data segment at the top level of the hierarchy is known as the root data segment (as in an inverted tree).

Figure: Root Segment



3.5.1.3 Network model

The hierarchical data model represents well any business data that inherently contains levels one below the other. We have just discussed how the manufacturing application deals with hierarchical levels of plant inventory with assemblies broken down into lower-level components. The hierarchical data model suits this application well. However, in the real world, most data structures do not conform to a hierarchical arrangement. The levels of data structures do not fall into nice dependencies one below another as in a hierarchy. In the hierarchical data model, you have noticed that each data segment at any level can have only one parent at the next higher level. In practice, many sets of related elements may not be subjected to such restrictions.

Let us consider a common set of related data elements in a typical business. The data elements pertain to customers placing orders and making payments, salespersons being assigned, and salespersons being part of sales territories. All of these data elements cannot be arranged in a hierarchy. The relationships cross over among the data elements as though they form a network. Refer to the next figure and note how it represents a network arrangement and not a hierarchical arrangement. Observe the six data elements of sales territory, salesperson, customer, order, order line item, and payment as nodes in a network arrangement.

The network data model overcomes some of the limitations of the hierarchical data model. The network data model is more representative of real-world information requirements than the hierarchical model. The network data model can represent most business information.

Key features of the network model

Levels: As in most real-world situations, no hierarchical levels exist in the network model. The lines in a network data model simply connect the appropriate data structures wherever necessary without the restriction of connecting only successive levels as in the hierarchical model. Note the lines connecting the various data structures with no restrictions.

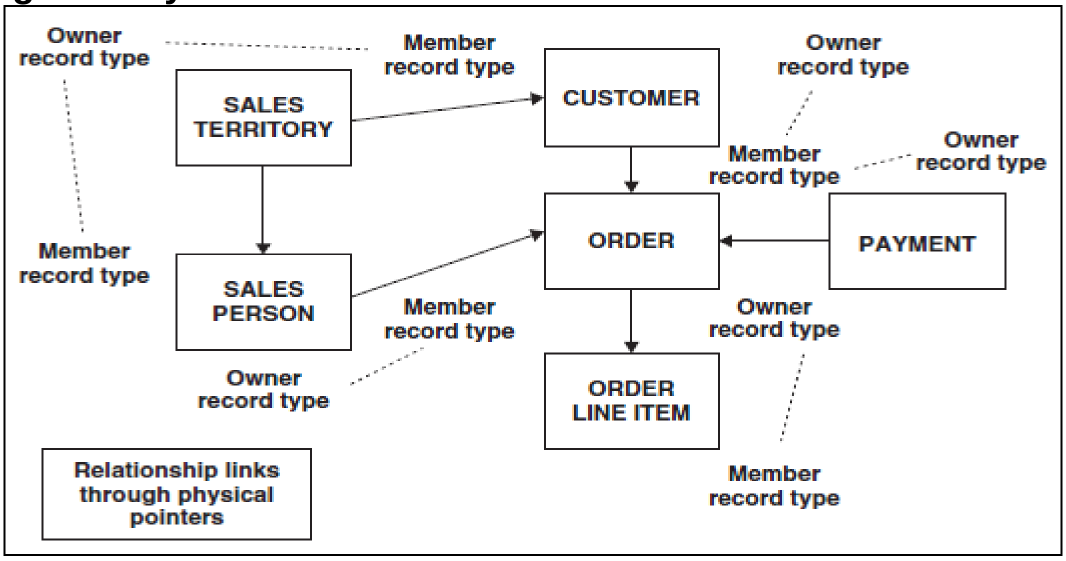
Record Types: In the network data model, each data structure is known as a record type. For example, the CUSTOMER record type represents the data content of all customers. The ORDER record type represents the data content of all orders.

Relationships: The network data model expresses relationships between two record types by designating one as the owner record type and the other as the member record type. For each occurrence of an owner record type, there are one or more occurrences of the member record type. The owner record type may be reckoned as the parent and the member record type as the child. In a sense, the owner record type “owns” the corresponding member record type. Each member type with its corresponding owner record type is known as a set. A set represents the relationship between an owner and a member record type.

Multiple Parents: Look at the ORDER member record type. For ORDER there are two parents or owner records, namely, CUSTOMER and PAYMENT. In other words, for one occurrence of CUSTOMER, one or more occurrences of ORDER exist. Similarly, for one occurrence of PAYMENT there are one or more occurrences of ORDER. By definition, a hierarchical data model cannot represent this kind of data arrangement with two parents for one child data structure.

Physical Pointers: Just as in the case of the hierarchical data model, related occurrences of two different record types in a network model are connected by physical pointers or physical storage addresses embedded within physical records in the database. Physical pointers link occurrences of an owner record type with the corresponding occurrences of the member record type. Within each record type itself the individual occurrences may be linked to one another by means of forward and backward pointers.

Figure: Physical Pointers



3.5.1.4 Relational model

This data model is superior to the earlier models. Dr. E. F. Codd, the celebrated father of the relational model, stipulated the rules and put this model on a solid mathematical foundation. At this stage, however, we want to introduce the relational model as a superior data model that addresses the limitations of the earlier data models.

The earlier hierarchical data model is suitable for data structures that are naturally hierarchical, with each data structure placed at a certain level in the hierarchy.

However, in the business arena, many of the data structures and their relationships cannot be readily placed in a hierarchical arrangement. The network data model evolved to dispense with the arbitrary restriction of the hierarchical model. Nevertheless, in both of these models, you need physical pointers to connect related data occurrences. This is a serious drawback because you have rewrite the physical addresses in the data records every time you reorganise the data, move the data to a different storage area, or change over to another storage medium. The relational model establishes the connections between related data occurrences by means of logical links implemented through foreign keys.

Key features of the relational data model

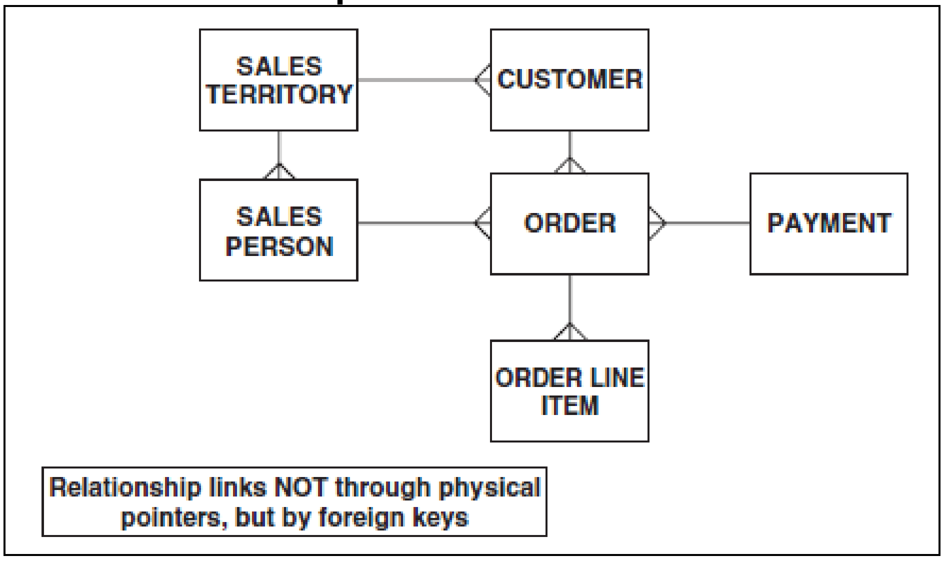
Levels: Just like the network data model, no hierarchical levels are present in the relational model. The lines in a relational data model simply indicate the relationships between the appropriate data structures wherever necessary without the restriction of connecting only successive levels as in the hierarchical model. As in the network model, note the lines connecting the various data structures with no restrictions.

Relations or Tables: The relational model consists of relations. A relation is a two-dimensional table of data observing relational rules. For example, the CUSTOMER relation represents the data content of all customers. The ORDER relation represents the data content of all orders.

Relationships: Consider the relationship between CUSTOMER and ORDER. For each customer one or more orders may exist. So this customer occurrence must be connected to all the related order occurrences. In the relational model, physical pointers do not establish these connections. Instead, a foreign key field is included in the ORDER data structure. In each of the order occurrences relating to a certain customer, the foreign key contains the identification of that customer. When you look for all the orders for a particular customer, you search through the foreign key field of ORDER and find those order occurrences with identification of that customer in the foreign key field.

No Physical Pointers. Unlike the hierarchical or the network data models, the relational model establishes relationships between data structures by means of foreign keys and not by physical pointers.

Figure: Data relationships


SALES TERRITORY
CUSTOMER
SALES PERSON
ORDER
PAYMENT
ORDER LINE ITEM
Relationship links NOT through physical pointers, but by foreign keys

3.5.1.5 Object-oriented model

Object DBMSs add database functionality to object programming languages. They bring much more than persistent storage of programming language objects. A major benefit of object-oriented model is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modelling, and code bases are easier to maintain.

Object developers can write complete database applications with a modest amount of additional effort.
The object-oriented database (OODB) paradigm combines object-oriented programming language (OOPL) systems and persistent systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs.

In contrast to a relational DBMS where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the in-memory structure, object DBMSs have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, World Wide Web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.

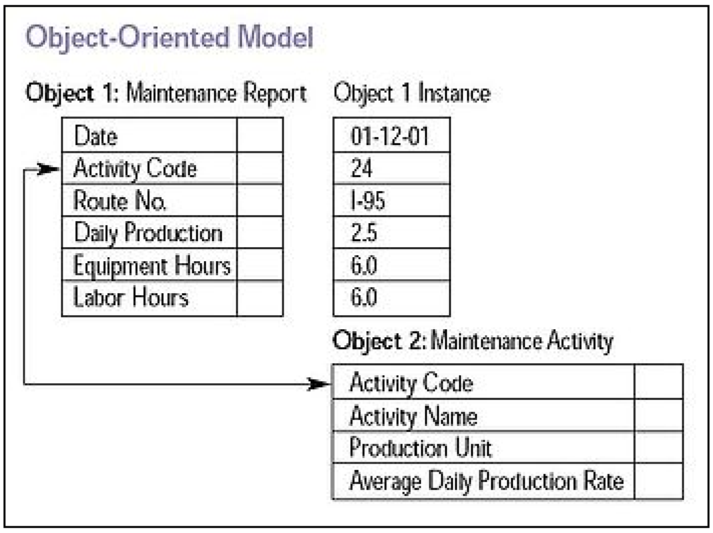
3.5.1.6 Object-relational

Take the case of the State of Minerals which manages the mines in the country. It maintains a library of more than half a million pictures. Users access this library several times a day. A user requests a picture by content: “Show me Anglo Platinum Mine”. Despite an index system of captions and keywords, retrieval of the right picture within a reasonable time is virtually impossible with the current relational database systems. Nor are purely object-oriented data systems totally adequate to handle the challenge.

As the demand for information continues to grow, organizations need database systems that allow representation of complex data types, user-defined sophisticated functions, and user-defined operators for data access.

Object-relational database management systems (ORDBMS) present viable solutions for handling complex data types. The object-relational model combines the ability of object technology to handle advanced types of relationships with features of data integrity, reliability, and recovery found in the relational realm.

Figure: Object-relational



3.5.2 Types of databases

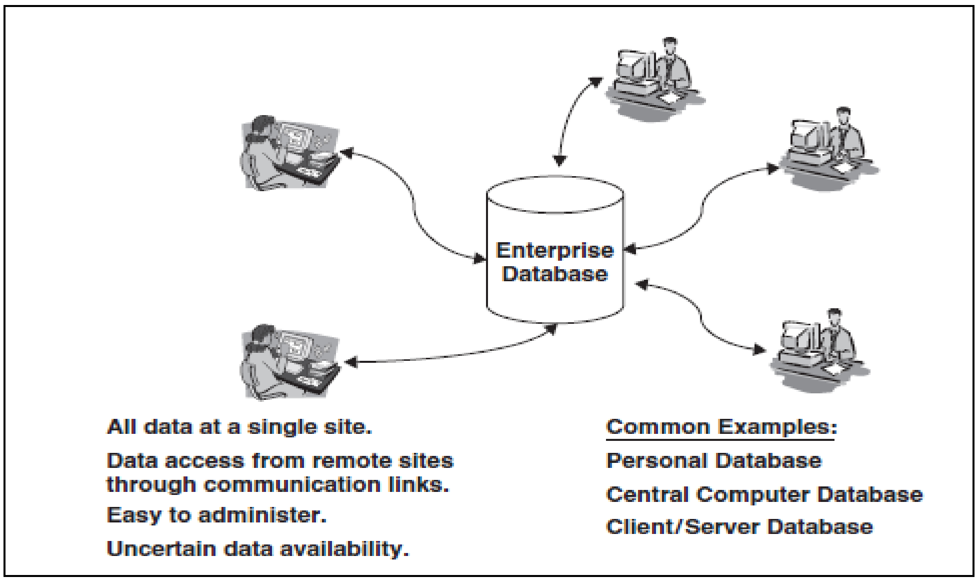
The corporate database that holds an organisation’s data is the underlying foundation for corporate information. Organisations are also faced with questions regarding how and where to hold the corporate data. Where should an enterprise hold its data? Should all the corporate data be kept centrally in one place? If so, what are the advantages and disadvantages? Or should the corporate data be divided into suitable fragments and the pieces kept at different locations? What are the implications of this arrangement?

Organisations primarily adopt one of two approaches. If the entire database is kept in one centralized location, this type of database is a centralised database. On the other hand, if fragments of the database are physically placed at various locations, this type of database is a distributed database. Each type has its own benefits and shortcomings. Again, whether an enterprise adopts a centralised or a distributed approach depends on the organisational setup and the information requirements.

3.5.2.1 Centralised

Personalised databases are always centralised in one location. If your company has a centralised computer system, then the database must reside in that central location. In the client/server architecture, the database resides on a server machine. The entire database may be kept on a single server machine and placed in a central location.

Figure: Centralized database


Enterprise Database
All data at a single site.
Data access from remote sites through communication links.
Easy to administer.
Uncertain data availability.
Common Examples:
Personal Database
Central Computer Database
Client/Server Database

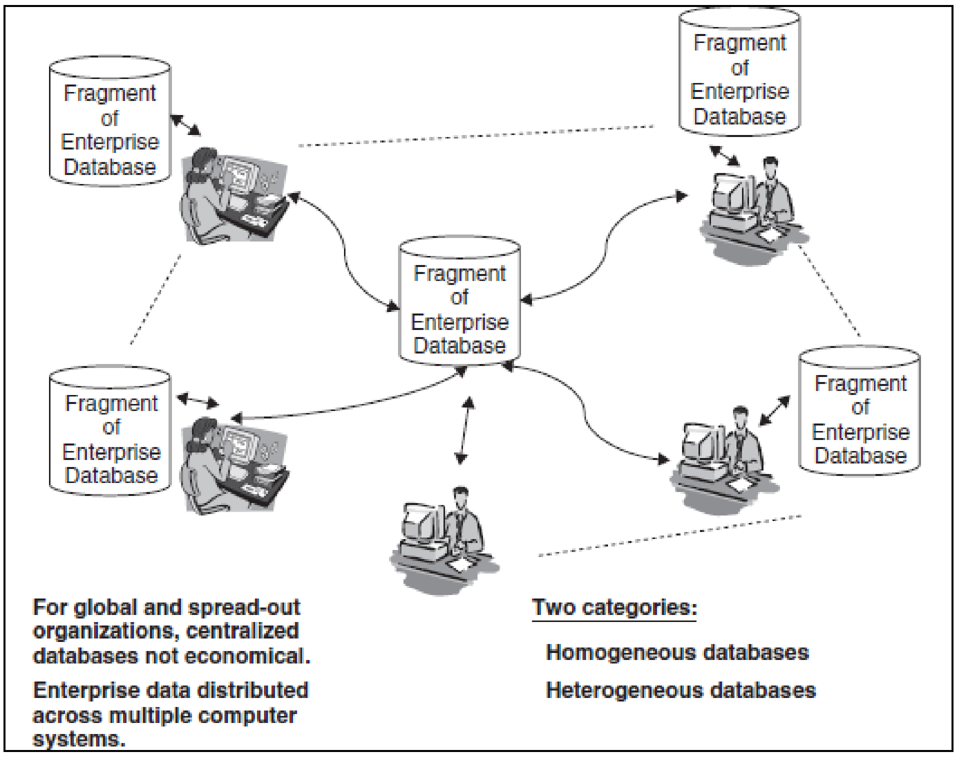
When all corporate data is in one place in a centralised database, companies find it easier to manage and administer the database. You can control concurrent accesses to the same data in the database easily in a centralised database. You can maintain security controls easily. However, if your company's operations are spread across remote locations, these locations must access the centralised database through communication links. Here, data availability depends on the capacity and dependability of the communication links.

3.5.2.2 Distributed

Shows how fragments of a corporate database are spread across remote locations. Global organisations or enterprises with widespread domestic operations can benefit from distributed databases. In such organisations computer processing is also distributed, with processing done locally at each location. A distributed database gets fragmented into smaller data sets. Normally, you would divide the database into data sets on the basis of usage. If a fragment contains data that are most relevant to one location, then that data set is kept at that location. At each location, a fragment of the enterprise data is placed based on the usage.

Each fragment of data at every location may be managed with the same type of database management system. For example, you may run Oracle DBMS at every location. In that case, you run your distributed database as a homogenous database. On the other hand, if you elect to manage the data fragments at different locations with different DBMSs, then you run your distributed database as a collection of heterogeneous database systems. Heterogeneous arrangement provides extra flexibility. However, heterogeneous distribution is difficult to coordinate and administer.

Figure: Distributed database


Fragment of Enterprise Database
Fragment of Enterprise Database
Fragment of Enterprise Database
Fragment of Enterprise Database
Fragment of Enterprise Database
For global and spread-out organizations, centralized databases not economical.
Enterprise data distributed across multiple computer systems.
Two categories:
Homogeneous databases
Heterogeneous databases

3.5.3 SQL

SQL is an industry standard database language that includes statements for database definition, database manipulation, and database control. SQL is based on a mathematical theory. This theory, which consists of a set of concepts and definitions, is called the relational model. The relational model was defined by E. F. Codd in 1970, when he was employed by IBM.

This relational model provides a theoretical basis for database languages. It consists of a small number of simple concepts for recording data in a database, together with a number of operators to manipulate the data. These concepts and operators are principally borrowed from set theory and predicate logic.

SQL is a relational database language. Among other things, the language consists of statements to insert, update, delete, query, and protect data. The following is a list of statements that can be formulated with SQL:

- Insert the address of a new employee.
- Delete all the stock data for product ABC.
- Show the address of employee Johnson.
- Show the sales figures of shoes for every region and for every month.
- Show how many products have been sold in London the last three months.
- Make sure that Mr. Johnson cannot see the salary data any longer.

SQL has already been implemented by many vendors as the database language for their database server. For example, IBM, MySQL, and Oracle are all vendors of SQL products.

We call SQL a relational database language because it is associated with data that has been defined according to the rules of the relational model.

Examples of SQL statements

As you already know by now, SQL is the language used to communicate with database systems. To execute commands on a database, you will compose SQL statements and send to a database or database server. The database system will parse the SQL statement and execute.

What is an SQL statement?

An SQL statement represents a set of commands that can be executed by a database system. Here is a sample SQL statement:

select *name* from *employee* where *age > 20*

The above SQL statement will find the name of all employees from the EMPLOYEE table who's Age is above 20. The words marked in blue colour are keywords. Let us break the statement into 3 sections:

select *name* - SELECT is a keyword and **NAME** represents the database field to select.

from *employee* - FROM is a keyword and **EMPLOYEE** represents the database table to select data from.

where *age > 20* - WHERE a keyword and AGE > 20 is the condition based on which the records are selected from the table.

Basically, the select statement has the following syntax:
SELECT [FIELD 1, FIELD 2, ...] FROM [TABLE NAME] WHERE [CONDITION]

The WHERE condition is optional. If there is no WHERE condition, then the SQL statement will return all the records from the table.

3.5.4 Benefits of databases

Minimal Data Redundancy: Unlike file-oriented data systems where data are duplicated among various applications, database systems integrate all the data into one logical structure. Duplication of data is minimized. Wastage of storage space is eliminated. Going back to the bank example, with a database, customer data is not duplicated in the checking account, savings account, and loan account applications. Customer data is entered and maintained in only one place in the database.

Sometimes, in a database, a few data elements may have to be duplicated. Let us say that product data consist of product number, description, price, and the corresponding product line number. All the fields relating to product line data are kept separately. Whenever the details of products and product lines are needed in applications, both data structures are retrieved from the database. Suppose a heavily used product forecast application needs all the details of the product from product data and just the product line description from the product line data. In that case, it will be efficient for the product data to duplicate the product line description from the product line data. Thus, in some instances, data duplication is permitted in a database for the purpose of access efficiency and performance improvement. However, such data duplications are kept to a minimum.

Data Integrity: Data integrity in a database means reduction of data inconsistency. Because of the elimination or control of data redundancy, a database is less prone to errors creeping in through data duplication. Field sizes and field formats are the same for all applications. Each application uses the same data from one place in the database. In a bank, names and addresses will be the same for checking account, savings account, and loan applications.

Data Integration: In a database, data objects are organised into single logical data structures. For example, in file-oriented data systems, data about employees are scattered among the various applications. The payroll application contains employee name and address, social security number, salary rate, deductions, and so on. The pension plan application contains pension data about each employee, whereas the human resources application contains employee qualifications, skills, training, and education. However, all data about each employee are integrated and stored together in a database.

So, in a database, data about each business object are integrated and stored separately as customer, order, product, invoice, manufacture, sale, and so on. Data integration enables users to understand the data and the relationships among data structures easily. Programmers needing data about a business object can go to one place to get the details. For example, data about orders are consolidated in one place as order data.

Data Sharing: This benefit of database systems follows from data integration. The various departments in any enterprise need to share the company’s data for proper functioning. The sales department needs to share the data generated by the accounting department through the billing application. Consider the customer service department. It needs to share the data generated by several applications. The customer service application needs information about customers, their orders, billings, payments, and credit ratings. With data integration in a database, the application can get data from distinct and consolidated data structures relating to customer, orders, invoices, payments, and credit status.

Data sharing is a major benefit of database systems. Each department shares the data in the database that are most pertinent to it. Departments may be interested in data structures as follows:

Sales department:	*Customer/Order*
Accounting department:	*Customer/Order/Invoice/Payment*
Order processing department:	*Customer/Product/Order*
Inventory control department:	*Product/Order/Stock Quantity/Back Order Quantity*

Database technology lets each application use the portion of the database that is needed for that application. User views of the database are defined and controlled. We will have more to say about user views in later chapters.

Uniform Standards: We have seen that, because of the spread of duplicate data across applications in file-oriented data systems, standards cannot be enforced easily and completely. Database systems remove this difficulty. As data duplication is controlled in database systems and as data is consolidated and integrated, standards can be implemented more easily. Restrictions and business rules for a single data element need to be applied in only one place. In database systems, it is possible to eliminate problems from homonyms and synonyms.

Security Controls: Information is a corporate asset and, therefore, must be protected through proper security controls. In file-oriented systems, security controls cannot be established easily. Imagine the data administrator wanting to restrict and control the use of data relating to employees. In file-oriented systems, control has to be exercised in all applications having separate employee files. However, in a database system, all data about employees are consolidated, integrated, and kept in one place. Security controls on employee data need to be applied in only one place in the database. Database systems make centralised security controls possible. It is also easy to apply data access authorizations at various levels of data.

Data Independence: Remember the lack of data independence in file-oriented systems where computer programs have data structure definitions embedded within the programs themselves. In database systems, file or data definitions are separated out of the programs and kept within the database itself. Program logic and data structure definitions are not intricately bound together. In a client/server environment, data and descriptions of data structures reside on the database server, whereas the code for application logic executes on the client machine or on a separate application server.

Reduced Program Maintenance: This benefit results primarily from data independence in applications. If the customer data structure changes by the addition of a field for cellular phone numbers, then this change is made in only one place within the database itself. Only those programs that need the new field need to be modified and recompiled to make use of the added piece of data. Within limits, you can change programs or data independently.

Simpler Backup and Recovery: In a database system, generally all data are in one place. Therefore, it becomes easy to establish procedures to back up data. All the relationships among the data structures are also in one place. The arrangement of data in database systems makes it easier not only for backing up the data but also for initiating procedures for recovery of data lost because of malfunctions.

ACTIVITY 12 – REVIEWED - US14944 SO 2

Complete this activity in your Portfolio of Evidence Workbooks.

REFERENCE

- Computer hardware and software. by Angela Du Preez, Vaughan Van Dyk, Adrian Cook
- Bernard A. Megrey and Erlend Moksness. Past, Present and Future Trends in the Use of Computers in Fisheries Research. 2009. http://www.pmel.noaa.gov/foci/publications/2009/megr0679.pdf
- http://www.ready.gov/america/_downloads/computerinventory.pdf.
- Data Structures Demystified by Jim Keogh and Ken Davidson, 2004
- Databases Demystified by Andy Oppel 2004
- Character set encoding basics -
- http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=IWS-Chapter03
- A tutorial on character code issues http://www.cs.tut.fi/~jkorpela/chars.html
- Unicode Chart. http://www.ssec.wisc.edu/~tomw/java/unicode.html
- www.wikipedia.org and Wiki answers.
- Wikipedia.org
- Computer hardware and networks By Angela Du Preez, Vaughan Van Dyk, Adrian Cook (Unit1, Unit3)
- http://wapedia.mobi/en/Kernel_%28computers%29

Developed for

MSC EDUCATION HOLDINGS (PTY) LTD

26 Bonza Bay Road,

Beacon Bay,

East London

Phone: (043) 7485778

Fax: (043) 7483610

E-mail: mscho@msccollege.co.za