CSC101 - INTRODUCTION TO COMPUTER SCIENCE

WHAT IS COMPUTER SCIENCE?

Computer Science is defined in different ways by different authors. Wikipedia (http://en.wikipedia.org/wiki/Computer_science) defines computer science as the collection of a variety of disciplines related to computing, both theoretical and practical: theoretical foundations of information and computation, language theory, algorithm analysis and development, implementation of computing systems, computer graphics, databases, data communications, etc. Another definition says: Computer science is the scientific and practical approach to computation and its applications. It is the systematic study of the feasibility, structure, expression, and mechanization of the methodical processes (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information, whether such information is encoded as bits in a computer memory or transcribed engines and protein structures in a human cell.

The US National Coordination Office for *Networking and Information Technology Research and Development* (NITRD) defines computer science in a similarly broad way: the systematic study of computing systems and computation. The body of knowledge resulting from this discipline contains theories for understanding computing systems and methods; design methodology, algorithms, and tools; methods for the testing of concepts; methods of analysis and verification; and knowledge representation and implementation. (http://www.nitrd.gov/pubs/bluebooks/1995/section.5.html)

Another broad definition comes from the *Association for Computing Machinery* (ACM) Model Curriculum. It says that computer science is the "study of computers and algorithmic processes, including their principles, their hardware and software design, their applications, and their impact on society."

A famous definition of computer science by Gibbs and Tucker (Gibbs and Tucker, "A Model Curriculum for a Liberal Arts Degree in Computer Science," *Comm. of the ACM*, vol. 29, no. 3, March 1986) emphasizes algorithm development and analysis as the central focus of computer science.

How is computer science a science? In contrast to physics, biology, and chemistry, computer science is not based on the study of the natural world. In that sense, computer science is more like mathematics than science. Some argue that computer

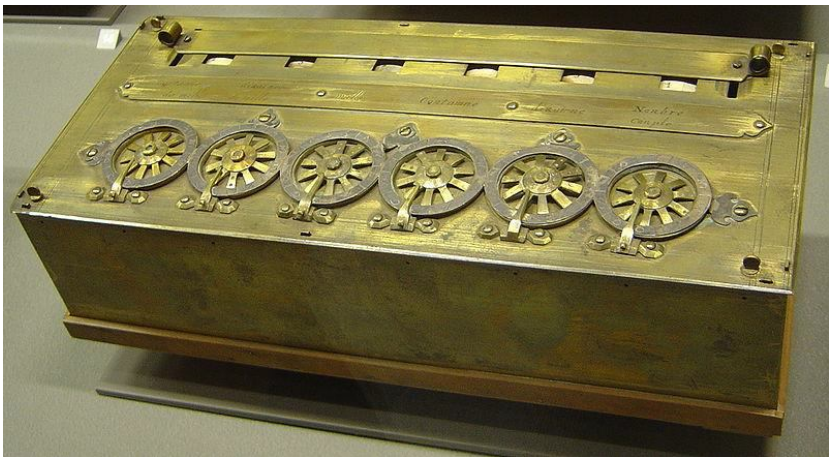
science is really computer art (where "art" means practice). On the other hand, computer scientists do use the scientific method to propose and test hypotheses, and some very non-obvious discoveries in computer science have important real-world implications.

Despite many variations, essentially all definitions of computer science emphasize the study of algorithms. Algorithms, in one form or another, are central to computer science. Computer science combines the theoretical concepts of algorithm design and analysis with the practical considerations of how to implement algorithms on a computer and solve practical problems. An algorithm defines a detailed and unambiguous sequence of actions for solving a particular problem or for performing some task. If you have ever followed a recipe when cooking, followed a set of driving directions, or filled out an income tax form, you have worked with an algorithm.

HISTORY OF COMPUTERS

From the earliest times the need to carry out calculations has been developing. The first steps involved the development of counting and calculation aids such as the counting board and the abacus.

Pascal (1623-62) was the son of a tax collector and a mathematical genius. He designed the first **mechanical calculator** (Pascaline) based on gears. It performed addition and subtraction.



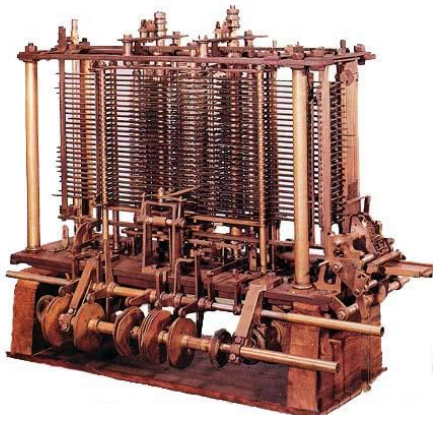
1642: Blaise Pascal's Pascaline

Leibnitz (1646-1716) was a German mathematician and built the first calculator to do multiplication and division. It was not reliable due to accuracy of contemporary parts.

1694: Gottfried Leibniz's mechanical calculator

Babbage (1792-1872) was a British inventor who designed an '**analytical engine**' incorporating the ideas of a *memory* and *card input/output* for data and instructions. Again the current technology did not permit the complete construction of the machine. Babbage is largely remembered because of the work of **Augusta Ada** (Countess of Lovelace) who was probably the first computer programmer.

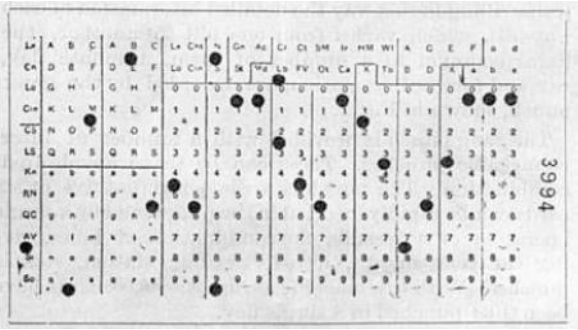
Difference Engine
To solve polynomial equations



Analytical Engine
General purpose machine, precursor to the computer

Burroughs (1855-98) introduced the first commercially successful mechanical adding machine of which a million were sold by 1926.

Hollerith developed an electromechanical punched-card **tabulator** to tabulate the data for 1890 U.S. census. Data was entered on punched cards and could be sorted according to the census requirements. The machine was powered by **electricity**. He formed the Tabulating Machine Company which became **International Business Machines** (**IBM**). IBM is still one of the largest computer companies in the world.



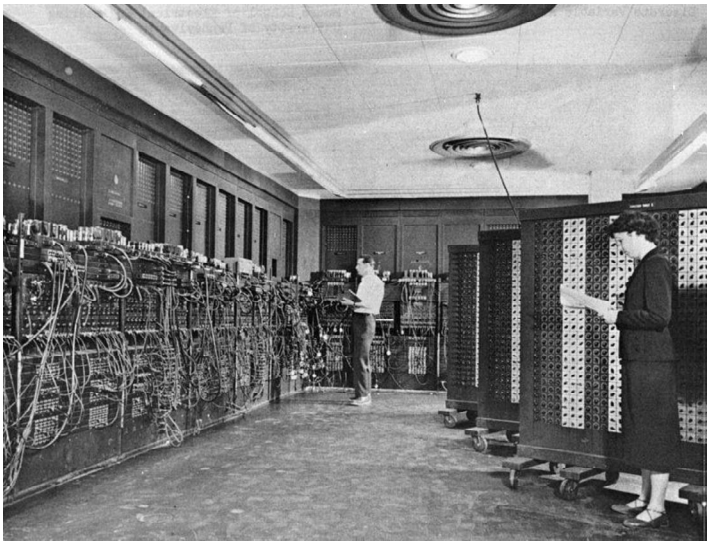
1890: Hollerith's Census Machines

Aiken (1900-73) a Harvard professor with the backing of IBM built the **Harvard Mark I** computer (51ft long) in 1944. It was based on relays (operate in milliseconds) as opposed to the use of gears. It required 3 seconds for a multiplication.

1944: Harvard Mark I

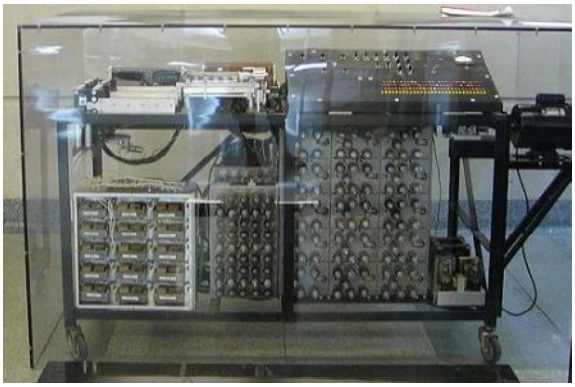
Eckert and **Mauchly** designed and built the **ENIAC** in 1946 for military computations. It used vacuum tubes (valves) which were completely electronic (operated in microseconds) as opposed to the relay which was electromechanical. It weighed 30 tons, used 18000 valves, and required 140KW of power. It was 1000 times faster than the

Mark I multiplying in 3 milliseconds. ENIAC was a decimal machine and could not be programmed without altering its setup manually.



1946: ENIAC

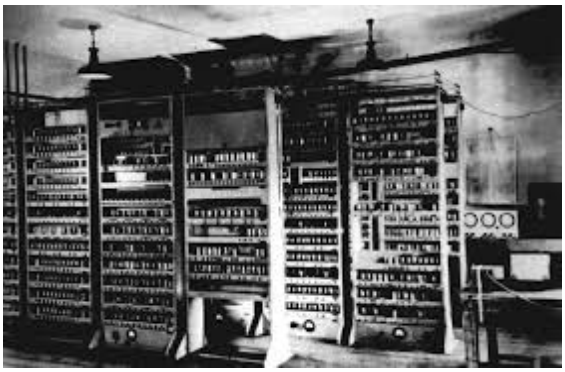
Atanasoff had built a specialised computer in 1941 and was visited by Mauchly before the construction of the ENIAC. He sued Mauchly in a case which was decided in his favour in 1974.



1939: Atanasoff-Berry Computer

Von Neumann was a scientific genius and was a consultant on the ENIAC project. He formulated plans with Mauchly and Eckert for a new computer (EDVAC) which was to store programs as well as data. This is called the **stored program concept** and Von Neumann is credited with it. Almost all modern computers are based on this idea and are referred to as **von neumann machines**. He also concluded that the **binary** system

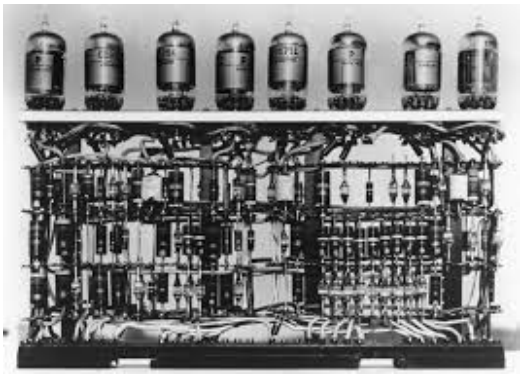
was more suitable for computers since switches have only two values. He went on to design his own computer at Princeton which was a general purpose machine.

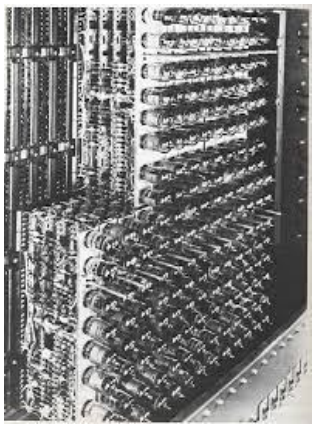


EDVAC

Alan Turing was a British mathematician who also made significant contributions to the early development of computing, especially to the **theory of computation**. He developed an abstract theoretical model of a computer called a **Turing machine** which is used to capture the notion of **computable** i.e. what problems can and what problems cannot be computed. Not all problems can be solved on a computer.
Note: A Turing machine **is an abstract model** and not a physical computer. From the 1950's, the computer age took off in full force. The years since then have been divided into periods or **generations** based on the technology used.

First Generation Computers (1951-58): Vacuum Tubes
These machines were used in business for accounting and payroll applications. **Valves** were unreliable components generating a lot of heat (still a problem in computers). They had very limited memory capacity. **Magnetic drums** were developed to store information and **tapes** were also developed for secondary storage. They were initially programmed in **machine language** (binary). A major breakthrough was the development of **assemblers** and **assembly language**.

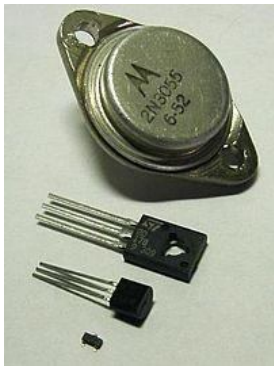




Vacuum Tubes Computers

Second Generation (1959-64): Transistors

The development of the **transistor** revolutionised the development of computers. Invented at Bell Labs in 1948, transistors were much smaller, more rugged, cheaper to make and far more reliable than valves. Core memory was introduced and disk storage was also used. The hardware became smaller and more reliable, a trend that still continues. Another major feature of the second generation was the use of **high-level** programming languages such as **Fortran** and **Cobol**. These revolutionised the development of software for computers. The computer industry experienced explosive growth.

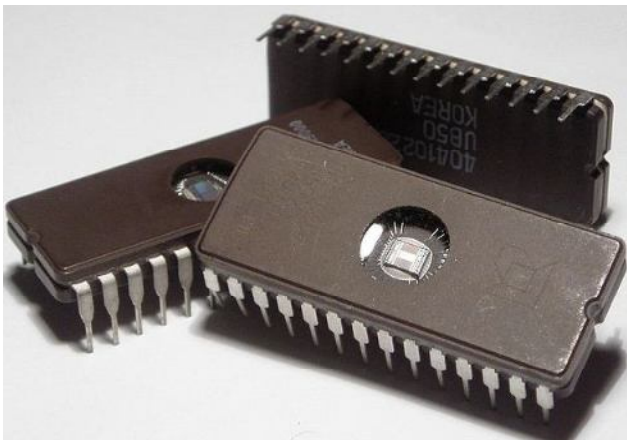


Transistors

Third Generation (1965-71): Integrated Circuits (ICs)

IC's were again smaller, cheaper, faster and more reliable than transistors. Speeds went from the microsecond to the nanosecond (billionth) to the picosecond (trillionth) range. ICs were used for main memory despite the disadvantage of being **volatile**. Minicomputers were developed at this time. **Terminals** replaced punched cards for

data entry and disk packs became popular for secondary storage. IBM introduced the idea of a **compatible** family of computers, 360 family, easing the problem of upgrading to a more powerful machine. Substantial **operating systems** were developed to manage and share the computing resources and **time sharing** operating systems were developed. These greatly improved the efficiency of computers. Computers had by now pervaded most areas of business and administration. The number of transistors that be fabricated on a chip is referred to as the **scale of integration** (**SI**). Early chips had **SSI** (small SI) of tens to a few hundreds. Later chips were **MSI** (Medium SI): hundreds to a few thousands,. Then came **LSI** chips (Large SI) in the thousands range.



1958: Integrated Circuit

Fourth Generation (1971 -): VLSI (Very Large SI)

VLSI allowed the equivalent of tens of thousands of transistors to be incorporated on a single chip. This led to the development of the **microprocessor** a processor on a chip.

Intel produced the 4004 which was followed by the 8008, 8080, 8088 and 8086 etc. Other companies developing microprocessors included **Motorolla** (6800, 68000), **Texas Instruments** and **Zilog**. Personal computers were developed and IBM launched the IBM PC based on the 8088 and 8086 microprocessors. Mainframe computers have grown in power. Memory chips are in the megabit range. VLSI chips had enough transistors to build 20 ENIACs. Secondary storage has also evolved at fantastic rates with storage devices holding gigabytes (1000Mb = 1 Gb) of data. On the software side, more powerful operating systems are available such as **Unix**. Applications software has become cheaper and **easier to use**. Software development techniques have vastly improved.

Fourth generation languages 4GLs make the development process much easier and faster. [Languages are also classified according to generations from machine language (1GL), assembly language (2GL), high level languages (3GL) to 4Gls]. Software is often developed as **application packages**. VisiCalc, a spreadsheet program, was the pioneering application package and the original **killer application**. **Killer application** is piece of software that is so useful that people will buy a computer to use that application.

Fourth Generation Continued (1990s): ULSI (Ultra Large SI)

ULSI chips have millions of transistors per chip e.g. the original Pentium had over 3 million and this has more than doubled with more recent versions. This has allowed the development of far more powerful processors.

The Future

Developments are still continuing. Computers are becoming faster, smaller and cheaper. Storage units are increasing in capacity.

Distributed computing is becoming popular and **parallel** computers with large numbers of CPUs have been built. The **networking** of computers and the convergence of computing and communications is also of major significance.

More areas of computing science are**: Smart clothing, Brain-powered prosthesis, Gesture recognition, Quantum computers and the Singularity**?

From Silicon to CPUs!

One of the most fundamental components in the manufacture of electronic devices, such as a CPU or memory, is a **switch**. Computers are constructed from thousands to millions of switches connected together. In modern computers, components called **transistors** act as electronic switches. A brief look at the history of computing reveals a movement from mechanical to electromechanical to electronic to solid state electronic components being used as switches to construct more and more powerful computers as illustrated below:

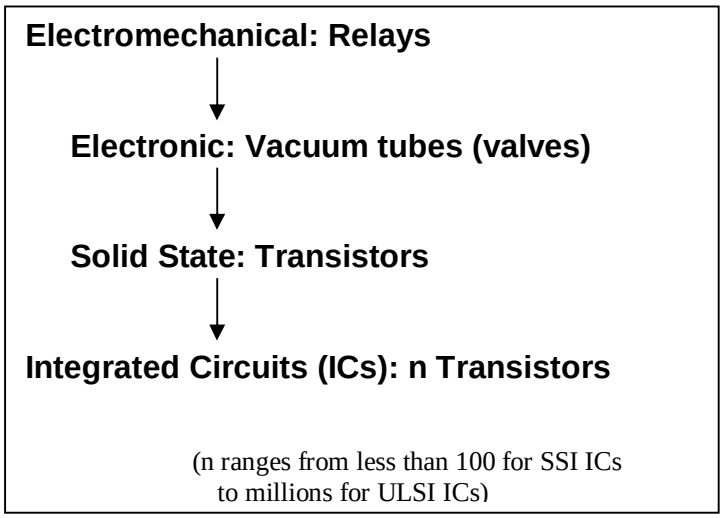


Figure 1: Evolution of switching technology

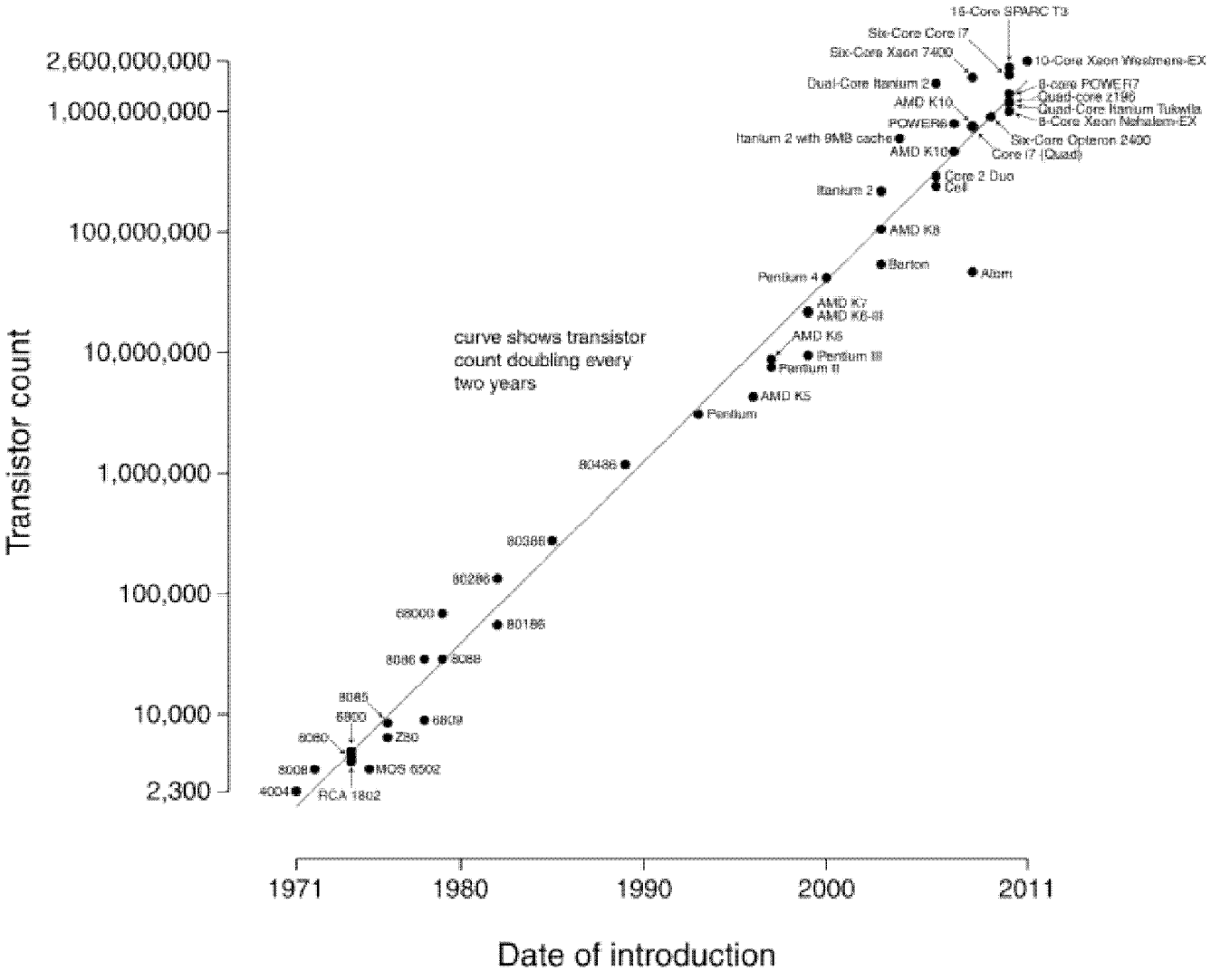
Transistors act as electronic switches, i.e. they allow information to pass or not to pass under certain conditions. The development of **integrated circuits** (ICs) allowed the construction of a number of transistors on a single piece of silicon (the material out of which IC's are made).

IC's are also called **silicon chips** or simply **chips**. The number of transistors on a chip is determined by its **level of integration**

No. of Transistors	Integration level	Abbreviation	Example
2-50	small-scale integration	SSI	
50-5000	medium-scale integration	MSI	
5000-100000	large scale integration	LSI	Intel 8086 (29,000)
100K-10Million	very large scale	VSLI	Pentium (3 million)
10Million – 1000 million	ultra large scale integration	ULSI	Pentium III (30 million)
1000 million -	super large scale integration	SLSI	

Moore's law is the observation that, over the history of computing hardware, the number of transistors on integrated circuits doubles approximately every two years. The law is named after Intel co-founder Gordon E. Moore, who described the trend in his 1965 paper. His prediction has proven to be accurate, in part because the law is now used in the semiconductor industry to guide long-term planning and to set targets for research and development. Summarily,

The capabilities of many digital electronic devices are strongly linked to Moore's law: ***processing speed***, ***memory capacity***, ***sensors*** and even the ***number and size of pixels in digital cameras***. All of these are improving at roughly exponential rates as well. This exponential improvement has dramatically enhanced the impact of digital electronics in nearly every segment of the world economy. Moore's law describes a driving force of technological and social change in the late 20th and early 21st centuries.



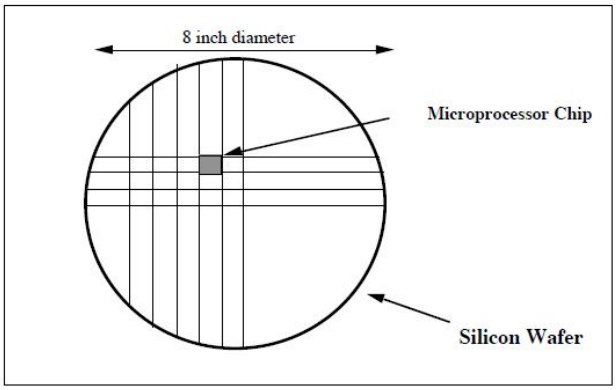
Microprocessor Transistor count 1971 – 2011 and Moore's Law

Summarily**, the number of transistors on an IC doubles every 18 months** (about two years)**.** (Gordon Moore, chairman of Intel at the time 1965). This prediction has proved very reliable to date and it seems likely that it will remain so over the next??? years.

Chip Fabrication

Silicon chips have a surface area of similar dimensions to a thumb nail (or smaller) and are three dimensional structures composed of microscopically thin layers (perhaps as many as 20) of insulating and conducting material on top of the silicon. The manufacturing process is extremely complex and expensive.

Silicon is a **semiconductor** which means that it can be altered to act as either a conductor allowing electricity to flow or as an insulator preventing the flow of electricity. Silicon is first processed into circular wafers and these are then used in the fabrication of chips. The silicon wafer goes through a long and complex process, which results in the circuitry for a semiconductor device such as a microprocessor or RAM being developed on the wafer. It should be noted that each wafer contains from several to hundreds of the particular device being produced. Figure 3 illustrates an 8-inch silicon wafer containing microprocessor chips.



A single silicon wafer can contain a large number of microprocessors

The percentage of functioning chips is referred to as the **yield** of the wafer. Yields vary substantially depending on the complexity of the device being produced, the feature size used and other factors. While manufacturers are slow to release actual figures, yields as low as 50% are reported and it is accepted that 80-90% yields are very good.
A single short circuit, caused by two wires touching in a 30 million plus transistor chip, is enough to cause chip failure!

Feature Size

The **feature size** refers to the size of a transistor or to the width of the wires connecting transistors on the chip. One micron (one thousandth of a millimeter) **was** a common feature size.

State of the art chips are using **sub-micron** feature sizes from **0.25 (1997) to 0.13 (2001) (250-130 nanometers).**

***The smaller the feature size, the more transistors there are available on a given chip area*.**

This allows more microprocessors for example to be obtained from a single silicon wafer. It also means that a given microprocessor will be smaller, runs faster and uses less power than its predecessor using a larger feature size. Since more of these smaller chips can be obtained from a single wafer, each chip will cost less which is one of the reasons for cheaper processor chips.

In addition, reduced feature size it makes it possible to make more complex microprocessors, such as the **Pentium III which uses around of 30 million transistors**.

Die Size

An obvious way to increase the number of transistors on a chip is to increase the area of silicon used for each chip - the **die size**. A **die** in the context of integrated circuits is a small block of *semiconducting* material, on which a given functional circuit is fabricated. Typically, integrated circuits are produced in large batches on a single wafer of **e**lectronic-**g**rade **s**ilicon (EGS) or other semiconductor (such as **GaAs** - **Gallium arsenide**, a compound of the elements *gallium* and *arsenic*) through processes such as *photolithography*. The wafer is cut ("diced") into many pieces, each containing one copy of the circuit. Each of these pieces is called a **die**.

However, this can lead to problems. Assume that a fixed number faults occur randomly on the silicon wafer illustrated in the figure above, a single fault will render an individual chip useless.

The larger the die size for the individual chip, the greater the waste in terms of area of silicon, when a fault arises on a chip.

For example, if a wafer were to contain 40 chips and ten faults occur randomly, then up to 10 of the 40 chips may be useless giving up to 25% wastage.

On the other hand, if there are 200 chips on the wafer, we would only have 5% wastage with 10 faults. Hence, there is a trade-off between die size and yield, i.e. a larger die size leads to a decrease in yield.

BASIC COMPONENTS OF A DIGITAL COMPUTER

Introduction

Recent Advanced research by microelectronics research society, Computer is an essential part of our society. Each day of our lives are start with digital alarm clock or mobile, drive car or any other vehicle in digital processor controlled automobiles and speedometers .work in extensively automated offices. in market all prices are tagged with a digital system which can store the product information and price of the product. Computer is used in every productive and non productive field. A great majority of the computers of our daily use are known as general purpose machines, other are special purpose machines is an important for solve a specific task, i.e. scientist use machine for reducing the complexity of their research work.

Computer is a combination of electronics and mechanical, so it is an electro mechanical device. The technological revolution witnessed in the computer industry is the result of a long chain of enormous and successful efforts made by two major forces. These are the academia, and the industry, represented by computer companies. This section describing the architecture of general purpose digital computer system.

Computer System Architecture

Basic function of computer is process data input by user and output or store the result of the processed data, for better used in specific application, so computer architecture is divided in four part input, output control and storage.

For example, controlling the room temperature system sensor sense the current room temperature, this is input of the system. Control temperature is output of system and if the temperature is set at 18 degree centigrade for lowest, system sensor control the compressor and turn off the compressor when room temperature is 18, it is the control part of system.

The computer revolution continues. Each time the cost of computing improves by another factor of 10, the opportunities for computers multiply. Applications that were economically infeasible suddenly become practical. In the recent past, the following applications were "computer science fiction."

• **Automatic teller machines:** A computer placed in the wall of banks to distribute and collect cash would have been a ridiculous concept in the 1950s, when the cheapest computer cost at least $500,000 and was the size of a car.

• **Computers in automobiles:** Until microprocessors improved dramatically in price and performance in the early 1980s, computer control of cars was ludicrous. Today, computers reduce pollution and improve fuel efficiency via engine controls and increase safety through the prevention of dangerous skids and through the inflation of air bags to protect occupants in a crash.

• **Laptop computers:** Who would have dreamed that advances in computer systems would lead to laptop computers, allowing students to bring computers to coffeehouses and on airplanes?

• **Human genome project:** The cost of computer equipment to map and analyze human DNA sequences is hundreds of millions of dollars. It's unlikely that anyone would have considered this project had the computer costs been 10 to 100 times higher, as they would have been 10 to 20 years ago.

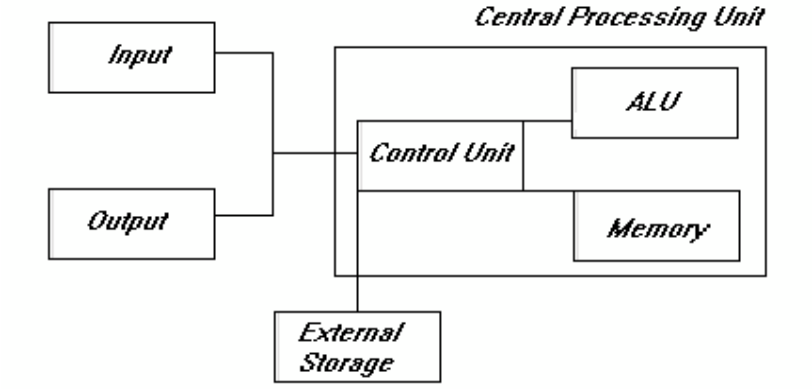
• **World Wide Web:** Not in existence at the time of the first edition of this book, the World Wide Web has transformed our society. Among its uses are distributing news, sending flowers, buying from online catalogues, taking electronic tours to help pick vacation spots, finding others who share your esoteric interests, and even more mundane topics like finding the lecture notes of the authors of your textbooks.

General purpose computer is a supplementary combination of Hardware and Software. There are two basic components of computer system architecture

A. Computer Hardware.
B. Computer Software.

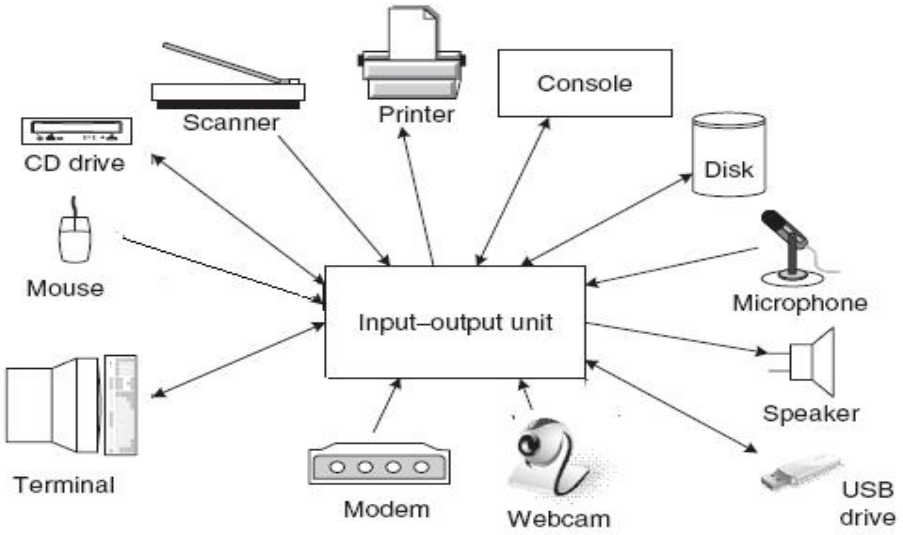
A. Computer Hardware

All above application of computers are use special purpose computer hardware, which contains less hardware components, General purpose computer containing a four basic hardware blocks which are memory unit (MU), arithmetic and logic unit (ALU), input/output unit (IOU), and control unit (CU). Input/output (I/O) devices input and output data into and out of the memory unit. In some systems, I/O devices send and receive data into and from the ALU rather than the MU. Programs reside in the memory unit. The ALU processes the data taken from the memory unit (or the ALU) and stores the processed data back in the memory unit (or the ALU). The control unit coordinates the activities of the other three units. It retrieves instructions from programs resident in the MU, decodes these instructions, and directs the ALU to perform corresponding processing steps. It also oversees I/O operations.



Computer hardware architecture

1. I/O Unit: The input unit contains the hardware devices those are used to enter the data in to computer system, Keyboard and mouse are most common devices. The output contain the hardware devices those are use to output the data from the computer system, Monitor and printer are most common output devices. Nowadays so many other I/O devices are shown in the figure below.

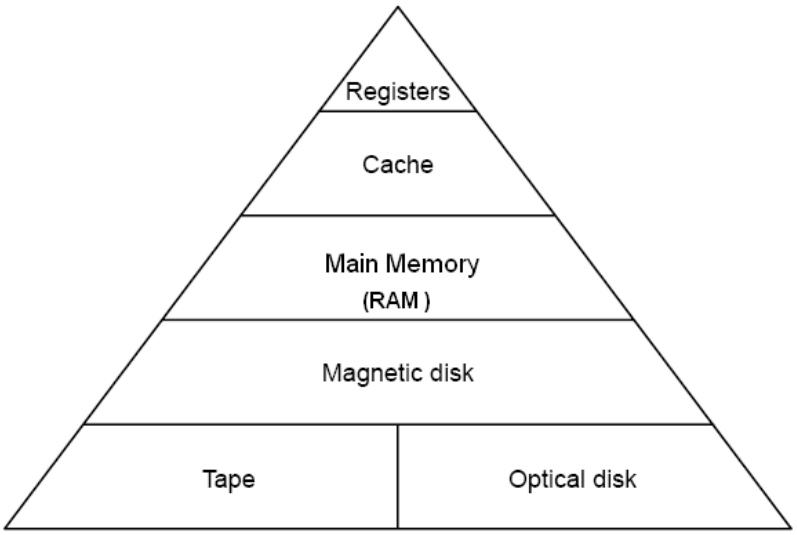


Input and output devices

2. Control Unit: The control unit maintains the sequence of the operation, controlling the actions of all other units it can perform the instruction which constitute the program and direct as per the operation perform by machine.

3. Arithmetic and logic Unit: The ALU unit functions are perform arithmetical operation, i.e. addition, subtraction, multiplication, division as well as logical operation i.e. and, or, not. The control unit gives the instruction to the ALU which operations they have to perform and where it supplied (store purpose), it is directed to perform the operations.

4. ***Memory Unit:*** memory unit or storage section of the computer consists of the devices used to store the data or information during the process. Memory unit is used to hold intermediate and final result of computer program. There are various types of memory which are used in computer system are as given below. In this structure, the level of importance of the memory is of top-down approach as depicted in the figure.

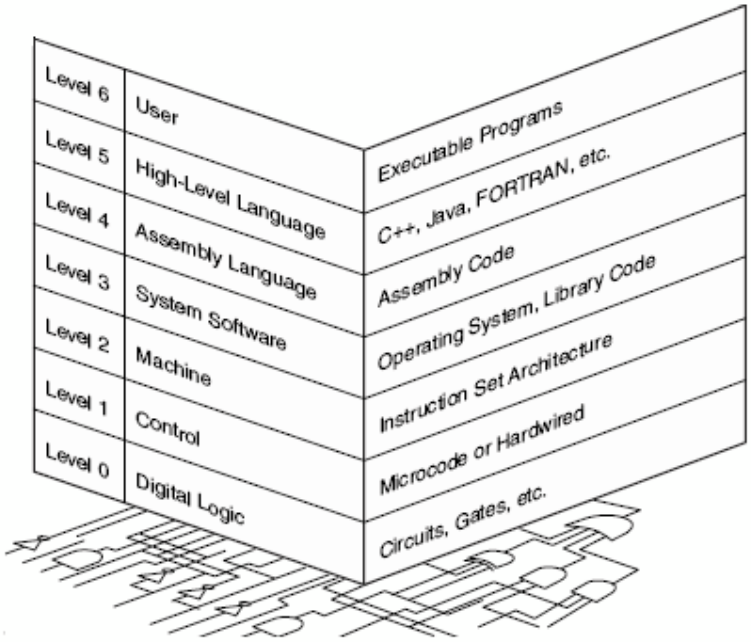


B. Computer Software:

If a machine is to be capable of solving a wide range of problems, it must be able to execute programs written in different high level languages, like 'C' or 'C++', while a the only physical components we have to work with are wires and gates. There is a difficult gap between these physical components and high level languages. For a system to be practical, the semantic gap must be invisible to most of the users of the system.

As per the practical rules that if the problem is large then it should be break it down and use "divide and conquer" thoughts. In programming approach, we divide a problem into modules and then design each module separately. Each module performs a specific task and modules need only know how to interface with other modules to make use of them.

Computer system organization can be approached in hierarchy of levels, in which each level has a specific function and exists as a distinct hypothetical machine. By studying computer organization, these layers are implemented and interface with each other.



Level 6, User Level: is composed of applications and is the level with which everyone is most familiar. At this level, we run programs such as word processors, graphics packages, or games. The lower levels are nearly invisible from the User Level.

Level 5, the High-Level Language: Programming languages such as C, C++, FORTRAN, Lisp, Pascal, and Prolog consisting of this level. These languages must be translated (using either a compiler or an interpreter) to a language the machine can understand. Compiled languages are translated into assembly language and then assembled into machine code. (They are translated to the next lower level.) The user at this level sees very little of the lower levels. Even though a programmer must know about data types and the instructions available for those types, she need not know about how those types are actually implemented.

Level 4, the Assembly Language: As earlier mentioned, compiled higher-level languages are first converted to assembly, which is directly translated to machine language. This is a one-to-one translation, meaning that one assembly language instruction is translated to exactly one machine language instruction. By having separate levels, we reduce the semantic gap between a high-level language, such as C++, and the actual machine language (which consists of 0s and 1s).

Level 3, the System Software: deals with operating system instructions. This level is

responsible for multiprogramming, protecting memory, synchronizing processes, and various other important functions. Often, instructions translated from assembly language to machine language are passed through this level unmodified.

Level 2, the Instruction Set Architecture (ISA), or Machine Level: consists of the machine language recognized by the particular architecture of the computer system. Programs written in a computer's true machine language on a hardwired computer (see below) can be executed directly by the electronic circuits without any interpreters, translators, or compilers.

Level 1, the Control level: control unit makes sure that instructions are decoded and executed properly and that data is moved where and when it should be. The control unit interprets the machine instructions passed to it, one at a time, from the level above, causing the required actions to take place. Control units can be designed in one of two ways: They can be *hardwired* or they can be *micro programmed*.

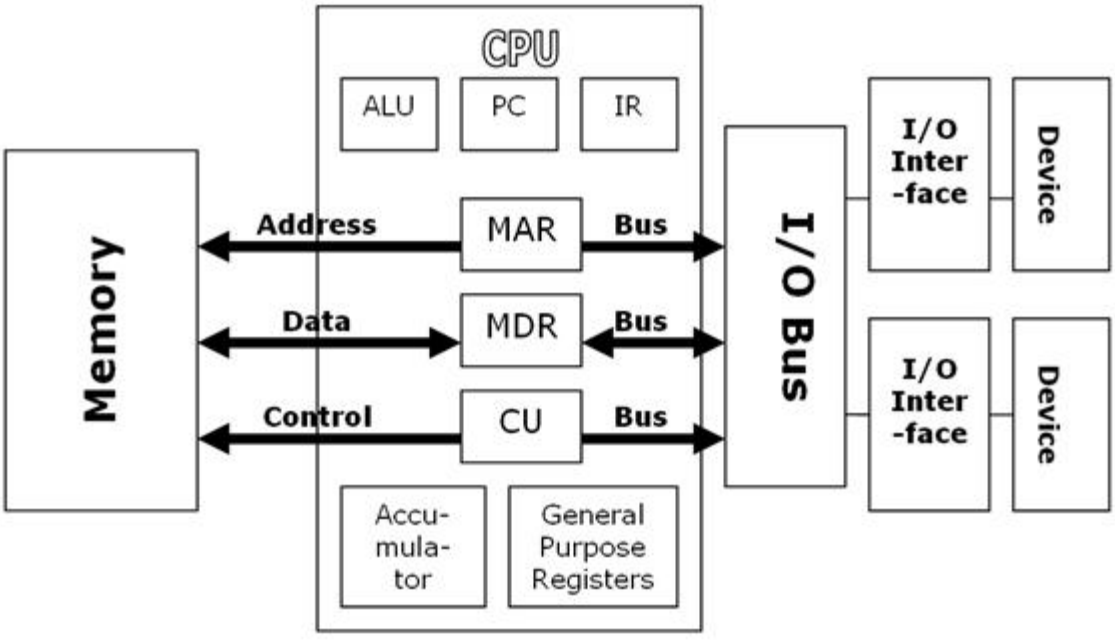
Level 0, the Digital Logic Level: physical components of the computer system: the gates and wires. These are the fundamental building blocks, the implementations of the mathematical logic that are common to all computer systems.

Functional Components of a Computer System

A computer system contains many different objects such as a CPU, memory, disks, etc. These must all be connected for the system to function. The **wires** used to connect the components are called **buses**.

The following section shows the connections within a computer system and describes how data is transferred between the components. The internal components of the CPU are described in the below.

The descriptions are actually concerned with real, physical things such as boards, chips and cables, as seen in the diagram below.



Functional diagram of a computer

Buses

The **buses** on a computer system are sets of wires that carry information to or from memory or I/O devices. They may be uni-directional (data travels one way) or bi-directional (data travels both ways).
Buses can be seen on the computer motherboard as parallel metal tracks. When the buses leave the motherboard to travel to a component such as a hard disk cables like these are used.

In general the number of wires on the bus corresponds to the *width* of the bus on the CPU. Computer systems have several buses dedicated to specific tasks.
Address Bus

The **address bus** carries the address of the piece of memory or I/O device to be read from or written to.
It is **a unidirectional** bus, which is to say that data travels only one way; from the CPU to memory.

The number of lines on the bus determines the number of addressable memory elements. For example an 8 bit bus can represent 2 to the power of 8 unique addresses. This equates to 256 unique **memory addresses**. A 16 bit bus can address 65536 unique addresses and so on.
Data Bus

The **data bus** carries the data that is to be written or has been read from memory. It is a **bidirectional** bus as it can carry data to or from memory.
The width of the data bus is directly related to the largest number that the bus can carry. For example an 8 bit bus can represent 2 to the power of 8 unique values. This equates to the numbers 0 to 255. A 16 bit bus can carry the values 0 to 65535 and so on.

Control Bus

The **control bus** carries signals that control the actions of the computer. For example one line of the control bus may be the read/write line. If the wire is low (no electricity flowing) then the memory is read. If the wire is high (with electricity flowing) then the memory is written.

Memory

Computer systems use **memory** to store data. Memory should not be confused with hard disk space. In general, memory is used for programs and data that are currently being used. Hard disks store data permanently.

Data that is lost when power is removed is volatile. Data that is not lost when power is removed is non-volatile. Many different types of memory, with varying characteristics, are used for different jobs in a computer system. These are described below.

RAM

RAM (Random Access Memory) is the main memory of a computer. Main memory is used to store all of the working information of the computer such as the operating system, user programs and data.

Main memory should not be confused with hard disk space which is used to permanently store data. RAM is volatile. The contents of RAM are lost when the computer is switched off.

DRAM

Dynamic RAM is the most common type of RAM used in computers. It is relatively easy to manufacture and so is cheap. However, DRAM contents must be continually refreshed. The process of refreshing the memory takes time and while the memory is being refreshed, it cannot be read from or written to. This makes DRAM cheap but slow. ***DRAM is cheap but slow and is used for main memory***.

SRAM

Static RAM is physically different to dynamic RAM as the memory contents do not have to be continually refreshed. This means that it can always be accessed and so SRAM is a faster type of memory. However the process of manufacturing the memory and the components used make it more expensive. For this reason SRAM is used in smaller

quantities where fast memory is required, such as cache. *SRAM is fast but expensive and is typically used for cache memory*.

ROM

Read Only Memory**:** Whereas RAM is volatile, **ROM** is non-volatile which is to say that the contents are not lost when power is removed. ROM chips come with instructions already burned into the chip. It is commonly used for computer **BIOS** chips. ***ROM is cheap in high quantities and is generally used for PC BIOSes.***

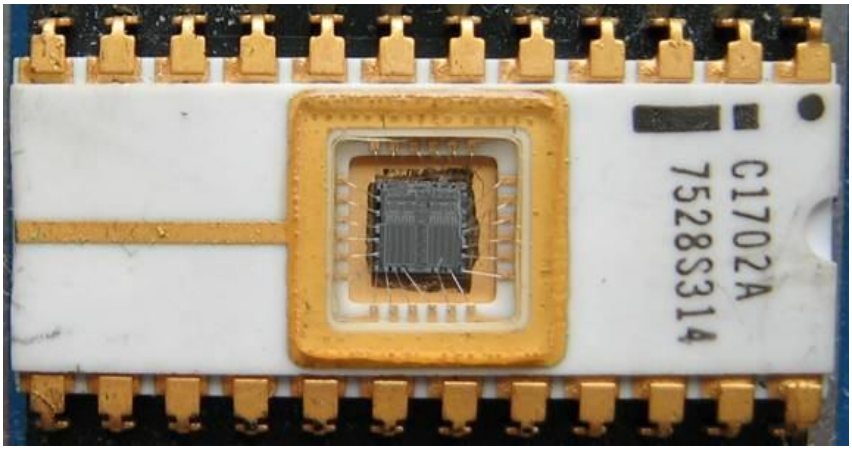
PROM

PROM (Programmable ROM) chips do not lose their data when power is removed but, unlike standard ROM chips, come without a program already installed.
The programs for the PROM chips can be 'burned' into them with a special piece of programming hardware. Once burned this way the contents are never lost but nor can they be altered.

PROM chips are commonly used where engaging a **chip foundry** to manufacture a custom ROM chip would be too expensive. **PROM is expensive and is used for low volume applications.**

EPROM

EPROM (Erasable Programmable ROM) chips can be erased once programmed, thus making them ideal for testing new applications where a ROM chip will be required.
They can be easily recognised by the plastic 'window' on the top of the chip. This is used to erase the contents of the chip by shining ultra-violet light onto it.

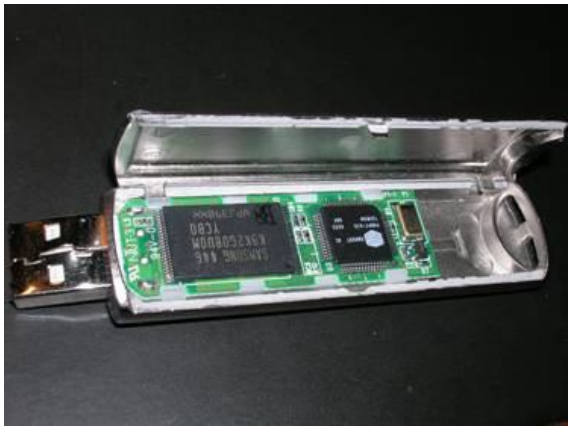


EPROMs are expensive but reusable and are used for testing.

EEPROM

EEPROMs (Electrically Erasable Programmable ROM) can be programmed and erased without using any special extra hardware. Instead a simple electrical current can be used. Unlike other programmable ROM chips EEPROMs need not be completely programmed at one time. Instead blocks of data can be programmed individually.

As well as being used in modern computers for updateable **BIOSes**, EEPROMs are typically used for USB drives, as well as for compact flash memory, **MultiMedia** cards, **SD** cards, etc.



EEPROM is cheap and is used for reprogrammable ROM applications like Flash.

Peripheral Input/Output

A peripheral is any piece of equipment attached to a computer. For example a mouse, keyboards, printers, etc. Peripherals employ several methods of communicating with the underlying computer system and those methods will be covered in this section.

I/O Device Addresses

Computers use two similar schemes to identify I/O devices.

The first scheme treats I/O devices as another type of memory. The addresses assigned to the devices are indistinguishable from memory addresses and the commands used to transfer data are the same as the commands used to transfer data to and from memory. **This scheme is called memory mapped I/O.**

The second scheme uses dedicated I/O commands to transfer data. When I/O instructions are issued the control bus will signal that the transfer is to I/O and not memory. **This is dedicated I/O.**

I/O Interface

The I/O interface contains the hardware necessary to allow communication with the I/O devices. In order to function, I/O interfaces require at least the following elements:

1. Buffer space to allow the data being transferred to be stored until the hardware is ready to process it.
2. A control bit to start and stop transfers.
3. A flag to signal when transfers are complete.

Device Drivers

It would be impossible for the writer of an operating system to cater for every piece of equipment that has been manufactured, never mind equipment that has not yet been created. Instead the authors will incorporate **hooks** into the operating system that other authors can use to pass data to and from the system.

When a manufacturer creates a new piece of hardware they will create a piece of software that talks to the operating system and passes data to and fro the peripheral. These small pieces of software are called **device drivers**. For example, when a new printer is installed, a **printer driver** will be added to the system at the same time. This printer driver will tell the computer how to send data to the printer and also the data format required. In this way the computer can simply provide a generic *Print* function, leaving the specific details of how to print to the printer driver.

Polling

Early computers cycled round all of the peripherals attached to the computer asking if they require any attention. For example when a device, such as a keyboard, reported that a key had been pressed and a character was ready to be processed by the CPU, a small program was run to collect the character from the device and move it in to

memory. This type of interaction with peripherals is called **polling**. This type of interaction with peripherals is slow; roughly analogous to a teacher walking around a class asking each pupil in turn if they need any help, rather than watching for the pupils to put up their hand.

Interrupts

Interrupts are a mechanism for peripherals to signal to the computer's CPU that they require attention. Dedicated wires are used to connect with all of the peripherals connected to the computer. When a peripheral requires attention it sends a unique signal to the CPU for it to stop what it is doing and service the peripheral. In other words the CPU has to interrupt the current operation to deal with the peripheral.

Once the CPU has identified the peripheral that requires attention, control of the processor is passed to a small program, called an **interrupt handler** or interrupt service routine (**ISR**), which deals with the peripheral. These interrupt handlers are programmed to be as small and as fast as possible so that the program that was originally running is interrupted for as short a time as possible. The sequence of events is therefore as follows:

i. Interrupt is raised
ii. Current CPU instruction is completed
iii. Contents of internal registers are stored in (pushed onto) the **stack**
iv. The memory address of the ISR is found and transfer controlled
v. ISR is run
vi. Internal register contents are restored (popped) from the stack
vii. Original process continues from where it was stopped

As interrupts stop a currently running process, the state of which must be saved before the interrupt is handled, there is a lag between the interrupt being generated and the ISR being run. This lag is the **interrupt latency** and is a consequence of steps 2, 3 and 4 above.

Interrupts usually have a hierarchy, defined by the computer system designer, which decides which interrupts have the highest priority. If an interrupt service routine is itself interrupted then the priority of the interrupt is checked. If the new interrupt is of a higher priority, then it is serviced, if not, it is ignored. This is a nested interrupt system.
As we can see, there is a possibility that an interrupt with a low priority may be lost. While for some peripherals this may not matter too much, for others it may be

disastrous. Peripherals that must be serviced within a specific time are assigned a **non-maskable interrupt** (NMI). These NMIs override all other interrupts.

Identifying Interrupts

When a device sends an interrupt to the CPU, the CPU must identify the interrupting device.

Software Identification

In a system where there are many devices on an interrupt line, the CPU requires to identify which device has interrupted. It does this by querying each device in turn, asking if they are the interrupting device. This is software polling.

Hardware Identification

An interrupt system that uses extra hardware to identify an interrupting device is known as a **vectored interrupt** system. Vectored interrupts contain enough address information within the interrupt to identify the device that has interrupted. As a consequence the CPU can jump immediately to the correct ISR.

Read/Write Operations

Data transfers require all components of the computer system to correctly synchronise. Memory must be accessed, buses must be made available, peripherals must transfer data, etc. All these operations must happen in a specific sequence in order for the transfer to be successful. In addition, I/O devices are attached in a physically different way; requiring different approach for memory and peripherals.

Memory Data Transfers

Memory Reads

For data to be read from memory the sequence is as follows:

1. Processor signals the memory location to be read by placing the memory address onto the address bus.
2. Specific memory chips that hold the memory location are made ready and selected.
3. A delay is added to allow the memory to settle and be available to read.
4. Data is placed onto the data bus.
5. Control unit of the processor turns on the read line of the control bus.
6. Data is placed into the MDR.

Notes: Step 2 is necessary as computer memory consists of more than one chip. To read from memory the correct chip must be chosen. Step 3 introduces the time necessary for all electronic components to respond to supplied signals (settle).

Memory Writes

For data to be written to memory the sequence is as follows:

1. Processor signals the memory location to be written by placing the memory address onto the address bus.
2. Processor places the data to be written onto the data bus.
3. Specific memory chips that hold the memory location are made ready and selected.
4. A delay is added to allow the memory to settle and be available to write.
5. Control unit of the processor turns on the write line of the control bus.
6. Data is written to the correct memory location.

I/O Data Transfers

I/O Reads

For data to be read from an I/O device the sequence is as follows:

1. Processor signals the device to be read by placing the memory address onto the address bus.
2. Control unit of the processor turns on the I/O read line of the control bus.
3. Data is placed onto the data bus from the peripheral.
4. CPU reads data from data bus and places it into the MDR.

I/O Writes

For data to be written to an I/O device the sequence is as follows:

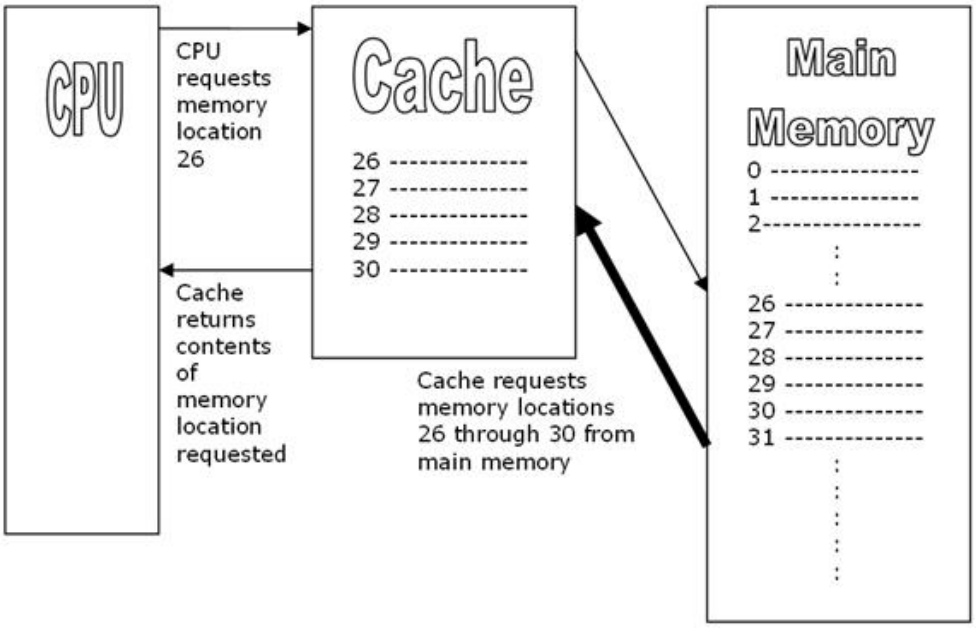
1. Processor signals the I/O device to be written by placing the memory address onto the address bus.
2. Processor places the data to be written onto the data bus.
3. Control unit of the processor turns on the I/O write line of the control bus.
4. Data is written to the peripheral.

Caching

Caching is the process of storing small amounts of a large data source in order to speed up the data access time. In this case it is achieved by storing a small amount of the main memory in fast RAM to speed up the process of accessing memory.

CPU caching makes use of the sequential and predictable nature of running a program. In general programs run the instruction at memory location 0, followed by the instruction at memory location 1, then 2, 3, 4 and so on. This is known as **memory locality**.

The slowest part of running the program is accessing the instruction in memory and placing it in the CPU. If we can speed up this part of the fetch process then we dramatically increase the overall speed of the computer.
To do this we place a small piece of fast RAM (SRAM, access speed of 10ns) between the CPU and the relatively slow main memory (DRAM, access speed of 60ns). When the CPU requests one of the memory locations the cache hardware automatically loads the memory locations close to the location requested into cache. When the CPU requests the next memory location the chances are that it will already have been loaded into the fast cache memory and can be supplied more quickly to the CPU.
CPU Caching Diagram



This diagram illustrates level 2 cache. Level 1 cache, where the cache memory is built into the CPU, is also used.

Cache memory can work in both directions. When writing to main memory the data is first of all written to cache, allowing the CPU to continue working. The cache hardware then writes the data to main memory in its own time.
Cache Types
Different methods of reading and writing using cache have been developed. Some of those are described here.

Reading Using Cache

Look Through

CPU request memory from the cache. Only if the data is not present is the main memory queried.

Look Aside

CPU requests memory from cache and main memory simultaneously. If the data is in the cache then it is returned, otherwise the CPU waits for the data from the main memory.

Writing Using Cache

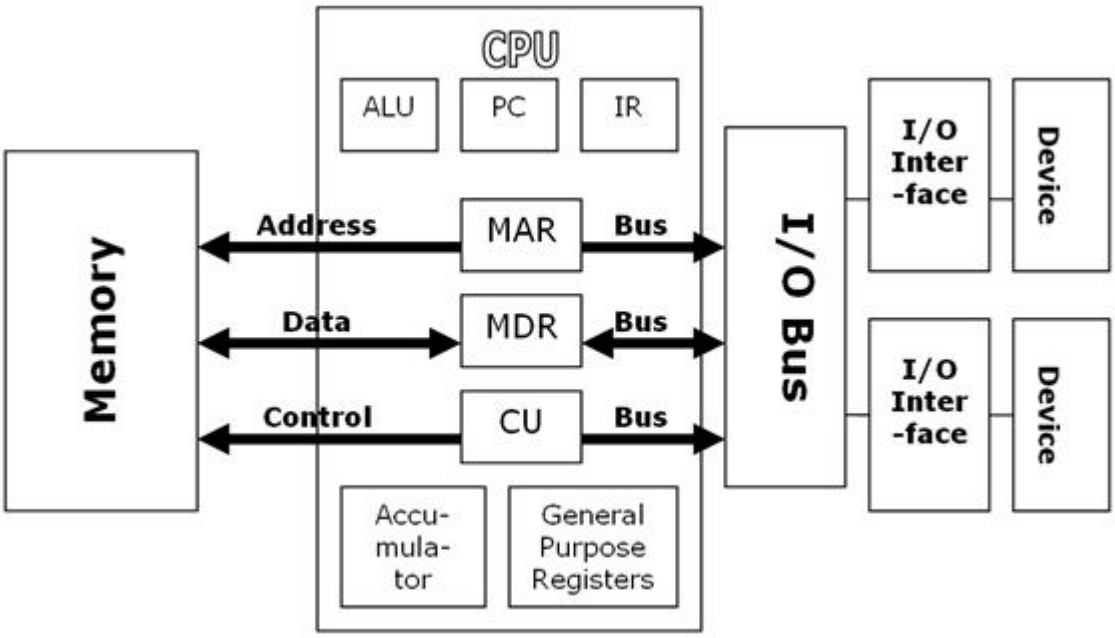
When data is stored back to memory it is written to cache and main memory at the same time.

Write Back

Data in the cache is compared to the data in the main memory. Data is written only if there is a difference.

Direct Memory Access (DMA)

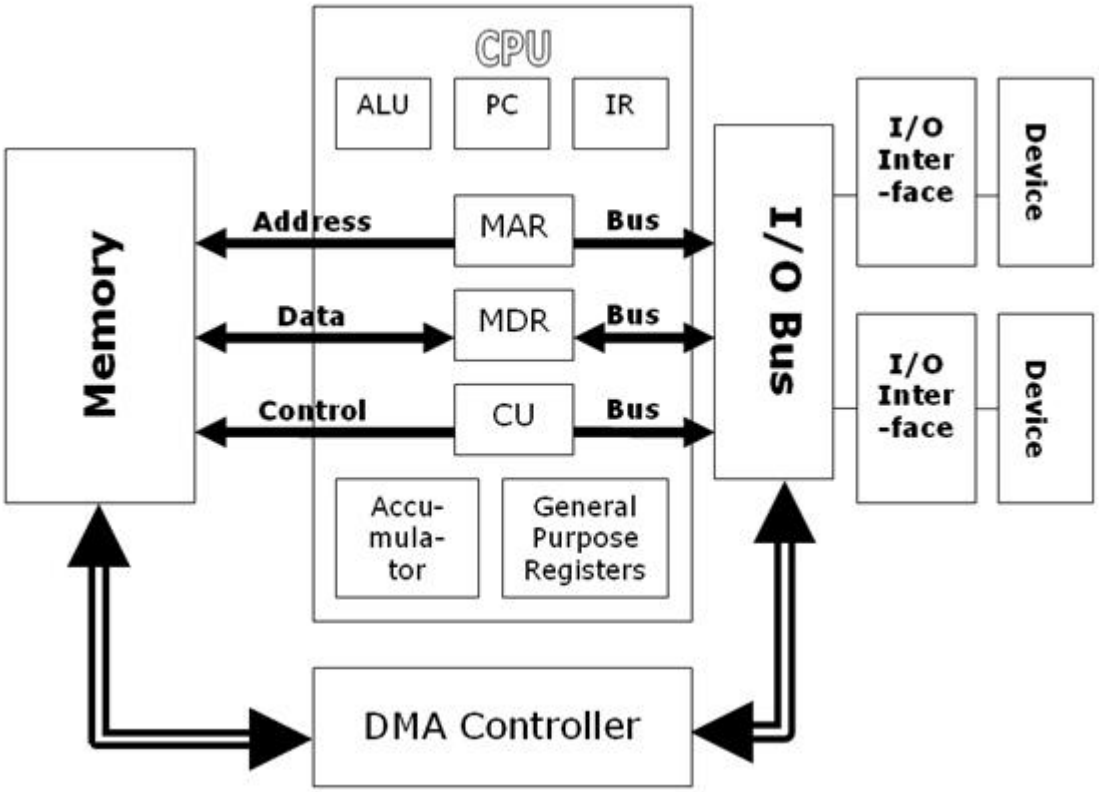
If we look at the computer system diagram we can see that when transferring data from a device such as a hard disk the CPU is heavily involved.



Each word of data has to be transferred from the disk, through the CPU and individually placed into memory. This places a heavy burden on the CPU and stops it from performing any useful tasks. To address this problem we use a system of Direct Memory Access (DMA).

Implementing DMA

To implement the direct transfer of data between memory and I/O a hardware DMA controller is added to the system. This hardware works like a bypass for a town, allowing large blocks of data to bypass the CPU without snarling it up.



In this scenario large blocks of data are transferred using the DMA hardware. The CPU, on receiving an interrupt, initiates the DMA hardware with

- A memory start address
- The amount of data to be transferred
- The device to be used (hard disk, CD, DVD, etc.)
- The direction of transfer (input or output)

The DMA controller then transfers the data. Upon completion the controller notifies the CPU that the transfer has been completed.

DMA Conflicts

The use of a DMA controller can sometimes cause a conflict between the DMA hardware and the CPU. This arises when both devices attempt to access memory at the same time.

As data would be lost from a high speed peripheral if its transfer was paused the DMA controller always has priority over the CPU. As this can occasionally cause the CPU to lose a memory access cycle it is sometimes called **cycle stealing**.

Characteristics of various computer systems

Main Characteristics of computer systems are:

1. **Single user systems** are systems that only allow for one user to access the computer at a time
2. **Multi User systems** are systems that allow continuous access of multiple users at once and that are capable of running more than one process at a time in order to maximize the potential of all users. **Example:** a server/computer where several people access a server/computer remotely
3. **Single Tasking** is a reference to when a system is only doing one task at a time
4. **Multi Tasking** allows a computer to run more than one task at a time by scheduling what tasks to do when in order not to have the computer make mistakes or do the wrong thing.

INTRODUCTION TO PROGRAMMING METHODOLOGY/TECHNIQUES, ALGORITHMS AND PROBLEM SOLVING

Introduction to Program Development and Problem Solving

- Computers do what we tell them to do NOT what we want them to do
- **Computer Programming** involves writing instructions and giving them to the computer to complete a task.

- A **computer program** or **software is:**
  - a **set of instructions** written in a computer language in order to
  - be **executed** by a computer to perform a useful task

Example: Application software packages, such as word processors, spreadsheets and databases are all computer programs

- A **Computer Programmer** is
  - is a person who translates the task you want a computer to do
  - into a form that a computer can understand

- A well designed **computer program** must be:
  - **correct and accurate**
  - **easy to understand**
  - **easy to maintain and update**
  - **efficient**
  - **reliable**
  - **flexible**

- The **program development process** involves the following steps:
  - document the program
    - program documentation is the text or graphics that provide description of
    - the purpose of a function or a particular step
    - the purpose of instruction in a program
  - determine the user needs
  - design the program specifications
  - review the program specifications
  - design the algorithm
    - steps that will convert the available input into the desired output
    - step by step solution for a given problem is called the algorithm

- a flowchart graphically details processing steps of a particular program
- code the program
- write the program in a programming language using a program editor
- a program editor is a program that allows to type, edit and save a program code on a disk
- compile, test and debug the program
- in order to find out possible errors in the program
- types of errors may be syntax errors, run-time errors and logic errors
- get program to the user
- install software on the users computer and offer training

➢ Types of **programming errors**

- **Syntax** of a programming language is the set of rules to be followed when writing a program
- **syntax error** occurs when these rules are violated
- **run-time errors** occur when invalid data is entered during program execution
- e.g. program expects numeric data and alphabetic data is entered
- program will crash
- **logic error** will not stop the program but results will be inaccurate
- The process of finding and correcting errors in a program is called **debugging**

➢ For **successful programming**
- give no ambiguity in program instructions
- give no possibility of alternative interpretations
- make sure there is only one course of action

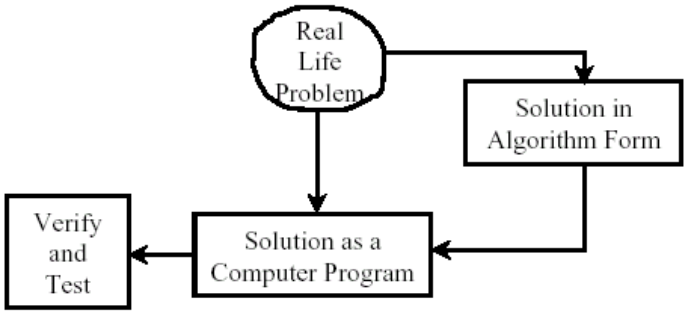
➢ Programming task can be made easier
- by breaking large and **complex programs**
- into **smaller** and **less complex** subprograms (modules)

➢ Programming task can be separated into **2 phases** (see Fig. 1)
- **problem solving** phase
- produce an ordered sequence of steps that describe solution of problem
- this sequence of steps is called an **algorithm**
- Example of an Algorithm: a recipe, to assemble a brand new computer ...

- what else?

- **implementation** phase
- implement the program in some programming language (Pascal, Basic, C)

Figure 1: Problem Solving and Programming



- What is **structured programming**
- a programming technique that splits the program into smaller segments (modules) to
- decrease program development time
- decrease program maintenance cost
- improve the quality of software
- structured programming achieves these goals by using
- **top-down** design and use of modules
- use of limited control structures (sequence, selection and repetition)
- management control

- **Top-down design** starts with major functions involved in a problem
- and divide them into sub-functions
- until the problem has been divided as much as possible

- Categories of **programming languages** are
- machine } low-level languages
- assembly }
- high-level
- fourth-generation
- fifth-generation

- **Machine language** is
  - made up of binary 1s and 0s
  - this is the only programming language the computers can understand
  - **advantages** of machine languages are:
    - fast execution speed and efficient use of main memory
  - **disadvantages** of machine languages are
    - writing machine language is tedious, difficult and time consuming
- Types of **language-translator** programs
  - **assemblers**
    - translates the program in **assembly-language** into **machine-language**
  - **compilers**
    - translates **high-level** language program into **machine-language** all at once
  - **interpreters**
    - translates **high-level** language into **machine-language a line at a time**

- Major **high-level languages** are
  - **FORTRAN, COBOL, PL/I, BASIC, PASCAL, C, LISP, Prolog, Logo**

- **FORTRAN**
  - **FOR**mula **TRAN**slator introduced in 1957
  - for use by scientists, engineers and mathematicians
  - well suited for complex numeric calculations
- **COBOL**
  - **CO**mmon **B**usiness **O**riented **L**anguage
  - a programming language used for business data processing
  - designed to handle large data files
- **PL/I**
  - **P**rogramming **L**anguage **One** created in 1960
  - general purpose language for powerful computations and
  - sophisticated data structures
  - today largely used in the oil industry
- **BASIC**
  - **B**eginners **A**llpurpose **S**ymbolic **I**nstruction **C**ode created in 1960
  - easy to learn interactive language on a time-sharing computer
- **PASCAL**
  - Named after Blaise **Pascal** and created in 1960
  - suited to both scientific and file processing applications

- designed to teach structured programming and top-down design

➢ **C**

- Developed at Bell Labs in 1970s
- used advantages of both high-level and low-level languages
- C gives programmers extensive control over computer hardware
- incorporates sophisticated control and data structures
- C++ is the object oriented version of C

➢ **LISP**

- is a language that processes symbol sequences (lists) rather than numbers
- designed to handle data strings more efficiently than others

➢ **Prolog**

- is a language for symbol processing

➢ **Logo**

- is an interactive education oriented language
- designed to teach inexperienced users logic and programming techniques
- includes list processing capabilities
- employs a triangular object called turtle
- to draw, animate and color images very simply

➢ **Object-Oriented Programming Languages (OOPL)**

- there are two main parts of a program, **instructions** and **data**
- traditional prog. languages treat instructions and data as separate entities
- an OOPL treats a program as a series of **objects** and **messages**
  - an **object** is a combination of **data** and **instructions** that work on data
  - and is stored together as a reusable unit
  - **messages** are instructions sent between objects
- to qualify as an OOPL a language must incorporate concepts of
  - **1. encapsulation**
  - **2. inheritance**
  - **3. polymorphism**
- **encapsulation** is
  - combining data and instructions into a reusable structures
  - encapsulation generates prefabricated components of a program
- **inheritance**
  - is the ability of a programming language to define a new object
  - that has all the attributes of an existing object

- programmer inherits the existing object
- writes the code that describes how the new object differs from the existing one

- **polymorphism** is
- the ability of an object to respond to a message in many ways
- for example, say you have a circle object and a square object
- each object has the same characteristics of drawing
- when a drawing message is sent to a circle object it draws a circle
- when a drawing message is sent to a square object it draws a square
- thus, each object has the same characteristics of drawing
- but the characteristics is implemented differently

➢ **Advantages of OOPL**

- code re-use rather than reinvention and adaptable code
- these speed development & maintenance of applications and reduce cost
- Smaltalk, Objective-C and C++ are examples of OOPL

Algorithms and Flowcharts

- A typical programming task can be divided into 2 phases:
  - **Problem solving phase**
    - produce an ordered sequence of steps that describes solution to problem
    - this sequence of steps is called an **algorithm**

Examples of Algorithm: a recipe, to assemble a brand new computer, ...
what else?

  - **Implementation phase**
    - implement the program in some programming language

- Steps in Problem solving
  - First produce a general algorithm (You can use **pseudocode**)
  - Refine the algorithm successively to get step by step detailed **algorithm** that is very close to a computer language.

Pseudocode is an artificial and informal language that helps programmers develop algorithms. Pseudocode is very similar to everyday English.

Example 1: Write an algorithm to determine the students final grade and indicate weather it is passing or failing. The final grade is calculated as the average of four marks.

Solution:

Pseudocode:
```
Input a set of 4 marks
    Calculate their average by summing and dividing by 4
    if average is below 50
        Print "FAIL"
    else
        Print "PASS"
```

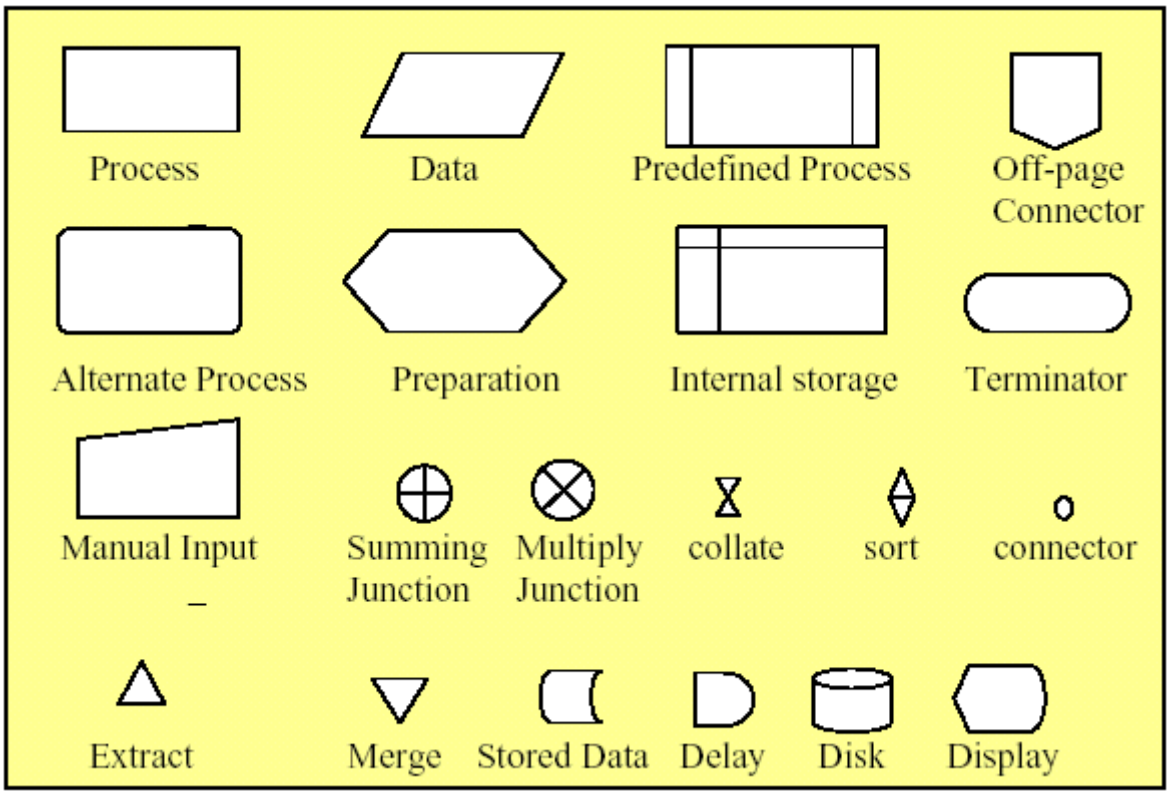
Detailed Algorithm:
```
Step 1:    Input M1, M2,M3,M4
    Step 2: GRADE = (M1+M2+M3+M4)/4
    Step 3: if (GRADE <50) then
                    Print "FAIL"
                else
                    Print "PASS"
```

The Flowchart

- Easier to convey ideas by picture than by words
  - **Examples:**
    - It is easier to show how to go somewhere on a map than explaining
    - It is easier to construct a toy if diagrams are shown
    - It is easier to construct a new PC if diagrams are provided
  - **Hence use pictorial format for describing an algorithm**
    - this is called a **FLOWCHART**

- A Flowchart
  - shows logic of an algorithm
  - emphasizes individual steps and their interconnections
    - e.g. control flow from one action to the next

  - **standard flowchart symbols** are shown below

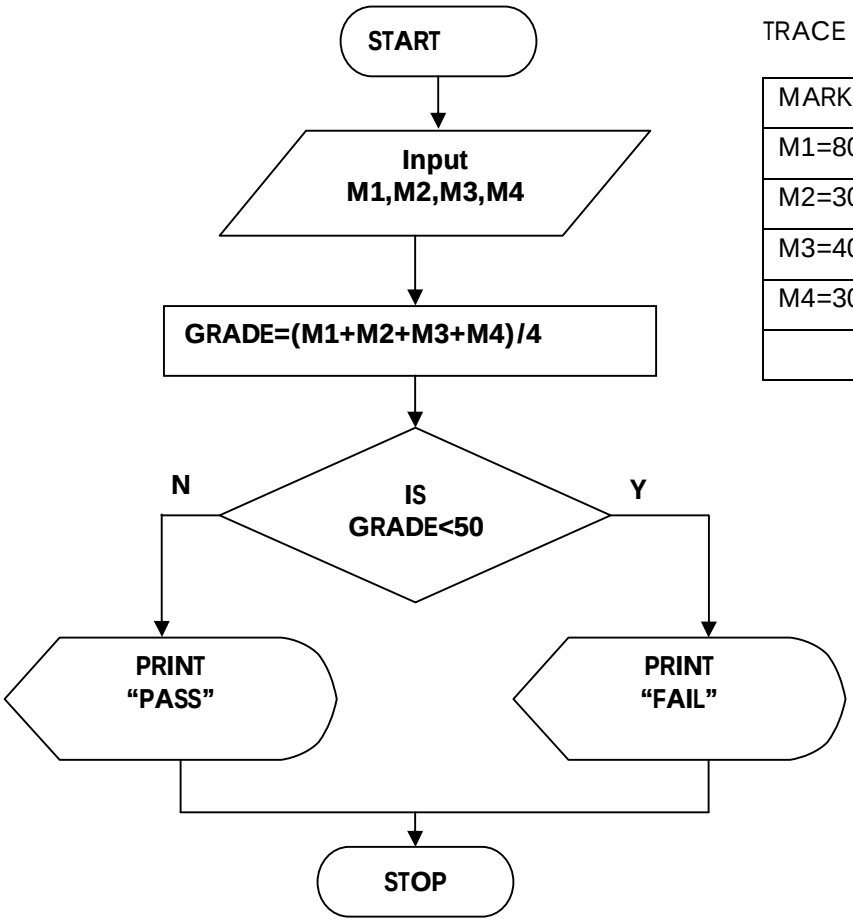


Example 1: Write an algorithm to determine the students final grade and indicate weather it is passing or failing. The final grade is calculated as the average of four marks.

Pseudocode: *Input a set of 4 marks*
Calculate their average by summing and dividing by 4
if average is below 50
Print "FAIL"
else
Print "PASS"

Detailed **Algorithm**: Step 1: Input M1, M2,M3,M4
Step 2: GRADE = (M1+M2+M3+M4)/4
Step 3: if (GRADE <50) then
Print "FAIL"
else
Print "PASS"

Flowchart:



TRACE TABLE:

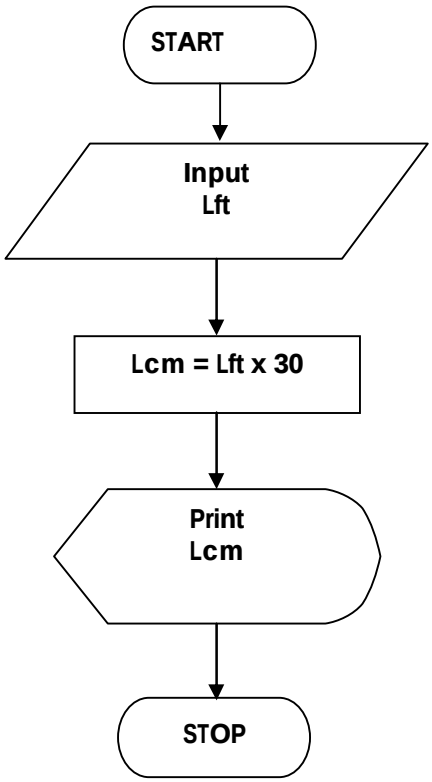
MARK	GRADE	STATUS
M1=80	80	
M2=30	+ 30	
M3=40	+ 40	
M4=30	+ 30 /4	
	= 45	FAIL

Example 2: Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode: Input the lenght in feet (Lft)
Calculate the length in cm (Lcm) by multiplying LFT with 30
Print LCM

Algorithm: Step 1: Input Lft
Step 2: Lcm = Lft x 30
Step 3: Print Lcm

Flowchart:

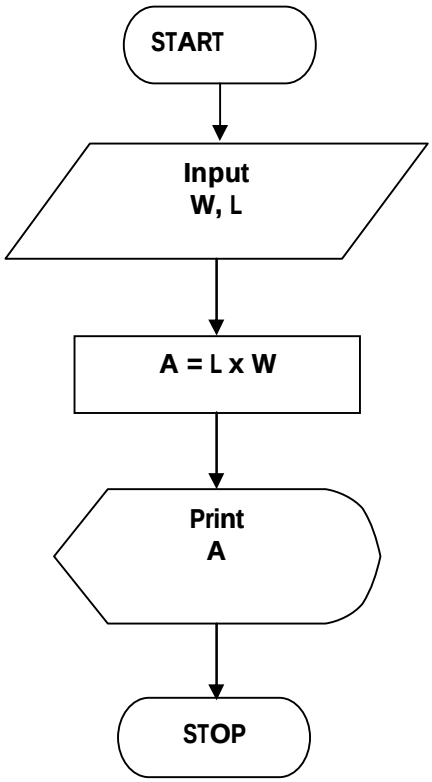


Example 3: Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

Pseudocode: Input the width (W) and Length (L) of a rectangle
Calculate the area (A) by multiplying L with W
Print A

Algorithm: Step 1: Input W,L
Step 2: A = L x W
Step 3: Print A

Flowchart:



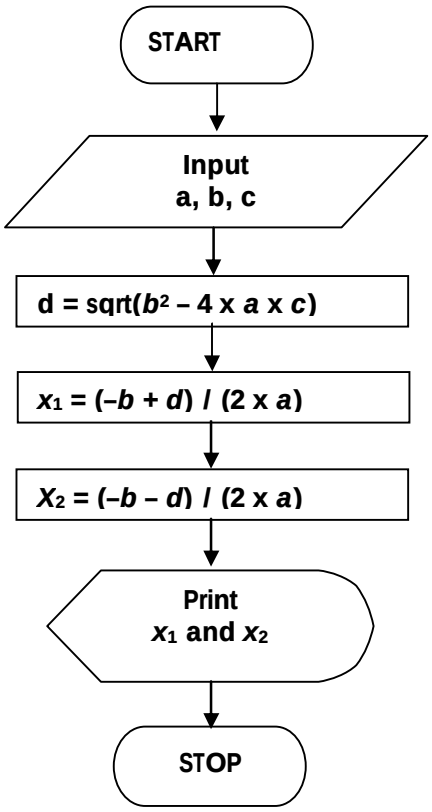
Example 4: Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation $ax^2 + bx + c = 0$

Hint: d = sqrt ($b^2 – 4ac$), and the roots are: $x_1 = (–b + d)/2a$ and $x_2 = (–b – d)/2a$

Pseudocode: Input the coefficients (a, b, c) of the quadratic equation
Calculate the discriminant d
Calculate x_1
Calculate x_2
Print x_1 and x_2

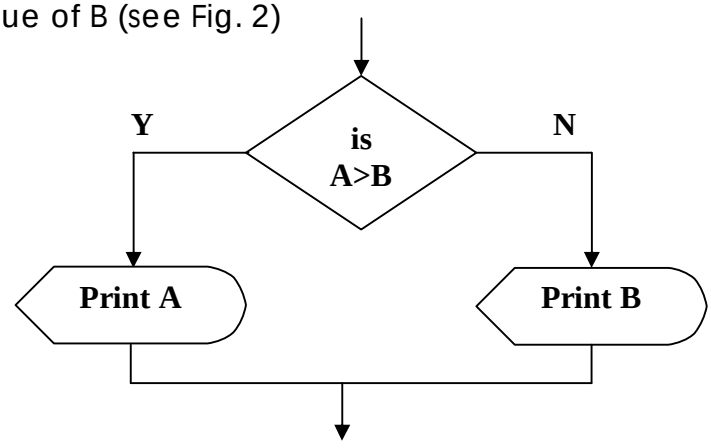
Algorithm: Step 1: Input a, b, c
Step 2: d = sqrt ($b^2 – 4$ x a x c)
Step 3: $x_1 = (–b + d) / (2 \times a)$
Step 4: $x_2 = (–b – d) / (2 \times a)$
Step 5: Print x_1 and x_2

Flowchart:



DECISION STRUCTURES

- The expression A>B is a logical expression
  - it describes a **condition** we want to test
  - **if A>B is true (if A is greater than B)** we take the action on left of
    - print the value of A (see Fig. 2)
  - **if A>B is false (if A is not greater than B)** we take the action on right
    - print the value of B (see Fig. 2)



IF–THEN–ELSE STRUCTURE

- The structure is as follows

- **If condition**
- **then true alternative**
- **else false alternative**
- **endif**

- The algorithm for the flowchart in figure 2 is as follows:

If A>B
then print A
else print B
endif

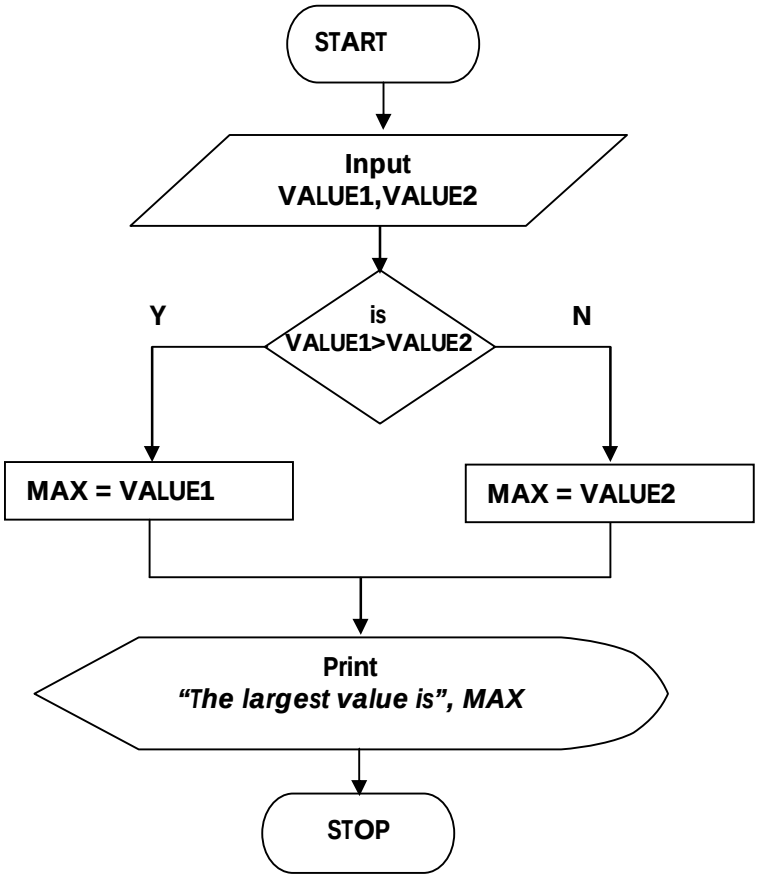
Here "**>**" is called the relational operator. Table 1 gives the possible relational operators:

Relational Operators	
Operator	**Meaning**
>	**Greater than**
<	**Less than**
=	**Equal to**
≥	**Greater than or equal to**
≤	**Less than or equal to**
≠	**Not equal to**

Example 5: Write an algorithm that reads two values, determines the largest value and prints the largest value with an identifying message.

Algorithm: Step 1: Input VALUE1, VALUE2
Step 2: if VALUE1 > VALUE2
then MAX = VALUE1
else MAX = VALUE2
endif
Step 3: Print "The largest value is", MAX

Flowchart:



NESTED IFS

- One of the alternatives within an IF–THEN–ELSE statement
- may involve further IF–THEN–ELSE statement

Example 6: Write an algorithm that reads **three** numbers and prints the value of the largest number.

Algorithm: Step 1: Input N1, N2, N3
Step 2: if N1>N2
then if N1>N3
then MAX = N1 [N1>N2, N1>N3]
else MAX = N3 [N3>N1>N2]
endif
else if N2>N3
then MAX = N2 [N2>N1, N2>N3]
else MAX = N3 [N3>N2>N1]
endif
endif
Step 3: Print "The largest number is", MAX

Flowchart: Draw the flowchart of the above Algorithm.

Example 7: Write an algorithm and draw a flowchart to:

a. read an employee name (NAME), overtime hours worked (OVERTIME), hours absent (ABSENT) and

b. determine the bonus payment (PAYMENT).

Bonus Schedule	
OVERTIME – (2/3)*ABSENT	Bonus Paid
>40 hours	$50
>30 but ≤ 40 hours	$40
>20 but ≤ 30 hours	$30
>10 but ≤ 20 hours	$20
≤ 10 hours	$10

Algorithm: Step 1: Input NAME,OVERTIME,ABSENT

Step 2: if OVERTIME–(2/3)*ABSENT > 40
then PAYMENT = 50
else if OVERTIME–(2/3)*ABSENT > 30
then PAYMENT = 40
else if OVERTIME–(2/3)*ABSENT > 20
then PAYMENT = 30
else if OVERTIME–(2/3)*ABSENT > 10
then PAYMENT = 20
else PAYMENT = 10
endif
endif
endif
endif
Step 3: Print "Bonus for", NAME "is $", PAYMENT

Flowchart: Draw the flowchart of the above algorithm