Lec-1

06 May 2021 15:29

```
# Computer Organization and Software Systems
## Lecture 1: 01-MAY-2021
### How Computer organization different from Computer Architecture.
### How Computer organization can be useful in data analytics.
### Components of a computer system
### How does a compiler compiles a program (Running Hello.c )
### Von Neuman Architecture and Von Neuman Bottleneck
### Harvard Architecture
### What is an instruction Cycle ?
#### Fetch
#### Execute
#### Interrupt
### Explain program flow control with and without Interrupt
### Interrupt Cycle
### Explain memory Hierarchy
### What is the role of cache memory
### What is an OS, and what happens if there is no OS?
### Objectives of having a system software/OS
### Role of Operating System in managing Hardware
### Operating System Modes
### Name few services provided by operating system
### What is a background process and how it is stored in memory
### What is a thread within a process
### Virtual Memory
```

How Computer organization different from Computer Architecture

Comp architecture → attributes of system → visible to Programmer
∴ those attributes → which have a direct impact on logical execution of a program

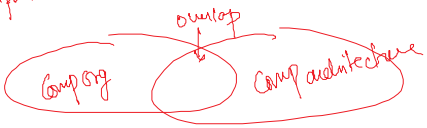
Comp arch is called ISA instruction set architecture | ISA

Computer architecture or instruction set architecture (ISA). The ISA defines
instruction formats,
instruction opcodes,
registers,
instruction and data memory;
the effect of executed instructions on the registers and memory; and
an algorithm for controlling instruction execution

A Computer organization refers to Operational units and their interconnections that realise the architectural specification

Example: of architectural attributes that are part of computer organization :
① Instruction Set ② No of bits used to represent the data type
③ I/O mechanism and techniques for addressing mem.

☆ Comp org attributes also include some hardware details that are visible to programmer
Eg: Control signals, memory technology and interface used between Comp and peripherals.



For example, it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system

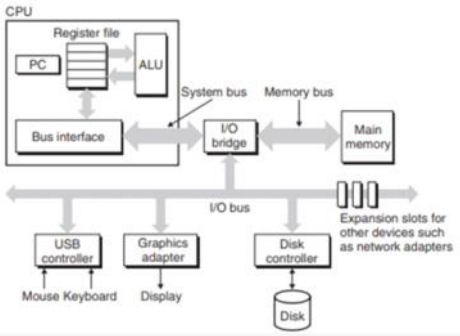
How Computer organization can be useful in data analytics.

With Rise in availability of data and high performance Computers becoming cheaper it is important that we optimise the architecture / Run the program in a suitable architecture machine because complex analysis could run for many days to train the models.

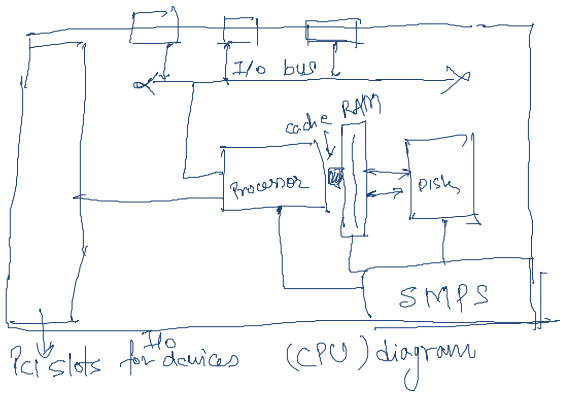
A Data expert should be able to answer certain analytical computations to run on certain type of System optimally.

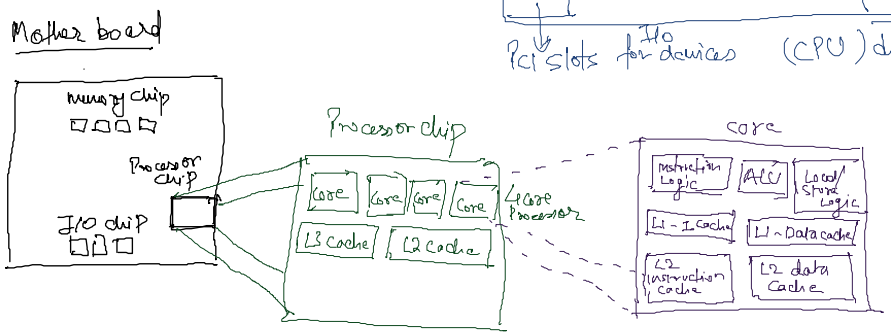
❖ Components of a computer system

- ❖ Hardware
  - ❖ CPU
    - ❖ ALU
    - ❖ PC
    - ❖ Registers
    - ❖ Control Unit
  - ❖ Memory
    - ❖ Cache
    - ❖ Primary Memory
    - ❖ Secondary Memory
  - ❖ I/O Devices
    - ❖ Peripherals for input
      - ❖ Mouse
      - ❖ Keyboard
      - ❖ Pen
      - ❖ Touch Screen
    - ❖ Peripherals for output
      - ❖ Printer
      - ❖ Monitor / Screen
      - ❖ Video Out



- Software
  - System Software
    - System Management Software
    - Tools and Utilities for developing the software
  - Application Software
    - General Purpose
      - e.g. Word processor, notepad etc
    - Specific Purpose
      - e.g. Firmware, Device Drivers





How does a compiler compiles a program (Running Hello.c)

Lexical Analysis
The first phase of scanner works as a text scanner. This phase scans the source code as a stream of characters and converts i t into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens as:
<token-name, attribute-value>

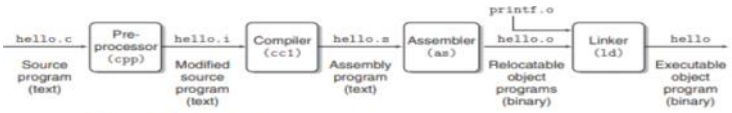
Syntax Analysis
The next phase is called the syntax analysis or parsing. It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar, i.e. the parser checks if th e expression made by the tokens is syntactically correct.

Semantic Analysis
Semantic analysis checks whether the parse tree constructed follows the rules of language. For example, assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyzer keeps track of identifiers, their types a nd expressions; whether identifiers are declared before use or not etc. The semantic analyzer produces an annotated syntax tree as an output.

Intermediate Code Generation
After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language. This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.

Code Optimization
The next phase does code optimization of the intermediate code. Optimization can be assumed as something that removes unneces sary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU , memory).

Code Generation
In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machi ne language. The code generator translates the intermediate code into a sequence of (generally) re-locatable machine code. Sequence of instructions of machine code performs the task as the intermediate code would do.

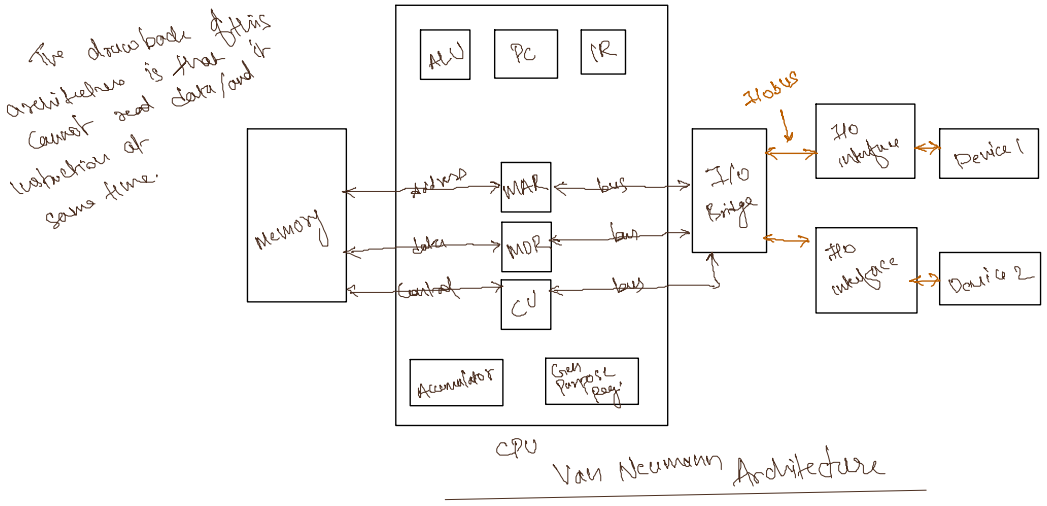


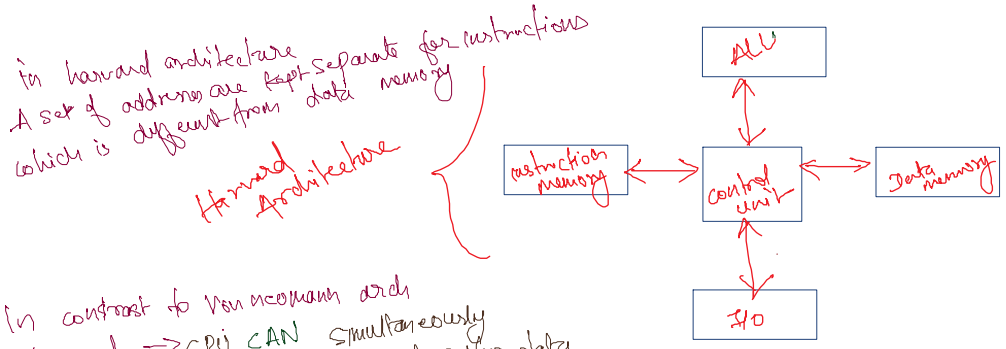
The compilation system.

Von Neuman Architecture and Von Neuman Bottleneck

Fixed program architecture : Earlier computers are built for specific purpose which are non-programmable. They are designed such a way that the instructions are stored fixed in registers. Example : A calculator which has given fixed instructions.

Stored program architecture : A stored program architecture is designed to keep data and instruction both in primary memory (RAM). The vast majority of m odern day computer are based on stored program architecture. These are programmable computers since we can change the set of instru ctions from memory.
This is also known as IAS, Von Neumann Architecture or Princeton architecture. Another type of stored program architecture is Harvard Architecture.





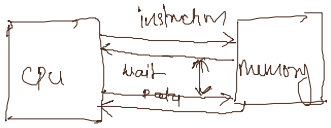
In contrast to Von neomann arch
harvard → CPU CAN simultaneously read instructions and access data from memory

The principle advantage of harvard architecture is reduced/compromised by modified architecture where CPU cache memory is being used.

harvard architecture was developed to overcome the Von neomann bottleneck

Bottleneck

Instructions are temporarily stored in Registers which are logically inside the CPU and these are v fast in comparison to RAM
∴ this speed parity has caused bottleneck which was later overcome by cache

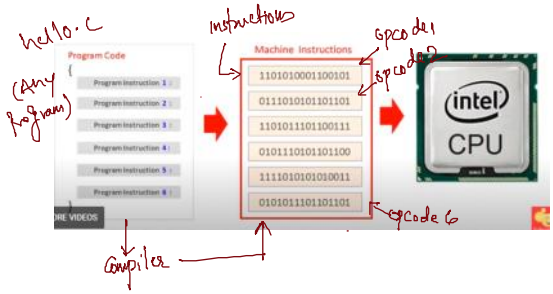


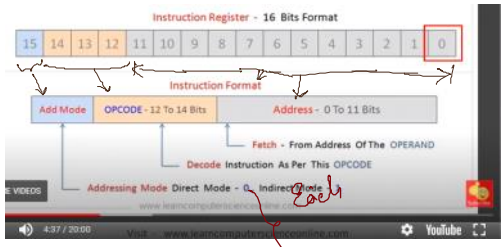
1. **Accumulator:** Stores the results of calculations made by ALU.
2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).
3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.
4. **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
5. **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.
6. **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.
   - **Control Unit –**
     A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions and controlling how data moves around the system.
   - **Arithmetic and Logic Unit (ALU)** –
     The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation
   - **Program Counter** - A program counter is a register which keeps the address of the next instruction that is to be executed.

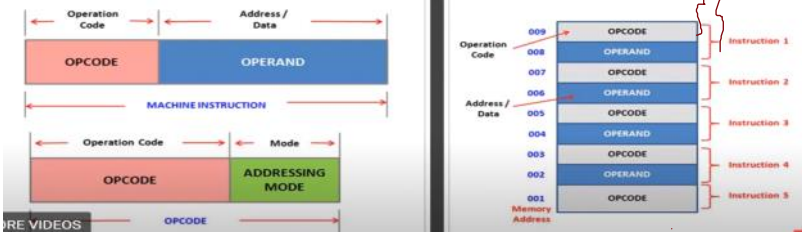
What is an instruction Cycle

- **Memory buffer register (MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
- **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
- **Instruction buffer register (IBR):** Employed to hold temporarily the righthand instruction from a word in memory.
- **Program counter (PC):** Contains the address of the next instruction pair to be fetched from memory.
- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations. For example, the result

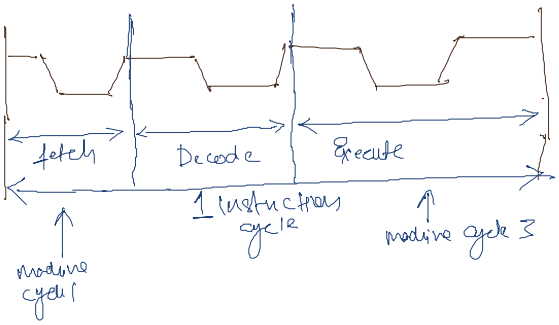
- **Memory buffer register (MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
- **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
- **Instruction buffer register (IBR):** Employed to hold temporarily the righthand instruction from a word in memory.
- **Program counter (PC):** Contains the address of the next instruction pair to be fetched from memory.
- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations. For example, the result

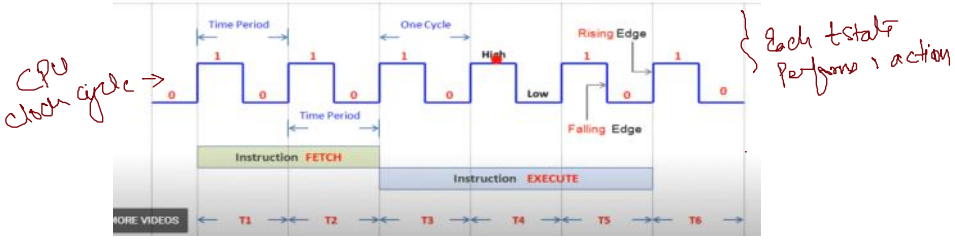






Instruction cycle: Time required to complete 1 instruction
1 instruction → has multiple machine cycle





Fetch -> Decode -> Execute -> Interrupt

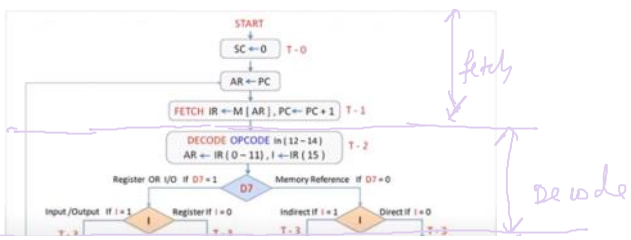
Fetch :

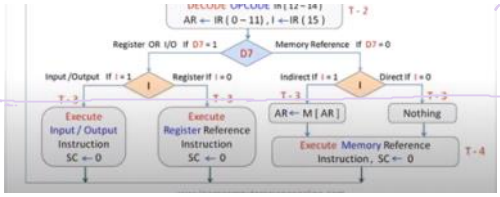
Program Counter (PC) holds address of next instructions be fetched

- Processor fetches instruction from memory location pointed to by PC
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions
- Increment PC
- – Unless told otherwise

Decode :

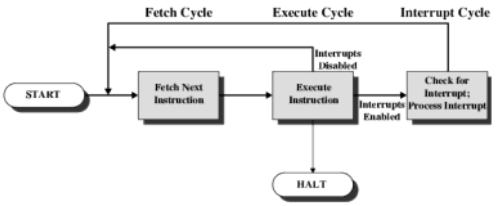
- The control unit inside the CPU decodes the program instruction that CPU has just received during fetch. (IR < - instruction)
- The instruction format tells the type of instruction that is to be performed by CPU (arithmetic ops, logical ops, etc.)
- The Opcode part (3 bits) of instruction format tells which operation to perform (000 -> 111 : 8 different opcodes possible)
- There can be 3 type of Instruction (Memory reference, Register reference and I/O instruction)

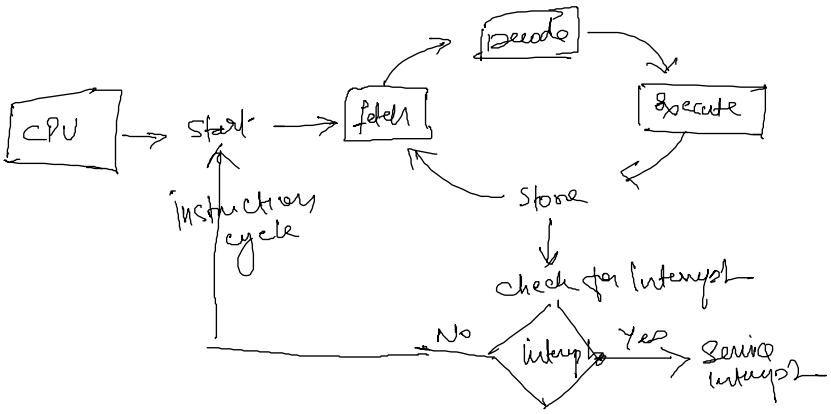




Execute :

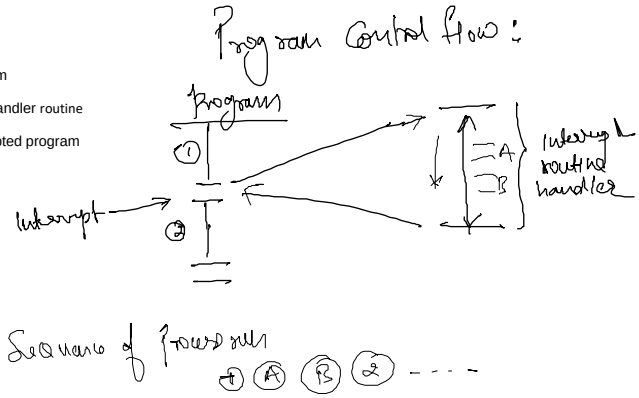
- Processor - memory – Data transfer between CPU and main memory
- Processor - I/O – Data transfer between CPU and I/O module
- Data processing – Some arithmetic or logical operation on data
- Control – Alteration of sequence of operation. e.g. jump
- Combination of above



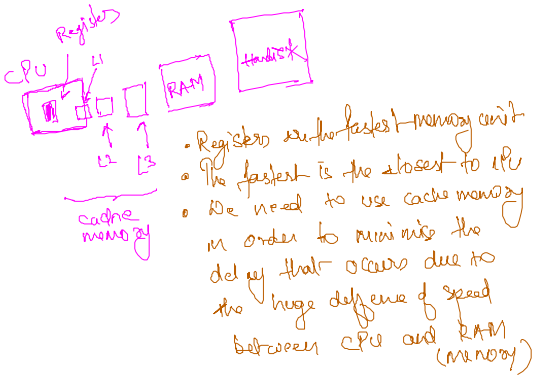


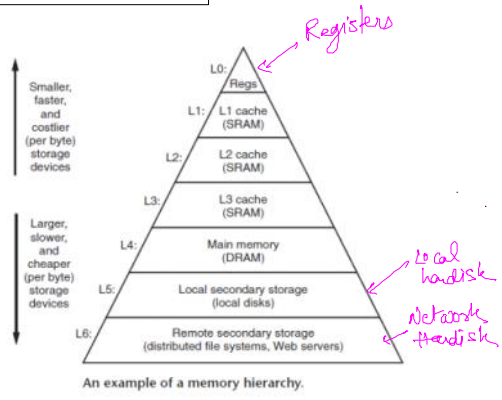
Interrupt Cycle : Interrupts: Mechanism by which other modules (e.g. I/O / program) may interrupt normal sequence of process ing

- Interrupts enhances processing efficiency
- Added to instruction cycle
- Processor checks for interrupt
  - Indicated by an interrupt signal
  - If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program



Role of Cache Memory and Memory Hierarchy





An example of a memory hierarchy.

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed. Cache memory is used to reduce the average time to access data from the Main memory. There are various different independent caches in a CPU, which store instructions and data.

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory** –
  It is external memory which is not as fast as main memory but data stays permanently in this memory

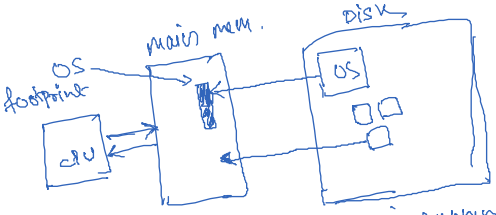
What is an Operating System

Operating system is a system software which acts as an intermediary between user, user applications and the computer hardware . OS is basically a program that help run the other programs on the computer, and makes it : end user programmable. The three main **functions** of an operating system are :

1. Resource Management
2. Establish a user interface.
3. Execute and provide services for application software.

The operating system can manage these by performing tasks such as :

1. File management
2. Process management
3. Memory Management
4. Handling Input / Output
5. Controlling peripheral devices



OS helps in running other Program

① it loads the program in memory
② Runs the program with help of Processor
③ Stores the result in persistent disk
④ unloads program, releases memory for further usage.

* When Computer starts the OS is also loaded in main memory from the disk. (Harddisk)

Main objectives of OS.

① Convenience : • Ease of operation
• More convenient to use computer

OS handles many tasks and make it convenient to just start running a Computer

② Efficiency : provides efficient resource mgmt

③ Ability to Evolve : An OS must be developed in a way to allow user to use the OS without interference when there is an upgrade

④ Protection / Access Control : provides security by access control and built in malware protection application.

Operating System Extras

A bootstrap program is loaded in ROM of the computer, known as firmware.
[* it initialises all aspect of the system] → we understand this better in boot sequence
* firmware loads the operating system kernel and starts execution

* What are System Daemons? or System Process?

A daemon (pronounced DEE-muhn) is a program that runs continuously and exists for the purpose of handling **periodic** service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate. Traditionally, the process names of a daemon end with the letter *d*, for clarification that the process is in fact a daemon, and for differentiation between a daemon and a normal computer program. In windows os these are simply called processes.

Fun Fact : The term was coined by the programmers at MIT's Project MAC. They took the name from Maxwell's demon, an imaginary being from a thought experiment that constantly works in the background, sorting molecules

Some of the examples of daemon processes are :

Httpd
Ntpd
Ftpd
Sshd
Crond

Boot ORder Sequence

Power on self test : When Computer is switched on it runs the test and

POST

Boot ORder Sequence

(0) POST : Power ON Self test : When Computer is switched on it runs the test and signals if any of the key component is not Powered ON.

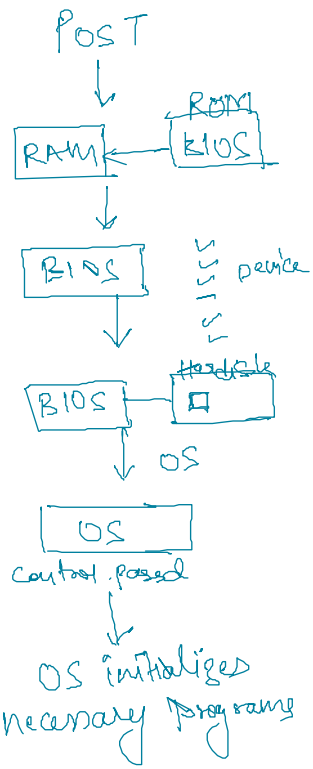
(1) As soon as the CPU Powers on it fetches the startup program (BIOS) from ROM and loads into the RAM

(2) The program BIOS then starts and checks the availibity of all the devices that are attached to computer, like harddisk, RAM, keyboard, monitor (display) checks if every device is running.

(3) BIOS then initialize the boot sequence, and it will look for either an OS or a boot Loader in the MBR (master boot Record) track 0 Sector 1 of the harddisk (Secondary memory)

(4) BIOS will fetch the operating system and loads it into RAM

(5) The BIOS then transfers the control to operating System.



Modes of Operating System

User Mode : Mode -1 : User mode is basically normal mode or less privilege mode where process running do not have access to all the hardware r esources . In User mode, the executing code has no ability to *directly* access hardware or reference memory. Due to the protection afforded by this sort of isolation, crashes in user mode are alway s recoverable. Most of the code running on your computer will execute in user mode.

Kernel Mode : Mode 0 : In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. **It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system.** Crashes in kernel mode are catastrophic; they will halt the entire PC.

Operating system has two modes : User mode and Kernel mode. A system (basically processor) is in user mode when operating sys tem is running a user application (e.g. notepad). **The transition from user to kernel happens when the application requests the help of operating system to through an interrupt or a system call.**

Example of Windows Operating System :

When you start a user-mode application, Windows creates a *process* for the application. The process provides the application with a private *virtual address space* and a private *handle table*. Because an application's virtual address space is private, one application cannot alter data that belongs to another applic ation. Each application runs in isolation, and if an application crashes, the crash is limited to that one application. Other applications and the operating system are not affected by the crash.

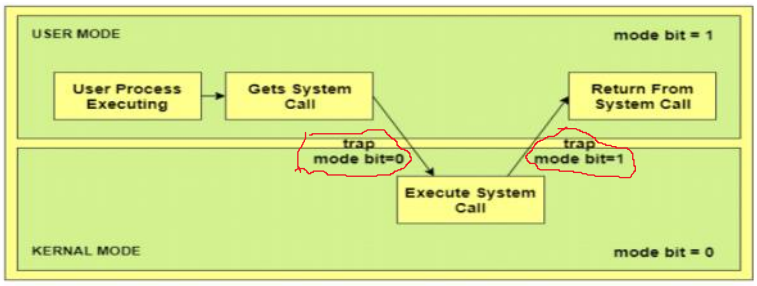
In addition to being private, the virtual address space of a user-mode application is limited. A processor running in user mode cannot access virtual addresses that are reserved for the operating system. Limiting the virtual address space of a user-mode application prevents the application from altering, and possibly damaging, critical operating system data.

All code that runs in kernel mode shares a single virtual address space. This means that a kernel-mode driver is not isolated from other drivers and the operating system itself. If a kernel-mode driver accidentally writes to the wrong virtual address, data that belongs to the operating system or another driver cou ld be compromised. If a kernel-mode driver crashes, the entire operating system crashes.

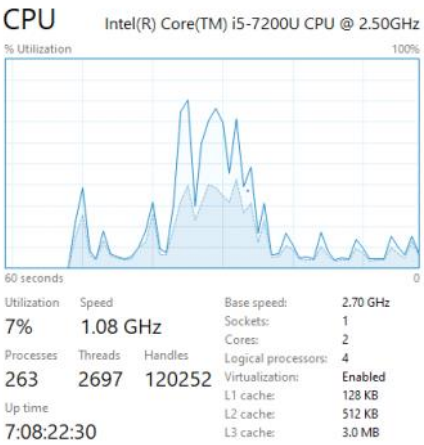
Transition from User mode to Kernel Mode

The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode. The syste m starts in kernel mode when it boots and after the operating system is loaded, it executes, all user applications in user mode. There are some privileged instructions that can only be executed in kernel mode (like Interrupts / memory management apps etc). If the privileged instructions are executed in user mode, it is illegal and a trap is generated. The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

when a privileged instruction need to be run in user mode we need an interrupt or system call



Trap = Execution



The lack of a dual mode i.e user mode and kernel mode in an operating system can cause serious problems. Some of these are

- A running user program can accidentally wipe out the operating system by overwriting it with user data.
- Multiple processes can write in the same system at the same time, with disastrous results.

- Mode bit provided by hardware – Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as privileged, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user mode
- Software error or request creates **exception** or **trap**

Different Services Provided by an Operating System

- Process Management
- Memory Management
- Storage Management
- Protection and Security

Process Management : A process is a program in execution. A program becomes a process when the executable files is loaded int o memory. Any process might require resources like CPU, memory ,I/O etc. All the process execution happens in a sequential fashion. Word, Browser, email package all are example or a process.

There are two type of process System process and user process. It is the job of OS to manage all the running processes of the system. It handles operations by performing tasks like **process scheduling** and such as **resource allocation.**

Below is the process architecture and this tells how processes are stored in memory.

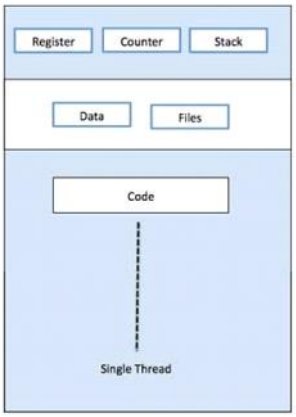
- A text segment, also known as a code segment or simply as text. It is one section of a process in memory which contains the executable instructions. Basically it contains the compiled program code.
- The data section stores global and static variables allocated and initialized prior to executing the main program.
- Heap is the segment where dynamic memory allocation usually takes place. The heap area begins at the end of the BSS segment and grows to larger addresses from there
- The stack: is used for local variables. Space on the stack is reserved for local variables when they are declared and the space is freed up when the variables go out of scope. Stack is also used for function return values.

Process Architecture

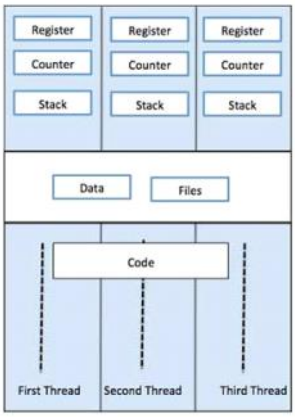
THREADS

- Definition: A thread is a single sequential flow of control within a program. a thread is considered lightweight because it runs within the conte xt of a full-blown program and takes advantage of the resources allocated for that program and the program's environment. A thread must have its own execution stack and program co unter. The code running within the thread works only within that context.
- A Web browser is an example of a multithreaded application. Within a typical browser, you can scroll a page while it's downlo ading an applet or an image, play animation and sound concurrently, print a page in the background while you download a new page, or watch three sorting algorithms race to the finish.
- **Each thread belongs to exactly one process and no thread can exist outside a process** . Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memo ry multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data



Single Process P with single thread



Single Process P with three threads

VIRTUAL MEMORY

Virtual memory is a mechanism to load and execute the programs/process from secondary memory. Size of a primary memory is limited and OS provides lot's of applications to run simultaneously and we can switch from one to another seamlessly, when program is in memory.

★ How it works?

A process may be broken into no. of pieces and these peices need not to be allocated continously in the main memory during execution. [The address of each instruction of a program can be fed to program counter]

★ All memory references within a process are logically addressed that are dynamically translated to physical address during runtime.

For example:

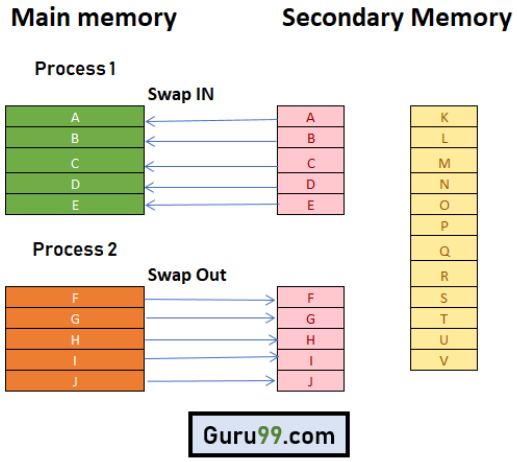
Let's assume that an OS requires **300 MB of memory to store all the running programs**. However, there's currently **only 50 MB of available physical memory** stored on the RAM.

- The OS will then set up **250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM)** to manage that 250 MB.
- So, in this case, the **VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required**.
- The **OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM**, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

A demand paging mechanism is used to swap in and swap out the parts of process that are required by the CPU.

∴ Virtual memory is also called as swap memory

Operating Systems suggest to create a separate partition for swap



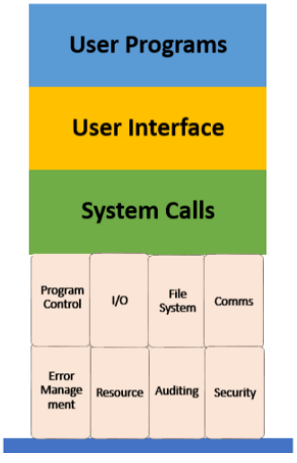
A demand paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.

SYSTEM CALLS

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.
System call offers the services of the operating system to the user programs via API (Application Programming Interface). Sys tem calls are the only entry points for the kernel system

For example if we need to write a program code to read data from one file, copy that data into another file. The first inform a the two files, the input and output files.
In an interactive system, this type of program execution requires some system calls by OS.

- First call is to write a prompting message on the screen
- Second, to read from the keyboard, the characters which define the two files.



Types of System calls

Here are the five types of system calls used in OS:

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications

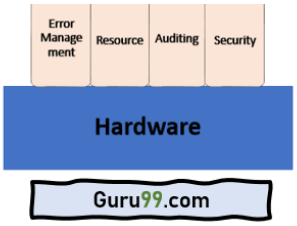
Process Control

This system calls perform the task of **process creation, process termination**, etc.
Functions:

- End and Abort
- Load and Execute

This system calls perform the task of **process creation, process termination**, etc.
Functions:

- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signed Event
- Allocate and free memory



File Management

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.
Functions:

- Create a file
- Delete file
- Open and close file
- Read, write, and reposition
- Get and set file attributes

Device Management

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.
Functions

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance

It handles information and its transfer between the OS and the user program.
Functions:

- Get or set time and date
- Get process and device attributes

Communication:

These types of system calls are specially used for inter-process communications.
Functions:

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

Categories	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
Device manipulation	SetConsoleMode() ReadConsole() WriteConsole()	loctl() read() write()
File manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	Open() Read() write() close!)
Information maintanence	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	Pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup ()	Chmod() Umask() Chown()