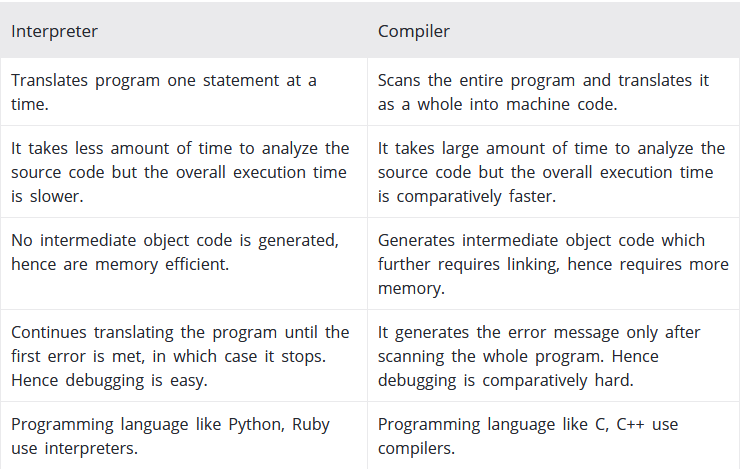
# Compilers Vs Interpreters



# Virtual Machines

Virtual machines are emulations of the respective levels to help humans interact with the real underlying machine. With each higher level the machine is much easier to interpret. For an example 1 line of code in one of the higher levels may result in hundreds of lower level calculations. Each of these levels are also built upon the predecessor level.

## Level -1

The wafer level of computer architecture,

## Level 0 Digital Logic

Analog components such as transistors and gates exist here. Gates computer new outputs such as AND or OR. These gates can be combined to store memories to form 1-bit memory. Groups of these are used to form registers which vary depending on the architecture (8, 16, 32, 64). Gates are also combined to form the main computing engine itself

## Level 1 Microarchitectural Level

A collection of registers, from 8 to 32 form a local memory, a control unit and a circuit called the Arithmetic Logic Unit which performs simple operations. Registers connect to ALU to form a data path, each data path / pathway is an instruction controlled often by a micro program or directly by the hardware.

## Level 2 Instruction Set Architecture

The instruction set architecture consists of several instructions defined by the hardware manufacturer. These can be found in the “Machine Reference Manual”. These are performed by the lower level.

## Level 3 OS Level

This is often considered a hybrid level, most of the language here is in the ISA level. New facilities are often here interpreted by an interpreter at level 2, which has been called an operating system. This level and below are not understood by Application programmers everything here is built to support higher levels. This level is written by a system programmer.

## Level 4 Assembly

First human friendly programming languages can be seen here Assembly, This language provides a way to create programs for level 1,2 and 3. Words and abbreviations are finally meaningful here.

## Level 5 Programming Languages such as C

High level languages exist here which are built upon the assembly language. Languages in this level are often generally translated down to level 4 language by a compiler while sometimes they are occasionally interpreted such as Java.

# Eight Great Ideas in Computing

## Moore’s Law

Moore’s Law states that the number of transistors on a chip doubles every 18-24 months.

### Use of Abstraction to Simplify

Programmers and Architects invented techniques to become more productive with ever expanding resources with Moore’s Law.

### Making’s Common Case Fast (Amdahl’s Law)

Making the common case as faster as possible is better than optimizing rare cases. It is also simpler to improve common case than rare cases.

### Performance in Parallelism

Designing computing to run in parallel to improve performance.

### Pipelining

Technique where multiple instructions are overlapped in instruction to help reduce time in computation.

### Performance Through Prediction

Predicting the future moves that a processor does with high degree of accuracy speeds up workflow if the recovery mechanism doesn’t take too long.

### Memory Hierarchy

Different levels of memory are introduced as memory cannot be fast, large and cheap all at the same time. Memory is placed in a hierarchy with fastest and smallest at top and slowest and largest at the bottom.

### Dependability via Redundancy

Computers need to be dependable, fail safes are put in place to predict and tackle on failures.

## CPU

CPU is responsible for many tasks,

Fetching Instructions

Decoding them

Executing them one after another

A bus is a collection of parallel wires used for communications, and data transfer, they are used to connect I/O devices, memory and other processing units.

The CPU is composed of several distinct Parts

Control unit fetches instructions from main memory and determines their type and what to do

Arithmetic Logic unit performs operations such as addition subtraction multiplication and division. Also, logical operations such as AND OR NOT SHIFT and Rotate

## Registers

Registers are high speed memory locations used to store temporary results and certain information.

They are the fastest type of memory and are internal to the CPU, they are used for operands.

They are limited in size and number, depending on architectural decision.

## Multiple Registers

Multiple registers existing such as Program Counter, or also called the Instruction Pointer holds the concurrent address of the next instruction to be fetched for execution

Instruction Register holds the instruction concurrently being executed

Status Register reflects the outcome of the execution of the previous instruction

Stack pointer is used to implement subroutine calls, returns and creating stacks

## Fetch-Decode-Execute

Fetch phase includes fetching the code from the Memory and placing it into the CPU.

Determining the pathway is called the instruction decode phase.

Instruction Execution Phase compromises of enabling and disabling various parts of the CPU until the electrical signals flower through the CPU until it settles into a stable state.

## DRAM

Also known as Dynamic Random-Access Memory.

Stored value is represented by a capacitor’s charge state. A capacitor is like a bucket which leaks energy which needed to be quickly refreshed to prevent memory from erroring.

It needs to be read and written 1000 times a second by the dedicated memory controller.

## SRAM

Also known as Static Random-Access Memory.

Two cross coupled inverted store a single bit

Bit maintained by a feedback path, 4 transistors for storage 2 for access.

SRAM is fast and expensive, used for cache, while DRAM is cheaper and slower used for main memory.

## Need for Memory Hierarchy

Memory can’t infinitely fast big etc, to get both speed and size it cannot be achieved with a single level. Therefore, multiple levels exist to satisfy this. L1 is on core memory, L2 is on chip memory, and L3 exists near the bus. With each level getting bigger and smaller

## RAM

Computer expresses memory as binary numbers. If an address has m bits the maximum number of addressable locations is 2^m.

Nearly all computers are standardized on an 8-bit location called a byte. Bytes are grouped into words, 32-bit word has 4 bytes/word while 64-bit has 8 bytes/word

Modern memory is reliable, modern controllers check for errors on start-up. However, some memory chips requiring even greater reliability have built in error checking and parity checks to discover errors.

## Parity Checking RAM

Chips have extra bit for every byte of data, a parity is set i.e. an odd or an even that checks and compares the parity bit by adding the byte to see if its even or odd. A comparison is made between the addition and parity bit to check if something has gone wrong. For e.g 11011011 being the byte in question, the ninth byte would be set to 0 to ensure that the amount of 1s is even. While for 11010011 it would be 1. It is the opposite for an odd parity scheme.

Parity checks are limited in the sense they detect errors but fail to correct them. Also, multiple bit errors are statistically unlikely to occur within the same byte but still possible which would be left undetected.

## Error Correcting Codes (ECC)

These are used to identify multi bit errors and correctly single bit errors.

## Words and Word Alignment

A word is a fixed size piece of data manipulated as a single unit by an instruction set. The number of bits is known as word length. Word lengths reflects the structure and operator of computer register sizes and maximum data transfer between processor and memory in a single operation and largest address size.

In modern computers words are usually 4 to 8 bytes sized. Words are aligned on natural boundaries for example a 4-byte word would be found at addresses 0, 4, 8 and 8 byte words are similarly found at 0, 8, 16, 24 etc not at 1, 2, 3…

## Endianness

The endianness refers to how a word is stored.

In a big endian the most significant byte is stored in the lowest memory slot, second most significant in next etc

While in little Endian least significant byte is stored at the lowest address, etc.

## Locality of Reference

There are two kinds

Temporal Locality, which is the same data, referred to multiple times over a short period of time, this data should be in fast access storage to speed up subsequent accesses.

Spatial Locality, which is a particular set of addresses that are accessed multiple times over a short period of time, nearby address should be stored in fast access storage to speed up subsequent addresses

Knowledge of locality offers chance of predictability and thus performance optimization using prefetching and caching. Locality follows from program structure i.e for, while, if etc and data structures

## Von Neumann Machines

Consist of 3 basic hardware components, A CPU, Main memory and an I/O system.

## Samphire Assembly Cheat sheet

### Arithmetic and Logic

ADD, SUB, MUL, DIV, INC, DEC, AND, OR, XOR

### Jumps and Conditionals

Jumps are used in combination with labels while calls are used with memory addresses

JMP unconditional jump

CMP compares two values through subtraction, x-y if cmp x, y

N variation is the if the flag is not set

JZ jump if zero

JS jump if sign flag is set

JO jump if overflow is set

### Procedures

Mov, moves a specific number into a register, can be used with [] to get value from specified address.

Call, calls for a specific region in memory used for subroutines

Pushes the register onto stack

Pops inserts stack intro register

Org generates code starting from the specified address

Db, defines a byte

End, ends the process

In 01 inputs from keyboard into Al

Out outputs from Al to whatever

## Binary and Hexadecimal

We use these counting systems for assembly and low-level computing

Counting in hex, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Counting in binary, 0, 1, 10, 11, 100, 101, 110, 111, 1000

## OSM

The Operating System Machine layer adds a new variety of new instruction beyond the ISA level. These new instructions are available to application programmers it also contains nearly all ISA level instructions. This layer also includes a new set of instructions called system calls. A system call invokes a predefined service such as reading a data file.

Not all computers have operating systems, dedicated hardware such as calculators or microwave ovens have very simple input and output methods which run only one program.

## What does an OSM do?

It manages all hardware and software resources such as processor, memory and disk space. Various programs compete for the attention of the CPU, memory and I/O, it is the OS’s job to make sure each application gets necessary resources.

It provides a stable and consistent way for applications to deal with hardware without having knowledge of hardware. For e.g. providing drivers and APIs.

The OSM provides a consistent application interface. A consistent application programming interface allows a software developer to run programs with high level confidence that it will work on other similar computers running the same OS. The API provided by the OS ensures that programs runs on thousands of hardware vendor’s PCs with different combinations of hardware combinations.

## Types of Operating Systems

Real-Time Operating System (RTOS). Used to control machinery and scientific instruments, and industrial systems. Very important in management of resources and that everything gets executed in a precisely same amount of time

Single-user, Single task – Designed to manage a computer so one used can effectively do one thing at a time. The Palm OS for handhelds is a good example

Single-user, Multi-tasking – This type is used for most desktop and laptop computers. E.g windows 98, MacOS will allow a single user to execute multiple tasks

Multi-user systems allow many different users to take advantage of a computer’s resources simultaneously.

## Tasks of an OSM

### Processor Management

Two tasks, ensuring that each process and application receives enough of processors time to function properly, and use as many cycles for real work as possible. A basic unit of work is the process or thread.

In a single tasking system, the schedule is simple.

Interrupts are special signals sent by hardware or software to the CPU to get its attention

Sometimes the OS will mask interrupts meaning ignore interrupts from some sources to get a task done faster.

There are certain interrupts that are none maskable such as error conditions or problems with memory. These are so important they cannot be ignored (NMIs)

All the information needed to keep track of a process when switching is kept in a data package called a **process control block**. This typically contains Process ID (PID), pointers to the locations of the program, register content, states of various flags and switches, status of all I/O devices needed by process, priority of process, list of files opened by process

Process switching happens without direct user interference, the OS requires some CPU cycles to perform the **context switching**.

If too many processes are started most of the CPU cycles will be used to perform switching with little real work done. This is called **thrashing**. This requires user interference to stop and bring the system back in order.

One way to reduce trashing is by using threads. A thread performs all the CPU-intensive work of a normal process but doesn’t require the extensive process control block of a regular process. A process may start many threads or other processes, but a thread cannot start a process.

In a system with two or more CPUs the OS must divide the workload among the CPUs. This brings rise to two different OS types, Asymmetric and Symmetric OS. An asymmetric OS divides itself on one core than applications on other remaining cores. While a symmetric OS divides itself evenly on all cores

### Memory and Storage Management

The OS tries to accomplish two goals, each process has enough memory to execute and not to interfere other processes, and that to different types of memory must be used properly to run the system most effectively

**Memory boundaries** are setup by the OS for individual applications, applications are loaded into memory in block sizes determined by the OS

When RAM is scarce, virtual memory is created on the hard disk, this is called the **virtual memory management**. (memory hierarchy) High speed cache (controllers predict what to grab from main memory), Main memory, Second Memory e.g. Disk, tape serves as virtual ram.

### Device Management

Device management which helps operate all devices connected on the motherboard called a **driver**. A driver is a translator between electrical signals of the hardware and high-level programming languages. Drivers take data files and turn them into streams of bits or series of lasers in a printer. Since there are differences the OS manages drivers and priorities for which ones are ran and at what time. The drivers are separate from the OS so hardware can be changed, Input and output between different devices and the processor is typically performed using **queues** and **buffers**.

### Application Interface/ User Interface

Just as drivers are to hardware, application interfaces are to programmers. Application Program interfaces allow to make use of functions of the computer and operating system.

One use case is a user interface which brings structure and consistent to the interactions between a user and computer. In recent years **graphical user interfaces (GUIs)** are used for interaction. Traditionally Unix used a text-based shell interface, which is preferred mode of operation among professionals.

The user interface is a program or set of them forming a layer above the OS

Core operating system functions of the management of the computer system lie in the kernel. The ties between the operating system kernel and the user interface, drivers, utilities and other software define the differences in OS(s).

# Virtual Memory

## Origin

Memory was expensive and virtual memory was the solution to this problem. Disks were used to help run larger programs.

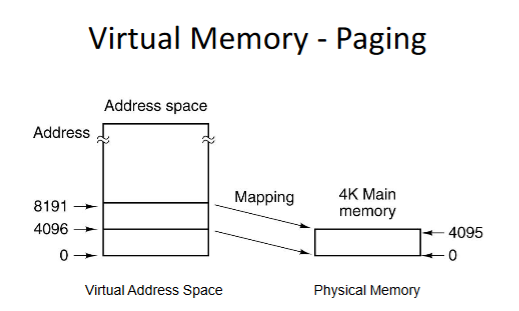
The programmer divided the program into several pieces called **overlays**. Each which could fit in the memory. To run this program, first overlay was brought into memory and ran, than the second overlay was read in and ran and so on.

## Virtual memory

In 1961 an automatic system was created and by 1970 it was available on most.

## Paging

For example, using a 16-bit system with 4k memory, which would have 4096 unique locations that can be accessed. Had two address spaces, **useful address space** and the **useless address space** useless being that it didn’t correspond to actual memory locations. So, whenever a memory address at 4096 was accessed, the memory word at location 0 is used, and for 8191 is referenced location 4095 is used and so forth. Hence, we made a mapping from the address space onto the actual memory location as shown.



So, if a program is larger than 4096 bytes it will error unless you use virtual memory. Which will load the next page which is next block of 4K memory onto the actual physical memory. Automatic overlaying is called paging and the chunks of program read in from disk are called pages.

To avoid confusion between **virtual address space** and **physical address space** a **memory map/page table** specifies for each virtual address what corresponds to a physical memory address.

It is important to sync data in memory with data on the disk when changes are made to the copy in the main memory. They should be reflected on the disk eventually.

The virtual address space is broken up into equally sized pages. Page sizes are represented in power of 2^k so the addresses can be represented in k bits.

Pieces of main memory into which the pages go are called **page frames**. In the example above the main memory contains only a single page frame.

### Implementation

One possible way to divide 64 KB of virtual memory in a 16-bit system is an address space which is 16 pages of 4 KB each. I.e. 16 pages, this leads to a limitation of memory to 32 KB because it is the physical extent of the machine and rest is allocated to other programs.

To do this mapping automatically a device has an **MMU** a **Memory management unit**.In a 32-bit virtual space, the address is separated into a 20-bit virtual page number with a 12-bit offset within the page due to 2^12 being 4K. With a 15-bit output register where the 3 extra bits are used for page number to put it in the right page frame.

The MMU checks if the page is already in the memory by looking at the present/absent bit in the page-table entry.

Refer to lecture 20 yellow sheets

### Demanding Paging

When a reference to a page that’s not present in main memory a **page fault** occurs. The OS reads in the page and repeats the instruction that caused the fault. The method of obtaining a virtual memory is called **demand paging**. A page is brought into memory only when a request for it occurs not in advance.

### Working Set Model

Most programs do not reference their address space uniformly – references tend to cluster on a small number of pages. This concept is called the **locality principle**.

Memory reference may fetch an instruction, data or it may store data. At any instant in time, t, there exists a set of consisting of all pages used by k most recent memory references this is called the **working set**.

The working set normally varies slowly with time. It is possible to make a reasonable guess at what pages are needed when the program is started/restarted based on the working set when it was last stopped. These pages can be loaded in advance before starting the program assuming they fit.

### Page Replacement Policy

Programmers rarely know which pages are in the working set so the OS discoverers this set dynamically. To make room for a new one some page will generally have to be sent back to the disk. The OS predicts the least useful pages in memory and sends them back. These pages are evicted by the **Least Recently Used Algorithm** or by the **First-In-First-Out Algorithm** which removes the oldest page. The FIFO Algorithm is decided by a built-in counter that is set 0 once it is brought in at first. For each page fault the counter is incremented until the page with highest counter value is chosen. These are used to cause the least amount of disruption to the running program.

These algorithms have their own problem if the working set is larger than the number of available pages. No algorithm will give good results and page faults will be frequent, although LRU tend to minimize the number of page faults.

**Thrashing** a program that generates faults frequently and continuously is said to be thrashing. If a program even one using a large virtual address space has a small slowly changing working set that fits in available main memory it will perform well.

If a page about to be evicted has not modified since it was read, (I.e. a program) it is not necessary to write back onto disk. If it has been modified since it was read in, the copy on the disk will be out of the date and the page must be written to the disk.

# Cloud Computing

Definition:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud computing has essentially five essential characteristics, three service models and four deployment models.

## Essential Characteristics

**On demand self-service**: requires no human interaction with the service provider.

**Broad Network Access**: The capability to be accessed by a client

**Resource Pooling**: Resources are pooled to serve multiple consumers with dynamic reassignment of resources to the users’ needs

**Rapid Elasticity**: Capabilities can be elastically provisioned and released to scale outward and inward commensurate with demand.

**Measured Service:** Cloud systems automatically control and optimize resource use by monitoring resource consumption

## Service Models

**Software as a Service:** The consumer uses the provider’s applications running on a cloud infrastructure. The consumer accesses it through various client devices through a client interface.

**Platform as a Service:** The consumer can deploy consumer created or acquired application onto the cloud. The consumer doesn’t manage underlying cloud infrastructure as in previous example.

**Infrastructure as a Service:** The consumer can deploy and run arbitrary software, including OS and application. The consumer doesn’t manage the hardware but manages anything software related i.e OS, applications etc.

## Types of Clouds

**Private Cloud:** The cloud service is provisioned for exclusive use by a single organization.

**Community Cloud:** The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organisations that have a shared concern.

**Public Cloud:** The cloud infrastructure is open for use by general public

**Hybrid Cloud:** Cloud infrastructure is a composition of two or more distinct cloud infrastructures that remain unique entities but are bound together by standardization or proprietary technology that enables data and application portability.

# Hardware Virtualization

**Hardware Virtualization** is a combination of hardware and software that enables simultaneous execution of multiple operating systems on a single physical machine.

To use each **virtual machine** running on the host computer appears to be a standalone computing system. The **hypervisor** is a software component much like an operating system kernel creates and manages instances of virtual machines. The hardware provides information to the hypervisor so that it can implement sharing policies for the CPU, Storage and I/O

## Benefits of Virtualization

SysAdmins can place virtual machines on the same physical server and move running virtual machines between servers to better distribute total load. Virtual machines give admins fine grained control over I/O device access e.g. bandwidth of a network port can be partitioned based on user-service levels.

## Challenges in Virtualization

No instruction should be able to access resources outside the current virtual machine

Memory access instructions must only access physical memory allocated to the currently executing virtual machine. Additional page-mapping facilities are also created for the virtual machine to map virtual memory to the host machine physical memory pages

I/O must not directly access physical I/O devices

Fine grained I/O control is typically implemented with interrupts to the hypervisor. Certain virtual machines can access certain I/O devices, the guest operating systems are expected to support these devices.

# Unix

# Windows

# Instruction Set Architecture Level

The ISA is between Microarchitecture and the OS level, this level was historically developed before any other level. (The OG Level) This level is sometimes referred to as the Architecture. The ISA serves as an interface between software and hardware.

While it’s possible to have hardware directly execute C, C++, Java or some other high-level language it is not often a good idea.

It serves its purpose as an intermediate level between high level programming languages and machine languages, it is an interface between compilers and hardware. A good ISA provides a long last instruction set that will serve a-many generations of computers, it also has to be backwards compatible. Once an efficient instruction set has been created it will most likely stay there for good.