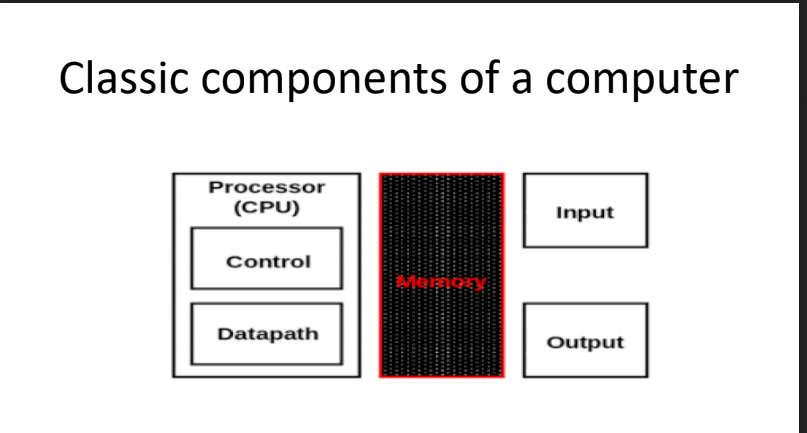
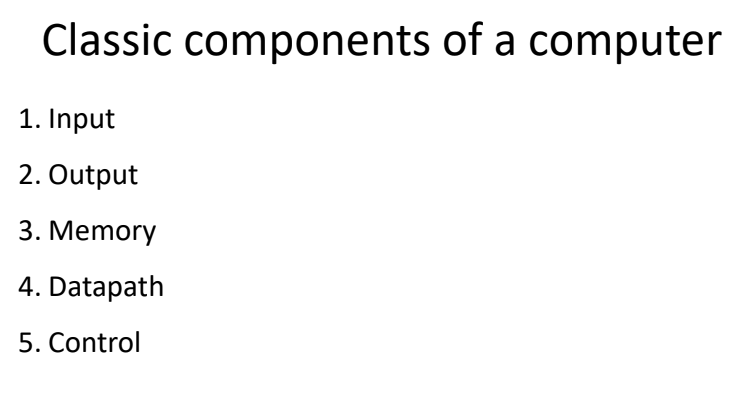
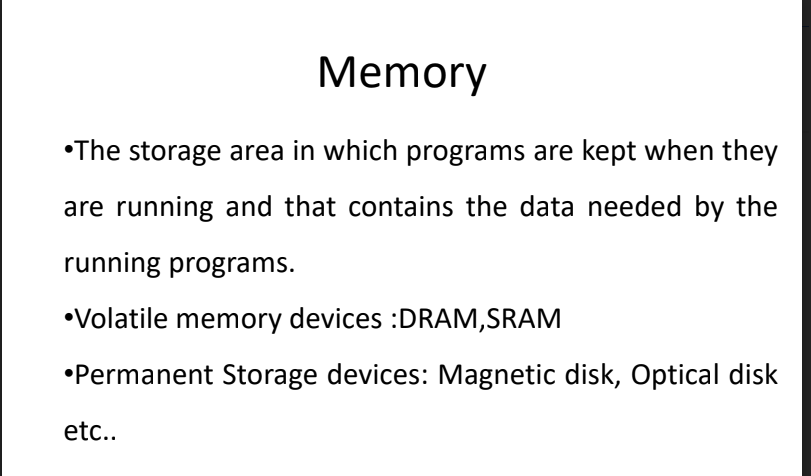
**Computer Architecture**

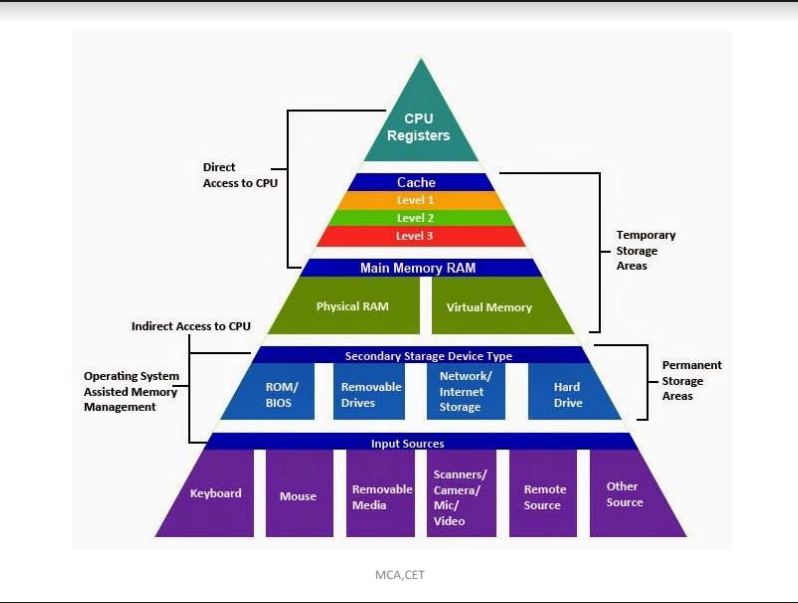
****

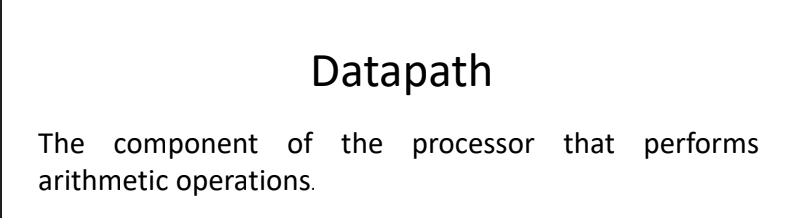
****

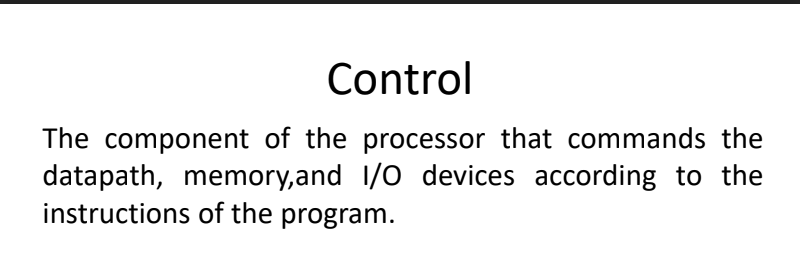
**Input Unit**: Accept data or instructions as input

**Output Unit**: This gives results in the form of output









* **Eight Great Ideas in Computer Architecture**
* **Design for Moore’s Law**

The one constant for computer designers is rapid change, which is driven largely by Moore’s Law. It states that integrated circuit resources double every 18–24 months.

Moore’s Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel.

As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project.

We use an “up and to the right” Moore’s Law graph to represent designing for rapid change.



* **Use Abstraction to Simplify Design**

A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels.

* **Make the Common Case Fast**

Making the common case fast will tend to enhance performance better than optimizing the rare case.

* **Performance via Parallelism**

Since the dawn of computing, computer architects have offered designs that get more performance by performing operations in parallel.

* **Performance via Pipelining**

A particular pattern of parallelism is so prevalent in computer architecture that it merits its own name: pipelining.

* **Performance via Prediction**
* **Hierarchy of Memories**

Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is oft en the majority of computer cost. Architects have found

that they can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom.

* **Dependability via Redundancy**

Computers not only need to be fast; they need to be dependable. Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures.

An **operating system**  interfaces between a user’s program and the hardware and provides a variety of services and supervisory functions.

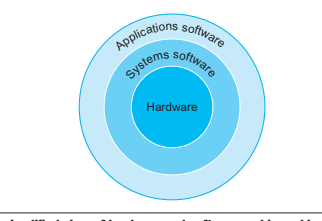
Among the most important functions are:

■ Handling basic input and output operations

■ Allocating storage and memory

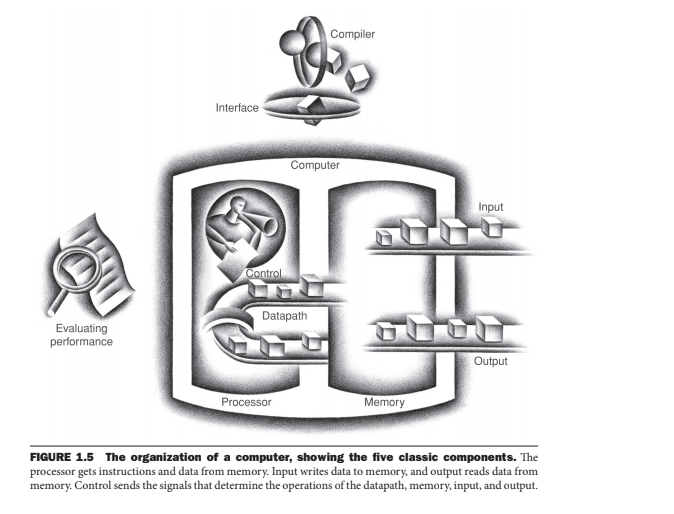
■ Providing for protected sharing of the computer among multiple applications using it simultaneously.

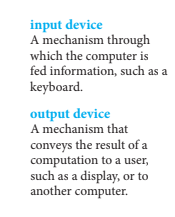
Examples of operating systems in use today are Linux, iOS, and Windows.

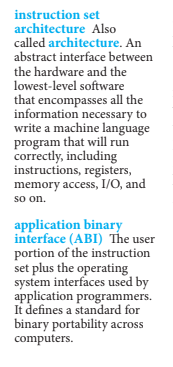


The combination of the basic instruction set and the operating system interface provided for application programmers is called the **application binary interface**

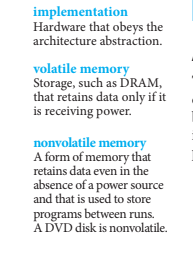
**(ABI).**





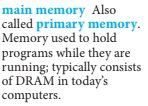


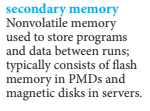
One of the most important abstractions is the interface between the hardware and the lowest-level soft ware. Because of its importance, it is given a special name: the **instruction set architecture, or simply architecture**, of a computer. The instruction set architecture includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on.

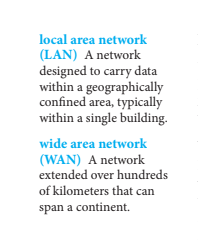


If we were to lose power to the computer, however, everything would be lost because the memory inside the computer is volatile—that is, when it loses power,it forgets. In contrast, a DVD disk doesn’t forget the movie when you turn off the power to the DVD player, and is thus a nonvolatile memory technology.

To distinguish between the volatile memory used to hold data and programs while they are running and this nonvolatile memory used to store data and programs between runs, the term main memory or primary memory is used for the former, and secondary memory for the latter. Secondary memory forms the next lower layer of the memory hierarchy. DRAMs have dominated main memory since 1975, but magnetic disks dominated secondary memory starting even earlier. Because of their size and form factor, personal Mobile Devices use fl ash memory, a nonvolatile semiconductor memory, instead of disks.





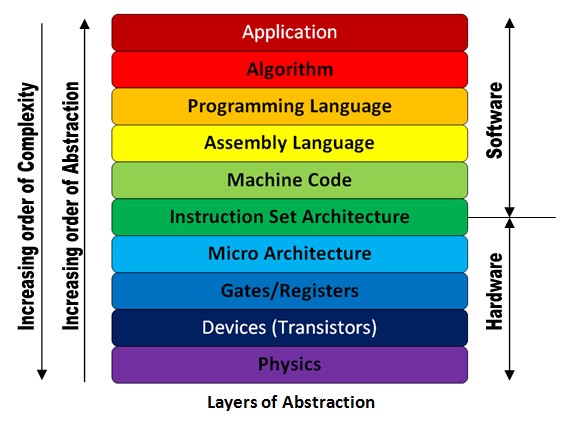


**Layers of Abstraction in Computer System**

Computer System is divided into two functional entities. Hardware and Software are two functional entities of computer system. Operating system is the link between hardware and software. There are a certain layers in computer system through which a process goes to perform a task. Here we will discuss about each layer.

Following are the different layers of abstraction in computer system:

* **Problem Statement –**  
  Problem Statement is stated using natural language. It may be ambiguous or imprecise. It is basically the user’s requirement from the system.
* **Algorithm –**  
  Algorithm is the step by step procedure to perform a specific task. It is guaranteed to finish. It has definiteness, effective computability and finiteness.
* **Program –**  
  Program expresses the algorithm using a computer language such as high level language and low level language. User writes the code for their problem statement.
* **Instruction Set Architecture –**  
  Instruction set architecture specifies the set of instructions the computer can perform using data types and addressing modes.
* **Micro-architecture –**  
  Micro-architecture is the detailed organization of a processor implementation.
* **Logic Circuits –**  
  Logic circuits combine basic operations to realize micro-architecture.
* **Device –**  
  Device has the properties of materials and manufacturability.



A **transistor** is simply an on/off switch controlled by electricity. The integrated circuit (IC) combined dozens to hundreds of transistors into a single chip. When Gordon Moore predicted the continuous doubling of resources, he was predicting the growth rate of the number of transistors per chip. To describe the tremendous increase in the number of transistors from hundreds to millions, the adjective very large scale is added to the term, creating the abbreviation VLSI, for very large-scale integrated circuit.

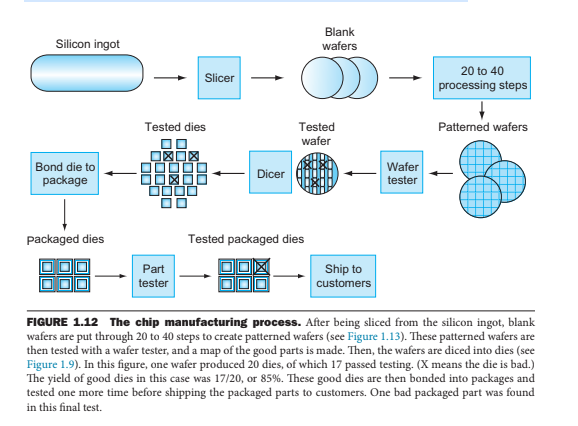
The manufacturing process for integrated circuits is critical to the cost of the chips and hence important to computer designers.

The process starts with a silicon crystal ingot, which looks like a giant sausage.

Today, ingots are 8–12 inches in diameter and about 12–24 inches long.

An ingot is finely sliced into wafers no more than 0.1 inches thick. These wafers then go through a series of processing steps, during which patterns of chemicals are placed on each wafer, creating the transistors, conductors, and insulators discussed earlier.

Today’s integrated circuits contain only one layer of transistors but may have from two to eight levels of metal conductor, separated by layers of insulators. The patterned wafer is then chopped up, or diced, into these components, called dies and more informally known as chips.

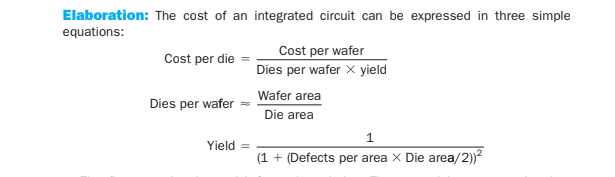


Dicing enables you to discard only those dies that were unlucky enough to contain the flaws, rather than the whole wafer. This concept is quantified by the yield of a process, which is defined as the percentage of good dies from the total number of dies on the wafer.

The cost of an integrated circuit rises quickly as the die size increases, due both to the lower yield and the smaller number of dies that fit on a wafer. To reduce the cost, using the next generation process shrinks a large die as it uses smaller sizes for both transistors and wires. This improves the yield and the die count per wafer. A 32-nanometer (nm) process was typical in 2012, which means essentially that the smallest feature size on the die is 32 nm.

Once you’ve found good dies, they are connected to the input/output pins of a package, using a process called bonding. Th ese packaged parts are tested a final time, since mistakes can occur in packaging, and then they are shipped to customers.

The cost of an integrated circuit can be expressed in three simple equations:

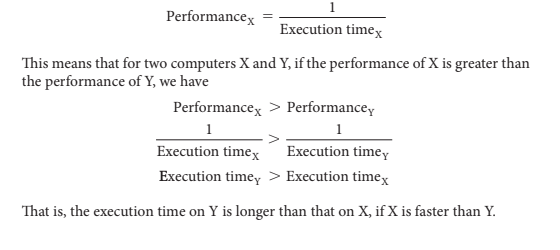
****

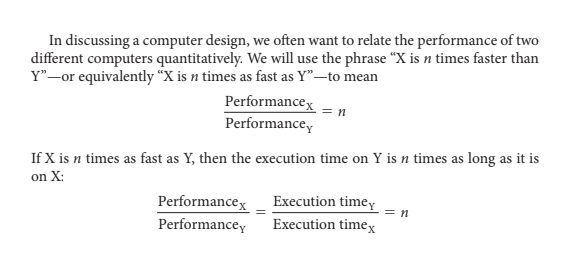
**Response time** also called **execution time**. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

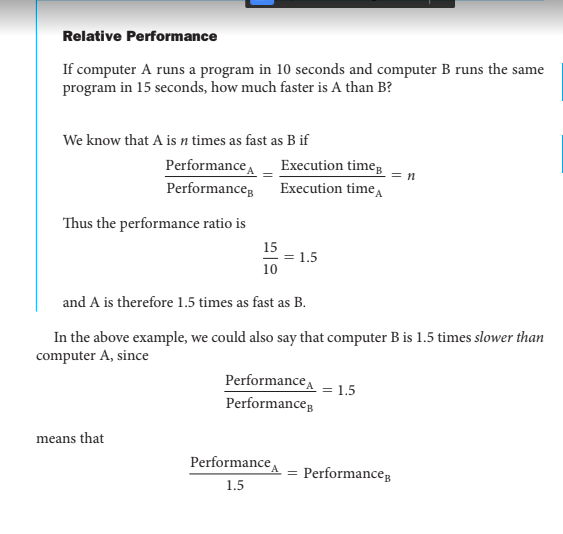
**Throughput** also called **bandwidth**. Another measure of performance, it is the number of tasks completed per unit time.

|  |
| --- |
| **NOTE:** **Decreasing response time almost always improves throughput.** |

To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:





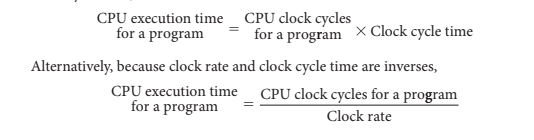
**CPU execution time** also called **CPU time**. The actual time the CPU spends computing for a specific task.

**User CPU time** : The CPU time spent in a program itself.

**System CPU time**: The CPU time spent in the operating system performing tasks on behalf of the program.

|  |
| --- |
| CPU execution time or simply CPU time, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs.  CPU time can be further divided into the CPU time spent in the program, called user CPU time, and the CPU time  spent in the operating system performing tasks on behalf of the program, called System CPU time.  All computers are constructed using a clock that determines when events take place in the hardware. These discrete time intervals are called clock cycles(ticks, clock ticks, clock periods, clocks, cycles)  Clock cycle: The time for one clock period, usually of the processor clock, which runs at a constant rate.  Clock period: The length of each clock cycle.  Clock rate: is the inverse of the clock period. |

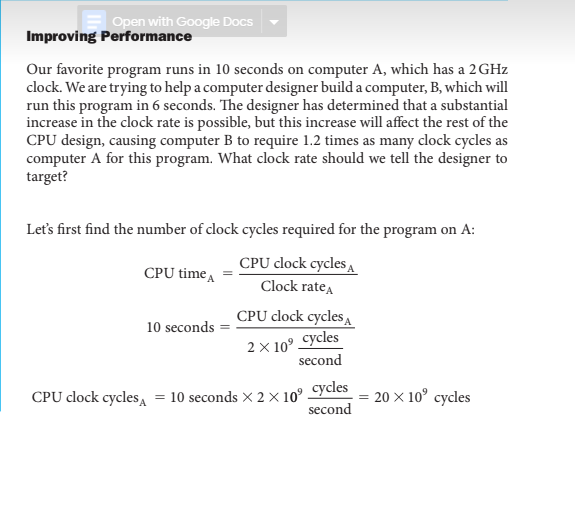
A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

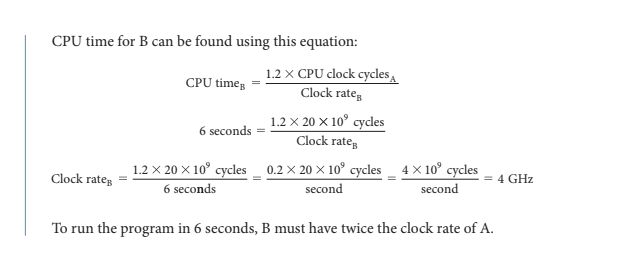


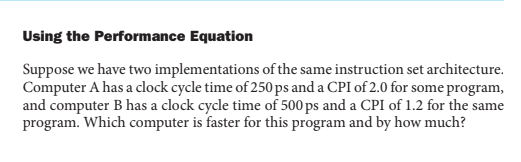
|  |
| --- |
| Clock cycles per instruction (CPI) :Average number of clock cycles per instruction for a program or program fragment. |

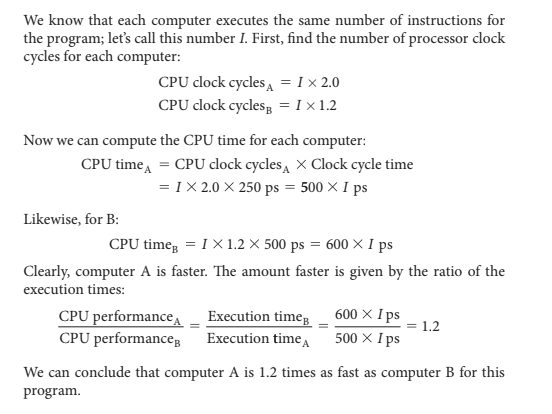
The number of clock cycles required for a program can be written as:





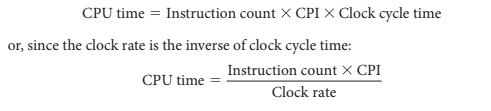


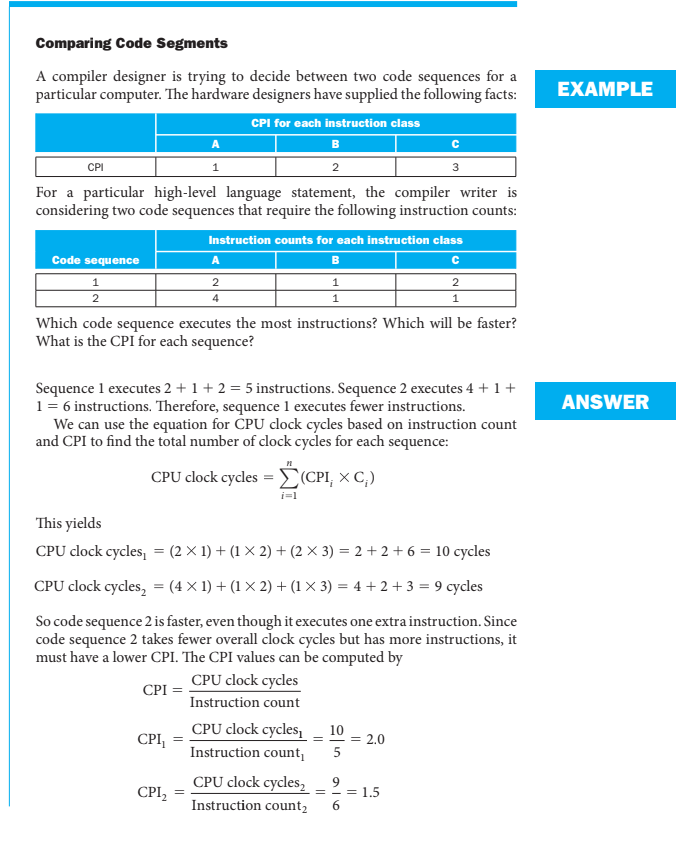


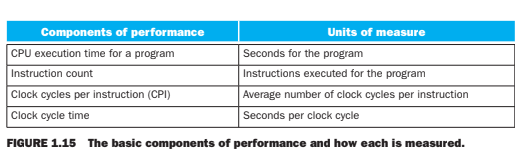


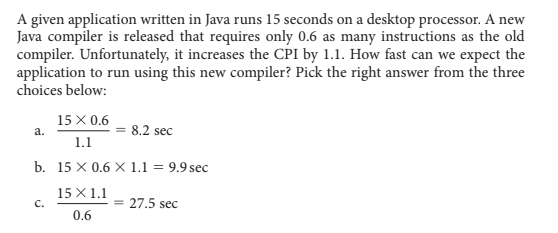
|  |
| --- |
| **Instruction count** :The number of instructions executed by the program. |

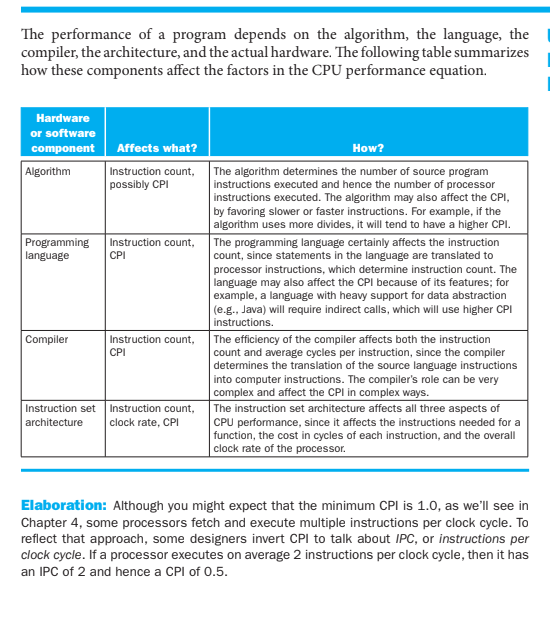
Basic performance equation in terms of instruction count CPI, and clock cycle time:











Stored-program concept : The idea that instructions and data of many types can be stored in memory as numbers, leading to the stored- program computer.

**Instruction Set Architecture**

ISA is an abstract interface between the hardware and the lowest-level software that encompasses all the information necessary to write a machine language program that will run correctly , including instructions , registers, memory access , I/O and so on.

ISAs differ based on the internal storage in a processor.

• Classification:

**1**. **Single accumulator organization**

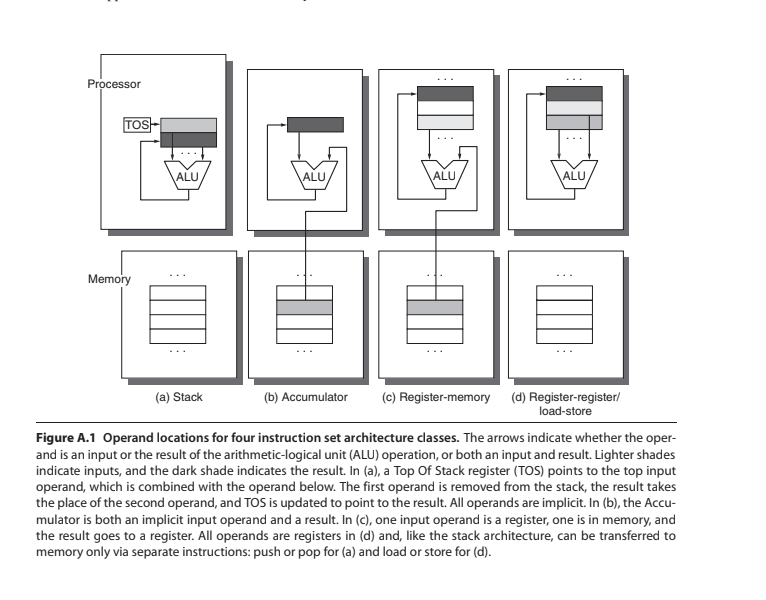
**2. General register organization**

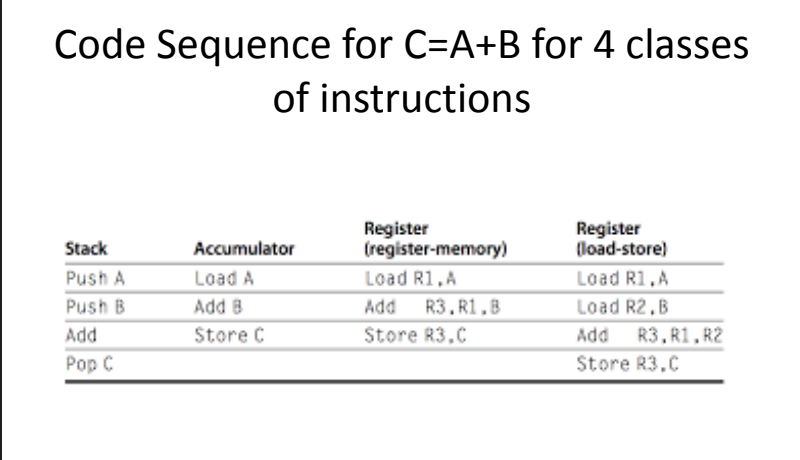
**(i) Register – register(load – store)**

**(ii) Register – memory**

**(iii) Memory – memory**

**3. Stack organization**





Different features that need to be considered :

A. Types of instructions

B. Types and sizes of operands

C. Addressing modes

D. Addressing memory

E. Encoding and Instruction Formats

**A. Types of Instructions**

* ***Data transfer instructions***

• Perform data transfer between the various storage places in the computer system, viz. registers , memory

• Both the instructions as well as data are stored in memory

• The processor needs to read the instructions and data

from memory.

* After processing, the results must be stored in memory

• Two basic operations involving the memory are needed, namely

**1**. **Load** (or Read or Fetch): transfers a copy of the

data from the memory to the processor

**2.** **Store(**or Write): moves the data from the

processor to memory.

• Other data transfer instructions are needed to

transfer data from one register to another or from/to

I/O devices and the processor

* ***Data manipulation instructions***
* Perform operations on data and indicate the computational capabilities for the processor.
* These operations can be arithmetic operations, logical operations or shift operations.
* Arithmetic operations include addition (with and without carry), subtraction (with and without borrow), multiplication, division, increment, decrement.
* The logical and bit manipulation instructions include AND, OR, XOR, Clear carry, set carry, etc
* You can perform different types of shift and rotate operations.
* ***Program sequencing and control instructions***
* Sequential flow of instructions: instructions that are stored in consequent locations are executed one after the other.

• Program sequencing and control instructions help you change the flow of the program.

Consider the task of adding a list of n numbers.

Move DATA1, R0

Add DATA2, R0

Add DATA3, R0

Add DATAn, R0

Move R0, SUM

**Loop Instruction**

• It is possible to place a single Add instruction in a program loop

Move N, R1

Clear R0

LOOP: Determine address of “Next” number and add “Next” number to R0

Decrement R1

Branch > 0, LOOP

Move R0, SUM

**Branch Instructions**

• This type of instruction loads a new value into the program counter

• As a result, the processor fetches and executes the instruction at this new address, called the branch target

• The branch instruction can be conditional or unconditional

• Some ISAs refer to such instructions as Jumps. The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions.

• This is accomplished by recording the required information in individual bits, often called condition code flags.

• These flags are usually grouped together in a special processor register called the condition code register or status register.

• Some of the commonly used flags are: Sign, Zero, Overflow and Carry.

***Call and return instructions***

• Used in conjunction with subroutines

• A subroutine is a self-contained sequence of instructions that performs a given computational task.

• Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions.

• After the subroutine has been executed, a branch is made back to the main program, through the return instruction

• A program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internally generated request.

• Control returns to the original program after the service program is executed.

Interrupts

• The interrupt procedure is quite similar to a subroutine call except for three variations:

(1) The interrupt is usually initiated by an internal or external signal apart from the execution of an instruction

(2) The address of the interrupt service program is determined by the hardware or from some information from the interrupt signal or the instruction causing the interrupt

(3) An interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

When the processor is interrupted, it saves: The current status of the processor, including

1. The return address

2. The register contents

3. The status information called the Processor Status Word (PSW)

• Then jumps to the interrupt handler or the interrupt service routine.

• Upon completing this, it returns to the main program.

* ***Input and output instructions***• It is possible to use special instructions that exclusively perform I/O transfers, or use memory –related instructions itself to do I/O transfers.

**B. Types and sizes of operands**

The data types and sizes indicate the various data types supported by the processor and their lengths.

• Common operand types –

1. Character (8 bits)

2. Half word (16 bits)

3. Word (32 bits)

4. Single Precision Floating Point (1 Word)

5. Double Precision Floating Point (2 Words)

6. Integers – two’s complement binary numbers

7. Characters usually in ASCII

8. Floating point numbers following the IEEE Standard 754

**C. Addressing Modes**

The way the operands are chosen during program execution is dependent on the addressing mode of the instruction

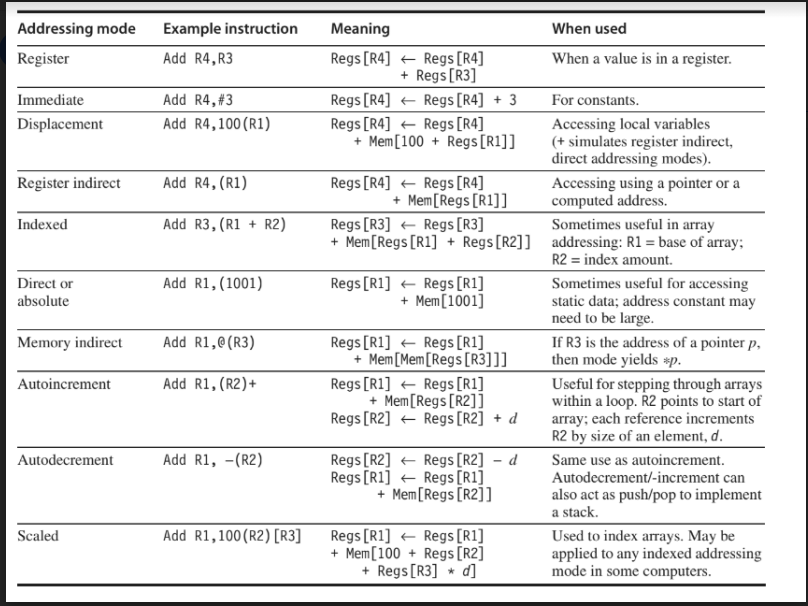
• The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. Computers use addressing mode techniques for the purpose of accommodating one or both of the following:

1. To give programming versatility to the user.

2. To reduce the number of bits in the addressing field of the instruction.

The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

• In assembly language, a variable is represented by allocating a register or a memory location to hold its value.



**D. Addressing Memory**

• Memories are normally arranged as bytes and a unique address of a memory location is capable of storing 8 bits of information.

• But when you look at the word length of the processor, the word length of the processor may be more than one byte.

• Suppose you look at a 32-bit processor, it is made up of four bytes.

These four bytes span over four memory locations.

• Most of the computers also provide access for double words(64 Bytes)

• Basically have two types of interpretation of the memory addresses

– Big endian arrangement and the little endian arrangement.

• Little Endian byte order puts the byte whose address is “x . . . x000” at the least-significant position in the double word (the little end).

• The bytes are numbered:



ie; specify the address of the least significant byte as the address of the word

• Big Endian byte order puts the byte whose address is “x . . . x000” at the most-significant position in the double word (the big end).

* The bytes are numbered:



• ie; specify the address of the most significant byte as the address of the word.

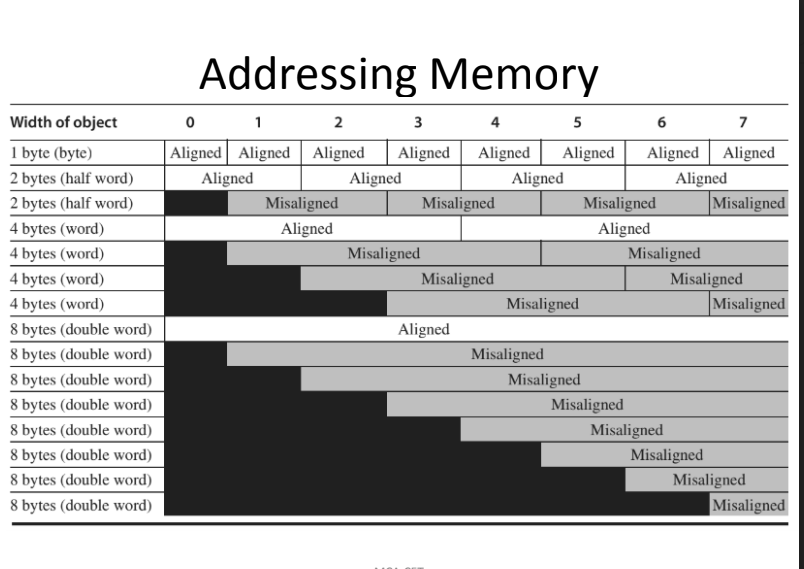
• IBM, Motorola, HP follow the big endian arrangement and Intel follows the little endian arrangement.

• Byte order is a problem when exchanging data among computers with different orderings

• Little Endian ordering also fails to match the normal ordering of words when strings are compared. Strings appear “SDRAWKCAB” (backwards) in the registers.

• A second memory issue is that in many computers, accesses to objects larger than a byte must be aligned.

• An access to an object of size s bytes at byte address A is aligned if A mod s = 0.



|  |
| --- |
| **NOTE:**  **A n**: Add the number in storage location n into the accumulator.  **E n** : If the number in the accumulator is greater than or equal to zero execute next the order which stands in storage location n; otherwise proceed serially.  **Z** : Stop the machine and ring the warning bell. |

**E. Encoding and Instruction Formats**

• How the instructions are encoded into a binary representation for execution by the processor?

• This representation affects not only the size of the compiled program but also the implementation of the processor which must decode this representation to quickly find the operation and its operands.

• The operation is typically specified in one field, called the **opcode.**

• The architect must balance several competing forces when encoding the instruction set:

1. The desire to have as many registers and addressing modes as possible.

2. The impact of the size of the register and addressing mode fields on the average instruction size and hence on the average program size.

3. A desire to have instructions encoded into lengths that will be easy to handle in a pipelined implementation.

There are three popular choices for encoding the instruction set.

1. Variable length: can support any number of operands, with each address specifier determining the addressing mode and the length of the specifier for that operand. This style is best when there are many addressing modes and operations.

2. Fixed length : always has the same number of operands, with the addressing modes (if options exist) specified as part of the opcode. It generally results in the largest code size.

3. Hybrid approach: has multiple formats specified by the opcode, adding one or two fields to specify the addressing mode and one or two fields to specify the operand address.

