# UNIT – I

# COMPUTER ORGANIZATION & INSTRUCTIONS

## 1.1 INTRODUCTION

Computer architecture acts as the interface between the hardware and the lowest level software. Computer architecture refers to:

- Attributes of a system visible to programmers like data type of variables.
- Attributes that have a direct impact on the execution of programs like clock cycle.

> ***Computer Architecture is defined as study of the structure, behavior, and design of computers.***

**Computer Organization:** It refers to the operational units and their interconnections that realize the architectural specifications. It describes the function of and design of the various units of digital computer that store and process information. The attributes in computer organization refers to:

- Control signals
- Computer/peripheral interface
- Memory technology

**Computer hardware:** Consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equipment and communication facilities.

**Computer Architecture:** It is concerned with the structure and behavior of the computer. It includes the information formats, the instruction set and techniques for addressing memory. The attributes in computer architecture refers to the:

- Instruction set
- Data representation
- I/O mechanisms
- Addressing techniques

The basic **distinction between architecture and organization** is: the attributes of the former are visible to programmers whereas the attributes of the later describes how features are implemented in the system.

## 1.2 BASICS OF A COMPUTER SYSTEM

The modern day computer system's functional unit is given by Von Neumann Architecture.


**Fig 1.1: Von Neumann Architecture**

### Input Unit

Computers accepts the coded information through input unit. Computer must receive both data and program statements to function properly and must be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. Input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

### Central Processing Unit (CPU)

The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices. A central control section and work areas are required to perform calculations or manipulate data. The CPU is the computing center of the system. It consists of a control section, an arithmetic-logic section, and an internal storage section (memory unit). Each section within the CPU serves a specific function and has a particular relationship with the other sections within the CPU.

### Memory Unit

It stores the programs and data. Memory unit is broadly classified into two types: Primary memory and Secondary memory.

1. **Primary Memory:**

It is a fast memory that operates at electronic speeds. Programs must be stored in the memory while they are being executed. The memory contains large no of semiconductor storage cells. Each cell carries 1 bit of information. The cells are processed in a group of fixed size called **Words**. To provide easy access to any word in a memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations. The number of bits in each word is called the **word length.** The word length ranges from 16 to 64 bits. There are 3 types of primary memory:

I. **RAM:** Memory in which any location can be reached in short and fixed amount of time after specifying its address is called RAM. Time required to access 1 word is called Memory Access Time.

II. **Cache Memory:** The small, fast, RAM units are called Cache. They are tightly coupled with processor to achieve high performance.

III. **Main Memory:** The largest and the slowest unit is the main memory.

**Arithmetic & Logic Unit**

Most computer operations are executed in ALU. The arithmetic-logic section performs arithmetic operations, such as addition, subtraction, multiplication, and division. Through internal logic capability, it tests various conditions encountered during processing and takes action based on the result. Data may be transferred back and forth between these two sections several times before processing is completed. Access time to registers is faster than access time to the fastest cache unit in memory.

**Output Unit**

Its function is to send the processed results to the outside world.

**Control Unit**

The operations of Input unit, output unit, ALU are co-ordinate by the control unit. The control unit is the Nerve centre that sends control signals to other units and senses their states. The control section directs the flow of traffic (operations) and data. It also maintains order within the computer. The control section selects one program statement at a time from the program storage area, interprets the statement, and sends the appropriate electronic impulses to the arithmetic-logic and storage sections so they can carry out the instructions. The control section does not perform actual processing operations on the data.

The control section instructs the input device on when to start and stop transferring data to the input storage area. It also tells the output device when to start and stop receiving data from the output storage area. Data transfers between the processor and the memory are controlled by the control unit through **timing signals**. Information stored in the memory is fetched, under program control into an arithmetic and logic unit, where it is processed.

### 1.2.1 Evolution of Computers

- The word ¦computer¦ is an old word that has changed its meaning several times in the last few centuries.
- Today, the word computer refers to computing devices, whether or not they are electronic, programmable, or capable of ¦storing and retrieving¦ data.

### The Mechanical Era (1623-1945)

- Wilhelm Schick hard, Blaise Pascal, and Gottfried Leibnitz were among mathematicians who designed and implemented calculators that were capable of addition, subtraction, multiplication, and division during the seventeenth century.
- The first multi-purpose or programmable computing device was probably **Charles Babbage's Difference Engine**, which was begun in 1823 but never completed.
- In 1842, Babbage designed a more ambitious machine, called the **Analytical Engine** but unfortunately it also was only partially completed.
- Babbage, together with Ada Lovelace recognized several important programming techniques, including conditional branches, iterative loops and index variables.
- Babbage designed the machine which is the first to be used in computational science.
- In 1933, George Scheutz and his son, Edvard began work on a smaller version of the difference engine and by 1853 they had constructed a machine that could process 15-digit numbers and calculate fourth-order differences.
- The US Census Bureau was one of the first organizations to use the mechanical computers which used punch-card equipment designed by Herman Hollerith to tabulate data for the 1890 census.
- In 1911 ¦Hollerith¦s company merged with a competitor to found the corporation which in 1924 became **International Business Machines** (IBM).

## First Generation Electronic Computers (1937-1953)

- These devices used electronic switches, in the form of **vacuum tubes**, instead of electromechanical relays.
- The earliest attempt to build an electronic computer was by J. V. Atanasoff, a professor of physics and mathematics at Iowa State in 1937.
- Atanasoff set out to build a machine that would help his graduate students solve systems of partial differential equations.
- By 1941 he and graduate student Clifford Berry had succeeded in building a machine that could solve 29 simultaneous equations with 29 unknowns.
- However, the machine was not programmable, and was more of an electronic calculator.
- A second early electronic machine was Colossus, designed by Alan Turing for the British military in 1943.
- The first general purpose programmable electronic computer was the Electronic Numerical Integrator and Computer **(ENIAC),** built by J. Presper Eckert and John V. Mauchly at the University of Pennsylvania.
- ENIAC was controlled by a set of external switches and dials; to change the program required physically altering the settings on these controls.
- Research work began in 1943, funded by the Army Ordinance Department, which needed a way to compute ballistics during World War II.
- The machine was completed in 1945 and it was used extensively for calculations during the design of the hydrogen bomb.
- Eckert, Mauchly, and John von Neumann, a consultant to the ENIAC project, began work on a new machine before ENIAC was finished.
- The next development was **EDVAC**- Electronic Discrete Variable Computer.
- The main contribution of EDVAC, their new project, was the notion of a **stored program.**
- EDVAC was able to run orders of magnitude faster than ENIAC and by storing instructions in the same medium as data, designers could concentrate on improving the internal structure of the machine without worrying about matching it to the speed of an external control.

- Eckert and Mauchly later designed the first commercially successful computer, the **UNIVAC (**Universal Automatic Computer**)**; in 1952.
- Software technology during this period was very primitive.
- The instructions were written in machine language that could be executed directly.

### Second Generation (1954-1962)

- The second generation witnessed several important developments at all levels of computer system design, ranging from the technology used to build the basic circuits to the programming languages used to write scientific applications.
- Electronic switches in this era were based on **discrete diode and transistor technology** with a switching time of approximately 0.3 microseconds.
- The first machines to be built with this technology include **TRADIC** at Bell Laboratories in 1954 and TX-ξ at M)T¦s Lincoln Laboratory.
- Index registers were designed for controlling loops and floating point units for calculations based on real numbers.
- A number of high level **programming languages** were introduced and these include FORTRAN (1956), ALGOL (1958), and COBOL (1959).
- **Batch processing systems** came to existence.
- Important commercial machines of this era include the IBM 704 and its successors, the 709 and 7094.
- In the 1950s the first two supercomputers were designed specifically for numeric processing in scientific applications.
- Multi programmed computers that serve many users concurrently came to existence. This is otherwise known as **time-sharing systems**.

### Third Generation (1963-1972)

- Technology changes in this generation include the use of **integrated circuits**, or ICs.
- This generation led to the introduction of **semiconductor memories**, **micro programming** as a technique for efficiently designing complex processors and the introduction of **operating systems** and time-sharing.

## 1.7 Introduction

- The first ICs were based on small-scale integration (SSI) circuits, which had around 10 devices per circuit (or chip), and evolved to the use of medium-scale integrated (MSI) circuits, which had up to 100 devices per chip.
- Multilayered printed circuits were developed and core memory was replaced by faster, solid state memories.
- In 1964, Seymour Cray developed the CDC 6600, which was the first architecture to use **functional parallelism**.
- By using 10 separate functional units that could operate simultaneously and 32 independent memory banks, the CDC 6600 was able to attain a computation rate of one million floating point operations per second (Mflops).
- Five years later CDC released the 7600, also developed by Seymour Cray.
- The CDC 7600, with its pipelined functional units, is considered to be the first vector processor and was capable of executing at ten Mflops.
- The IBM 360/91, released during the same period, was roughly twice as fast as the CDC 660.
- Early in this third generation, Cambridge University and the University of London cooperated in the development of **CPL** (Combined Programming Language, 1963).
- CPL was an attempt to capture only the important features of the complicated and sophisticated ALGOL.
- However, like ALGOL, CPL was large with many features that were hard to learn.
- In an attempt at further simplification, Martin Richards of Cambridge developed a subset of CPL called **BCPL** (Basic Computer Programming Language, 1967).
- In 1970 Ken Thompson of Bell Labs developed yet another simplification of CPL called simply **B**, in connection with an early implementation of the UNIX operating system.

### Fourth Generation (1972-1984)

- **Large scale integration** (LSI - 1000 devices per chip) and **very large scale integration** (VLSI - 100,000 devices per chip) were used in the construction of the fourth generation computers.
- Whole processors could now fit onto a single chip, and for simple systems the entire computer (processor, main memory, and I/O controllers) could fit on one chip.

- Gate delays dropped to about 1ns per gate. Core memories were replaced by semiconductor memories.
- Large main memories like CRAY 2 began to replace the older high speed vector processors, such as the CRAY 1, CRAY X-MP and CYBER.
- In 1972, Dennis Ritchie developed the **C language** from the design of the CPL and Thompson's B.
- Thompson and Ritchie then used C to write a version of UNIX for the DEC PDP-11.
- Other developments in software include very high level languages such as FP (functional programming) and Prolog (programming in logic).
- IBM worked with Microsoft during the 1980s to start what we can really call PC (Personal Computer) life today.
- IBM PC was introduced in October 1981 and it worked with the operating system (software) called 'Microsoft Disk Operating System (MS DOS) 'etc.
- Development of **MS DOS** began in October 1980 when IBM began searching the market for an operating system for the then proposed IBM PC and major contributors were Bill Gates, Paul Allen and Tim Paterson.
- In 1983, the **Microsoft Windows** was announced and this has witnessed several improvements and revision over the last twenty years.

### Fifth Generation (1984-1990)

- This generation brought about the introduction of machines with hundreds of processors that could all be working on different parts of a single program.
- The scale of integration in semiconductors continued at a great pace and by 1990 it was possible to build chips with a million components - and semiconductor memories became standard on all computers.
- Computer networks and single-user workstations also became popular. Parallel processing started in this generation.
- The Sequent Balance 8000 connected up to 20 processors to a single shared memory module though each processor had its own local cache.
- The machine was designed to compete with the **DEC VAX-780** as a general purpose UNIX system, with each processor working on a different user's job.

- However Sequent provided a library of subroutines that would allow programmers to write programs that would use more than one processor, and the machine was widely used to explore parallel algorithms and programming techniques.
- The **Intel iPSC-1**, also known as ¦the hypercube¦ connected each processor to its own memory and used a network interface to connect processors.
- This distributed memory architecture meant memory was no longer a problem and large systems with more processors (as many as 128) could be built.
- Also introduced was a machine, known as a **data-parallel** or SIMD where there were several thousand very simple processors which work under the direction of a single control unit.
- Both wide area network (WAN) and local area network (LAN) technology developed rapidly.

### Sixth Generation (1990 - )

- Most of the developments in computer systems since 1990 have not been fundamental changes but have been gradual improvements over established systems.
- This generation brought about gains in parallel computing in both the hardware and in improved understanding of how to develop algorithms to exploit parallel architectures.
- Workstation technology continued to improve, with processor designs now using a combination of RISC, pipelining, and parallel processing.
- Wide area networks, network bandwidth and speed of operation and networking capabilities have kept developing tremendously.
- Personal computers (PCs) now operate with Gigabit per second processors, multi-Gigabyte disks, hundreds of Mbytes of RAM, color printers, high-resolution graphic monitors, stereo sound cards and graphical user interfaces.
- Thousands of software (operating systems and application software) are existing today and Microsoft Inc. has been a major contributor. Microsoft is said to be one of the biggest companies ever, and its chairman – Bill Gates has been rated as the richest man for several years.

- Finally, this generation has brought about **micro controller technology**. Micro controllers are embedded inside some other devices so that they can control the features or actions of the product.
- They work as small computers inside devices and now serve as essential components in most machines.

### 1.2.2 Great Ideas in Computer Architecture

The ideas that marked tremendous improvement in the field of computer architecture are briefly discussed here.

**な. Moore' s Law**

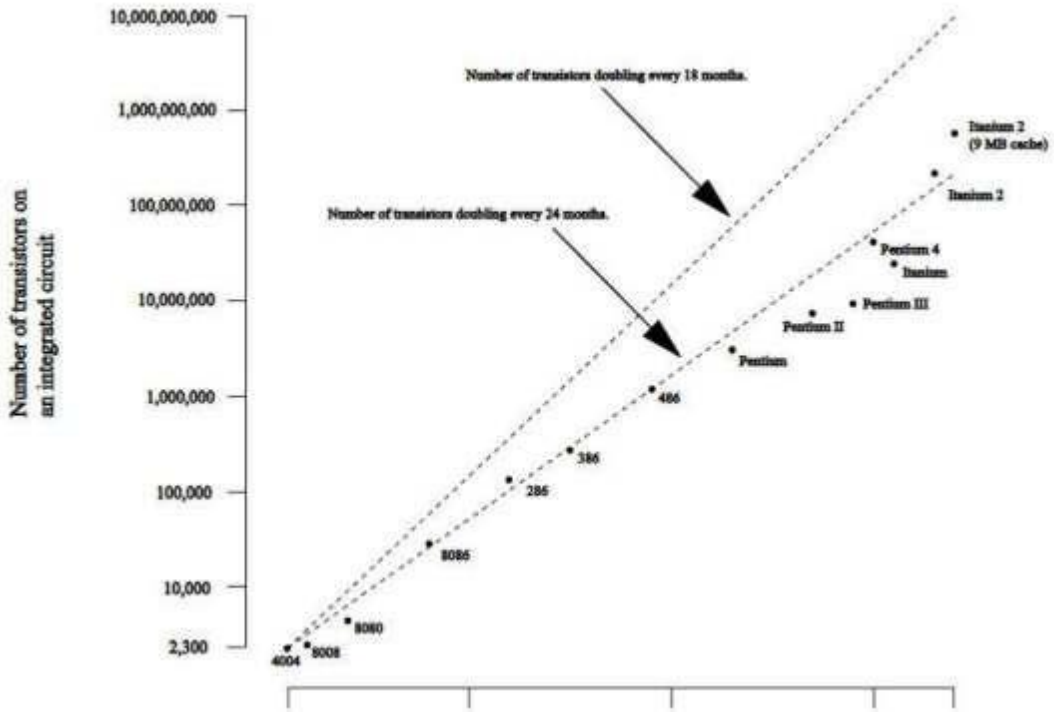



**Fig な.に: Illustration of Moore' s Law**

***Moore's law states that the numbers of transistors will double every 18 months.***

It is an observation that the number of transistors in a dense integrated circuit doubles about every two years. It is an observation and projection of a historical trend and not a physical or natural law.

**2. Abstract Design**

It is a major productivity technique for hardware and software. Abstractions are used to represent the design at different levels of representation. The detailed lower-level design details from the higher levels.

**3. Performance through parallelism**

Parallelism executes programs faster by performing several computations at the same time. This requires hardware with multiple processing units. The overall performance of the system is significantly increased by performing operations in parallel.

**4. Performance through Pipelining**

Pipelining increases the CPU instruction throughput. **Throughput** is a performance metric which is the number of instructions completed per unit of time. But it does not reduce the execution time of an individual instruction. It increases the execution time of each instruction due to overhead in the pipeline control. The increase in instruction throughput means that a program runs faster and has lower total execution time.

**5. Make the Common Case Fast**

Making the common case fast will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is often easier to enhance. In making a design trade-off, favor the frequent case over the infrequent case.

Amdahl's Law can be used to quantify this principle. This also applies when determining how to spend resources, since the impact on making some occurrence faster is higher if the occurrence is frequent. This will:

- Helps performance
- Is simpler and can be done faster

**6. Performance via prediction**

The computer can perform better (on average) by making rational guesses on the decisions. Instead of wasting clock cycles for certain results, the computers can remarkably improve the performance

**7. Hierarchy of memories**

Programmers want memory to be fast, large, and cheap. The memory speed is a primary factor in determining the performance of the system. The memory capacity limits the size of problems that can be solved.

Architects have found that hierarchy of memories will be a solution for all these issues. The fastest, smallest, and most expensive memory per bit is placed the top of the hierarchy and the slowest, largest, and cheapest per bit is at the bottom. **Caches** give the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy.

### 8. Dependability via Redundancy

Computers need to be fast and dependable. Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and help detect failures. Restoring the state of the system is done by redundancy.

## 1.2.3 Technologies

Up until the early なひばどs computers used magnetic core memory, which was slow, cumbersome, and expensive and thus appeared in limited quantities. The situation improved with the introduction of transistor-based dynamic random-access memory (DRAM, invented at IBM in 1966) and static random-access memory (SRAM). A **transistor** is simply an on/off switch controlled by electricity. The **integrated circuit** (IC) combined dozens to hundreds of transistors into a simple chip. **Very large-scale integrated** (VLSI) circuit is a device containing hundreds of thousands to millions of transistors.

**Manufacturing of IC:**

Integrated circuits are chips manufactured on silicon wafers. Transistors are placed on wafers through a chemical etching process. Each wafer is cut into chips which are packed individually.

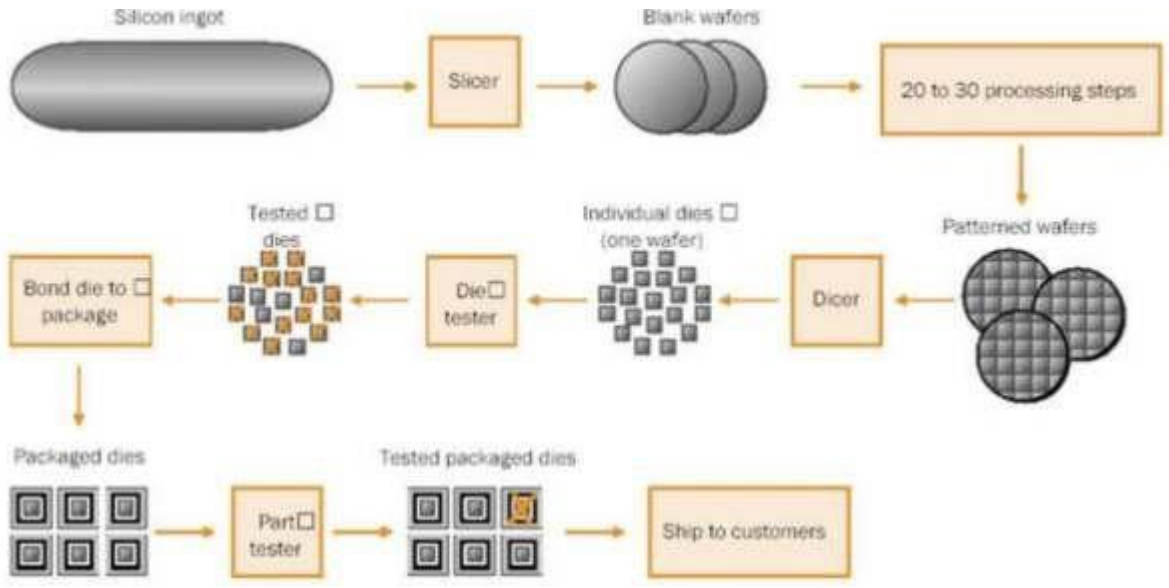



**Fig 1.3: Chip manufacturing process**

After being sliced from the silicon ingot, blank wafers are put through 20 to 40 steps to create patterned wafers. These patterned wafers are then tested with a wafer tester, and a map of the good parts is made. Then, the wafers are diced into dies. The good dies are then bonded into packages and tested one more time before shipping the packaged parts to customers.

Cost of an IC is found from:

- Cost per die= (cost per wafer) / ((dies per wafer)*yield) Yield refers the fraction of dies that pass testing.
- Dies / wafer= wafer area / die area
- Yield=1 / (1 + (defects per area * die area)/2 )$^2$

**Programmable Logic Device (PLD)**

A programmable logic device (PLD) is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.

The major limitations of PLD:

- Consume space due to large number of switches for programmability
- Low speed due to the presence of many switches.

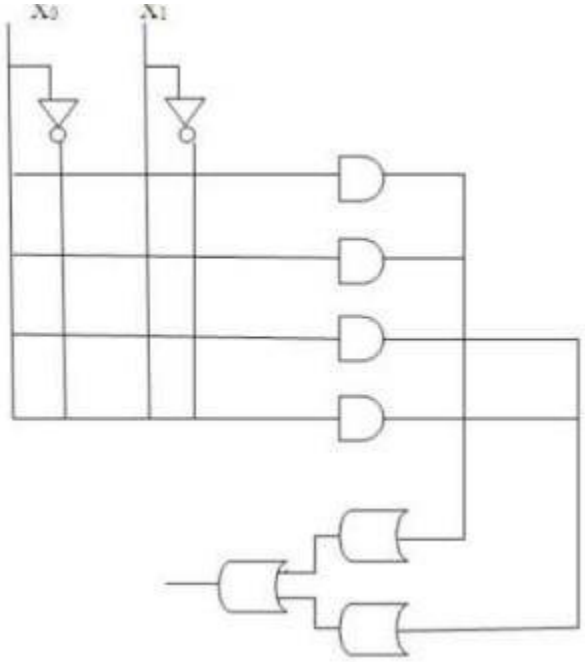


**Fig 1.4: Programmable Logic Device**

### Custom chips

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. Application-Specific Standard Products (ASSPs) are intermediate between ASICs and industry standard integrated circuits.

## 1.2.4 Performance

Elapsed time and throughput are two different ways of measuring speed.

- **Elapsed time** or wall-clock time or response time is the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead. It is the better measure for processor speed because it is less dependent on other system components.
- **CPU execution time** is the actual time the CPU spends computing for a specific task.
- The **User CPU time** is the CPU time spent in a program itself. **System CPU time** is the CPU time spent in the operating system performing tasks on behalf of the program.
- The **CPU Performance** equation (CPU Time) is the product of number of instructions executed, Average CPI of the program and CPU clock cycle.

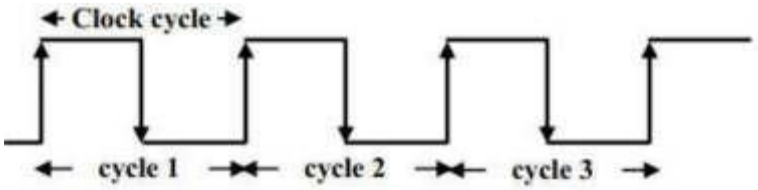
$$CPU\ Time = \frac{Seconds}{Program} \times \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

- Performance is inversely proportional to execution time. Performance ratios are inverted from time ratios.

$$Performance\ improve mentation = \frac{Performance\ after\ change}{Performance\ before\ change} \text{ x Execution } \frac{time\ before\ change}{Execution\ time\ after\ change}$$

- Clock cycle is the time for one clock period, usually of the processor clock, which runs at a constant rate.
- Clock period is the length of each clock cycle.
- The CPU clock rate depends on CPU organization and hardware implementation.

$$Clock\ Rate = \frac{1}{Clock\ Cycle}$$



- **Cycles per Instruction (CPI)** is count of clock cycles taken by an instruction to complete its execution.

$$\textit{Instructions per Cycle (IPC)} = \frac{1}{\textit{Cycles per Instruction}}$$

- Performance is improved by reducing number of clock cycles, increasing clock rate and hardware designer must often trade off clock rate against cycle count.
- Workload is a set of programs run on a computer that is either the actual collection of applications run by a user or is constructed from real programs to approximate such a mix. A typical workload specifies both the programs as well as the relative frequencies.
- To evaluate two computer systems, a user would simply compare the execution time of the workload on the two computers.
- Alternatively, set of benchmarks containing several typical engineering or scientific applications can be used. A CPU benchmark (CPU benchmarking) is a series of tests designed to measure the performance of a computer or device CPU. A set of standards, or baseline measurements are used to compare the performance of different systems, using the same methods and circumstances.

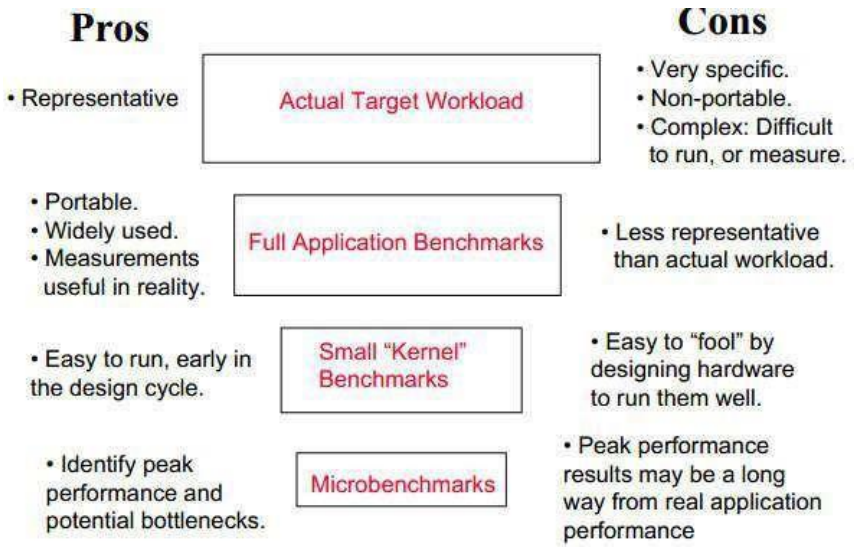



**Fig 1.6: Types of Benchmark Programs**

- The use of benchmarks whose performance depends on very small code segments encourages optimizations in either the architecture or compiler that target these segments.
- The arithmetic mean is proportional to execution time, assuming that the programs in the workload are each run an equal number of times.
- **Weighted arithmetic mean** is an average of the execution time of a workload with weighting factors designed to reflect the presence of the programs in a workload; computed as the sum of the products of weight.

**Example 1.1:** For a given program, the execution time on machine A is 1s and on B is 10s. Find the performance or speed up of the machines.

Execution $_A$= 1s

Execution $_B$=10s

$$Speedup = \frac{Performance\ of\ A}{Performance\ of\ B} x \frac{Execution\ of\ B}{Execution\ of\ A}$$

Speedup=10/1=10

The performance of machine A is 10 times faster than that of B.

**Example 1.2:** For a certain program with 1,00,00,000 instructions, find the execution time given the average CPI is 2.5 cycles/instruction and clock rate as 200MHz.

Number of instructions=1,00,00,000

Average CPI=2.5 cycles/ instruction

Clock rate=200MHz =200000000 Hz

Clock cycle=1/Clock rate=1/ 200000000= 5 x $10^{-9}$s

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} x \frac{Cycles}{Instruction} x \frac{Seconds}{Cycle}$$

CPU Time=10000000 x 2.5 x 5 x $10^{-9}$

=0.125 s

**Example 1.3:** For a certain program with 1,00,00,000 instructions has an average CPI is 2.5 cycles/instruction and clock rate as 200MHz. When a new optimization complier is deployed, the instruction count was reduced to 95,00,000 with new CPI=3.0 cycles/instruction at modified clock rate of 300MHz. Find the speedup.

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} \times \frac{I_{old} \times CPI_{old} \times \text{Clock cycle}_{old}}{I_{new} \times CPI_{new} \times \text{Clock Cycle}_{new}}$$

$(10000000 \times 2.5 \times 5 \times 10^{-9}) / (9500000 \times 3 \times 3.33 \times 10^{-9})$

$\approx 1.315$

The new compiler is 1.315 times faster than the old one.

**Example 1.4:** A program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which with run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 time as many clock cycles as computer A for this program. What clack rate should we tell the designer to target?

Clock rate of B= Clock Cycles $_B$ / CPU Time $_B$

$= 1.2 \times$ Clock Cycles $_A$ / 6 Clock Cycles $_A$= CPU Time $_A \times$ Clock Rate $_A$

$\approx 10 \times 2 = 20 \times 10^9$

Clock Cycles $_B = 1.2 \times 20 \times 10^9 / 6$

$= 4$ GHz

**Example 1.5:** Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Computer A: Cycle Time = 250ps, CPI = 2.0

Computer B: Cycle Time = 500ps, CPI = 1.2

CPU Time=Instruction Count x CPI $_A$ x Cycle Time $_A$

= I x 2.0 x 250= I x 500

CPU Time=Instruction Count x CPI $_B$ x Cycle Time $_B$

= I x1.2x 500= Ix 600

$$\frac{CPU\ time_B}{CPU\ Time_A} = 1.2$$

### 1.2.5 Power wall

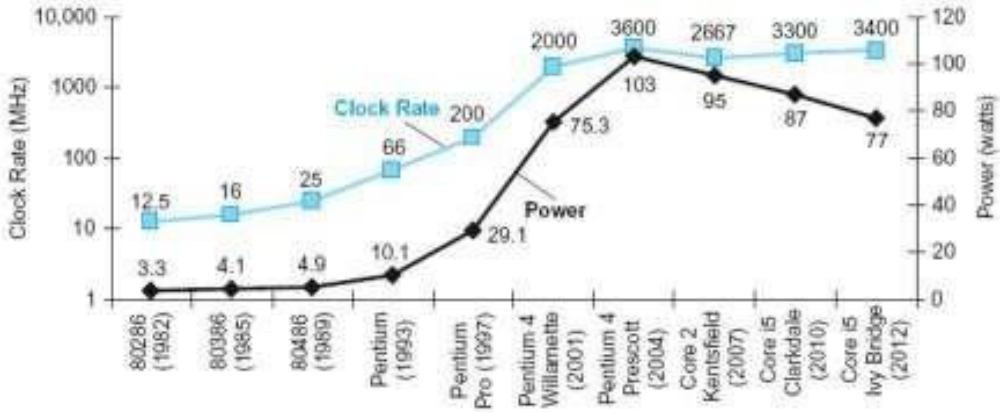



**Fig 1.7: Clock rate and Power**

- **Power wall** refers to the representational wall signifying the peak power constraint of a system.
- Clock rate and Power for Intel x86 microprocessors over eight generations and 25 years is shown in Fig 1.7.
- The Pentium 4 made a dramatic jump in clock rate and power but less so in performance.
- The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip.
- Continuous technology scaling like reduction of the transistor feature sizes makes it possible to pack more transistors in a given chip die area.
- Reduced supply voltage, simultaneous switching of these transistor devices causes a tremendous increase in the power density, leading to the power wall disaster.

- An increase in the power density increases the chip temperature, which slows down the transistor switching rate and hence, the overall speed of the computer.
- Cooling solutions are very expensive, and hence, computer architects have focused on innovating device, circuit and architecture level techniques to combat power wall.
- **Dynamic voltage and frequency scaling** are solutions for these problems. Here the operating voltage and frequency of the chip are dynamically controlled based on the chip activity.
- In CMOS (complementary metal oxide semiconductor) IC technology

**Power = Capacitive load x Voltage$^2$ x Frequency**

**Example 1.6:** Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it has adjustable voltage so that it can reduce voltage 1 5% compared to processor B, which results in 15% shrink in frequency. What is the impact on dynamic power? Given: 85% of capacitive load of old CPU, 15% voltage reduction, 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{(C_{old} \times 0.85) \times (V_{old} \times 0.85)^2 \times (F_{old} \times 0.85)}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

The new processor uses 0.52 the power of the old processor.

### 1.2.6 from Uniprocessors to Multiprocessors

The performance of the computers has drastically increased when the technology has drifted from uniprocessor systems to multiprocessor system. As the core computing units were made more powerful, the performance of the processors also increased significantly.

**Uniprocessor system is a type of architecture that is based on a single computing unit. All the operations were done sequentially on the same unit. Multiprocessor systems are based on executing instructions on multiple computing units.**

The multiprocessor architectures, is based on Flynn Taxonomy.

**Flynn's taxonomy is a classification of parallel computer architectures that are based on the number of concurrent instruction and data streams available in the architecture.**

**Single Instruction, Single Data (SISD):**

- This is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream.
- The machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers.
- Most conventional computers have SISD architecture.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of the processing element in the SISD model is limited by the rate at which the computer can transfer information internally.

**Multiple Instruction, Single Data (MISD):**

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different Processing Elements but all of them operating on the same dataset.

**Single Instruction, Multiple Data (SIMD)**:

- This machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations. So that the information can be passed to all the Processing Elements (PEs) organized data elements of vectors can be divided into multiple sets and each PE can process one data set.

**Multiple Instruction, Multiple Data (MIMD):**

- This is capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

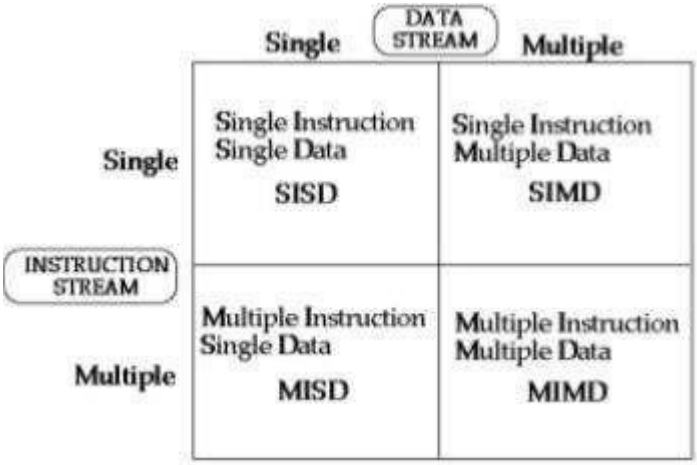



**Fig 1.8: Flynns Taxonomy**

Apart from these architectures, MIPS Technologies developed a Microprocessor without Interlocked Pipeline Stages on Reduced Instruction Set Computer (RISC).

### Concern for Power

- The power limit has forced a dramatic change in the design of microprocessors. Since 2002, the rate has slowed from a factor of 1.5 per year to a factor of 1.2 per year.
- Most of the desktop manufacturing companies are shipping microprocessors with multiple processors per chip, where the benefit increased throughput than on response time. This is done at the cost of increase in power.
- To reduce confusion between the words processor and microprocessor, companies refer to processors as cores and such microprocessors are generically called **multicore microprocessors.**
- A **quad core** microprocessor is a chip that contains four processors or four cores.
- In the past, programmers could rely on innovations in hardware, architecture, and compilers to double performance of their programs every 18 months without having to change a line of code.
- Today, for programmers to get significant improvement in response time, they need to rewrite their programs to take advantage of multiple processors.

- Moreover, to get the historic benefit of running faster on new microprocessors, programmers will have to continue to improve performance of their code as the number of cores increases.

## 1.3 ADDRESSING AND ADDRESSING MODES

Each instruction of a computer specifies an operation on certain data.

***The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.***

Different operands will use different addressing modes. One or more bits in the instruction format can be used as mode field. The value of the mode field determines which addressing mode is to be used. The effective address will be either main memory address of a register.

The most common addressing modes are:

1. Immediate addressing mode
2. Direct addressing mode
3. Indirect addressing mode
4. Register addressing mode
5. Register indirect addressing mode
6. Displacement addressing mode
7. Stack addressing mode

### □ Immediate Addressing:

- This is the simplest form of addressing. Here, the operand is given in the instruction.
- This mode is used to define constant or set initial values of variables.
- The advantage of this mode is that no memory reference other than instruction fetch is required to obtain operand.
- The disadvantage is that the size of the number is limited to the size of the address field because most instruction sets is small compared to word length.

- **Example:** ADD 3
- Adds 3 to contents of accumulator and 3 is the operand.

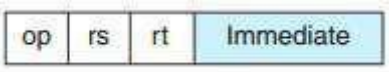



**Fig 1.9: Immediate Mode**

- **Direct Addressing:**
  - In direct addressing mode, effective address of the operand is given in the address field of the instruction.
  - It requires one memory reference to read the operand from the given location and provides only a limited address space.
  - Length of the address field is usually less than the word length.
  - **Example :** Move P, Ro

    Add Q, Ro

    Where P and Q are the address of operand, $R_o$ is any register. Sometimes Accumulator (AC) is the default register. Then the instruction will look like:

    Add A

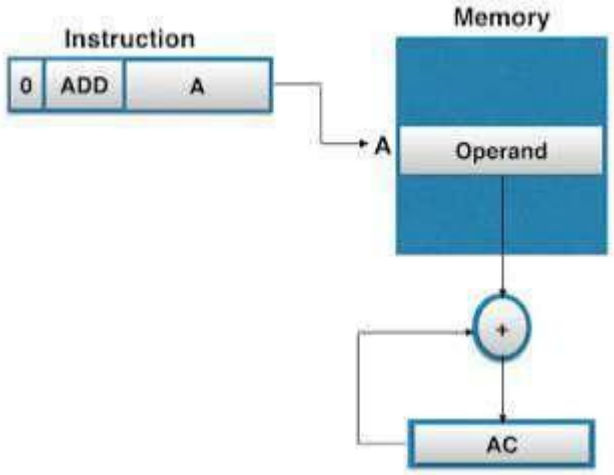



**Fig 1.10: Direct Addressing modes**

### □ Indirect or Pseudo direct Addressing:

- Indirect addressing mode, the address field of the instruction refers to the address of a word in memory, which in turn contains the full length address of the operand.
- The address field of instruction gives the memory address where on, the operand is stored in memory.
- Control fetches the instruction from memory and then uses its address part to access memory again to read Effective Address.
- The advantage of this mode is that for the word length of N, an address space of 2N can be addressed.
- The disadvantage is that instruction execution requires two memory references to fetch the operand.
- Multilevel or cascaded indirect addressing can also be used.
- **Example:** Effective Address (EA) = (A).
- The operand will be present in the memory location A.

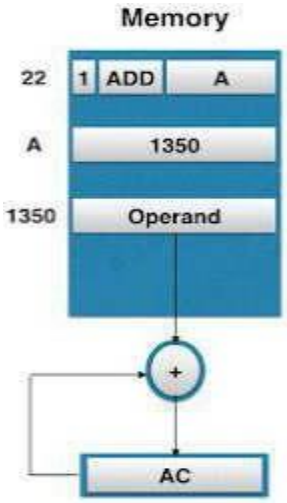



**Fig 1.11: Indirect Addressing Modes**

### □ Register Addressing:

- Register addressing mode is similar to direct addressing. The only difference is that the address field of the instruction refers to a register rather than a memory location.

- 3 or 4 bits are used as address field in the instruction to refer 8 to 16 generate purpose registers (GPR).
- The operands are in registers that reside within the CPU.
- The instruction specifies a register in CPU, which contain the operand.
- There is no need to compute the actual address as the operand is in a register and to get operand there is no memory access involved.
- The advantages of register addressing are small address field is needed in the instruction and faster instruction fetch.
- The disadvantages includes very limited address space and usage of multiple registers helps in performance but it complicates the instructions.

**Example:** MOV AX, BX

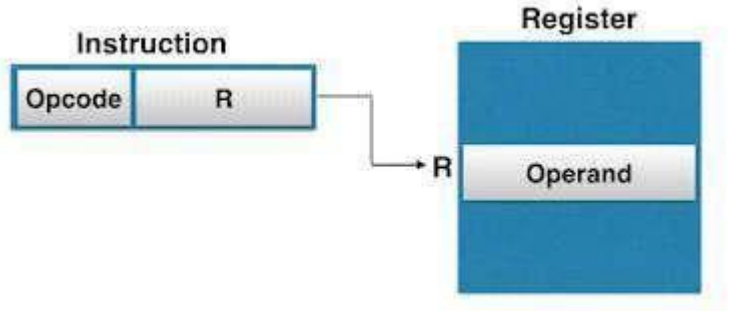



**Fig 1.12: Register Mode**

**Register Indirect Addressing:**

- This mode is similar to indirect addressing. The address field of the instruction refers to a register.
- The instruction specifies a register in CPU whose contents give the operand in memory.
- The selected register contain the address of operand rather than the operand itself.
- The register contains the effective address of the operand. This mode uses one memory reference to obtain the operand.

- Control fetches instruction from memory and then uses its address to access Register and looks in Register(R) for effective address of operand in memory.
- The address space is limited to the width of the registers available to store the effective address.
- **Example: MOV AL, [BX]**

  Code example in Register:

  MOV BX, 1000H

  MOV 1000H, operand
- The instruction (MOV AL, [BX]) specifies a register [BX] which contain the address of operand (1000H) rather than address itself.

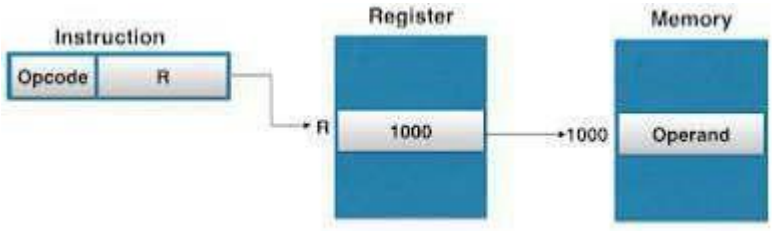



**Fig 1.13: Register Indirect Mode**

**Displacement Addressing:**

- It is a combination of direct addressing or register indirect addressing mode.
- Displacement Addressing Modes requires that the instruction have two address fields, at least one of which is explicit means, one is address field indicate direct address and other indicate indirect address.
- Value contained in one addressing field is A, which is used directly and the value in other address field is R, which refers to a register whose contents are to be added to produce effective address.
- **Example:** EA=A+(R)

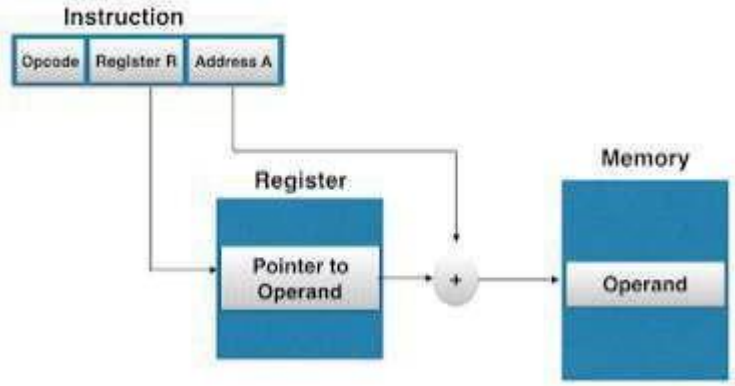



**Fig 1.14 a): Displacement Addressing Modes**

- In displacement addressing mode there are 3 types of addressing mode.
- **Relative addressing:**

  The contents of program counter is added to the address part of instruction to obtain the Effective Address. The address field of the instruction is added to implicitly reference register Program Counter to obtain effective address.

  **Example: EA=A+PC**

  Assume that PC contains the value 825 and the address part of instruction contain the value 24, then the instruction at location 825 is read from memory during fetch phase and the Program Counter is then incremented by one to 826. Here both PC and instruction contains address. The effective address computation for relative address mode is 826+24=850

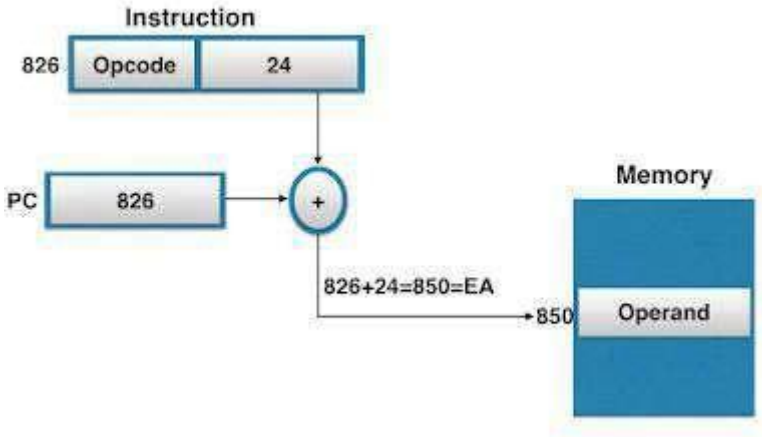



**Fig 1.14 b): Relative addressing**

- **Base register addressing**

The content of the Base Register is added to the direct address part of the instruction to obtain the effective address. The address field point to the Base Register and to obtain EA, the contents of Instruction Register, is added to direct address part of the instruction. This is similar to indexed addressing mode except that the register is now called as Base Register instead of Index Register.

**Example:** EA=A +Base

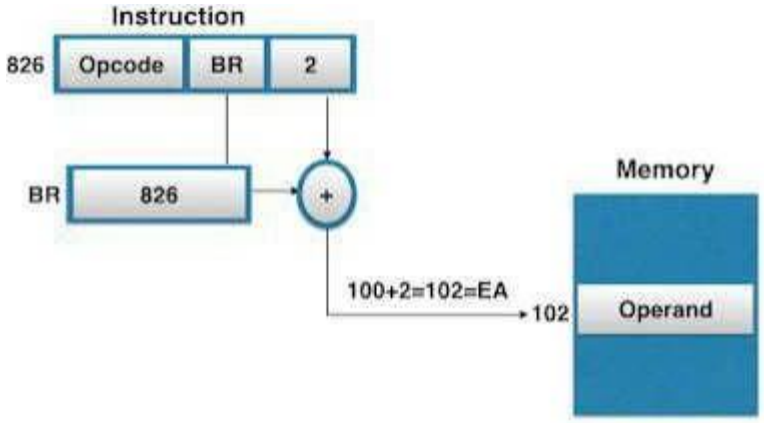



**Fig 1.14 c): Base Register Addressing Mode**

- **Indexed addressing:**

The content of Index Register is added to direct address part of instruction to obtain the effective address. The register indirect addressing field of instruction point to Index Register, which is a special CPU register that contain an Indexed value, and direct addressing field contain base address.

The data array is in memory and each operand in the array is stored in memory relative to base address. The distance between the beginning address and the address of operand is the indexed value stored in indexed register.

Any operand in the array can be accessed with the same instruction, which provided that the index register contains the correct index value i.e., the index register can be incremented to facilitate access to consecutive operands.

**Example:** EA=A+ Index

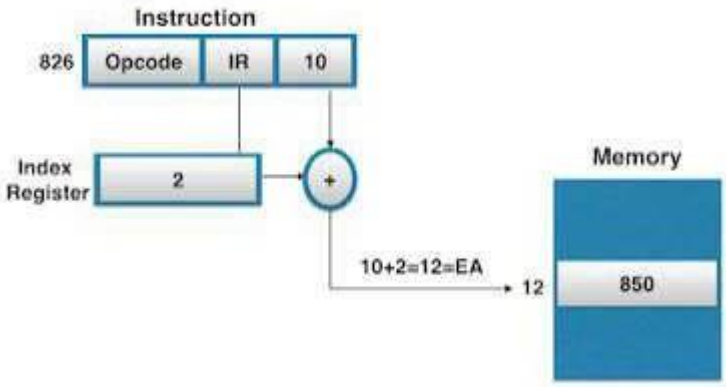



**Fig 1.14d): Indexed Addressing**

□ **Stack Addressing:**

- Stack is a linear array of locations referred to as last-in first out queue.
- The stack is a reserved block of location, appended or deleted only at the top of the stack.
- Stack pointer is a register which stores the address of top of stack location.
- This mode of addressing is also known as **implicit addressing**.
- Example: Add
- This instruction pops two items from the stack and adds.

**Additional Modes:**

There are two additional modes. They are:

- Auto-increment mode
- Auto-decrement mode

These are similar to Register indirect Addressing Mode except that the register is incremented or decremented after (or before) its value is used to access memory. These modes are required because when the address stored in register refers to a table of data in memory, then it is necessary to increment or decrement the register after every access to table so that next value is accessed from memory.

**Auto-increment mode:**

- Auto-increment Addressing Mode are similar to Register Indirect Addressing Mode except that the register is incremented after its value is loaded (or accessed) at another location like accumulator (AC).
- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically incremented to point to the next item in the list.
- **Example:** (R) +.
- The contents in register R will be accessed and them it will be incremented to point the next item in the list.

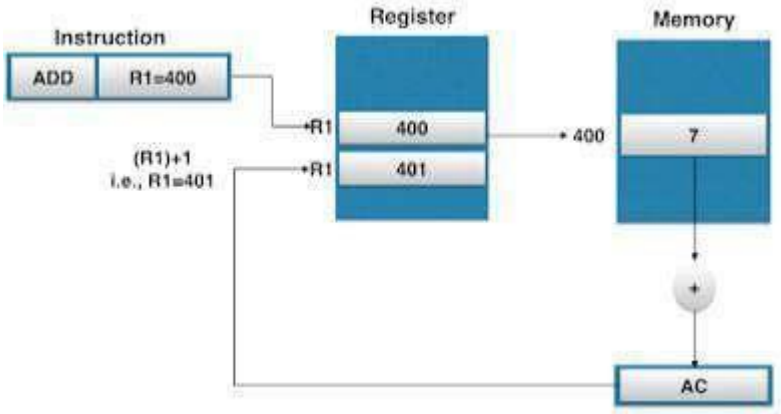



**Fig 1.16: Auto-increment Mode**

- The effective address is (R) =400 and operand in AC is 7. After loading R1 is incremented by 1, it becomes 401.

**Auto-decrement mode:**

- Auto-decrement Addressing Mode is reverse of auto-increment, as in it the register is decrement before the execution of the instruction.
- Effective address is equal to EA=(R) - 1
- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically decremented to point to the next item in the list.

- **Example**: - ( R)
- The contents in register R will be decremented and then it is accessed.

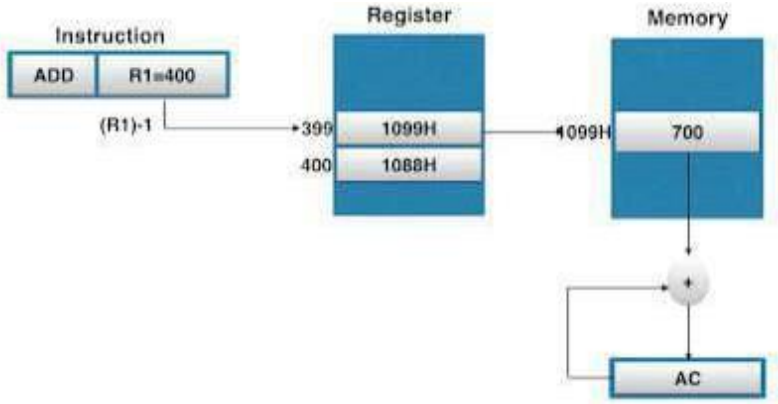



**Fig 1.17: Auto Decrement Addressing Mode**

## 1.4 INSTRUCTIONS

***An instruction is a binary code, which specifies a basic operation for the computer.***

- Operation Code (op code) defines the operation type. Operands define the operation source and destination.
- **Instruction Set Architecture** (ISA) describes the processor in terms of what the assembly language programmer sees, i.e. the instructions and registers.
- The op codes and operands follows Stores Program Concept.

***Stored Program Concept is an idea that instructions and data of many types can be stored in memory as numbers, leading to the stored program computer.***

### 1.4.1 Operations

- The computer performs the arithmetic through operations.
- The MIPS arithmetic instruction performs only one operation and must always have exactly three variables.

**Example:** Add a, b, c

Adds b and c and stores the sum in a.

- The hardware for a variable number of operands is more complicated than hardware for a fixed number.
- It is always essential to design the instructions with same number of operands so as to simplify the hardware requirement.

### 1.4.2 Operands

- The operands of arithmetic instruction must be from specially built memory locations called **registers.**
- The registers are accessed as 32 bit groups termed as **words**. MIPS architecture supports 32 registers.

### Memory Operands

- The operands are always stored in registers.
- Data transfer instruction is a command that moves data between memory and registers.
- Address of an operand is a value used to delineate the location of a specific data element within a memory array.
- The data transfer instruction that copies data from memory to a register is traditionally called **load (lw- load word)**.
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.
- The sum of the constant portion of the instruction and the contents of the second register forms the memory address.
- **Store (sw- store word)** instruction copies data from a register to memory.
- The format of a store is the name of the operation, followed by the register to be stored, then offset to select the array element, and finally the base register.
- The MIPS address is specified in part by a constant and in part by the contents of a register.

- Many programs have more variables than computers have registers. The compiler tries to keep the most frequently used variables in registers and places the rest in memory, using loads and stores to move variables between registers and memory.
- The process of putting less commonly used variables into memory is called **spilling registers.**

**Constant or Immediate Operands**

- Sometimes it is necessary to load a constant from memory to use one. The constants would have been placed in memory when the program was loaded.

  **Example:** add I$s3,$s3,10
- This instruction is interpreted as addition of content of $s3 and the value 10. The sum is stored in $s3. Add I means add immediate, since one of the operand is in immediate addressing mode.
- As per the design principle ¦Make common case faster¦, the constant operands must be loaded faster from the memory.
- Since constants occur more frequently in the instruction, they are mentioned in the instruction itself rather than to load from registers.

| Name | Example | Comments |
|---|---|---|
| 32 Registers | $S0, $S1…<br>$t0, $t1… | They can be accessed quickly. In MIPS architecture, the data must be loaded into the register to perform arithmetic operation. |
| $2^{30}$ memory words | Memory[0],<br>Memory [1]… | The contents can be accessed only after data transfer instructions. MIPS use byte addressing. |

| Category | Instruction | Operation |
|---|---|---|
| Arithmetic | Add $s1, $s2, $s3 | S1=s2+s3.<br>There are three operands in this instruction.<br>The data resides in the registers. |
| | Sub $s1, $s2, $s3 | S1=s2-s3.<br>There are three operands in this instruction.<br>The data resides in the registers. |

| | | |
|---|---|---|
| | Addi $s1, $s2, 50 | S1=s2+50.<br>This is add immediate instruction.<br>It has two operands and one constant value,<br>which is directly added to get the result. |
| Data Transfer | Lw $s1, 50($s2)<br><br>Sw$s1,50($s2) | S1=memory [s2+50]<br>Data is transferred from memory to registers.<br>Memory[s2+50]=$s1<br>Data is transferred from register to memory. |

### 1.4.3 Representation of Instructions

- Numbers are represented in computer hardware as a series of high and low electronic signals that are denoted as 1s and 0s. Hence they are considered base 2 numbers.

***A bit or binary digit is a single digit of a binary number and is the smallest indivisible unit of computing.***

- The binary digit may be used to denote high or low, on or off, true or false, or 1 or 0.
- Registers are part of every instruction, hence there must be a convention to map register names into numbers.

  **Example:** add $t0,$s1,$s2.

  This instruction is mapped to its equivalent decimal representation as: The binary

| 0 | 17 | 18 | 8 | 0 | 32 |
|---|---|---|---|---|---|

equivalent representation is given as:

| 000000 | 10001 | 10010 | 01000 | 00000 | 100000 |
|---|---|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

- Each cell is termed as a **field.**
- The binary representation used for communication within a computer system is termed as **Machine Language.**
- **Instruction Format** is a representative form an instruction of fields of binary numbers.

**Fields in MIPS**

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

- Op code is the field that denotes the operation and format of an instruction.

- rs: The first register source operand.
- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation.
- shamt: Shift amount. This is done to adds two zeros to the low-order end of the sign- extended offset field in calculating the address. This operation truncated the sign values.
- op field and is sometimes called the function code.
- The MIPS instructions are designed in the same format for easy manipulation this is in accordance with the design principle Good design demands good compromises.

| Register Name | Number | Usage |
|---|---|---|
| zero | 0 | Constant 0 |
| at | 1 | Reserved for assembler |
| v0 | 2 | Expression evaluation and |
| v1 | 3 | results of a function |
| a0 | 4 | Argument 1 |
| a1 | 5 | Argument 2 |
| a2 | 6 | Argument 3 |
| a3 | 7 | Argument 4 |
| t0 | 8 | Temporary (not preserved across call) |
| t1 | 9 | Temporary (not preserved across call) |
| t2 | 10 | Temporary (not preserved across call) |
| t3 | 11 | Temporary (not preserved across call) |
| t4 | 12 | Temporary (not preserved across call) |
| t5 | 13 | Temporary (not preserved across call) |
| t6 | 14 | Temporary (not preserved across call) |
| t7 | 15 | Temporary (not preserved across call) |
| s0 | 16 | Saved temporary (preserved across call) |
| s1 | 17 | Saved temporary (preserved across call) |
| s2 | 18 | Saved temporary (preserved across call) |
| s3 | 19 | Saved temporary (preserved across call) |
| s4 | 20 | Saved temporary (preserved across call) |
| s5 | 21 | Saved temporary (preserved across call) |
| s6 | 22 | Saved temporary (preserved across call) |
| s7 | 23 | Saved temporary (preserved across call) |
| t8 | 24 | Temporary (not preserved across call) |
| t9 | 25 | Temporary (not preserved across call) |
| k0 | 26 | Reserved for OS kernel |
| k1 | 27 | Reserved for OS kernel |
| gp | 28 | Pointer to global area |
| sp | 29 | Stack pointer |
| fp | 30 | Frame pointer |
| ra | 31 | Return address (used by function call) |

**Fig 1.18: Mapping of register names and numbers**

**Op code values of MIPS instruction**

In the MIPS instruction reg means a register number ranging from 0 and 31. Address means a 16-bit address, and not applicable (n.a.) means this field does not appear in this format. The add and sub instructions have the same value in the op field. The hardware uses the funct field to decide the whether it is addition or subtraction operation using: add (32) or subtract (34).

| Instruction | Format | Op | Rs | Rt | Rd | Shamt | Funct | Address |
|---|---|---|---|---|---|---|---|---|
| Add | R | 0 | Reg | Reg | Reg | 0 | $32_{10}$ | Na |
| Sub | R | 0 | Reg | Reg | Reg | 0 | $^{34}10$ | Na |
| Add immediate | I | $^{8}10$ | Reg | Reg | Na | Na | Na | Constant |
| Lw | I | $35_{10}$ | Reg | Reg | Na | Na | Na | Address |
| Sw | I | $43_{10}$ | Reg | Reg | Na | Na | Na | Address |

### 1.4.4 Logical Operations

The following are the logical operations performed by the processor:

| Logical Operations | MIPS Instructions |
|---|---|
| Shift left | sll |
| Shift right | srl |
| Bit by bit AND | and, andi |
| Bit by bit OR | or, ori |
| Bit by bit NOT | nor |

The first class of such operations is called **shifts.** They move all the bits in a word to the left or right, filling the emptied bits with 0s.

0000 0000 0000 00000 000 0000 0000 0000 $1001_2 = 9_{10}$ After left shifting by four, the new value is 144.

0000 0000 0000 0000 0000 0000 0000 1001 $0000_2 = 144_{10}$

- **Left shift:** Left shifting by i bits is equivalent to multiplying the number by $2^i$.
- **Right Shift**: Right shifting by i bits is equivalent to dividing the number by $2^i$.
- **AND:** This is used in masking of bits.
- **OR:** It is a bit-by-bit operation that places a 1 in the result if either operand bit is a 1
- **NOT:** A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.
- **NOR:** A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands.

| Category | Instruction | Operation |
|---|---|---|
| AND | and $s1, $s2, $s3 | S1=s2&s3 |
| OR | or $s1, $s2, $s3 | S1=s2\|s3 |
| NOR | nor $s1, $s2, $s3 | S1=~(s2\|s3) |
| NAND | nand $s1, $s2, $s3 | S1=~(s2&s3) |
| AND immediate | andi $s1, $s2, 100 | S1=s2&100 |
| OR immediate | Ori $s1, $s2, $s3 | S1=s2\|100 |
| Shift leftlogical | Sll $s1, $s2, 10 | S1=s2<<10 |
| Shift right logical | Srl $s1, $s2, 10 | S1=s2>>10 |

### 1.4.5 Control Operations

Decision making and branching makes the computers more powerful.

**Decision Making:**

Decision making in MIPS assembly language includes two decision-making instructions **(conditional branches):**

**i) Branch if Equal (BEQ):**

beq register1, register2, L1

In this instruction, the go to the statement labeled L1 if the value in register1 is equal to the value in register2.

**ii) Branch if not Equal (BNE):** bne register1, register2, L1

In this instruction, the go to the statement labeled L1 if the value in register1 does not equal the value in register2.

***Conditional branch is an instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.***

**Example:**

Consider the following statement,

**if (i == j) f = g + h; else f = g – h;**

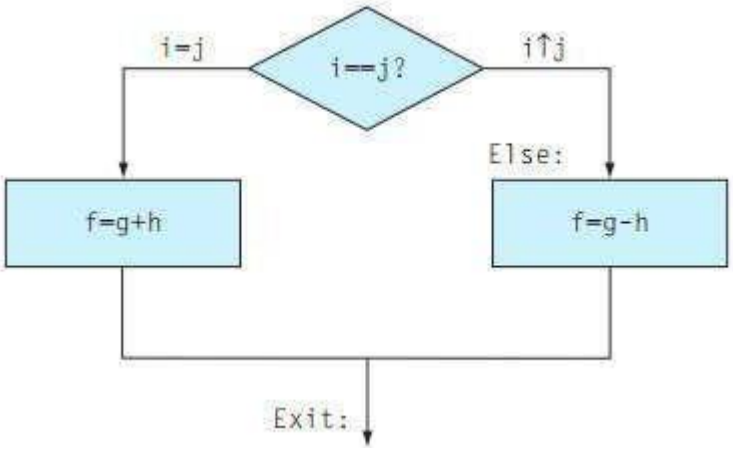



**Fig 1.19: Flowchart for if (i == j) f = g + h; else f = g – h;**

The instruction first compares for equality, using beq. In general, the code will be more efficient if we test for the opposite condition to branch over the code that performs the subsequent then part of if (the label Else is defined below):

bne $s3,$s4,Else # go to Else if i =j

The next assignment statement performs a single operation, and if all the operands are allocated to registers, it is just one instruction:

add $s0,$s1,$s2 # f = g + h (skipped if i =j)

This instruction says that the processor always follows the branch. To distinguish between conditional and unconditional branches, the MIPS name for this type of instruction is jump, abbreviated as j (the label Exit is defined below).

```
j Exit # go to Exit
```

The assignment statement in the else portion of if statement can again be compiled into a single instruction. We just need to append the label Else to this instruction. We also show the label Exit that is after this instruction, showing the end of the if-then-else compiled code:

```
Else: sub $s0, $ s1, $s2 # f = g – h (skipped if i = j)
Exit:
```

Compilers create branches and labels wherever necessary for maintaining flow of the program. Also, the assembler calculates the addresses and relieves the compiler and the assembly language programmer.

**Looping:**

When a set of statements has to be executed more number of times, looping statements are used.

**Example:**

```
while (save[i] == k)
i += 1;
```

i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. The MIPS instructions are:

- The first step is to load save[i] into a temporary register. This operation needs an address. Multiply the index i by 4 and add i to the base of array to obtain the address.
- Add the label Loop to it to branch back to that instruction at the end of the loop: *Loop:*

  ```
  sll  $t1,$s3,2 # Temp reg $t1 = 4 * i
  ```
- To get the address of save[i] , add $t1 and the base of save in $s6:

  ```
  add $t1,$t1,$s6 # $t1 = address of save[i]
  ```
- Use that address to load save[i] into a temporary register:

  ```
  lw $t0,0($t1) # Temp reg $t0 = save[i]
  ```

- The next instruction performs the loop test, exiting if save[i] k: *bne $t0,$s5, Exit # go to Exit if save[i] k*
- The next instruction adds 1 to i :
  *add $s3,$s3,1 # i = i + 1*
- The end of the loop branches back to the while test at the top of the loop. Add the Exit label after it:
  *j Loop # go to Loop*

  *Exit:*

Grouping on instructions that makes compiling easy is through partitioning the assembly language instructions into basic blocks.

***A sequence of instructions without branches except possibly at the end and without branch targets or branch labels except possibly at the beginning are called basic blocks.***

## Case / Switch Statements

These statements allow the programmers to select one among the many options. The simple way is to implement switch is through a sequence of conditional tests using a chain of if-then-else statements. The alternatives are encoded in **jump address table.** The program needs only to index into the table and then jump to the appropriate sequence.

***A table of addresses of alternative instruction sequences is maintained in jump address table.***

The jump table is an array of words containing addresses that correspond to labels in the code. MIPS include a **jump register instruction** (jr), to support the unconditional jump to the address specified in a register. The program loads the appropriate entry from the jump table into a register, and then it jumps to the proper address using a jump register.

| | | |
|---|---|---|
| | | |
| | | |
| | | immediate) |
| Unconditional branch | J L | Goto L<br>(Jump to target address L) |