Chapter 1. Basic Structure of Computers

Information Handled by a Computer

- **Program:**
  - A Program is a sequence of machine Instructions, located and executing from the memory. Each Instruction is a Assembly or Machine Language Instruction.
  - Two Types of Information Handled by a Program: Instruction and Data.
- Instruction
  - Govern the transfer of information within a computer as well as between the computer and its I/O devices
  - Specify the arithmetic and logic operations to be performed
- Data
  - Used as operands of the Instructions
- Any Instruction or Data Encoded in binary code: 0 and 1

Basic Functional Units

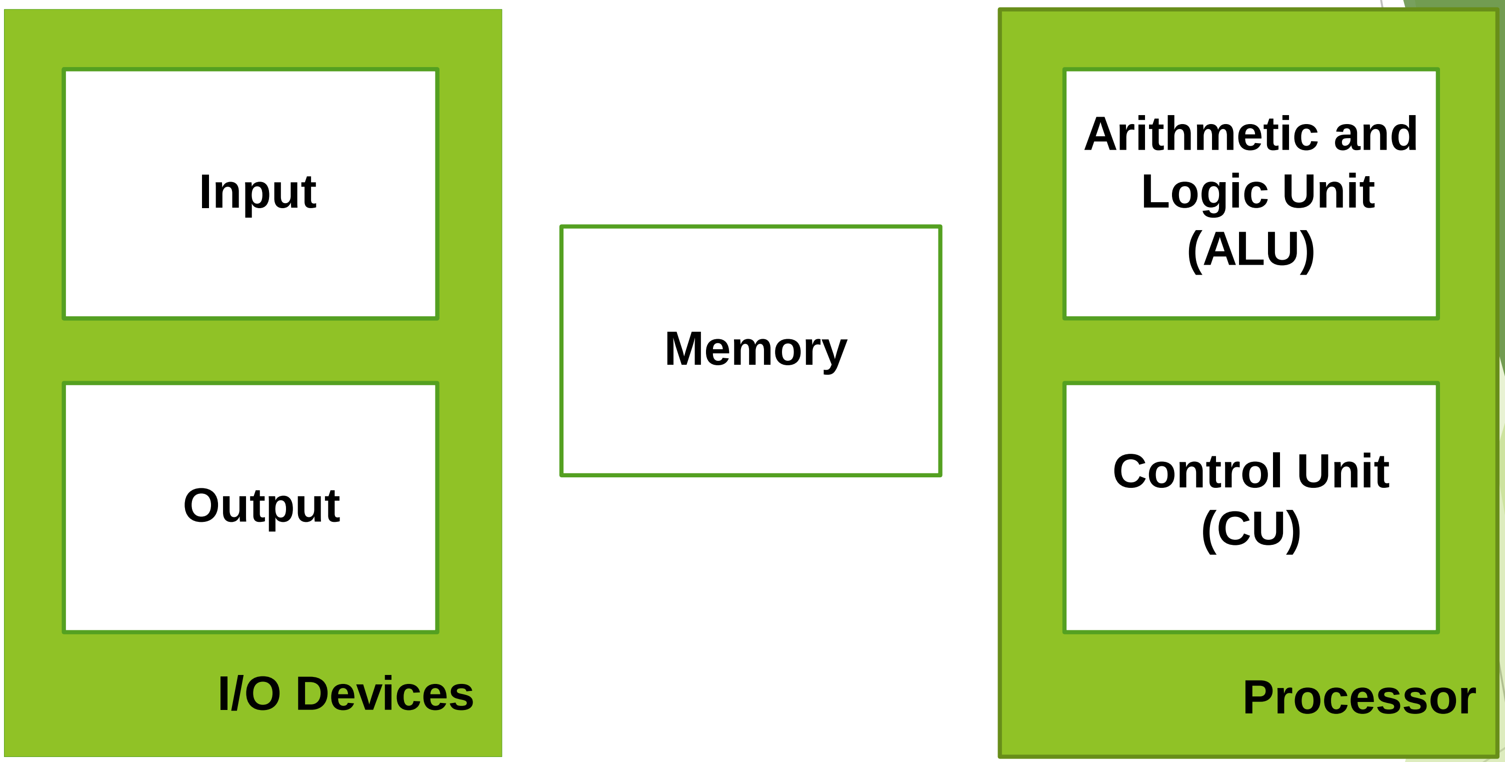


Figure 1.1. Basic functional units of a computer

Basic Functional Units of Computer

- A computer consists of five functionally independent main parts.
  - **Input Unit**
  - **Memory Unit**
  - **Arithmetic and Logic Unit**
  - **Output Unit**
  - **Control Unit**

Input Unit

- Computer accepts coded information through input units which read the data
- Keyboard is used to take input of a letter, digit or special symbol.
- Joysticks, mouse are used as graphic input device.
- Microphone can be used to capture audio input.

Memory Unit

- Store programs and data
- Two types of memory : Primary and secondary
  - Primary Memory / Main Memory
    - Fast - memory access time in microseconds
    - Programs must be stored in memory while they are being executed
    - Large number of semiconductor storage cells.
    - Memory is Processed (read/write) in Words. Usually, 1 word = 4 bytes = 32 bits; This is called 32 bit system. But, In a 64 bit system, 1 word = 64 bits = 8 bytes)
    - Each byte has an Address – called “byte addressable” system
    - Example : RAM(Random Access Memory)
  - Secondary Memory
    - Access time in Milliseconds (which is 1000 times slower than Microseconds access time of RAM, or million times slower than nano-second access time of cache memory!)
    - Example : Magnetic and Optical Disks
- Memory hierarchy- Cache Memory (fastest, smallest, most expensive), Primary Memory (RAM) and Secondary Memory

Arithmetic and Logic Unit (ALU)

- ALU: Most computer operations (i.e., instructions) are executed in ALU of the processor.
  - ✓ Load the Program into memory
  - ✓ Bring the instructions & operands to the processor
  - ✓ Perform operation in ALU
  - ✓ Store the result back to memory or retain in the processor register.

- Registers: Special places/logical units inside the CPU to hold data or values.
  - ✓ Temporarily holds data values during any Arithmetic and Logical operation executing inside the CPU.
  - ✓ Each register can hold 1 word of data
  - ✓ Registers are extremely fast with nanosecond access time (read/write time), compared to the microsecond access time of the Main memory (i.e., RAM) or millisecond access time of magnetic or optical storage (i.e., hard disks or CD/DVD)

Control Unit (CU)

- All computer operations are controlled by the CU
  - Performs a sequence of micro-instructions to implement some meaningful task or sub-task.
  - For example, Multiplication/Division by a sequence of Shift and Add/Sub.

- The timing signals that govern the I/O transfers, data transfer between the processor and the memory are also generated by the control unit.

- Control unit is usually distributed throughout the machine instead of standing alone.

Operations of a computer

- Accept information in the form of programs and data through an input unit and store it in the memory.
- Fetch the information stored in the memory, under program control, into the ALU, where the info is processed.
- Output the processed information through an output unit.
- Control all activities inside the machine through the CU.

Basic Operational Concepts

A Typical Instruction

- MOV LOCA, R0

- **General format:**

  Instruction = Operation destination_operand source_operand

- Moves the operand of a register R0 in the processor to a memory location LOCA.

- The original contents of R0 are preserved.

- The original contents of LOCA is overwritten.

Load and Store Instruction

- Instruction that Moves data from Memory to Register is called LOAD instruction (e.g. MOV R0, LOCA)

- Instruction that moves data from Register to Memory is called STORE instruction (e.g. MOV LOCA, R0)

Another Typical Instruction

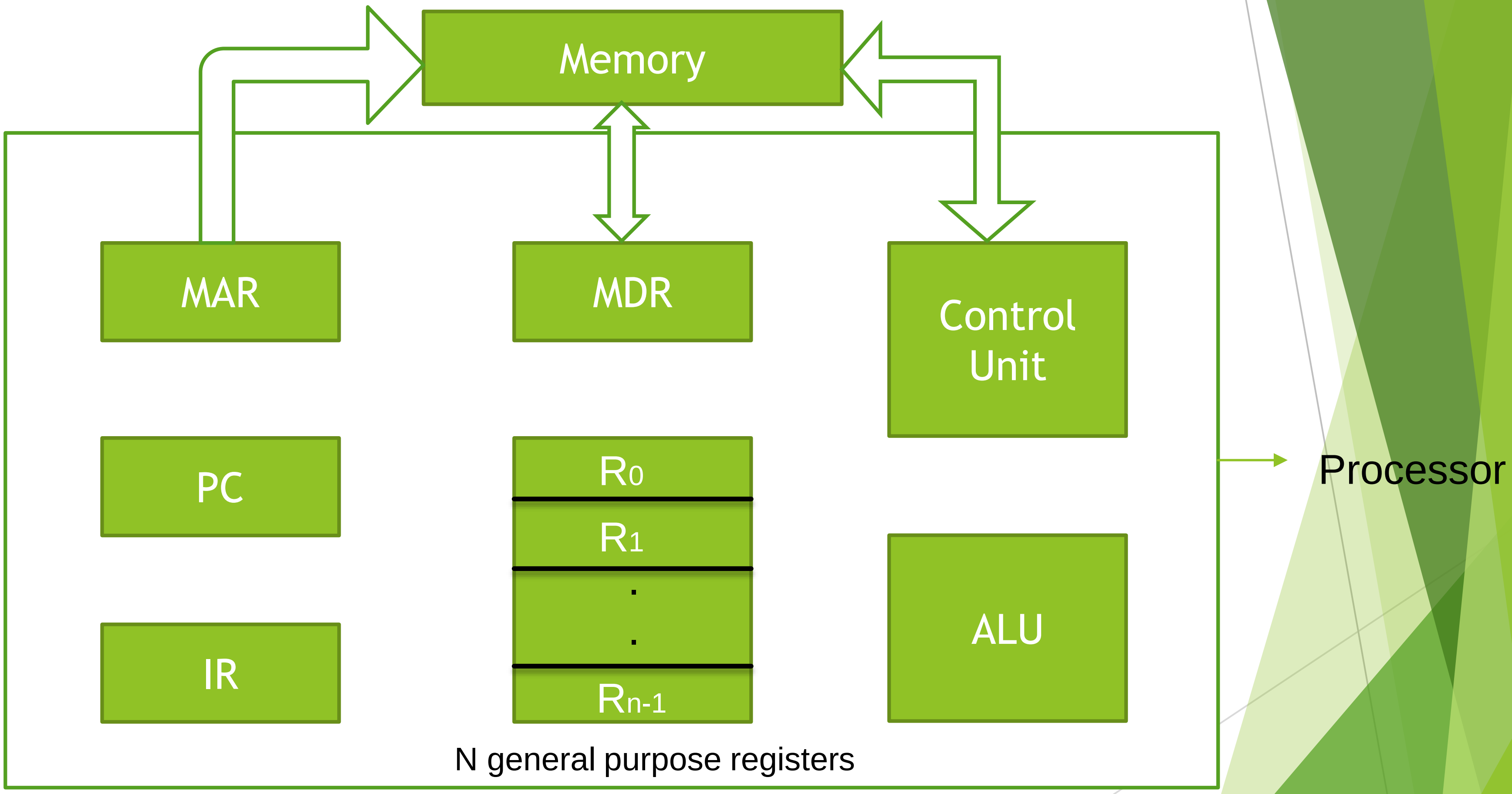
- ADD   LOCA, R0

- General format:

  Instruction = Operation Destination_operand Source_operand

- Add the operand in a register R0 in the processor with the operand at memory location LOCA.
- Place the sum into the memory location LOCA.
- The original contents of R0 are preserved.
- The original contents of LOCA is overwritten.

Connections Between the Processor and the Memory



Examples of a Few Registers:

- **Instruction register (IR):** Holds the instruction that is currently executing by the CPU
- **Program counter register (PC):** Points to (i.e., holds the address of) the next instruction that will be fetched from the memory to be executed by the CPU
- **General-purpose registers (R_0 - R_{n-1}):** generally holds the operands for executing the instructions of current program
- **Memory address register (MAR):** Holds the memory address to be read. A read signal from the CPU to the memory module reads the **word** address held by the MAR register
- **Memory data register (MDR):** Facilitates the transfer of operands/data to/from Memory from/to the CPU

Basic Operating Steps for Executing a Program

- Programs reside in the main memory (RAM) through input devices
- PC register's value is set to the first instruction

Repeat the following Steps Until the "END" instruction is executed

- The contents of PC are transferred to MAR
- A Read signal is sent by CU to the memory
- The Memory module reads out the location addressed by MAR register.
- The contents of that location is loaded into (returned by) MDR.
- The contents of MDR are transferred to IR register
- Decode and execute the instruction at IR
- PC is incremented properly to point to the next instruction

Decoding and Execution a Typical Instruction

- Get operands for ALU
  - The operand may already in a General-purpose register
  - Or, may be fetched from Memory (send address to MAR - send Read signal to Memory module - Wait for MFC signal (WMFC) from Memory - Get the operand/data from MDR)
- Perform operation in ALU
- Store the result back
  - Store in a general-purpose register
  - Or, store into memory (send the write address to MAR, and send result to MDR – Write signal to Memory – WMFC)
- Meanwhile, PC is incremented to the next instruction

Interrupt

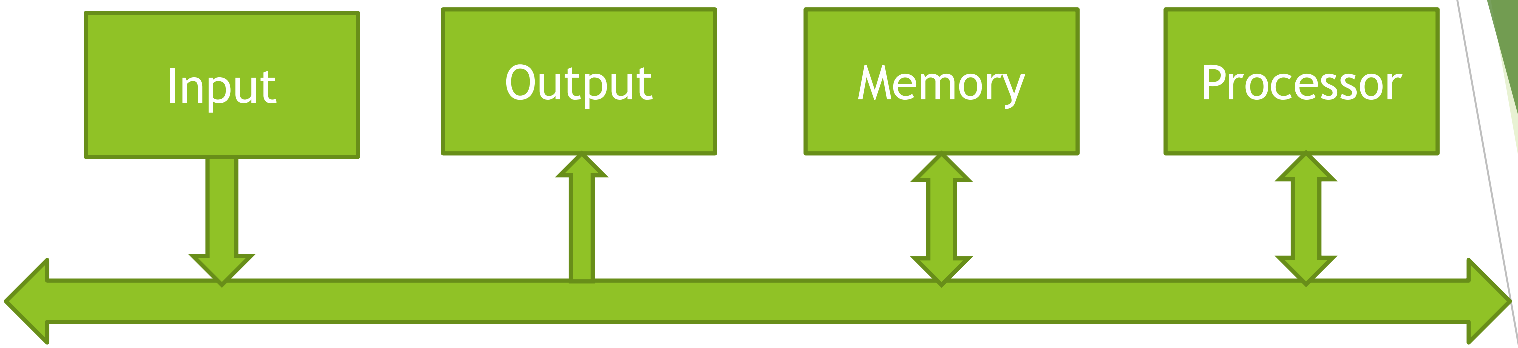
- Normal execution of programs may be preempted if some device requires urgent servicing. e.g., a key pressed in Keyboard, Data arrived in Modem, Printer is ready.
- The normal execution of the current program must be interrupted if the device raises an interrupt signal.
- It is a request from an I/O device for service by the processor.
- The processor provides the required service by executing an appropriate interrupt-service routine
- Current system information backup and restore (PC, general-purpose registers, control information, specific information)

Bus Structures

- There are many ways to connect different parts inside a computer together.
- A group of lines/wires that serves as a connecting path for several devices is called a bus.
- In addition to the lines that carry the data the bus must have lines for address and control purposes.
- For example, set of connection lines of the motherboard between CPU and RAM, Hard disk, I/O devices, etc.

Single Bus Structure

- The simplest way to interconnect functional units is to use a single bus
- Single-bus Architecture: **Basic Block Diagram**



- **All units are connected to this bus**
- **Bus can be used for only one transfer at a time**
- **Only two units can actively use the bus at any given time**
- **Low cost and flexible for attaching peripheral devices**

Speed Issue

- Different devices have different transfer/ operate speed.
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- To solve this problem a common approach is to use buffers.
- For example: Keyboard buffer holds the immediate key pressed, printer buffer holds the next character to be printed, Modem buffer holds the last, say 10K, data bytes arrived/to be sent

Performance

- The most important measure of a computer is how quickly it can execute programs i.e. runtime of programs.
- Three factors affect performance:
  - **Hardware design** (e.g., CPU clock rate)
    - 1GHz CPU => 1 Billion Hz => 10^9 clock cycles/sec (Hz=cycles/sec)
      - **1 basic operation (e.g., integer addition) possible in 1 cycle => 1 billion basic operations (10^9 integer additions!) possible in 1 sec.**
    - 1Mhz => 1 Million Hz => 10^6 clock cycles/sec
  - **Instruction set architecture (ISA)** (e.g., CISC or RISC ISA)
    - CISC => instructions complex, more capable, but runs slower
    - RISC => instructions Simple, runs faster, but less capable
  - **Compiler**
    - how efficient your compiler to optimize your code for pipelining.

Performance

- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.

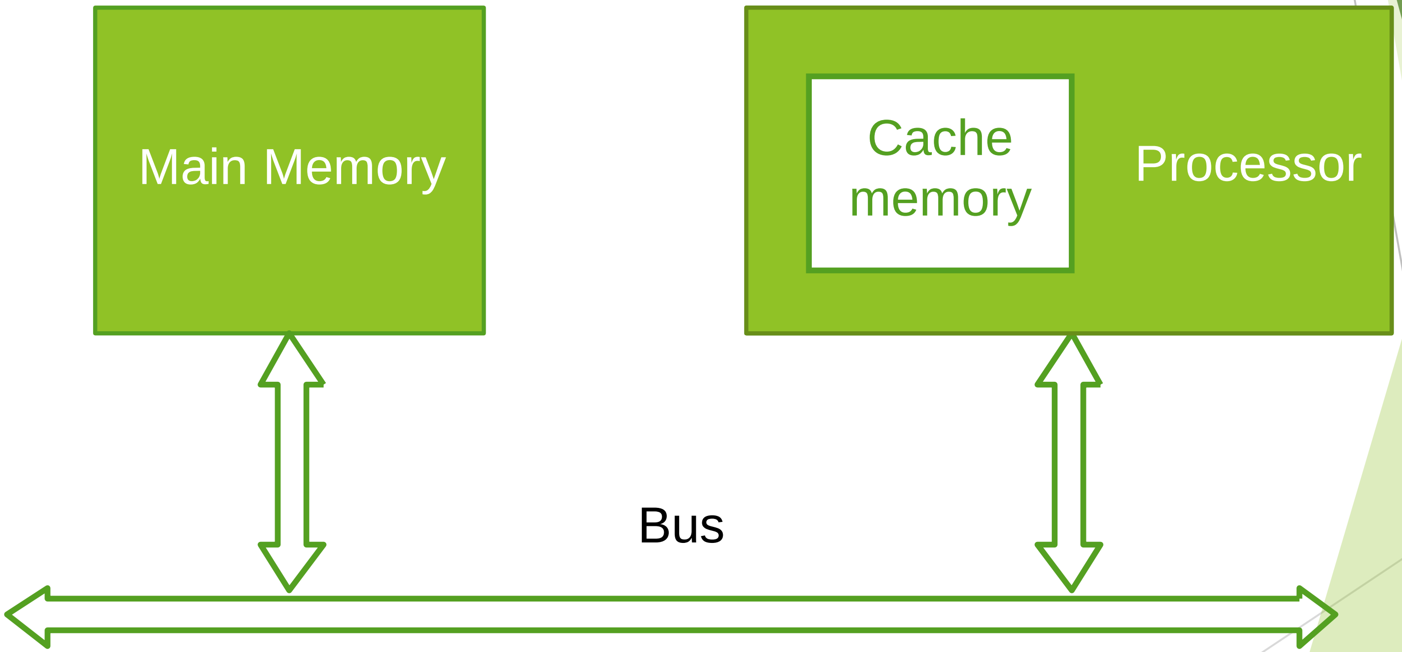


Figure 1.5: The Processor Cache.

Performance

- The Processor and a relatively small Cache Memory can be fabricated on a single integrated circuit chip.
- The internal speed of performing the basic steps of instruction processing on such chips is very high.
- Cache memory is Costly.
- Memory management.

Processor Clock

- Processor circuits are controlled by a timing signal called a clock.
- Clock defines regular time intervals known as clock cycles.
- The inverse of the length of one clock cycle is known as clock rate.
  - Clock Rate = 1 GHz = 10^9 Hz = 10^9 cycles/second or 10^9 clock pulses per second. It also means it has a Clock Cycle of $1/10^9 = 10^{-9}$ sec = 1 ns (nanosecond).
  - 4GHz CPU => 4×10^9 cy/sec => 1 clock cycle = 0.25 ns
  - 500 MHz => 500×10^6 cycles/sec => 2 ns clock pulses
  - 1 MHz = 10^6 cycles/sec; 1KHz=10^3 cycles/sec
  - 1GHz=1000MHz, 1MHz=1000KHz, 1KHz=1000Hz
- The execution of each instruction is divided into several basic steps, each of which completes in one clock cycle. (e.g., integer Addition/Subtraction = just 1 cycle, but a Division may require as many as around 30 cycles)
- Hz (Hertz) – cycles per second (clock cycles / second)
- Million is denoted by Mega(M)
- Billion is denoted by Giga(G)

Basic Performance Equation

- T – processor time required to execute a program that may have been prepared in high-level language. Unit: second
- N – Dynamic Instruction Count. It is the number of actual machine language instructions needed to complete the execution. N is computed considering loops, repeated function calls, recursion, etc. Unit: instructions
- S – average number of basic steps (or, clock cycles) needed to execute one machine instruction. Each basic step completes in one clock cycle. Unit: cycles/instruction
- R – clock rate: cycles/sec
- The execution time T of a program that has a dynamic instruction count N is given by:

$$\boldsymbol{T} = \frac{\boldsymbol{N} \times \boldsymbol{S}}{\boldsymbol{R}} \qquad \text{unit : second because } \frac{intructions \times cycles/instruction}{cycles/sec}$$

- To improve T we have to reduce N and S, Increase R
  - The value of N is reduced if the source program is compiled into fewer machine instructions
  - The value of S is reduced if instructions have a smaller number of basic steps to perform
  - Using a higher frequency clock increases the value of R which means that the time required to complete a basic execution step is reduced.

Performance Evaluation

- Example: A program with dynamic instruction count of 1000 instructions, each instruction taking 5 cycles on average and running at a speed of 1KHZ (R = 103 0r 1000 cycles/second), what will be the program execution time T?
- Answer:

$$T = \frac{N \times S}{R}$$

$$T = \frac{1000\ instructions \times 5\ cycles/instruction}{1000\ cycles/second}$$

= 5 second

Instruction Throughput

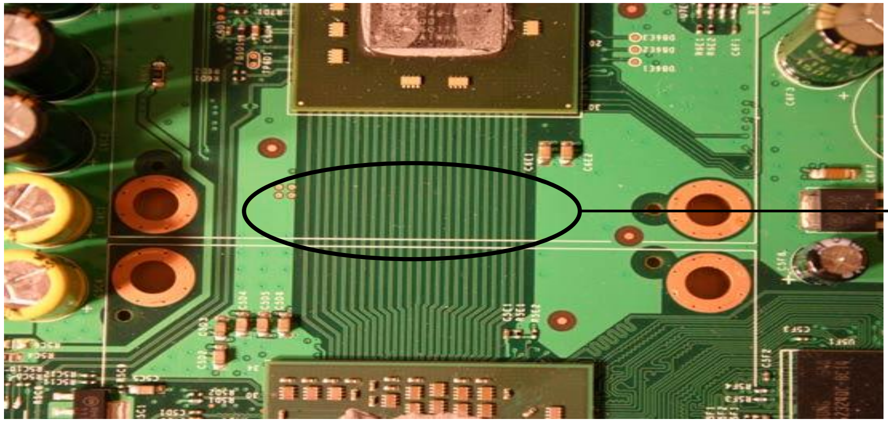
- Instruction throughput is defined as the number of instructions executed per second.

$$P_S = \frac{R}{S}$$

- Unit : instructions/second because $\frac{cycles/second}{cycles/instruction}$

Pipelining and Superscalar Operation

- Instructions are not necessarily executed one after another.
- Pipelining is the capability of overlapping the execution of successive instructions (i.e., parallel/simultaneous execution of multiple instructions).
- Superscalar operation is multiple instruction pipelines are implemented in the processor.
- Goal is to increase Throughput > 1



parallel bus lines
on a motherboard

Improving Performance: Increasing Clock Rate

- Improve the integrated-circuit (IC) technology to make the logic circuits faster
  - Reduce the time needed to complete a basic step
  - Reduce the amount of processing done in one basic step

- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor’s operation equally except the time to access the main memory.

Improving Performance: Effect of Instruction Set Architectures (ISA)

- Tradeoff between N and S
  - **Reduced Instruction Set Computers (RISC): simpler instructions** => N$\uparrow$, S$\downarrow$, Better than CISC, because Pipelining is more effective for RISC
  - **Complex Instruction Set Computers (CISC): Complex instructions** => N$\downarrow$, S$\uparrow$, Not Good, As not suitable for Pipelining. Instructions complex, more capable => the program gets smaller in size (reduced N*)*, but complex instructions increase S and hampers pipeline. Example of CISC: Intel processors

- So, A key consideration is the use of Pipelining

- S is close to 1, means the number of cycles per instruction is nearly ideal (close to 1) (e.g. RISC processors)

- RISC is Better, because easier to implement efficient pipelining with simpler instruction sets.

Improving Performance: Compiler based speedup

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.
- Goal of a Smart compiler is to reduce $N \times S$ to reduce program runtime T to improve performance.

Performance Measurement

- Measure computer performance using benchmark programs (a set of sample programs, e.g., word processing programs, games, media (audio/video) playback , I/O intensive programs, etc.).
- **System Performance Evaluation Corporation (SPEC)** selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Reference computer: A previous, renowned computer system, picked by SPEC

$$SPEC\ rating = \frac{Running\ time\ on\ the\ reference\ computer}{Running\ time\ on\ the\ computer\ under\ test}$$

$$SPEC\ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Multiprocessors and Multi-computers

- Multiprocessor computer
  - Good for Executing several different application tasks in parallel
  - Good for Executing subtasks of a single large task in parallel
  - All processors have access to all of the memory that is shared
  - Cost– processors, memory units, complex interconnection networks

- Multi computers
  - Each computer only have access to its own memory
  - Example: a Network of computers, such as a LAN (Local Area Network), WAN (wide area network) or MAN (metropolitan area network) etc.
  - Exchange message via a communication network – message-passing multi-computers