Computer Organization

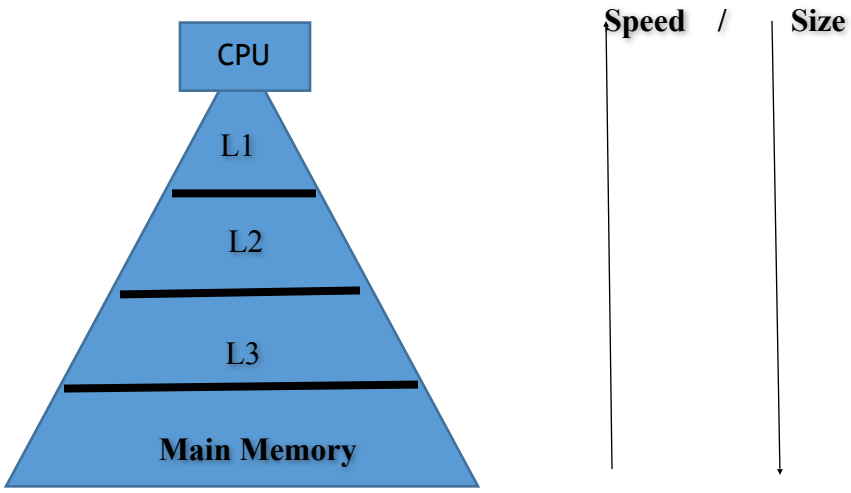
Presentation Summary

Parts of the Central Processing Unit (CPU)

1. The **controller** acts as the circuit board for the computer. This component is responsible for directing both sent and received signals all through the computer, as well as ensuring that all operations occur at the correct time and order.
2. The **Arithmetic Logic Unit (ALU)** is responsible for processing data through its ability to perform arithmetic and bitwise operations on integer binary numbers.
3. **Registers** serves as a temporary storage location for data and/or control information. This type of memory is usually considered to be a **faster** method of access than internal memory because it's location and frequent need to be accessed.
4. **Buses** are simple circuits connected to the motherboard, which permits data to be transferred all through the computer. The speed of the bus is measured by the amount of data that is able to move across it concurrently, commonly referred to as **MHz**.
5. The **internal memory (**or **cache)** serves as a fast and temporary storage space for data that isn't being used in the register. Data stored in the internal memory can either be (1) written to the RAM or back into the registers.

Memory Hierarchy

Adoption of this concept discusses the importance of data storage placement, in relation to a computer's level of performance. Cache has been divided into three different levels of storage space, with the intent to adhere to the dilemma concerning the **speed** and **size** of memory.



Main Idea:

1. Ability to use a combination of memory types
    a. **Large** amounts of cheaper **slower** memory
    b. **Smaller** amounts of more expensive **faster** memory
2. Profit through the usage of **locality of reference**
    a. **Spatial Locality –** Use of data elements that are located relatively close to one another
    b. **Temporal Locality –** Reuse of data elements that have been accessed in a relatively small amount of time

Assembly Language

Low-level method of programming which serves as a more "human friendly," language in comparison to machine code language. Because processors are only able to understand **machine code language** (strings of 1's and 0's), **assembly language** provides an easier method of understanding for programmers.

Basic Syntax

Sections

1. Data
    a. Declared initialized data or constants
    b. "**section.data**," signifies the beginning of the section
2. Bss
    a. Declared variables
    b. "**section.bss**," signifies the beginning of the section
3. Text
    a. Used for storing the program's code
    b. "**global_start**," informs kernel where to begin execution

Statements

1. Executable instructions (opcode)
    a. Instructs the behavior of the processor

b. Each instruction generates one machine language instruction
    i. Only requires an operation
    ii. Label, operands, and comments are optional
2. Assembler Directives (pseudo-op)
    a. Non-Executable and **cannot** generate machine language instructions
    b. Sets aside space for variables
    c. Establishes start address for a program
3. Macros
    a. Provides a means for generating commonly used sequences of assembly instructions/statements
    b. Code only needs to be written once
        i. Sequence of codes must start with **MACROS**
        ii. Second line must declare the name, to allow reuse of code
        iii. **MEND** must be written after last line of code

Pipelining (5 Stage)

Technique which implements **instruction-level parallelism** within a single processor, thus increasing the **throughput (number of instructions executed per given unit of time).**

Classic RISC pipeline:

1. Instruction Fetch (IF)
    a. *Fetch* instruction from memory
    b. Update program counter
        i. Default +4 bytes, unless there exists a **branch instruction**.
    c. Place instruction into IF/ID register
2. Instruction Decode (ID)
    a. *Decode* instruction

        i. If the instruction is a **branch**, update program counter accordingly

    b. Reads register

        i. Place read values into ID/EX register

3. Execution (EX)

    a. Preform ALU Operation

        i. Extra steps must be taken if in the instruction consists of:

            1. **(lw) Load** value from memory, then copy data from **memory** into a **register**

            2. **(sw) Store** value from memory, then copy data from a **register** to **memory**

    b. Place output of ALU into EX/MEM register

4. Memory Access (MEM)

    a. Access Memory, *if needed*

        i. Treat this stage as a "dummy-phase," if instruction was a simple ALU computation

    b. Place output of ALU into MEM/WB register

5. Write Back

    a. Read MEM/WB register and write output to register file.

A **branch instruction** causes a computer to update the program counter, thus creating a change in the instruction sequence that must be executed. This instruction allows the program counter to deviate from the default +4 bytes.