joseph innocent matabaro

ditnrb 531724

St paul's University

Computer Organisation and Architeture CAT 1

1. ASCII Sorting Order: In ASCII, characters are sorted based on their numerical values. Here's how the strings would be sorted:

214: Numbers come before uppercase letters in ASCII.

ADEC: Uppercase letters are sorted alphabetically. So the order would be: 214, ADEC

2. Decimal Equivalent of Binary Numbers:

10011101:

(1 x 2^7) + (0 x 2^6) + (0 x 2^5) + (1 x 2^4) + (1 x 2^3) + (1 x 2^2) + (0 x 2^1) + (1 x 2^0)

128 + 0 + 0 + 16 + 8 + 4 + 0 + 1 = 157

100101: (1 x 2^5) + (0 x 2^4) + (0 x 2^3) + (1 x 2^2) + (0 x 2^1) + (1 x 2^0)

32 + 0 + 0 + 4 + 0 + 1 = 37

3. EBCDIC Coding for "NETQWORK" (Hexadecimal):

Let's find the EBCDIC codes for each letter and represent them in hexadecimal:

N: D5

E: C5

T: E3

Q: D8

W: E6

O: D6

R: D9

K: C2

So, the EBCDIC coding for "NETQWORK" is: D5C5E3D8E6D6D9C2. Since each character requires 1 byte, representing "NETQWORK" will require 8 bytes.

4. De Morgan's Law in Boolean Algebra: De Morgan's Laws provide a way to simplify Boolean expressions. They state:

(NOT A) AND (NOT B) = NOT (A OR B)

(NOT A) OR (NOT B) = NOT (A AND B)

Example: Let's say A = TRUE and B = FALSE

(NOT A) AND (NOT B) = (FALSE) AND (TRUE) = FALSE

NOT (A OR B) = NOT (TRUE OR FALSE) = NOT (TRUE) = FALSE

(NOT A) OR (NOT B) = (FALSE) OR (TRUE) = TRUE

NOT (A AND B) = NOT (TRUE AND FALSE) = NOT (FALSE) = TRUE

5. Four Types of CPU Registers:

CPU registers are small, high-speed storage locations used by the processor to quickly access frequently used data and instructions. Here are four common types:

Accumulator (AC): Used to store intermediate results during calculations. For example, when adding two numbers, one might be loaded into the AC, the other added to it, and the result remains in the AC.

Program Counter (PC): Stores the address of the next instruction to be executed. The CPU fetches the instruction from the memory location pointed to by the PC. After the instruction is fetched, the PC is incremented to point to the next instruction.

Memory Address Register (MAR): Holds the memory address of the data or instruction that the CPU needs to access. The CPU uses the MAR to specify which memory location it wants to read from or write to.

Memory Data Register (MDR): Contains the data being transferred to or from memory. When reading from memory, the data is placed in the MDR. When writing to memory, the data to be written is placed in the MDR.

6. RISC vs. CISC Architecture:

a. RISC (Reduced Instruction Set Computing):

Focus: Simplified instruction set.

Instructions: Fewer instructions, typically of fixed length.

Complexity: Simpler hardware, more complex software (compilers).

Example: A RISC processor might have separate instructions for loading data from memory and performing arithmetic operations.

b. CISC (Complex Instruction Set Computing):

Focus: Complex instruction set.

Instructions: Many instructions, often of variable length, that can perform complex tasks.

Complexity: More complex hardware, simpler software (compilers).

Example: A CISC processor might have a single instruction that can load data from memory, perform an arithmetic operation, and store the result back to memory.

7. Fetch-Execute Cycle:

The fetch-execute cycle is the fundamental operation cycle of a computer. It describes the process by which the CPU retrieves an instruction from memory, decodes it, and executes it.

Fetch: The CPU fetches the instruction from memory location pointed to by the program counter (PC).

Decode: The instruction is decoded to determine what operation needs to be performed.

Execute: The CPU performs the operation specified by the instruction. This might involve reading data from registers or memory, performing arithmetic or logical operations, or writing data back to registers or memory.

Increment PC: The program counter is incremented to point to the next instruction.

This cycle repeats continuously, allowing the computer to execute programs.

8. Structure of a Process in an OS Environment:

A process is an instance of a program in execution. The OS manages processes, providing them with resources and ensuring they don't interfere with each other. A process typically has the following structure:

Text Section: Contains the executable code of the program. This is usually read-only.

Data Section: Contains global variables and static variables that are initialized.

BSS Section: (Block Started by Symbol) Contains uninitialized global and static variables. The OS typically initializes this section to zero.

Heap: Dynamically allocated memory used during the execution of the program (e.g., using malloc in C).

Stack: Used to store local variables, function arguments, and return addresses during function calls. The stack grows and shrinks as functions are called and return.

Process Control Block (PCB): A data structure maintained by the OS that contains information about the process, such as its ID, state (e.g., running, waiting, ready), priority, memory allocation, and open files. The PCB is essential for the OS to manage and control processes.