**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #2**

**Edafe Ukoh #20200**

1. **Discuss how stack architecture computer works by giving examples, such as arithmetic express in reverse polish notation. And compare the pros and cons between stack-based virtual machine and register-based virtual machine**

Computers using stack-based CPU organization rely on a data structure known as a stack, which is a list of data words. The stack follows the Last In First Out (LIFO) access method, a common approach in many CPUs. A register called the Stack Pointer (SP) holds the address of the topmost element in the stack. In this organization, Arithmetic Logic Unit (ALU) operations are conducted on stack data, meaning both operands must be on the stack for operations to occur, and results are also placed back onto the stack.

The primary operations performed on the stack are Push and Pop, both of which operate from one end of the stack:

1. Push:

- Inserts an operand at the top of the stack and increments the stack pointer register.

- Places a data word at the specified address at the top of the stack.

2. Pop:

- Removes an operand from the top of the stack and decrements the stack pointer register.

- Deletes the data word from the top of the stack and stores it at a specified address.

Operational instructions in stack-based CPUs do not need an address field since operations are performed on the top two stack elements. These instructions contain only the opcode, simplifying the instruction format. For example, a subtraction operation will pop the top two elements, subtract them, and push the result back onto the stack.

Example Execution

Consider an arithmetic expression in Reverse Polish Notation (RPN):

Input: `["7", "4", "5", "2", "+", "-3", "\*", "/", "\*", "8", "+", "3", "+"]`

Output: 11

Explanation of the computation follow these steps:

1. Push 7 onto the stack. ----------------[7]

2. Push 4 onto the stack. ----------------[7, 4]

3. Push 5 onto the stack. ----------------[7, 4, 5]

4. Push 2 onto the stack. -------------------- [7, 4, 5, 2]

5. Process `+` (add 5 and 2) ---------------- [7, 4, 7]

6. Push -3 onto the stack. --------------------[7, 4, 7, -3]

7. Process `\*` (multiply 7 and -3) -----------[7, 4, -21]

8. Process `/` (divide 4 by -21) --------------[7, 0]

9. Process `\*` (multiply 7 by 0) ------------- [0]

10. Push 8 onto the stack) ------------- [0, 8]

11. Process `+` (add 0 and 8) -----------------[8]

12. Push 3 onto the stack ---------------------[8, 3]

13. Process `+` (add 8 and 3) -----------------[11]

The result is 11.

**Pros of Stack-Based CPU Organization**

Efficient Computation: Facilitates the computation of complex arithmetic expressions.

Fast Execution: Operations are swift since operand data is stored in consecutive memory locations.

Compact Instructions: Instructions are shorter as they do not require an address field.

**Cons of Stack-Based CPU Organization**

Increased Program Size: More instructions might be needed due to stack operations.

**Register-Based Virtual Machines**

* **Pros**

1. Performance: Faster execution due to direct access to operands, reducing the overhead of stack operations.

2. Optimizations: Enhanced opportunities for optimizations like register allocation and instruction reordering. Reduced need for stack management simplifies these optimizations.

3. Parallelism: Easier to exploit instruction-level parallelism due to the explicit nature of register-based instructions.

4. Flexible Instruction Set: Supports a richer set of instructions, allowing for more complex operations.

* **Cons**

1. Complexity: Instruction decoding, and execution are more complex due to explicit operand addresses.

2. Compiler Complexity: Requires more sophisticated compiler design to efficiently manage registers and generate optimized code.

3. Larger Bytecode: Instructions tend to be larger because they include operand addresses.

4. Fixed Register Limit: Limited number of registers can constrain performance and require careful management.

**2. Processors are one of the most important components in computing systems. Its**

**performance can have a big impact on the whole system. Discuss about processor**

**design metrics and benchmarking tools.**

**Processor Performance Evaluation: Metrics and Benchmarking Tools**

Evaluating processor performance involves identifying and analyzing various metrics to understand and improve the overall efficiency and speed of computing systems. As technology advances, new architectures emerge, requiring continual assessment to pinpoint bottlenecks and innovate better designs. Here we explore key performance metrics and benchmarking tools that are crucial for such evaluations.

**Key Performance Metrics**

1. **Clock Speed (Frequency)**

Measured in gigahertz (GHz), it indicates the number of cycles a processor can execute per second. Higher clock speeds generally translate to faster execution of instructions, though efficiency also depends on the architecture.

2. **Instructions Per Cycle** (IPC)

The number of instructions a processor can execute in one clock cycle. Higher IPC indicates better performance as the processor can perform more work in each cycle.

3. **Power Consumption and Thermal Design Power (**TDP)

TDP measures the maximum amount of heat a processor is expected to generate under maximum workload, measured in watts. Lower TDP means less cooling is required, leading to quieter and more energy-efficient systems.

4. **Energy Efficiency**

Performance per watt, indicating how efficiently a processor uses power. More energy-efficient processors are crucial for battery-operated devices and environmentally friendly computing.

5. **Die Size and Transistor Count**

Relates to the manufacturing process size, measured in nanometers (nm), and the number of transistors on a chip. Smaller nodes and higher transistor counts typically improve performance and energy efficiency.

6. **Latency and Throughput**

Latency is the delay in data transfer, while throughput is the amount of data that can be transferred per unit time. Lower latency and higher throughput enhance overall system responsiveness and performance.

7. **Scalability**

Supported by multi-core and multi-threading technologies, it indicates the processor’s ability to handle increasing workloads. Essential for multitasking and parallel processing, improving performance in multi-threaded applications.

8. Architectural Features

- Description: Features like pipelining, superscalar execution, and out-of-order execution.

- Impact: These features improve performance by optimizing the execution of instructions and enhancing parallel processing capabilities.

9. Cache and Memory Hierarchy

- Description: Amount and structure of cache memory and its organization relative to the processor.

- Impact: Larger and more efficient caches can significantly reduce the time needed to access frequently used data, improving performance.

**Benchmarking Tools**

Benchmarking tools provide standardized methods to evaluate and compare processor performance. Here are some of the key tools:

* SPEC CPU: Evaluates compute-intensive tasks with SPECint for integer operations and SPECfp for floating-point operations. Provides a comprehensive performance score, widely recognized in the industry.
* PassMark: Measures various CPU functions like integer and floating-point operations. Useful for comparing overall system performance and individual component performance.
* Geekbench: Offers cross-platform benchmarks for single-core and multi-core performance. Provides a quick and comprehensive performance comparison across different devices and architectures.
* Cinebench: Assesses multi-core rendering capabilities. Popular for evaluating performance in multimedia and content creation applications.
* AIDA64: Provides detailed system information and benchmarks for CPU and memory. Useful for detailed system diagnostics and performance tuning.
* 3Dmark: Primarily a GPU benchmark, but includes tests for CPU performance, particularly in gaming scenarios. Widely used in the gaming industry to assess system performance under gaming loads.
* Linpack: Measures floating-point performance, especially in high-performance computing (HPC) contexts. Used for evaluating the computational power of supercomputers.
* Prime95: Used for CPU stress testing and stability checks. Essential for testing the stability and reliability of processors under heavy loads.

**Importance of Performance Metrics**

Understanding performance metrics is crucial for both consumers and designers. From a purchasing perspective, it helps in making informed decisions, cutting through marketing hype, and ensuring judicious choices by comparing performance, cost, and cost-per-performance ratios. From a designer's perspective, it aids in evaluating design options to achieve the best performance improvements with minimal costs.

**CPU Performance Evaluation**

The core of performance evaluation often revolves around time, primarily execution time (response time) and throughput. Execution time, the period from task initiation to completion, is critical for individual users, while throughput, the total amount of work done in a given time, is essential for data center managers. Performance is inversely related to execution time; thus, minimizing execution time maximizes performance.

For example, if computer A runs a program in 10 seconds and computer B takes 20 seconds for the same task, A is twice as fast as B. Execution time can be calculated by multiplying the total number of clock cycles by the clock cycle time or dividing the total clock cycles by the clock rate. To enhance performance, either the clock cycle time must be reduced or the number of clock cycles per program minimized.

**Factors Influencing CPU Execution Time**

Several factors influence the three key components of CPU execution time: instruction count, clock cycle time, and cycles per instruction (CPI):

1. Instruction Count

2. Clock Cycle Time

3. Cycles Per Instruction (CPI)

Designers must optimize these factors to reduce execution time and improve performance. This involves understanding how different instructions and their implementation, memory hierarchy, and I/O handling contribute to performance. For instance, improving cache efficiency or optimizing instruction execution can significantly impact overall performance.

These tools standardize performance evaluations, optimize processor designs, inform purchasing decisions, and drive competition and innovation. By providing insights into various performance aspects such as single-core efficiency, multi-core scalability, power consumption, and system stability, benchmarking tools help identify strengths and weaknesses, guiding improvements in processor technology.