**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**Homework Assignment #2**

**Due day: 6/16/2024**

**Instruction:**

1. **Push the answer sheet to Github in word file**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. 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 (1.5~2 pages)
5. 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 (1.5~2 pages)

**ANSWER 1**

A stack architecture computer operates primarily using a stack, a last-in, first-out (LIFO) data structure. Instructions in such a system do not explicitly reference operands by memory location or register, but instead implicitly use the top elements of the stack. This type of architecture is particularly well-suited for arithmetic operations and expression evaluation using Reverse Polish Notation (RPN).

**Reverse Polish Notation Example**

In Reverse Polish Notation, operators follow their operands. This eliminates the need for parentheses to dictate operation order. For instance, the arithmetic expression “(1 + 3) \* 3” in standard notation would be written as “1 3 + 3 \*” in RPN.

**Push Operation:** The **push** operation adds (or "pushes") an element onto the top of the stack. This increases the stack's size by one element.

**Pop Operation:** The **pop** operation removes (or "pops") the top element from the stack. This decreases the stack's size by one element and typically returns the removed element for further use.

Here's how a stack-based computer would evaluate “1 3 + 3 \*”:

1. **Push 1**: Stack = [1]
2. **Push 3**: Stack = [1, 3]
3. **Add**: Pop 3 and 1, push result (4): Stack = [4]
4. **Push 3**: Stack = [4, 3]
5. **Multiply**: Pop 3 and 4, push result (12): Stack = [12]

The final result of the expression is 12.

**Stack-Based Virtual Machine**

**Pros**

1. **Simple Instruction Set**: Stack-based virtual machines (VMs) have simpler instructions since operations implicitly refer to the top of the stack. This simplicity can lead to a smaller VM codebase and easier implementation.
2. **Compact Code**: Instructions in stack-based VMs tend to be more compact because they do not need to include operand addresses, leading to smaller programs.
3. **Ease of Compilation**: Some high-level language constructs, especially expressions, map more naturally to stack operations, simplifying the compilation process.

**Cons**

1. **Performance Overhead**: Stack operations can introduce performance overhead due to the additional push and pop operations required.
2. **Limited Parallelism**: The inherent sequential nature of stack operations can limit opportunities for parallelism and optimization.
3. **Debugging Difficulty**: Debugging stack-based VMs can be more challenging as the stack state is transient and less intuitive than registers.

**Register-Based Virtual Machine**

**Pros**

1. **Efficiency**: Register-based VMs are often more efficient as operations can be performed directly on registers, reducing the number of memory accesses.
2. **Parallelism**: Using multiple registers allows for more parallel operations and optimizations by the compiler and the hardware.
3. **Ease of Optimization**: Register allocation and instruction scheduling optimizations are more straightforward and can be more effective.

**Cons**

1. **Complex Instruction Set**: Register-based VMs have a more complex instruction set since instructions must explicitly specify which registers to use.
2. **Larger Code Size**: Instructions often require more bits to encode the register addresses, leading to larger code sizes compared to stack-based VMs.
3. **Increased Compiler Complexity**: Compiling high-level languages to a register-based VM can be more complex due to the need for efficient register allocation and handling.

**Comparison**

**Instruction Complexity and Size:** Stack-based VMs have a simpler instruction set, making each instruction smaller since it doesn’t need to specify operands. This compactness can lead to smaller programs, which is advantageous in memory-constrained environments. In contrast, register-based VMs require more complex instructions that specify registers, which can increase the instruction size but allow for more direct and efficient execution.

**Performance:** Register-based VMs typically offer better performance due to reduced overhead in accessing operands. Operations on registers are faster than frequent stack manipulations, which involve push and pop operations that can slow down execution. Moreover, register-based VMs can leverage parallelism more effectively, allowing for simultaneous operations on different registers, which is less feasible with a stack-based approach.

**Compiler and Optimization:** The simplicity of stack-based VMs makes them easier to compile to, but they are harder to optimize at the machine level. Conversely, register-based VMs, while more complex, provide more opportunities for sophisticated compiler optimizations. Techniques such as instruction scheduling and register allocation are more advanced and effective in register-based architectures, potentially leading to more optimized and efficient code.

**Debugging and Maintenance:** Debugging stack-based VMs can be challenging due to the transient nature of stack operations, making it harder to track the state of the program. Register-based VMs, with their fixed and named registers, offer a more stable and comprehensible state, simplifying the debugging process.

**ANSWER 2**

Processors, or central processing units (CPUs), are critical components of computing systems, directly influencing the performance, efficiency, and overall capabilities of the system. Evaluating processor performance involves understanding various design metrics and utilizing benchmarking tools to provide quantitative assessments.

**Processor Design Metrics**

Processor performance is evaluated based on several key design metrics, each offering insights into different aspects of the processor's capabilities and efficiency.

**1. Clock Speed (Frequency)**

Clock speed, measured in gigahertz (GHz), indicates the number of cycles a processor executes per second. Higher clock speeds typically mean faster execution of instructions. However, clock speed alone does not determine overall performance, as it must be considered alongside other factors like instruction per cycle (IPC) and architecture efficiency.

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

IPC represents the number of instructions a processor can execute in a single clock cycle. A higher IPC indicates a more efficient processor, capable of performing more work per cycle. IPC is influenced by the processor's architecture, including factors like instruction pipelining, superscalar execution, and out-of-order execution.

**3. Core Count**

Modern processors often feature multiple cores, each capable of executing instructions independently. Multi-core processors can handle multiple threads simultaneously, enhancing performance for parallelizable tasks. Core count is particularly important for multi-threaded applications and workloads that benefit from parallelism.

**4. Cache Size and Hierarchy**

Processors use cache memory to store frequently accessed data and instructions, reducing the time needed to access main memory. Cache is organized into levels (L1, L2, L3), with L1 being the smallest and fastest. Larger cache sizes and efficient cache hierarchies can significantly improve performance by reducing memory access latency.

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

Power consumption is a critical metric, especially for mobile and embedded systems where battery life is a concern. TDP represents the maximum amount of heat a processor generates under typical workload conditions. Efficient processors provide a balance between performance and power consumption, ensuring sustainable thermal management.

**6. Instruction Set Architecture (ISA)**

ISA defines the set of instructions a processor can execute. Common ISAs include x86, ARM, and RISC-V. The choice of ISA impacts compatibility, performance, and power efficiency. Modern processors often support advanced extensions (e.g., SIMD, AVX) to accelerate specific workloads.

**Benchmarking Tools:** These are some benchmarking tools

**1. SPEC CPU Benchmark**

The Standard Performance Evaluation Corporation (SPEC) CPU benchmark is a widely used suite that evaluates the performance of a processor's integer and floating-point operations. SPEC benchmarks simulate real-world applications, providing a comprehensive assessment of computational capabilities. SPECint and SPECfp scores offer insights into integer and floating-point performance, respectively.

**2. Geekbench**

Geekbench is a cross-platform benchmarking tool that measures single-core and multi-core performance. It runs a series of tests mimicking real-world tasks, including image processing, machine learning, and cryptography. Geekbench scores allow for direct comparisons between different processors and systems.

**3. Cinebench**

Cinebench assesses a processor's performance in rendering tasks, specifically using the Cinema 4D engine. It provides scores based on how quickly the processor can render a 3D scene, highlighting its capabilities in handling graphics-intensive workloads. Cinebench offers single-core and multi-core results, useful for evaluating parallel processing efficiency.

**4. PassMark PerformanceTest**

PassMark PerformanceTest evaluates overall system performance, including CPU, memory, disk, and graphics. Its CPU tests include integer and floating-point operations, compression, encryption, and physics simulations. PassMark provides a comprehensive set of scores, facilitating comparisons across different components and systems.

**5. Linpack Benchmark**

The Linpack benchmark measures a system's floating-point computing power, specifically in solving linear equations. It is particularly relevant in high-performance computing (HPC) environments, where floating-point operations are critical. Linpack scores are used to rank supercomputers in the TOP500 list.

**6. AIDA64**

AIDA64 offers extensive benchmarking and diagnostic capabilities, including tests for CPU, memory, and disk performance. It provides detailed information about processor architecture, cache hierarchy, and thermal performance. AIDA64 is useful for both performance evaluation and system diagnostics.