**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**2024 Summer Midterm Exam**

**Student ID: 20099 Student Name: Md Maniruzzaman**

1. Explain what the differences are between computer architecture and computer organization?

**Answer:**

**Computer Architecture**

Definition:

Computer architecture refers to the conceptual design and fundamental operational structure of a computer system. It encompasses the attributes of a system that a programmer must understand to write efficient software. This includes the instruction set architecture (ISA), which is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O.

**Computer Organization**

Definition:

Computer organization deals with the realization of the architecture in hardware. It focuses on the implementation and operational units of the computer system, detailing how the various components are interconnected and interact to execute the architectural specifications.

**Differences Between Computer Architecture and Computer Organization**

Scope:

Computer Architecture:

* Defines the high-level design and functionality of the computer system.
* Focuses on the abstract model visible to the programmer, including the instruction set architecture (ISA), data types, addressing modes, and system control.

Computer Organization:

* Focuses on the operational aspects and implementation of the computer system.
* Deals with how the components of the computer are interconnected and function together to execute the architectural specifications.

Abstraction Level:

Computer Architecture:

* Conceptual and abstract.
* Concerned with the programmer's perspective on how instructions are executed and data is processed.
* Determines the software-hardware interface.

Computer Organization:

* Concrete and physical.
* Focuses on the engineer's perspective on how the hardware is designed and built to perform the required tasks.
* Includes details on circuitry, control signals, and timing.

Focus:

Computer Architecture:

* Involves designing the overall structure and capabilities of the computer.
* Includes defining the instruction set, registers, memory addressing modes, and I/O mechanisms.
* Determines the system's performance characteristics and compatibility with software.

Computer Organization:

* Involves implementing the architectural design using physical hardware components.
* Includes designing data paths, control units, memory hierarchies, and I/O interfaces.
* Focuses on optimizing performance, efficiency, and physical constraints like power consumption and heat dissipation.

Components:

Computer Architecture:

* Instruction Set Architecture (ISA): Specifies the set of instructions the CPU can execute.
* Data Types: Defines the types of data the system can process.
* Registers: Specifies the number and types of registers available for use.
* Addressing Modes: Determines how memory locations are addressed.
* Memory Architecture: Defines the structure and management of memory systems.
* Control Unit: Interprets instructions and generates control signals.
* I/O Mechanisms: Defines how the system interfaces with external devices.

Computer Organization:

* Microarchitecture: Details the internal implementation of the CPU components.
* Data Paths: Describes the routes data takes within the CPU.
* Control Paths: Manages the flow of control signals.
* Memory Hierarchy: Organizes memory into different levels (e.g., cache, RAM, secondary storage) for efficiency.
* Clock and Timing: Regulates the timing of operations within the system.
* I/O Organization: Implements interfaces and protocols for communication with external devices.

Impact:

Computer Architecture:

* Functionality: Determines the capabilities and performance characteristics of the computer.
* Compatibility: Affects software compatibility and optimization.
* Optimization: Guides software developers in writing efficient code.

Computer Organization:

* Efficiency: Directly impacts the computer’s speed and operational efficiency.
* Physical Design: Influences power consumption, heat dissipation, and size.
* Implementation: Ensures the architectural design is realized effectively in hardware.

Design vs. Implementation:

Architecture: Provides the blueprint and design specifications for what the system can do.

Organization: Realizes these specifications through practical hardware implementation.

Abstract vs. Concrete:

Architecture: Abstract and focuses on the design and capabilities of the system.

Organization: Concrete and focuses on the actual construction and performance.

Programmer's vs. Engineer's Perspective:

Architecture: Deals with what a programmer needs to understand to use the system effectively.

Organization: Deals with what an engineer needs to build the system efficiently.

Understanding both aspects is crucial for designing, optimizing, and effectively utilizing computer systems. Architecture defines the potential and limits of the system, while organization ensures these potentials are efficiently realized.

**=========================**

1. Give two examples of RISC and CISC processors respectively. What are the main characteristics of RISC processors?

**Answer:**

**RISC (Reduced Instruction Set Computing) Processors**

ARM (Advanced RISC Machine):

* Widely used in mobile devices, embedded systems, and increasingly in servers and desktops.
* Known for its energy efficiency and high performance in power-constrained environments.

MIPS (Microprocessor without Interlocked Pipeline Stages):

* Used in a variety of applications, including embedded systems, network routers, and gaming consoles.
* Known for its simple and efficient design, which facilitates high performance and low power consumption.

**CISC (Complex Instruction Set Computing) Processors**

Intel x86:

* Dominant in desktop, laptop, and server markets.
* Known for its backward compatibility and extensive instruction set, which allows complex operations to be performed with single instructions.

AMD x86 (Advanced Micro Devices x86):

* Competes directly with Intel in the desktop, laptop, and server markets.
* Shares many architectural similarities with Intel x86, including a rich instruction set designed for high performance and versatility.

**Main Characteristics of RISC Processors**

Simplified Instruction Set:

* RISC processors use a small, highly optimized set of instructions. Each instruction is designed to execute in a single clock cycle, making the CPU's design simpler and faster.

Load/Store Architecture:

* RISC architectures use a load/store model where operations are performed only on CPU registers. Memory is accessed only via explicit load and store instructions, separating computation and memory access.

Fixed-Length Instructions:

* Instructions in RISC architectures are typically of a fixed length, simplifying the instruction decoding process and enabling more efficient pipelining.

Single-Cycle Execution:

* Most instructions are designed to be executed in a single clock cycle, reducing the instruction latency and improving performance.

Pipelining:

* RISC processors are highly pipelined, meaning multiple instructions can be processed simultaneously at different stages of execution, which increases throughput and overall performance.

Few Addressing Modes:

* RISC architectures typically support a limited number of addressing modes, simplifying the design of the instruction set and the CPU hardware.

Large Number of Registers:

* To minimize the need for slow memory accesses, RISC processors include a large number of general-purpose registers that allow for efficient instruction execution and reduce the overhead of memory access.

Emphasis on Software:

* RISC places more complexity in the software (compilers) rather than the hardware. Compilers are designed to optimize code to use the simple, fast instructions efficiently.

Understanding these differences helps in appreciating the trade-offs between RISC and CISC architectures, such as simplicity and power efficiency versus rich instruction sets and hardware optimization.

=======================

1. What are the best benchmarks to use, and why?

**Answer:**

Choosing the best benchmarks for evaluating computer performance depends on the specific aspects of performance you're interested in measuring, such as general-purpose computing, graphics processing, power efficiency, or specialized workloads. Here are some widely used and respected benchmarks across various domains:

**General-Purpose Computing Benchmarks**

SPEC (Standard Performance Evaluation Corporation) Benchmarks:

* SPEC CPU: Measures the performance of the processor, memory, and compiler. It includes integer and floating-point tests that are widely used to evaluate CPU performance.
* SPECint and SPECfp: Focus on integer and floating-point computations, respectively.
* Use of SPEC: SPEC benchmarks are industry-standard, providing reliable and reproducible results that are widely accepted for comparing the performance of different systems.

Geekbench:

* Measures both single-core and multi-core performance across different workloads, including integer, floating-point, and memory operations.
* Use of Geekbench: It's easy to use, covers a wide range of tests, and provides a comprehensive performance score, making it useful for quick comparisons.

PassMark:

* Provides a suite of tests for CPU, memory, disk, and graphics performance.
* Use of PassMark: It offers a broad range of tests that can help evaluate overall system performance, making it useful for general-purpose benchmarking.

**Graphics and Gaming Benchmarks**

3DMark:

* Measures the graphics performance of systems using DirectX and Vulkan APIs. It includes several tests like Fire Strike, Time Spy, and Sky Diver.
* Use of 3DMark: It's one of the most comprehensive and widely used benchmarks for gaming and graphics performance, providing detailed insights into GPU capabilities.

Unigine Benchmarks (Heaven, Valley, Superposition):

* Focuses on real-time 3D rendering performance.
* Use of Unigine: It provides visually demanding tests that push GPUs to their limits, making it excellent for assessing graphics performance and stability under load.

**Power Efficiency Benchmarks**

SPEC Power:

* Measures the power consumption and performance of server-class systems.
* Use of SPEC Power: It's specifically designed to assess the power efficiency of systems, providing insights into energy consumption under different workloads.

MobileMark:

* Evaluates battery life and performance of mobile devices under realistic usage scenarios.
* Use of MobileMark: It provides a balanced view of performance and power efficiency for laptops and other portable devices.

**Specialized Benchmarks**

TPC (Transaction Processing Performance Council) Benchmarks:

* TPC-C: Measures the performance of online transaction processing (OLTP) systems.
* TPC-H: Evaluates decision support systems with a focus on complex queries.
* Use of TPC: These benchmarks are tailored for database systems, providing detailed insights into transaction processing and query performance.

SPEC SFS and SPECmail:

* SPEC SFS: Evaluates file server performance.
* SPECmail: Measures email server performance.
* Use of SPEC SFS and SPECmail: These benchmarks are specialized for assessing the performance of file and email servers, making them ideal for enterprise environments.

**Synthetic Benchmarks**

Linpack:

* Measures floating-point computing power, particularly useful for high-performance computing (HPC) systems.
* Use of Linpack: It's a standard benchmark for evaluating supercomputers, providing a measure of their ability to solve linear equations.

Cinebench:

* Evaluates CPU and GPU performance using real-world rendering tasks.
* Use of Cinebench: It's based on the Cinema 4D engine, making it highly relevant for users in graphics and content creation industries.

The best benchmarks to use depend on your specific needs and the type of performance you want to measure. For general-purpose computing, SPEC CPU and Geekbench are highly recommended. For graphics and gaming, 3DMark and Unigine benchmarks provide in-depth insights. For power efficiency, SPEC Power and MobileMark are excellent choices. Specialized benchmarks like TPC for databases and SPEC SFS for file servers cater to specific workloads.

Using a combination of these benchmarks can provide a comprehensive view of a system's performance across different dimensions, helping you make informed decisions based on accurate and reliable data.

==============================

1. Assuming that *program 1* is running on *computer 1* with the following parameters:
   1. Instruction Count = 23400
   2. Average CPI = 2.1
   3. Clock Rate = 3.0 GHz

find the execution time of *program 1* on *computer 1*? If *computer 2* runs *program 2* three times faster than *program* 1 with the same clock rate and Average CPI, then what variable must be changed to meet this speedup requirement and what is the value of this variable?

**Answer:**

To find the execution time of program 1 on computer 1, we know the formula is:

Execution Time = (Instruction Count × CPI) / Clock Rate

Given the parameters,

Instruction Count (IC) = 23,400 instructions

Average CPI (Cycles Per Instruction) = 2.1

Clock Rate = 3.0 GHz

Now,

Execution Time = (23400×2.1) / (3.0×109) = 1.638 × 10-5 seconds = 16.38 microseconds

Convert the clock rate to Hz = 3.0 GHz = 3.0×109 Hz

So, the execution time of program 1 on computer 1 is 16.38 microseconds.

Now, Computer 2 runs program 2 three times faster than program 1 with the same clock rate and average CPI. This means the execution time of program 2 on computer 2 is:

Execution Time of Program 2 on Computer 2 = (Execution Time of Program 1 on Computer 1/ 3)

= (16.38 / 3) microseconds

= 5.46 microseconds

Given that, the clock rate and average CPI are the same for both computers, the only variable that can be changed to achieve this speedup is the instruction count.

Now,

Execution Time = (Instruction Count × CPI) / Clock Rate

Instruction Count = (Execution Time × Clock Rate) / CPI

For program 2:

Instruction Count Program 2 = (5.46 × 10−6 seconds × 3.0 ×109 Hz) / 2.1 ≈ 7800

So, the instruction counts for program 2 must be approximately 7,800 instructions to meet the speedup requirement.

According to question and calculation,

The execution time of program 1 on computer 1 is 16.38 microseconds.

To run program 2 three times faster with the same clock rate and average CPI, the instruction count must be reduced to approximately 7,800 instructions.

==================================

1. Explain the difference between Cache Memory and the Register File.

**Answer:**

**Difference Between Cache Memory and the Register File**

Definition and Purpose

Cache Memory:

* Cache memory is a small, high-speed storage area located between the CPU and the main memory (RAM). Its primary purpose is to reduce the time it takes for the CPU to access frequently used data and instructions, thus improving overall system performance.

Register File:

* The register file is a small, fast storage area within the CPU itself. It consists of a set of registers that hold data and instructions temporarily during the execution of programs. Registers are used to store immediate values and results of arithmetic and logical operations.

Location and Speed

Cache Memory:

* Cache memory is typically located closer to the CPU than the main memory but further away than the register file. Modern processors often have multiple levels of cache (L1, L2, and sometimes L3), with L1 being the closest and fastest, followed by L2 and L3 caches.
* Cache memory is slower than registers but significantly faster than the main memory.

Register File:

* Registers are located inside the CPU itself, making them the fastest type of memory in a computer system.
* The access time for registers is minimal, often on the order of one CPU cycle.

Size and Capacity

Cache Memory:

* Cache memory is relatively small in capacity compared to main memory, typically ranging from a few kilobytes (KB) to several megabytes (MB).
* Despite its small size, cache memory plays a crucial role in enhancing performance by storing the most frequently accessed data.

Register File:

* The register file is even smaller in size than cache memory, usually consisting of a few dozen to a few hundred registers.
* Registers are designed to hold a small amount of data (usually 32 or 64 bits per register), sufficient for immediate processing needs.

Data Storage and Access

Cache Memory:

* Cache memory stores a copy of frequently accessed data from the main memory. It operates on the principle of locality of reference, where recently accessed data is likely to be accessed again soon.
* Data in the cache is managed using cache algorithms like Least Recently Used (LRU) or First-In-First-Out (FIFO).

Register File:

* Registers hold data that the CPU is currently processing or will process shortly. This includes operands for instructions, intermediate results, and pointers for addressing.
* Access to registers is direct and occurs through the CPU's instruction set, without the need for complex management algorithms.

Function and Usage

Cache Memory:

* Cache memory acts as an intermediary between the CPU and main memory, reducing latency and improving data access times. It helps mitigate the performance gap between the CPU and slower main memory.
* It is used to store copies of data that are expected to be reused soon, such as instruction loops and frequently accessed variables.

Register File:

* Registers are used for immediate data storage and manipulation during instruction execution. They provide operands for arithmetic, logic, and control operations, and store the results of these operations.
* Registers are essential for the execution of instructions, as they facilitate fast data retrieval and storage directly within the CPU.

=====================================

1. Explain the difference between the Instruction Register and the Program Counter

**Answer:**

**Difference Between the Instruction Register (IR) and the Program Counter (PC)**

A diagram of a program code

Description automatically generated

Fig: Basic diagram for relation between Instruction register and program counter.

Definitions and Purposes

Instruction Register (IR):

* The Instruction Register is a special register within the CPU that holds the current instruction being executed. Its primary purpose is to store the instruction fetched from memory so that it can be decoded and executed by the CPU.

Program Counter (PC):

* The Program Counter is a register that holds the address of the next instruction to be fetched from memory. Its primary purpose is to keep track of the CPU's position in the instruction sequence of a program.

Functionality

Instruction Register (IR):

* Holds Current Instruction: The IR temporarily stores the instruction that has been fetched from memory and is currently being executed. This includes all parts of the instruction, such as the opcode and the operands.
* Instruction Execution: Once the instruction is loaded into the IR, the CPU's control unit decodes and executes it. The IR ensures that the CPU can access the instruction's details during execution.

Program Counter (PC):

* Holds Address of Next Instruction: The PC contains the memory address of the next instruction to be fetched. After the current instruction is executed, the PC is updated to point to the next instruction in the sequence.
* Instruction Sequencing: The PC ensures the correct sequence of instruction execution. It is incremented after each instruction fetch, although it can be modified by branch, jump, and call instructions to implement control flow changes.

Updates and Modifications

Instruction Register (IR):

* Loaded During Fetch: The IR is updated during the instruction fetch phase of the CPU cycle. It is loaded with the contents of the memory location pointed to by the PC.
* Remains Until Execution: The content of the IR remains constant while the instruction is being decoded and executed. It is only updated with the next instruction during the next fetch phase.

Program Counter (PC):

* Incremented or Modified: The PC is typically incremented by one (or by the size of the instruction) after each instruction fetch to point to the next instruction. However, it can be modified by control flow instructions (like jumps, branches, and subroutine calls) to point to a non-sequential memory address.
* Directs Fetch Cycle: The PC directs the instruction fetch cycle by providing the address of the next instruction to be read from memory.

Role in CPU Operation Cycle

Instruction Register (IR):

* Fetch-Decode-Execute Cycle: The IR is central to the decode and execute phases. After the fetch phase (guided by the PC), the instruction is placed in the IR, decoded to understand the operation, and then executed by the CPU.
* Immediate Operation: The IR's role is immediate and temporary for the current instruction only.

Program Counter (PC):

* Fetch Cycle Initiation: The PC is crucial in the fetch phase of the CPU cycle. It provides the address from which the next instruction should be fetched.
* Continuous Operation: The PC operates continuously throughout the program execution, ensuring the sequential or controlled progression of instruction execution.

===============================

1. Convert the following pseudo statements to MIPS assembly language:
   1. *t3 = t4 + t5 – t6*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt1: .asciiz "Enter value of t4: "

prompt2: .asciiz "Enter value of t5: "

prompt3: .asciiz "Enter value of t6: "

result: .asciiz "The result of, t3 = t4 + t5 - t6 is: "

.text

.globl main

main:

li $v0, 4

la $a0, prompt1

syscall # print prompt1

li $v0, 5

syscall

move $t4, $v0 # store input in t4

li $v0, 4

la $a0, prompt2

syscall # print prompt2

li $v0, 5

syscall

move $t5, $v0 # store input in t5

li $v0, 4

la $a0, prompt3

syscall # print prompt3

li $v0, 5

syscall

move $t6, $v0 # store input in t6

# Compute calculation t3 = t4 + t5 - t6

add $t3, $t4, $t5 # t3 = t4 + t5

sub $t3, $t3, $t6 # t3 = t3 - t6

li $v0, 4

la $a0, result

syscall # print result

move $a0, $t3

li $v0, 1

syscall # print integer

li $v0, 10

syscall

======================

* 1. *s3 = t2 / (s1 – 54321);*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt1: .asciiz "Enter value of t2: "

prompt2: .asciiz "Enter value of s1: "

result: .asciiz "The result of, s3 = t2 / (s1 - 54321) is: "

div\_zero: .asciiz "Division by zero error!\n"

.text

.globl main

main:

li $v0, 4

la $a0, prompt1

syscall # print prompt1

li $v0, 5

syscall

move $t2, $v0 # store input in t2

li $v0, 4

la $a0, prompt2

syscall # print prompt2

li $v0, 5

syscall

move $s1, $v0 # store input in s1

li $t0, 54321 # load 54321 into $t0

# Calculation s1 - 54321

sub $t1, $s1, $t0

# Check

beq $t1, $zero, div\_by\_zero # if (s1 - 54321 == 0), handle division by zero

# Calculate t2 / (s1 - 54321)

div $t2, $t1

mflo $s3

li $v0, 4

la $a0, result

syscall # print result

move $a0, $s3

li $v0, 1

syscall # print integer

li $v0, 10

syscall

div\_by\_zero:

li $v0, 4

la $a0, div\_zero

syscall

li $v0, 10

syscall

============================

* 1. *cout << t3; //print t3’s value in C++*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt: .asciiz "Enter value for t3: "

error: .asciiz "Error: Invalid input (negative value)! Please enter a non-negative value.\n"

result: .asciiz "The value of t3 is: "

.text

.globl main

main:

input\_loop:

li $v0, 4

la $a0, prompt

syscall # print prompt

li $v0, 5

syscall

move $t3, $v0

# Check

bltz $t3, input\_error # if t3 < 0, jump to input\_error

li $v0, 4

la $a0, result

syscall

move $a0, $t3

li $v0, 1

syscall # print integer

li $v0, 10

syscall

input\_error:

li $v0, 4

la $a0, error

syscall # print error message

# Jump back to input\_loop to prompt user again for type input

j input\_loop

================================

* 1. *t8 = Mem(a0);*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt: .asciiz "Enter a value of a0: "

result\_msg: .asciiz "Memory value loaded into t8: "

memory\_value: .word 0

.text

.globl main

main:

li $v0, 4

la $a0, prompt

syscall

li $v0, 5

syscall

move $t0, $v0

sw $t0, memory\_value

lw $t8, memory\_value

li $v0, 4

la $a0, result\_msg

syscall

move $a0, $t8

li $v0, 1

syscall

li $v0, 10

syscall

==================================

* 1. *Mem(a0+ 16) = 32768;*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

*.data*

*result\_msg: .asciiz "Value 32768 stored at memory address: "*

*.text*

*.globl main*

*main:*

*addi $t0, $a0, 16*

*andi $t0, $t0, -4*

*li $t1, 32768*

*sw $t1, 0($t0)*

*li $v0, 4*

*la $a0, result\_msg*

*syscall*

*li $v0, 1*

*move $a0, $t0*

*syscall*

*li $v0, 10*

*syscall*

===========================================

* 1. *cout << "Hello World"; //print “Hello World” in C++*

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt: .asciiz "Please type string: "

result: .asciiz "You typed: "

input: .space 256 # allocate space for the input string

.text

.globl main

main:

li $v0, 4

la $a0, prompt

syscall # print prompt

li $v0, 8

la $a0, input

li $a1, 256

syscall

move $t0, $a0

li $v0, 4

la $a0, result

syscall

move $a0, $t0

li $v0, 4

syscall # print the string

li $v0, 10

syscall

================================

1. Write the program to execute the following statement efficiently in MIPS assembly language:

$t0 = $s0 / 8 - 2 \* $s1 + $s2;

**Answer:**

***Source code (.asm) file is attached to Github link and Canvas also.***

***Paste the code below. If here any error, please consider it. Please run the source code (.asm) file.***

.data

prompt\_s0: .asciiz "Enter the value of $s0: "

prompt\_s1: .asciiz "Enter the value of $s1: "

prompt\_s2: .asciiz "Enter the value of $s2: "

result: .asciiz "Result is: "

.text

.globl main

main:

li $t3, 1

beq $t3, $zero, skip\_input\_s0

li $v0, 4

la $a0, prompt\_s0

syscall # print prompt

li $v0, 5

syscall # read integer

move $s0, $v0

skip\_input\_s0:

li $t3, 1

beq $t3, $zero, skip\_input\_s1

li $v0, 4

la $a0, prompt\_s1

syscall # print prompt

li $v0, 5

syscall # read integer

move $s1, $v0

skip\_input\_s1:

li $t3, 1

beq $t3, $zero, skip\_input\_s2

li $v0, 4

la $a0, prompt\_s2

syscall # print prompt

li $v0, 5

syscall # read integer

move $s2, $v0

skip\_input\_s2:

sra $t0, $s0, 3

# Calculatation $t0 = $t0 - 2 \* $s1

sll $t1, $s1, 1

sub $t0, $t0, $t1

# Calculatation $t0 = $t0 + $s2

add $t0, $t0, $s2

li $v0, 4

la $a0, result

syscall # print result string

move $a0, $t0

li $v0, 1

syscall # print integer

li $v0, 10

syscall