**Logo

Description automatically generated San Francisco Bay University**

**EE488 - Computer Architecture**

**2024 Summer Midterm Exam**

**Student ID:19949 Student Name: Chufeng Jiang**

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

|  |  |
| --- | --- |
| **Computer Architecture**  Deals with the design, structure, and functionality of a computer system.  Involves the specification of instruction set, hardware components, and data pathways.  Defines the logical design and behavior of a computer system as seen by a programmer.  Concerned with abstract concepts and high-level design choices.  Examples include instruction set architecture (ISA), microarchitecture design.  Determines what the system does.  Focus on functionality, performance, and capabilities. | **Computer Organization**  Focuses on the operational aspects and implementation of the architecture.  Involves the implementation details such as control signals, interfaces, memory technology.  Defines the physical implementation and functionality of components as seen by a designer.  Concerned with concrete implementation details and performance optimization.  Examples include memory hierarchy, data path design, control unit design.  Determines how the system performs its tasks.  Focus on efficiency, speed, and resource utilization. |

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

RISC (Reduced Instruction Set Computing) Processors：

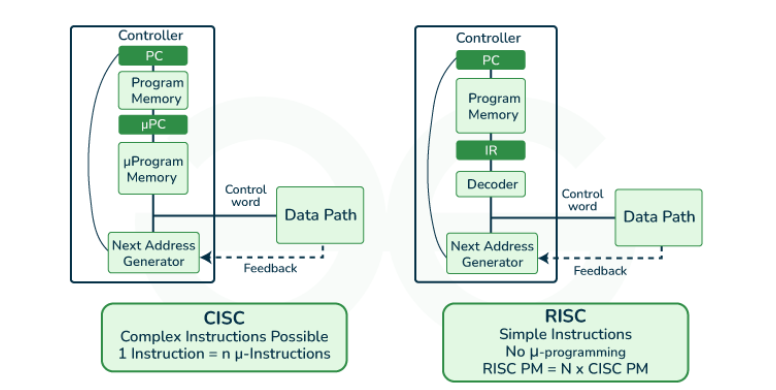
ARM Cortex-A Series

MIPS R-Series

CISC (Complex Instruction Set Computing) Processors

Intel x86 Series

AMD Ryzen Serie



RISC Characteristics:

* **Simplicity of Instructions**: The ARM Cortex-A series uses a reduced set of instructions, leading to fewer transistors and lower power consumption compared to CISC processors.
* **Uniform Instruction Length**: In MIPS R-Series, all instructions are 32 bits long, which streamlines the process of fetching and decoding instructions.
* **Load/Store Architecture**: Both ARM and MIPS processors separate memory access from arithmetic operations using distinct load and store instructions for memory operations and performing all other operations on data in registers.
* **Large Number of General-Purpose Registers**: ARM Cortex-A processors have a large register file, reducing the need to frequently access memory, which improves performance.
* **Focus on Software Optimization**: RISC architectures are designed with the expectation that the compiler will optimize instruction sequences to leverage the simple instruction set effectively.
* **Pipelining**: ARM and MIPS processors are designed to efficiently execute instructions in a pipeline, where different stages of multiple instructions overlap, improving throughput and overall performance.

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

I think the best bencjmarks are Geekbench and SPEC CPU. Because These tools are highly regarded for their comprehensive performance metrics, standardized testing procedures, and real-world relevance, making them essential for consumers, developers, and industry professionals seeking reliable and comparable performance data.

Geekbench is celebrated for its versatility and ease of use, making it accessible to a wide range of users. It provides detailed performance evaluations by measuring both single-core and multi-core capabilities across a variety of real-world tasks. These include integer and floating-point operations, as well as memory performance, ensuring a well-rounded assessment of a processor's strengths and weaknesses. Additionally, Geekbench's cross-platform availability allows for consistent and direct comparisons between different devices and operating systems, enhancing its utility for those looking to evaluate and compare diverse hardware configurations. Its user-friendly interface and regular updates further cement its position as a top choice for comprehensive and up-to-date performance benchmarking.

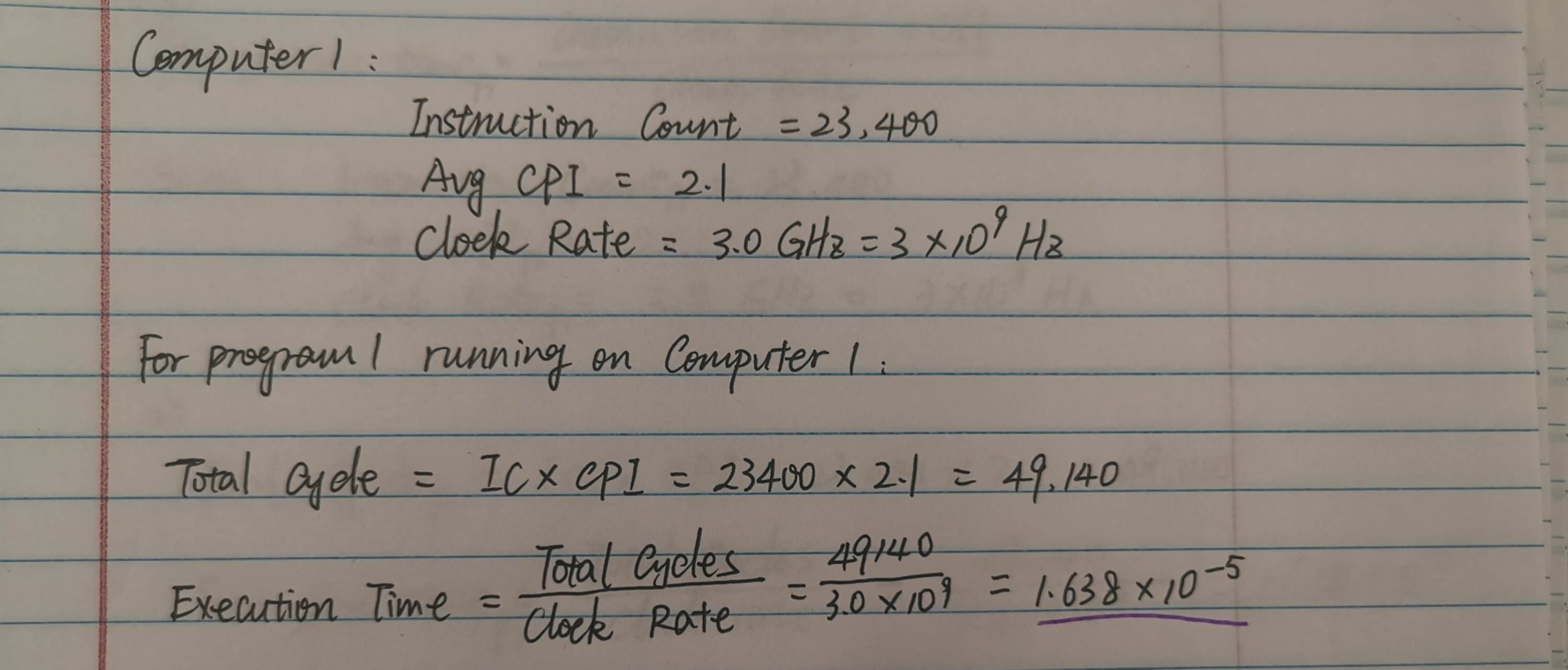
SPEC CPU, developed by the Standard Performance Evaluation Corporation, is renowned for its rigorous and detailed benchmarking methodology. It focuses on compute-intensive performance by using a suite of tests that simulate a wide range of real-world applications, from scientific computing to engineering and business workloads. SPEC CPU benchmarks are highly respected for their accuracy and standardization, providing a reliable measure of a system's capabilities under sustained heavy loads. This makes SPEC CPU particularly valuable for industry professionals and researchers who need precise and in-depth performance data. Its ability to offer a granular view of both integer and floating-point performance across various operating systems makes SPEC CPU an indispensable tool for in-depth performance analysis and comparison.

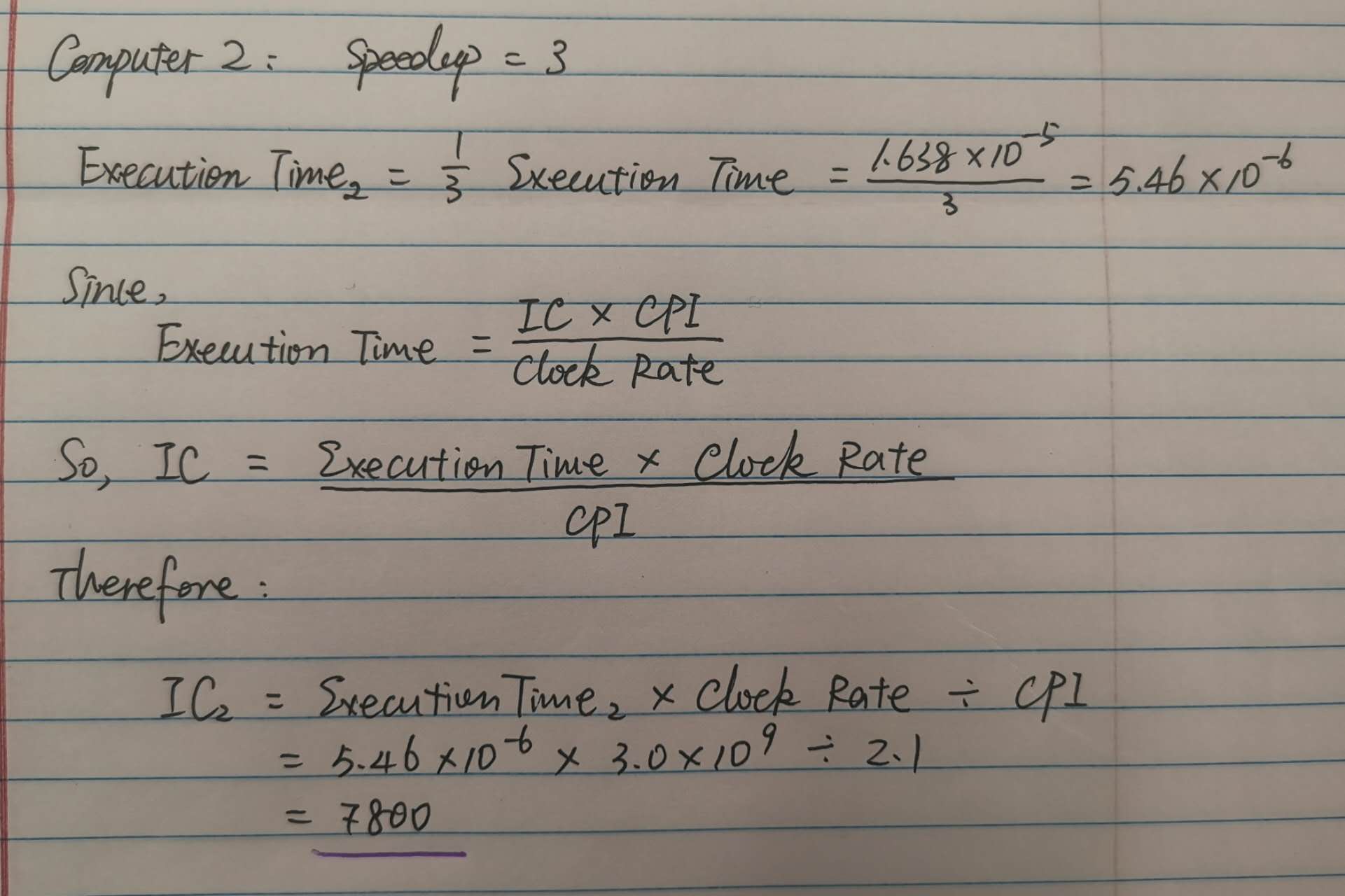
1. **Assuming that *program 1* is running on *computer 1* with the following parameters:** 
   1. **Instruction Count = 23400**
   2. **Average CPI = 2.1**
   3. **Clock Rate = 3.0 GHz**

**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?**

The execution time of *program 1* on *computer 1 is 1.638\*10-5* Seconds.

The variable must be changed is the instruction count, and it should be reduced to 7800.





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

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Cache Memory** | **Register File** |
| Definition | Small, fast memory between CPU and main memory | Set of registers within the CPU |
| Location | On-chip or close to the CPU | Inside the CPU |
| Size | Larger (KB to MB) | Smaller (bytes) |
| Access Speed | Very fast, but slower than registers | Fastest memory in the computer |
| Purpose | Stores frequently accessed data and instructions | Holds data for immediate processing by the CPU |
| Hierarchy | Between main memory and CPU | Top of memory hierarchy |
| Accessibility | Managed by hardware | Directly accessible by CPU instructions |
| Volatility | Volatile | Volatile |
| Content Preservation | Preserved when switching between programs | Usually not preserved between context switches |
| Addressability | Addressable | Addressed by register names in instructions |
| Management | Managed automatically by cache controller | Managed by compiler and programmer |
| Latency | Low latency, but higher than registers | Lowest latency |
| Cost | More expensive than main memory, less than regs | Most expensive per bit |
| Power Consumption | Lower than main memory, higher than registers | Very low |
| Typical Access Time | Few clock cycles | Single clock cycle |

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

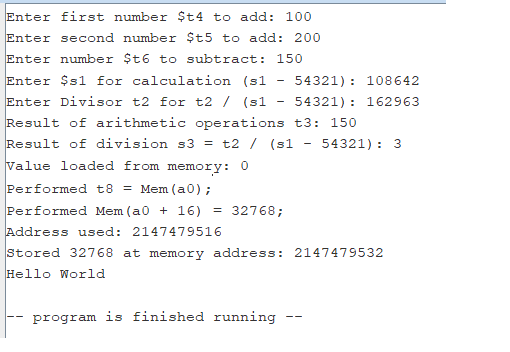
|  |  |  |
| --- | --- | --- |
| **Aspect** | **Instruction Register (IR)** | **Program Counter (PC)** |
| Definition | Holds the current instruction being executed | Holds the address of the next instruction to fetch |
| Purpose | To store and decode the current instruction | To keep track of the program's execution sequence |
| Content | Machine code of the instruction | Memory address |
| When Updated | At the fetch stage of each instruction cycle | After each instruction fetch, or during jumps/calls |
| Role in Execution | Provides instruction to control unit for decoding | Determines which instruction to fetch next |
| Accessibility | Read by the control unit | Can be read and modified by the CPU |
| Size | Typically matches the instruction length | Matches the address bus width of the system |
| Relation to Memory | Receives data from memory or cache | Points to locations in memory or cache |
| In Instruction Cycle | Used in the decode stage | Used in the fetch stage |
| Modification | Changed for every new instruction | Typically incremented, but can be changed for jumps |
| Type of Register | Data register | Address register |
| Visibility to Software | Generally not directly accessible | Can often be accessed/modified by software |
| Role in Branching | Indirectly involved (contains branch instruction) | Directly modified during branch operations |
| In Pipelining | May be duplicated for different pipeline stages | May use branch prediction to speculatively update |

1. **Convert the following pseudo statements to MIPS assembly language:**
   1. ***t3 = t4 + t5 – t6***
   2. ***s3 = t2 / (s1 – 54321);***
   3. ***cout << t3; //print t3’s value in C++***
   4. ***t8 = Mem(a0);***
   5. ***Mem(a0+ 16) = 32768;***
   6. ***cout << "Hello World";*** *//print “Hello World” in C++*

|  |  |
| --- | --- |
| *t3 = t4 + t5 – t6* | # Arithmetic operations  add $t3, $t4, $t5 # $t3 = $t4 + $t5  sub $t3, $t3, $t6 # $t3 = $t3 - $t6 |
| *s3 = t2 / (s1 – 54321);* | # Calculate s3 = t2 / (s1 - 54321)  li $t0, 54321 # $t0 = 54321  sub $t1, $s1, $t0 # $t1 = $s1 - 54321  div $t2, $t1 # $t2 / $t1 (Note the order change)  mflo $s3 # $s3 = quotient of division |
| *cout << t3;* | # Print result of arithmetic operations (t3)  li $v0, 1  move $a0, $t3  syscall |
| *t8 = Mem(a0);* | # Load word from memory (t8 = Mem(a0))  lw $t8, 0($t7) # Load word from memory at address in $t7 to $t8 |
| *Mem(a0+ 16) = 32768;* | # Store 32768 at memory address + 16 (Mem(a0 + 16) = 32768)  li $t0, 32768 # Load immediate value 32768 into $t0  sw $t0, 16($t7) # Store $t0 at memory address $t7 + 16 |
| *cout << "Hello World";* | # Print "Hello World"  li $v0, 4  la $a0, hello  syscall |

**The whole program designed is as follow:**

|  |
| --- |
| .data  prompt1: .asciiz "Enter first number $t4 to add: "  prompt2: .asciiz "Enter second number $t5 to add: "  prompt3: .asciiz "Enter number $t6 to subtract: "  prompt4: .asciiz "Enter $s1 for calculation (s1 - 54321): "  prompt5: .asciiz "Enter Divisor t2 for t2 / (s1 - 54321): "  result1: .asciiz "Result of arithmetic operations t3: "  result2: .asciiz "Result of division s3 = t2 / (s1 - 54321): "  result3: .asciiz "Value loaded from memory: "  result4: .asciiz "Stored 32768 at memory address: "  op\_info1: .asciiz "Performed t8 = Mem(a0);"  op\_info2: .asciiz "Performed Mem(a0 + 16) = 32768;"  addr\_info: .asciiz "Address used: "  hello: .asciiz "Hello World"  newline: .asciiz "\n"  .text  .globl main  main:  # [Original code remains the same up to the "Print Hello World" section]  # Get input values  li $v0, 4  la $a0, prompt1  syscall  li $v0, 5  syscall  move $t4, $v0  li $v0, 4  la $a0, prompt2  syscall  li $v0, 5  syscall  move $t5, $v0  li $v0, 4  la $a0, prompt3  syscall  li $v0, 5  syscall  move $t6, $v0  li $v0, 4  la $a0, prompt4  syscall  li $v0, 5  syscall  move $s1, $v0  li $v0, 4  la $a0, prompt5  syscall  li $v0, 5  syscall  move $t2, $v0  # Arithmetic operations  add $t3, $t4, $t5 # $t3 = $t4 + $t5  sub $t3, $t3, $t6 # $t3 = $t3 - $t6  # Print result of arithmetic operations  li $v0, 4  la $a0, result1  syscall  li $v0, 1  move $a0, $t3  syscall  li $v0, 4  la $a0, newline  syscall  # Calculate s3 = t2 / (s1 - 54321)  li $t0, 54321 # $t0 = 54321  sub $t1, $s1, $t0 # $t1 = $s1 - 54321  div $t2, $t1 # $t2 / $t1 (Note the order change)  mflo $s3 # $s3 = quotient of division  # Print result of division  li $v0, 4  la $a0, result2  syscall  li $v0, 1  move $a0, $s3  syscall  li $v0, 4  la $a0, newline  syscall  # New memory operations with a random stack address  addi $sp, $sp, -32 # Allocate space on the stack  la $t7, 0($sp) # Load address of allocated stack space into $t7  lw $t8, 0($t7) # Load word from memory at address in $t7 to $t8  # Print loaded value  li $v0, 4  la $a0, result3  syscall  li $v0, 1  move $a0, $t8  syscall  li $v0, 4  la $a0, newline  syscall  # Print operation information for t8 = Mem(a0)  li $v0, 4  la $a0, op\_info1  syscall  li $v0, 4  la $a0, newline  syscall  # Store 32768 at memory address + 16  li $t0, 32768 # Load immediate value 32768 into $t0  sw $t0, 16($t7) # Store $t0 at memory address $t7 + 16  # Print operation information for Mem(a0 + 16) = 32768  li $v0, 4  la $a0, op\_info2  syscall  li $v0, 4  la $a0, newline  syscall  # Print the address used for memory operations  li $v0, 4  la $a0, addr\_info  syscall  li $v0, 1  move $a0, $t7  syscall  li $v0, 4  la $a0, newline  syscall  # Print confirmation of store operation  li $v0, 4  la $a0, result4  syscall  li $v0, 1  move $a0, $t7  addi $a0, $a0, 16 # Add 16 to the address for display  syscall  li $v0, 4  la $a0, newline  syscall  # Print "Hello World"  li $v0, 4  la $a0, hello  syscall  li $v0, 4  la $a0, newline  syscall  # Exit program  li $v0, 10  syscall |



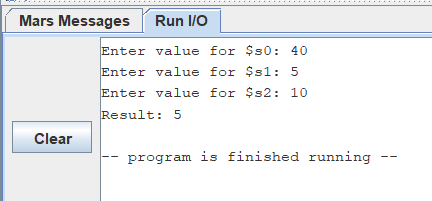
Test case user input:

|  |
| --- |
| Enter first number $t4 to add: 100  Enter second number $t5 to add: 200  Enter number $t6 to subtract: 150  Enter $s1 for calculation (s1 - 54321): 108642  Enter Divisor t2 for t2 / (s1 - 54321): 162963  Result of arithmetic operations t3: 150  Result of division s3 = t2 / (s1 - 54321): 3  Value loaded from memory: 0  Performed t8 = Mem(a0);  Performed Mem(a0 + 16) = 32768;  Address used: 2147479516  Stored 32768 at memory address: 2147479532  Hello World  -- program is finished running -- |

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

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

|  |
| --- |
| .data  prompt\_s0: .asciiz "Enter value for $s0: "  prompt\_s1: .asciiz "Enter value for $s1: "  prompt\_s2: .asciiz "Enter value for $s2: "  result\_prompt: .asciiz "Result: "  newline: .asciiz "\n"  .text  .globl main  main:  # Prompt user to enter $s0  li $v0, 4 # syscall code for print string  la $a0, prompt\_s0 # load address of prompt\_s0  syscall # print prompt\_s0  # Read integer input for $s0  li $v0, 5 # syscall code for read integer  syscall # read integer input  move $s0, $v0 # move input value to $s0  # Prompt user to enter $s1  li $v0, 4 # syscall code for print string  la $a0, prompt\_s1 # load address of prompt\_s1  syscall # print prompt\_s1  # Read integer input for $s1  li $v0, 5 # syscall code for read integer  syscall # read integer input  move $s1, $v0 # move input value to $s1  # Prompt user to enter $s2  li $v0, 4 # syscall code for print string  la $a0, prompt\_s2 # load address of prompt\_s2  syscall # print prompt\_s2  # Read integer input for $s2  li $v0, 5 # syscall code for read integer  syscall # read integer input  move $s2, $v0 # move input value to $s2  # Calculate $t0 = $s0 / 8 - 2 \* $s1 + $s2  srl $t1, $s0, 3 # $t1 = $s0 >> 3 (divide $s0 by 8)  sll $t2, $s1, 1 # $t2 = $s1 << 1 (multiply $s1 by 2)  sub $t3, $t1, $t2 # $t3 = $t1 - $t2  add $t0, $t3, $s2 # $t0 = $t3 + $s2  # Print the result  li $v0, 4 # syscall code for print string  la $a0, result\_prompt # load address of result\_prompt  syscall # print result\_prompt  # Print $t0 (the result)  li $v0, 1 # syscall code for print integer  move $a0, $t0 # load $t0 into $a0  syscall # print $t0  # Print newline  li $v0, 4 # syscall code for print string  la $a0, newline # load address of newline  syscall # print newline  # Exit program  li $v0, 10 # syscall code for exit  syscall # exit program |



|  |
| --- |
| Enter value for $s0: 40  Enter value for $s1: 5  Enter value for $s2: 10  Result: 5  -- program is finished running -- |