COMPUTER ORGANIZATION AND ARCHITECTURE LAB

Fall 2024, 5th Semester

Lab Report

Submitted by: **Hassan Zaib Jadoon**

Registration Number**: 22PWCSE2144**

Section: **A**

"On my honor, as a student at the University of Engineering and Technology Peshawar, I have neither given nor received unauthorized assistance on this academic work."

Signature: Hassan

Submitted To: Dr. Amad Khalil

Department of Computer Systems Engineering

University of Engineering and Technology Peshawar

Objective:

To understand the basic concepts of computer organization and architecture, the differences between CISC and RISC architectures, and the use of MIPS architecture with QtSPIM simulator.

1. Introduction to Computer Organization and Architecture

Computer Organization:
Refers to the physical and logical arrangement of a computer's hardware. It deals with the actual operational units, the interconnections between hardware components, and the implementation of architecture specifications.

Examples include:

- Control signals
- RAM size
- Input/Output (I/O) devices
- Timer
- Interrupts

Computer Architecture:
Refers to the behavior and structure of a computer as seen by a programmer. It focuses on the design of the system's functionality, including how the CPU and memory work together. From the programmer's perspective, computer architecture defines what the computer does.

Examples include:

- Instruction set design
- Memory addressing
- Instruction execution
- CPU functionality

In simple terms:

- **Computer architecture** defines *what* a computer does.
- **Computer organization** explains *how* it does it.

---

2. Types of Computer Architectures

CISC (Complex Instruction Set Computing): CISC provides a large set of instructions, allowing the processor to perform complex operations within a single instruction. The goal of CISC is to reduce the number of instructions per program by making each instruction more capable.

- Instructions are complex and can perform multiple operations.
- Instruction length is variable.
- CISC is harder to pipeline due to its complexity.

RISC (Reduced Instruction Set Computing):

RISC simplifies the instruction set, ensuring each instruction takes only one clock cycle. This leads to faster execution and easier pipelining.

- Instructions are simpler and faster.
- Fixed-length instructions.
- Each instruction is executed in a single clock cycle, making it highly optimized for pipelining.

Feature	RISC	CISC
Instruction Complexity	Fewer, simpler instructions	Many complex instructions
Instruction Length	Fixed	Variable
Clock Cycles per Instruction	Single clock cycle	Multiple clock cycles
Memory Access	Load/store architecture	Direct memory access with most instructions
Pipelining	Highly optimized	More difficult due to complexity
Hardware Design	Simple, fewer transistors	Complex design
Examples	MIPS, ARM, PowerPC	Intel, AMD

Table 1: Features of RISC & CISC

---

3. MIPS Architecture

MIPS stands for *Microprocessor without Interlocked Pipeline Stages*, and it follows the RISC design principles. It was developed in the early 1980s and is known for its simplicity, efficiency, and pipeline design. MIPS processors are widely used in embedded systems and high-performance applications.

- **Fixed-length instructions**: Instructions in MIPS have the same length, making it easier to predict and optimize.
- **Efficiency**: MIPS focuses on simplicity, enabling faster execution and efficient pipelining.

---

4. QtSPIM Simulator

QtSPIM is a simulator that allows users to run MIPS assembly language programs. It simulates the operations of a MIPS processor, providing a learning platform for students to understand MIPS architecture.

Key Points:

1. **Simulates MIPS Processor**: Helps in understanding the workings of a MIPS-based system.
2. **Runs Assembly Code**: Users can write and execute MIPS assembly language programs.
3. **Registers and Memory**: Shows how data is stored in registers and memory.
4. **Simple Interface**: Allows users to load, run, and step through programs easily.
5. **Helps in Debugging**: Helps find and fix errors by observing changes in registers and memory.

5. QtSPIM File Structure

Source File Name:
The source file name where MIPS assembly code is written usually ends with .asm or .s. This file is loaded into QtSPIM for execution.

Memory Segments in QtSPIM:

- **Text Segment**: Contains the instructions of the program (e.g., addiu, lw, syscall).
- **Data Segment**: Stores the program's data, such as variables and constants. It holds the static/global variables and memory used during the execution of the program.

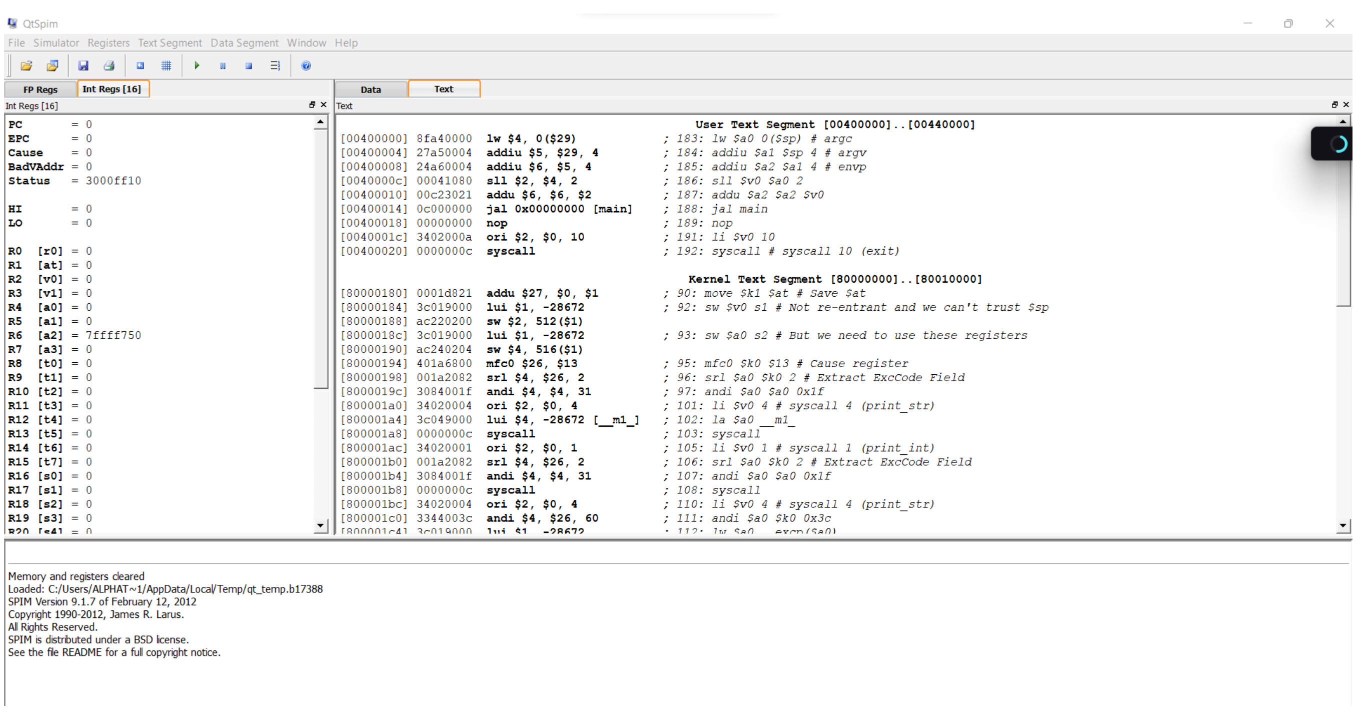


Figure 1: QTSPIM Structure

Part II

Write a program in MIPS assembly language, task2.asm, that computes and prints the sum of two numbers specified at runtime by the user. The program uses registers $t0 and $t1 to store the two user inputs, while $t2 holds the computed sum of these two values. It utilizes the $v0 register for system call numbers and $a0 for passing arguments to system calls. Upon execution, the program prompts the user to enter two integers, calculates their sum, and displays the result on the console before terminating gracefully. This program effectively demonstrates basic input, arithmetic operations, and output handling in MIPS assembly language.

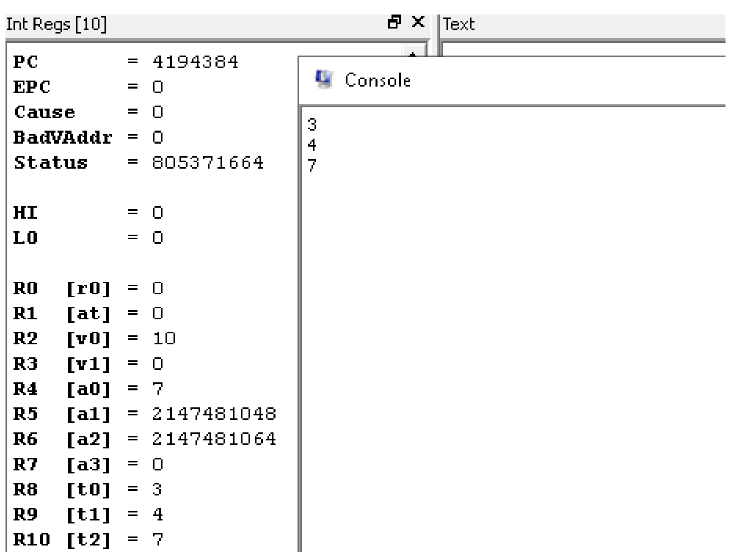
```
# Name of the programmer -- Hassan Zaib Jadoon Github: @hzjadoon
# task2.asm-- A program that computes and prints the sum of two numbers specified at runtime by the user.
# Registers used:
#       $t0 -used to hold the first number.
#       $t1 -used to hold the second number.
#       $t2 -used to hold the sum of the $t1 and $t2.
#       $v0 -syscall parameter and return value.
#       $a0 -syscall parameter.
main:
        ## Get first number from user, put into $t0.
        li $v0, 5       # load syscall read_int into $v0.
        syscall         # make the syscall.
        move $t0, $v0    # move the number read into $t0.

        ## Get second number from user, put into $t1.
        li $v0, 5       # load syscall read_int into $v0.
        syscall         # make the syscall.
        move $t1, $v0   # move the number read into $t1.
        add $t2, $t0, $t1 # compute the sum.

        ## Print out $t2.
        move $a0, $t2   # move the number to print into $a0.
        li $v0, 1       # load syscall print_int into $v0.
        syscall         # make the syscall.

        li $v0, 10      # syscall code 10 is for exit.
        syscall         # make the syscall.
# end of task2.asm
```

Output:



System calls:

Through the system call (syscall) instruction, SPIM offers a limited number of operating system-like services. A program loads the arguments into registers $a0,….$a3 (or $f12 for floating point values) and the system call code into register $v0 in order to request a service. Values returned by system calls are stored in register $v0 (or $f0 for floating point results). To print "the answer = 5," for instance

Task 3:

```
# Name of the programmer -- Hassan Zaib Jadoon Github: @hzjadoon
.data
a: .asciiz "The Answer is "

.text
main:
  li  $v0, 4   # system call code for print_str
  la  $a0, a    # address of string to print
  syscall          # print the string

  li  $v0, 1       # system call code for print_int
  li  $a0, 5       # integer to print
  syscall          # print

  li   $v0, 10     # system call code for exit
  syscall           # exit the program
```

Output: