

Faculty of Engineering

Computer and Systems Engineering Department

**CSE 311: Hardware Organization (2)**

Report

**MIPS Processor Project**

**Submitted by**

**Mohamed Nashaat Refaey Ibrahim (**33822)

**Mohamed Karam Helmy Farahat (**33816)

**Mahmoud Ahmed Mahmoud Ahmed (**33823)

**Mohamed Anwar Sa’ed Abd El-Ghany (**33809)

Fall (2015-2016)

Table of Contents

[1.0 Preface 3](#_Toc438990332)

[2.0 Implementation Description 3](#_Toc438990333)

[3.0 Data Path 3](#_Toc438990334)

[4.0 Simulation 3](#_Toc438990335)

[5.0 Test Programs 4](#_Toc438990336)

[5.1 Test 1: Program adds numbers from 1 to 100 4](#_Toc438990337)

[5.2 Test 2: Program multiplies a number in $s1 by 9 4](#_Toc438990338)

[6.0 Assembler 5](#_Toc438990339)

[6.1 Documentation 5](#_Toc438990340)

[6.1.1 Installation 5](#_Toc438990341)

[6.1.2 Usage 5](#_Toc438990342)

# 1.0 Preface

# 2.0 Implementation Description

We’ve built this MIPS processor description using Verilog HDL on Active-HDL simulator. We’ve implemented the design in figure (2.1) with some editing to support instructions like jal, jr.

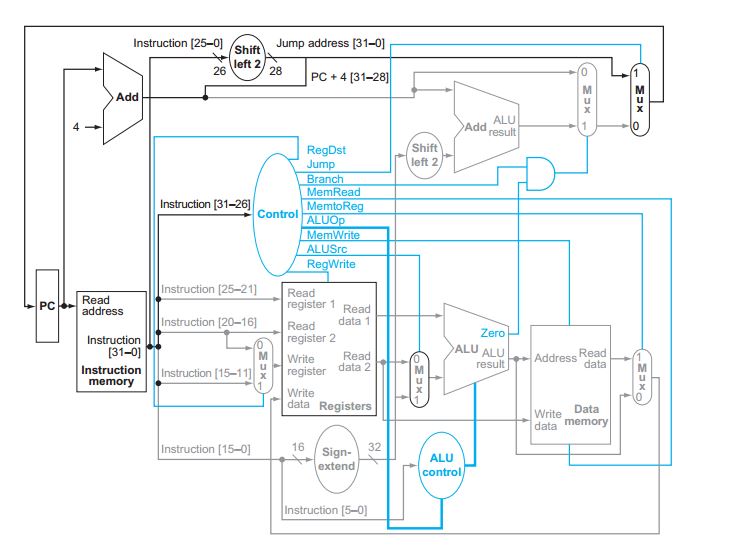


Figure ‑ - MIPS Processor Data Path

We split the description into modules, every module has its own description and the whole processor is a module where we used all other modules.

# 3.0 Data Path

We used the data path in figure (2.1) with extensions in figure (3.1). We have added

# 4.0 Simulation

# 5.0 Test Programs

We’ve built 12 test programs in MIPS assembly as follows:

## 5.1 Test 1: Program adds numbers from 1 to 100

*# Calculate the sum of numbers from 0:100 and store them in address 0x00* in memory

add $s1, $zero, $zero

addi $t1, $zero, 101

andi $t2, $t2, 0

andi $t4, $t4, 0

LOOP:

add $s1, $s1, $t2

addi $t2, $t2, 1

slt $t3, $t1, $t2

beq $t1, $t2, END

beq $t3, $zero, LOOP

END:

sw $s1, 0($t4)

Assembly Code

*00000000000000001000100000100000*

*00100001001000000000000001100101*

*00110001010010100000000000000000*

*00110001100011000000000000000000*

*00000010001010101000100000100000*

*00100001010010100000000000000001*

*00000001001010100101100000101010*

*00010001001010100000000000100100*

*00010001011000000000000000010000*

*10101101100100010000000000000000*

Binary Code

## 5.2 Test 2: Program multiplies a number in $s1 by 9

*# Multiply number in $s1 by 9*

lw $s1, 0($t1)  
addi $t2, $s1, 0  
sll $s1, $s1, 3  
add $s1, $s1, $t2

Assembly Code

*10001101001100010000000000000000*

*00100001010100010000000000000000*

*00000000000100011000100011000000*

*00000010001010101000100000100000*

Binary Code

## 5.2 Test 3: Program multiplies a number in $s1 by 9

# 6.0 Assembler

As a bonus we’ve built an assembler in python. It takes input assembly code from a file named “test.mips” and outputs binary code in a file named “output.binary”.

## 6.1 Documentation

### 6.1.1 Installation

* 1. Install python 2.7 from <https://www.python.org/ftp/python/2.7.11/python-2.7.11.msi>
  2. Add python destination folder to your PATH

### 6.1.2 Usage

* 1. Build your MIPS code and put it in a file named “test.mips”
  2. Put “test.mips” in the same directory of “assembler.py”
  3. Double click on “assembler.py” and a file called “output.binary” will be created if not already created. If “output.binary” is created previously, our assembler will overwrite its contents with the new generated binary code.