CS 342000 | CS343000 - Spring 2022

Instructor: Professor Izidor Gertner

Spring 2022 – 5/22/22

Single Cycle CPU - Azwad Shameem

**Table of Contents**

[Objective 3](#_Toc104139862)

[Components 4](#_Toc104139863)

[Package 4](#_Toc104139864)

[Instruction Memory 7](#_Toc104139865)

[AdderSub 8](#_Toc104139866)

[Extender 8](#_Toc104139867)

[Mux 9](#_Toc104139868)

[Bitwise Operations 10](#_Toc104139869)

[Adder 11](#_Toc104139870)

[Equal 11](#_Toc104139871)

[Data Memory 12](#_Toc104139872)

[Register File 13](#_Toc104139873)

[Instruction Register 14](#_Toc104139874)

[PC Register 15](#_Toc104139875)

[Control Unit 16](#_Toc104139876)

[Single Cycle CPU 19](#_Toc104139877)

[Simulation 21](#_Toc104139878)

[MIPS in Mars Simulator 32](#_Toc104139879)

[Conclusion 34](#_Toc104139880)

# Objective

The objective of the Single Cycle CPU lab was to write a program to compute the sum of five integers. Furthermore, we had to prove the correctness of the simulation using MIPS instructions in MARS Simulator. In order to simulate MIPS instructions in VHDL we needed to create data memory, instruction memory, 3-ported register file, IR-Instruction Register, Program Counter, AdderSub, Extender, and multiplexers components. Not to mention it was also imperative to implement a control unit that generated the control signals and regulated what action should be taken. Therefore, by completing the previous steps it became possible to simulate the cycle that an instruction takes when executing in the CPU such as reading the instruction from memory, breaking it down to specific parts and then executing the instruction and also determining the address of PC.

# Components

## Package

Graphical user interface, text, application

Description automatically generated

Figure 1: Package VHDL code. This package contains all of the components used.

Graphical user interface, text, application

Description automatically generated

Figure 2: Package VHDL code continued.

Graphical user interface, text, application

Description automatically generated

Figure 3: Package VHDL code continued.

Graphical user interface, text, application

Description automatically generated

Figure 4: Package VHDL code continued.

Graphical user interface, text, application, email

Description automatically generated

Figure 5: Package VHDL code continued.

## Instruction Memory

Graphical user interface, text, application

Description automatically generated

Figure 6: Instruction Memory VHDL code.

Graphical user interface, text, application, email

Description automatically generated

Figure 6: Instruction Memory Port VHDL code.

## AdderSub

Graphical user interface, text, application

Description automatically generated

Figure 7: AdderSub VHDL code. Used to compute instructions Add, Addi, Addu, Addiu, Sub, Subu.

## Extender

Graphical user interface, text, application, email

Description automatically generated

Figure 7: Extender VHDL code.

## Mux

Graphical user interface, text, application

Description automatically generated

Figure 8: Mux 2:1 VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 9: Mux 2:1 5-bit VHDL code. Used for input decide RW from RD or RT.

Text, application, email

Description automatically generated

Figure 10: Mux 3:1 VHDL code. Used for input decide address for regular instructions, BNE/BEQ and jump.

## Bitwise Operations

Graphical user interface, text, application

Description automatically generated

Figure 11: Bitwise Operations VHDL code. Used for instructions such as AND, OR, NOR, ORI, ANDI, SRL, SRL, SLL.

## Adder

Graphical user interface, text, application, email

Description automatically generated

Figure 12: Adder VHDL code. Used for address change per instruction.

## Equal

Graphical user interface, text, application, email

Description automatically generated

Figure 12: Equal VHDL code. Used BNE and BEQ to check condition to branch.

## Data Memory

Graphical user interface, text, application

Description automatically generated

Figure 13: Data Memory VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 14: Data Memory Port VHDL code. This is used in Data Memory.

## Register File

Graphical user interface, text, application, email

Description automatically generated

Figure 15: Register File VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 16: Triple Port Ram VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 16: Dual Port Ram VHDL code. Used in Triple Ported Ram.

## Instruction Register

Graphical user interface, text, application, email

Description automatically generated

Figure 17: Instruction Register VHDL code.

## PC Register

Graphical user interface, text, application

Description automatically generated

Figure 18: PC Register VHDL code.

## Control Unit

Graphical user interface, text, application, email

Description automatically generated

Figure 19: Control Unit VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 20: Control Unit VHDL code continued.

Graphical user interface, text, application

Description automatically generated

Figure 21: Control Unit VHDL code continued.

ALU

Text

Description automatically generated

Figure 22: ALU VHDL code.

Graphical user interface, text, application

Description automatically generated

Figure 23: ALU VHDL code continued.

## Graphical user interface, text, application Description automatically generatedSingle Cycle CPU

Figure 24: Single Cycle CPU VHDL code.

Text

Description automatically generated

Figure 25: Single Cycle CPU VHDL code continued.

Diagram

Description automatically generated

# Simulation

Timeline

Description automatically generated

Figure 26: Simulation Waveform

This is the simulation of using 5 load word instructions and then 5 add instructions and finally one save word instruction. We will be verifying every instruction one by one in a zoomed in snipit.

A picture containing graphical user interface

Description automatically generated

Figure 27: Simulation Waveform Zoomed in 100 – 200 ps - LW

First of all let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000000 to 0x00000004 at 50 ps this is because the BNE, BEQ or J instructions are not called so we only add 4 to the current program counter. Next we see that at RW at 50 ps the register is the register at 00001. Then if you look at BusW it is noticeable that at 150 ps the number 0x00000005 is loaded into RW which is the register at 00001.

Diagram

Description automatically generated with low confidence

Figure 28: Simulation Waveform Zoomed in 200 - 400ps - LW

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000004 to 0x00000008 at 200 ps this is because the BNE, BEQ or J instructions are not called so we only add 4 to the current program counter. Next, we see that at RW at 250 ps the register is the register at 00010. Then if you look at BusW it is noticeable that at 150 ps the number 0x00000006 is loaded into RW which is the register at 00010.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 29: Simulation Waveform Zoomed in 400 - 600 ps - LW

First of all let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000008 to 0x0000000C at 400 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next we see that at RW at 450 ps the register is the register at 00011. Then if you look at BusW it is noticeable that at 550 ps the number 0x00000007 is loaded into RW which is the register at 00011.

Diagram

Description automatically generated

Figure 30: Simulation Waveform Zoomed in 600 - 800ps - LW

First of all let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x0000000C to 0x00000010 at 600 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next we see that at RW at 650 ps the register is the register at 00100. Then if you look at BusW it is noticeable that at 750 ps the number 0x00000008 is loaded into RW which is the register at 00100.

Graphical user interface

Description automatically generated

Figure 31: Simulation Waveform Zoomed in 800-1000ps - LW

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000010 to 0x00000014 at 800 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next, we see that at RW at 850 ps the register is the register at 00101. Then if you look at BusW it is noticeable that at 950 ps the number 0x00000009 is loaded into RW which is the register at 00101. Therefore, at this point 5 instructions have been completed and 5 integers are stored in the registers 00001, 00010, 00011, 00100, 00101

Graphical user interface

Description automatically generated

Figure 32: Simulation Waveform Zoomed in 1000 - 1200ps - Add

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000014 to 0x00000018 at 800 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next, we see that RS is 1 which is register 00001, and RT is 2 which is register 00010 at 1050 ps. Then at 1150 ps it is noticeable that the numbers 0x00000005 and 0x00000006 we have loaded into the registers are adding and we get the result 0x0000000B in BusW. Furthermore, RW is 00001 from 1050 ps and after so the sum of addition will be saved in the register 00001.

A picture containing graphical user interface

Description automatically generated

Figure 33: Simulation Waveform Zoomed in 1200 - 1400ps - Add

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000018 to 0x0000001C at 1200 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next, we see that RS is 3 which is register 00011, and RT is 1 which is register 00001 at 1050 ps. Then at 1350 ps it is noticeable that the number 0x00000007 in BusA we have loaded is there and the number 0x0000000B in BusB, which is the sum we added for previously is now being added and gives us the value of 0x00000012 in BusW. Furthermore, RW is 00001 from 1200 ps and after so the sum of addition will be saved in the register 00001.

A picture containing diagram

Description automatically generated

Figure 34: Simulation Waveform Zoomed in 1400 - 1600ps - Add

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x0000001C to 0x00000020 at 1400 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next, we see that RS is 4 which is register 00100, and RT is 1 which is register 00001 at 1450 ps. Then at 1550 ps it is noticeable that the number 0x00000008 in BusA we have loaded is there and the number 0x00000012 in BusB, which is the sum we added for previously is now being added and gives us the value of 0x0000001A in BusW. Furthermore, RW is 00001 from 1400 ps and after so the sum of addition will be saved in the register 00001.

Graphical user interface

Description automatically generated with medium confidence

Figure 34: Simulation Waveform Zoomed in 1600 - 1800ps - Add

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000020 to 0x00000024 at 1600 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. Next, we see that RS is 5 which is register 00101, and RT is 1 which is register 00001 at 1650 ps. Then in 1750 ps it is noticeable that the number 0x00000009 in BusA we have loaded is there and the number 0x0000001A in BusB, which is the sum we added for previously is now being added and gives us the value of 0x00000023 in BusW. Furthermore, RW is 00001 from 1600 ps and after so the sum of addition will be saved in the register 00001.

A picture containing diagram

Description automatically generated

Figure 35: Simulation Waveform Zoomed in 1800 – 2000 ps - SW

First of all, let’s verify the program counter. On the top it is easily noticeable that the PC goes from 0x00000024 to 0x00000028 at 1800 ps this is because the BNE, BEQ or J instructions are called so we only add 4 to the current program counter. We see at 1800 ps RW is is 00001 and busW is 0x00000023. Furthermore in 1950 ps we see that BusB shows 0x00000023, which shows that the number has saved properly.

# MIPS in Mars Simulator

Graphical user interface

Description automatically generated

Figure 36: MIPS Code in MARS Simulator.

Table

Description automatically generated

Figure 37: Executed in MARS Simulator.

At the Data Segment at 0x10010000, MARS Simulator also shows that 0x00000023 is the sum of the addition of the 5 numbers, just like the simulation waveforms.

**Explanation**

Let’s recap we got a total of 0x23. The To obtain this total we did 0x5 + 0x6 = 0xB then 0xB + 0x7 = 0x12 then 0x8 + 0x12 = 0x1A then 0x9 + 0x1A = 0x23. To verify if this is correct, we check the decimal version can add 5 + 6 + 7 + 8 + 9 = 35. 35 is in decimal, but in hexadecimal 35(decimal) is 0x23. So far by checking decimal version the correctness stacks up to the hexadecimal version. This is furthered by the same numbers showing up in MIPS in MARS Simulator. Therefore, we can say that the simulation correctly computes the number in VHDL.

# Conclusion

The Single Cyle CPU lab was a great assignment to test our VHDL skills to simulate MIPS instructions in VHDL. Specifically, the assignment was to write a program to compute the sum of five integers. Furthermore, we had to prove the correctness of the simulation using MIPS instructions in MARS Simulator. Therefore, by using VHDL and creating components to simulate the MIPS instruction to get the sum of five integers we learn a lot about how the CPU works and how we can use VHDL to simulate things that may be happening in the CPU of our computers.