CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Design, Modeling, and Simulation of a MIPS Multi-Cycle Processor

A graduate project in partial fulfillment of the requirements

For the degree of Science in Electrical Engineering

By

Or Chakon

May 2017

The graduate project of Or Chakon is approved by:

_____	_____
Dr. Somnath Chattopadhyay	Date
_____	_____
Dr. Xiaojun Geng	Date
_____	_____
Dr. Nagi El Naga, Chair	Date

California State University, Northridge

Table of Contents

Signature Page ii
List of Figures v
List of Tables ix
Abstract x
1 INTRODUCTION 1
1.1 Objective 1
1.2 Project Outline 1
2 PROCESSOR OVERVIEW 2
3 MIPS INSTRUCTIONS 4
3.1 Supported J-type Instructions 4
3.2 Supported R-type Instructions 5
3.3 Supported I-type Instructions 12
4 COMPONENTS AND OPERATION 19
4.1 Multiplexor 19
4.2 ALU with Control Module 25
4.3 Program Counter with Control Circuitry 29
4.4 Sequence Controller with State Register 31
4.5 Memory and Register File 39
4.6 Registers and Miscellaneous Components 42
5 PROCESSOR EXCEPTIONS 43
6 COMPONENT LEVEL DESIGN AND TEST 44
6.1 ALU Design and Test 44
6.2 ALU Controller Design and Test 54
6.3 Combinational Block Design and Test 59
6.4 Concatenate Module Design and Test 64
6.5 Sequence Controller Design and Test 67
6.6 Multiplexor Design and Test 81
6.7 Program Counter Design and Test 85
6.8 RAM Design and Test 88
6.9 Register File Design and Test 93
6.11 Scalable Register Design and Test 98

6.12 Shift-By-Two Module Design and Test 101
6.13 Sign Extend Module Design and Test 104
6.14 State Register Design and Test 107
7 RISC PROCESSOR OPERATION 110
7.1 RISC Processor Design Code 110
7.2 RISC Processor Test Setup and Documentation 113
7.3 RISC Processor Test Results and Validation 119
8 PROCESSOR TIMING 170
9 CONCLUSION 174
Works Cited 175

List of Figures

Figure 1. MIPS Processor Design 3
Figure 2. 2x1 Mux 19
Figure 3. ALU with Controller 25
Figure 4. Program Counter with Combinational Control Logic 29
Figure 5. Sequence Controller with State Register 31
Figure 6. Sequence Controller FSM 34
Figure 7. RAM Module 39
Figure 8. Register File 41
Figure 9. ALU Design Code Part 1 45
Figure 10. ALU Design Code Part 2 46
Figure 11. ALU Design Code Part 3 47
Figure 12. ALU Test Bench Part 1 48
Figure 13. ALU Test Bench Part 2 49
Figure 14. ALU Test Bench Part 3 50
Figure 15. ALU Test Bench Part 4 51
Figure 16. ALU Simulation Results Part 1 52
Figure 17. ALU Simulation Results Part 2 53
Figure 18. ALU Controller Design Code 54
Figure 19. ALU Controller Test Bench Part 1 55
Figure 20. ALU Controller Test Bench Part 2 56
Figure 21. ALU Controller Test Bench Part 3 57
Figure 22. ALU Controller Simulation Results 58
Figure 23. Combinational Block Design Code 59
Figure 24. Combinational Block Test Bench Part 1 60
Figure 25. Combinational Block Test Bench Part 2 61
Figure 26. Combinational Block Test Bench Part 3 62
Figure 27. Combinational Block Simulation Results 63
Figure 28. Concatenate Module Design Code 64
Figure 29. Concatenate Module Test Bench 65
Figure 30. Concatenate Module Simulation Results 66
Figure 31. Sequence Controller Design Code Part 1 68
Figure 32. Sequence Controller Design Code Part 2 69
Figure 33. Sequence Controller Design Code Part 3 70
Figure 34. Sequence Controller Design Code Part 4 71
Figure 35. Sequence Controller Design Code Part 5 72
Figure 36. Sequence Controller Design Code Part 6 73
Figure 37. Sequence Controller Design Code Part 7 74
Figure 38. Sequence Controller Design Code Part 8 75
Figure 39. Sequence Controller Design Code Part 9 76
Figure 40. Sequence Controller Design Code Part 10 77
Figure 41. Sequence Controller Test Bench Part 1 78

Figure 42. Sequence Controller Test Bench Part 2 79
Figure 43. Sequence Controller Simulation Results 80
Figure 44. Multiplexor Design Code 81
Figure 45. Multiplexor Test Bench Part 1 82
Figure 46. Multiplexor Test Bench Part 2 83
Figure 47. Multiplexor Simulation Results 84
Figure 48. Program Counter Design Code 85
Figure 49. Program Counter Test Bench 86
Figure 50. Program Counter Simulation Results 87
Figure 51. RAM Design Code 88
Figure 52. RAM Test Bench Part 1 89
Figure 53. RAM Test Bench Part 1 90
Figure 54. RAM Simulation Results Showing Individual Reads 91
Figure 55. RAM Simulation Results Showing Block Read 92
Figure 56. Register File Design Code 93
Figure 57. Register File Test Bench Part 1 94
Figure 58. Register File Test Bench Part 2 95
Figure 59. Register File Simulation Results Showing Individual Reads 96
Figure 60. Register File Simulation Results Showing Block Reads 97
Figure 61. Scalable Register Design Code 98
Figure 62. Scalable Register Test Bench 99
Figure 63. Scalable Register Simulation Results 100
Figure 64. Shift Module Design Code 101
Figure 65. Shift Module Test Bench 102
Figure 66. Shift Module Simulation Results 103
Figure 67. Sign Extend Module Design Code 104
Figure 68. Sign Extend Module Test Bench 105
Figure 69. Sign Extend Module Simulation Results 106
Figure 70. State Register Design Code 107
Figure 71. State Register Test Bench 108
Figure 72. State Register Simulation Results 109
Figure 73. RISC Processor Design Code Part 1 111
Figure 74. RISC Processor Design Code Part 2 112
Figure 75. RISC Processor Test Bench 114
Figure 76. RAM_Data.txt Showing Initial RAM Data Part 1 116
Figure 77. RAM_Data.txt Showing Initial RAM Data Part 2 117
Figure 78. RegFile_Data.txt Showing Initial Register File Data 118
Figure 79. Simulation Results for ADD Instruction 120
Figure 80. Simulation Results for SUB Instruction 121
Figure 81. Simulation Results for SW Instruction 122
Figure 82. Simulation Results for LW Instruction 123
Figure 83. Simulation Results for AND Instruction 124

Figure 84. Simulation Results for BEQ Instruction Part 1 125
Figure 85. Simulation Results for BEQ Instruction Part 2 126
Figure 86. Simulation Results for ADDi Instruction 127
Figure 87. Simulation Results for BNE Instruction Part 1 128
Figure 88. Simulation Results for BNE Instruction Part 2 129
Figure 89. Simulation Results for Jump Instruction 130
Figure 90. Simulation Results for OR Instruction 131
Figure 91. Simulation Results for SH Instruction 132
Figure 92. Simulation Results for LB Instruction 133
Figure 93. Simulation Results for LBu Instruction 134
Figure 94. Simulation Results for XOR Instruction 135
Figure 95. Simulation Results for SB Instruction 136
Figure 96. Simulation Results for JR Instruction 137
Figure 97. Simulation Results for ANDi Instruction 138
Figure 98. Simulation Results for ORi Instruction 139
Figure 99. Simulation Results for XORi Instruction 140
Figure 100. Simulation Results for LH Instruction 141
Figure 101. Simulation Results for LHu Instruction 142
Figure 102. Simulation Results for BLTZ Instruction Part 1 143
Figure 103. Simulation Results for BLTZ Instruction Part 2 144
Figure 104. Simulation Results for NOR Instruction 145
Figure 105. Simulation Results for ADDiu Instruction 146
Figure 106. Simulation Results for SLTiu Instruction Part 1 147
Figure 107. Simulation Results for SLTiu Instruction Part 2 148
Figure 108. Simulation Results for SLTi Instruction Part 1 149
Figure 109. Simulation Results for SLTi Instruction Part 2 150
Figure 110. Simulation Results for BLEZ Instruction Part 1 151
Figure 111. Simulation Results for BLEZ Instruction Part 2 152
Figure 112. Simulation Results for SLL Instruction 153
Figure 113. Simulation Results for SLLV Instruction 154
Figure 114. Simulation Results for SRL Instruction 155
Figure 115. Simulation Results for SRLV Instruction 156
Figure 116. Simulation Results for SRA Instruction 157
Figure 117. Simulation Results for SRAV Instruction 158
Figure 118. Simulation Results for BGTZ Instruction Part 1 159
Figure 119. Simulation Results for BGTZ Instruction Part 2 160
Figure 120. Simulation Results for SLTu Instruction Part 1 161
Figure 121. Simulation Results for SLTu Instruction Part 2 162
Figure 122. Simulation Results for SLT Instruction Part 1 163
Figure 123. Simulation Results for SLT Instruction Part 2 164
Figure 124. Simulation Results for JAL Instruction 165
Figure 125. Simulation Results for ADDu Instruction 166

Figure 126. Simulation Results for SUBu Instruction 167
Figure 127. Simulation Results for JALR Instruction Part 1 168
Figure 128. Simulation Results for JALR Instruction Part 2 169
Figure 129. Processor Timing Example 171

List of Tables

Table 1. Mux Functionality 20
Table 2. ALU Operations 26
Table 3. ALU Controller Logic 27
Table 4. Sequence Controller Outputs 32
Table 5. Function Field in Signed/Unsigned Operations 33
Table 6. Register File Mapping 40
Table 7. Processor Registers 42
Table 8. Scoped Signals Included in Simulation 115
Table 9. Component Gate Delays 172

Abstract

Design, Modeling, and Simulation of a MIPS Multi-Cycle Processor

By

Or Chakon

Master of Science in Electrical Engineering

In this project, the design, modeling, simulation, and timing analysis of a MIPS Multi-Cycle Processor using Hardware Description Language (HDL) are presented. During the design phase the processor schematic was generated incrementally, adding hardware to support a wide range of MIPS instructions. The processor modeling was done using Verilog HDL and the bottom-up design approach, where each of the processor components were designed and tested before being incorporated into the top level processor design. The processor design was then tested by executing each of the supported MIPS instructions and verifying proper functionality. Once the processor was tested and fully functional, the processor timing was analyzed to assure optimized performance.

The MIPS Multi-Cycle Processor is a type of Reduced Instruction Set Computer (RISC) architecture. RISC is one of two prominent types of processors. The other is Complex Instruction Set Computing (CISC). The main difference between the two processors is that CISC does not have a fixed length and may contain different formats whereas RISC is always a fixed length and consistent format. RISC was designed to reduce complexity of the instruction sets and ultimately improve performance.

The RISC processor is favored as is it newer technology. As a result, this project provides detailed instruction sets, testing cases, design, findings, and implementation of the MIPS Multi-Cycle Processor.

1 INTRODUCTION

RISC is a type of CPU design in which it is believed that a simplified instruction set will enhance performance of the processor. It uses a small, highly-optimized set of instructions rather than a more complex set as found in other processors. The word “reduced” in the name refers to the amount of work for any single instruction set.

Typical features of RISC architecture include: fixed length, standard format, identical general purpose registers, simple addressing modes, one cycle execution time, and pipelining.

Aside from quicker performance, RISC processor componets are generally cheaper to design and produce as they utilize less transistors. RISC is the newer technology and is widely used in the industry compared to other processor types.

1.1 Objective

The objective of this project is to successfully design, model and simulate a MIPS Multi-Cycle Processor using Verilog HDL. The button-up design approach was used where each sub-module of the processor was first designed, coded, and tested. Once all sub-modules were designed and determined to be fully functional, they were instantiated into a structural module to form the MIPS processor. The processor was then tested by executing a set of MIPS instructions while verifying proper functionality and timing.

1.2 Project Outline

This project consists of nine chapters. The first chapter is an introduction chapter which states the objective. The second chapter provides an overview of the entire processor and shows the complete schematic of the processor architecture. The third chapter describes the MIPS instructions supported by the processor. There are three types of instruction formats and the supported instructions are grouped based on their instruction type. The fourth chapter describes the theory of operation of each individual component in my processor. The fifth chapter describes processor exceptions and how they are dealt with in my design. The sixth chapter documents each component design and verification testing. The seventh chapter documents the processor top level design, verification method, and final test results. The eighth chapter deals with timing for physical implementation of the processor in actual hardware. The last chapter provides a conclusion to the project.

2 PROCESSOR OVERVIEW

The Complex Instruction Set Computing (CISC) processor consists of an architecture which contains a large set of computer instructions that range from simple to complex. CISC computer instructions are not of a fixed length and may contain a different number of addresses and different size operands. The instruction types may range, some transferring data from memory to register, register to register, or memory to memory direct. The complexity and size of the CISC instructions makes the computer architecture more complex and ultimately effects performance.

The Reduced Instruction Set Computer (RISC) was designed with the goal of simplifying the instruction set and reducing the number of clock cycles it takes to execute an instruction. RISC instructions are always a fixed length (32-bit or 64-bit) and follow a specific format which makes decoding easier. Memory access is very limited as most instruction move data from register to register. Register to register transactions are advantageous as register systems are smaller and faster than memory. RISC processors use registers as much as possible only accessing memory for less frequently used variables.

The MIPS processor, also known as the "Microprocessor without Interlocked Pipeline Stages", is a type of RISC architecture. The three types for MIPS architectures studied for this project are Single-cycle, Multi-cycle, and Pipelined architectures. The Single-cycle processor can execute an instruction in one clock cycle. As can be imagined, the clock cycle of a Single-cycle processor has to be big enough to allow the entire instruction to be processed. The Multi-cycle processor breaks the instructions down into smaller steps and ultimately decreases the required cycle time. Using the Multi-cycle approach, different instruction may take different amounts of time to process unlike in the Single-cycle approach where instruction processing is as fast as the slowest instruction. A Pipelined processor breaks the process down into steps which allow multiple instructions to be processed concurrently. When compared to a Multi-cycle processor, the Pipelined processor does not help the latency of a single instruction but rather the throughput of a set of instructions.

The RISC processor design is of a Multi-cycle MIPS architecture. Figure 1 shows a complete schematic of the design. The following chapters will discuss this project's design in detail.

Figure 1 was generated using AutoCAD. See separate supporting documents for a more legible version.

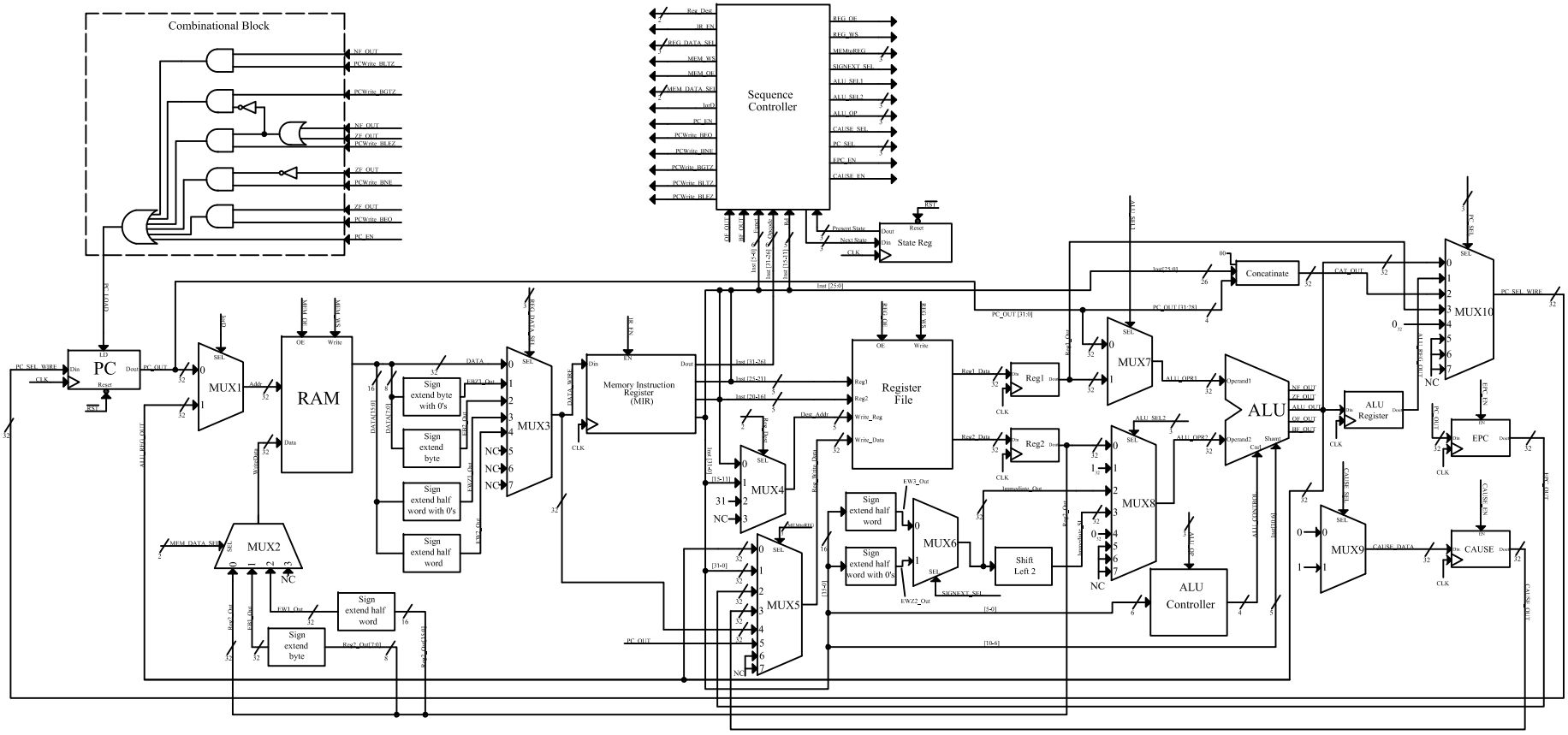


Figure 1. MIPS Processor Design

3 MIPS INSTRUCTIONS

The project's processor design is based on the MIPS Reduced Instruction Set Computer (RISC) architecture and includes a subset of the MIPS Instruction set. MIPS Instructions are always 32-bits wide and use one of the three following instruction types:

R-Type

opcode[31:26]	Rs[25:21]	Rt[20:16]	Rd[15:11]	Shamt[10:6]	Function[5:0]
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type

opcode[31:26]	Rs[25:21]	Rt[20:16]	Address/constant [15:0]
6 bits	5 bits	5 bits	16 bits

J-Type

opcode[31:26]	Address[25:0]
6 bits	26 bits

This chapter outlines all MIPS Instructions supported by the processor design in three sections, one for each instruction type as reference in Appendix B of *Computer Organization and Design* (Patterson and Hennessy). In each section, the supported instructions are listed under their corresponding instruction type. Under each instruction, the instruction structure, operation, number of cycles, and sequence controller states are given. The instruction structure shows the opcode of each instruction along with the instruction bit allocation. The Operation field shows the actual register operation of the instruction. The Number of Cycles field shows how many processor cycles it takes to complete the instruction. The Sequence Controller States field shows which states of the Sequence Controller state machine are executed. Additionally, the Comments field includes key points worth mentioning.

3.1 Supported J-type Instructions

1. jump – j instruction

J-Type

000010	Jump target (26 bits)

Operation: PC ← PC[31:28] || Inst[25:0] || 00
Number of Cycles: 3
Sequence Controller States: 0, 1, and 2.
Comments: Used to jump to the address derived by shifting the 26-bit Jump Target left by two bits and concatenating the result with the 4 most significant bits of the original program counter.

2. jump and link – jal instruction

J-Type

000011	Jump target (26 bits)

Operation: $ra ← PC + 1
PC ← PC[31:28] || Inst[25:0] || 00
Number of Cycles: 3
Sequence Controller States: 0, 1, 3 and 36.
Comments: The next PC is calculated using the same equation as in the jump instruction. The difference in this instruction is that the next PC (PC + 1) is stored in the return address register ($ra).

3.2 Supported R-type Instructions

1. Jump and link register – jalr instruction

R-Type

000000	Rs	Rt	Rd	00000	001001

Operation: PC ← Rs
Rd ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, 21 and 35.
Comments: None.

2. Jump register – jr instruction

R-Type

000000	Rs	Rt	Rd	00000	001000

Operation: PC ← Rs
Number of Cycles: 3
Sequence Controller States: 0, 1, and 20.
Comments: ALU not used in the execution of this instruction.

3. Addition – add instruction

R-Type

000000	Rs	Rt	Rd	00000	100000

Operation: Rd ← Rs + Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: In the case of an overflow, an exception is created and the Sequence Controller will set the appropriate registers. This means the instead of going to state 5 the Sequence Controller will go to state 30 in the write back cycle.

4. Unsigned addition – addu instruction

R-Type

000000	Rs	Rt	Rd	00000	100001

Operation: Rd ← Rs + Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Same as add instruction, but overflow is ignored.

5. Subtraction – sub instruction

R-Type

000000	Rs	Rt	Rd	00000	100010

Operation: Rd ← Rs - Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: In the case of an overflow, an exception is created and the Sequence Controller will set the appropriate registers. This means the instead of going to state 5 the Sequence Controller will go to state 30 in the write back cycle.

6. Unsigned subtraction – subu instruction

R-Type

000000	Rs	Rt	Rd	00000	100011

Operation: Rd ← Rs - Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Same as sub instruction, but overflow is ignored.

7. Logical AND – AND instruction

R-Type

000000	Rs	Rt	Rd	00000	100100

Operation: Rd ← Rs AND Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: None.

8. Logical OR – OR instruction

R-Type

000000	Rs	Rt	Rd	00000	100101

Operation: Rd ← Rs OR Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: None.

9. Logical XOR – XOR instruction

R-Type

000000	Rs	Rt	Rd	00000	100110

Operation: Rd ← Rs XOR Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: None.

10. Logical NOR – NOR instruction

R-Type

000000	Rs	Rt	Rd	00000	100111

Operation: Rd ← Rs NOR Rt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: None.

11. Shift Left Logic – sll instruction

R-Type

000000	Rs	Rt	Rd	active	000000

Operation: Rd ← Rt shifted left by Shamt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted left by the number specified by the shamt field of the instruction and stored in Rd. The most significant bits fall off.

12. Shift Left Logic by variable – sllv instruction

R-Type

000000	Rs	Rt	Rd	00000	000100

Operation: Rd ← Rt shifted left by Rs
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted left by the number specified by the Rs field of the instruction and stored in Rd. The most significant bits fall off.

13. Shift Right Logic – srl instruction

R-Type

000000	Rs	Rt	Rd	active	000010

Operation: Rd ← Rt shifted right by shamt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted right by the number specified by the shamt field of the instruction and stored in Rd. The most significant bits are filled with 0's.

14. Shift Right Logic variable – srlv instruction

R-Type

000000	Rs	Rt	Rd	00000	000110

Operation: Rd ← Rt shifted right by Rs
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted right by the number specified by the Rs field of the instruction and stored in Rd. The most significant bits are filled with 0's.

15. Shift Right Arithmetic – sra instruction

R-Type

000000	Rs	Rt	Rd	active	000011

Operation: Rd ← Rt shifted right by shamt
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted right by the number specified by the shamt field of the instruction and stored in Rd. The sign bit is shifted in from the most significant end while bits fall off the least significant end.

16. Shift Right Arithmetic Variable – srav instruction

R-Type

000000	Rs	Rt	Rd	00000	000111

Operation: Rd ← Rt shifted right by Rs
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rt is shifted right by the number specified by the Rs field of the instruction and stored in Rd. The sign bit is shifted in from the most significant end while bits fall off the least significant end.

17. Set if less than unsigned: sltu instruction

R-Type

000000	Rs	Rt	Rd	00000	101001

Operation: if Rs < Rt then Rd ← 1
Else Rd ← 0
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rs and Rt are unsigned numbers.

18. Set if less than: slt instruction

R-Type

000000	Rs	Rt	Rd	00000	101010

Operation: if Rs < Rt then Rd ← 1
Else Rd ← 0
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 4, and 5.
Comments: Rs and Rt are signed numbers.

19. Move from C0: mfc0 instruction

R-Type

010000	Rs	Rt	Rd	00000	000000

Operation: if Rd = 01110 then Rd ← EPC
Else if Rd = 01101 then Rd ← Cause
Number of Cycles: 3
Sequence Controller States: 0, 1, (32 or 33), and 8.
Comments: This instruction is dependent on Rd and is used to transfer the content of either Exception Register to the Register File.

3.3 Supported I-type Instructions

1. Branch on less than zero: bltz instruction

I-Type

000001	Rs	Rt	Address/constant

Operation: if Rs < 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)
else PC ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, and 13.
Comments: If condition is met, branch to PC + 1 + 4*offset.

2. Branch on equal: beq instruction

I-Type

000100	Rs	Rt	Address/constant

Operation: if Rs = Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)
else PC ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, and 9.
Comments: If condition is met, branch to PC + 1 + 4*offset.

3. Branch on not equal: bne instruction

I-Type

000101	Rs	Rt	Address/constant

Operation: if Rs ≠ Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)
else PC ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, and 10.
Comments: If condition is met, branch to PC + 1 + 4*offset.

4. Branch on less than or equal zero: blez instruction

I-Type

000110	Rs	Rt	Address/constant

Operation: if Rs ≤ 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)
else PC ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, and 11.
Comments: If condition is met, branch to PC + 1 + 4*offset.

5. Branch on greater than zero: bgtz instruction

I-Type

000111	Rs	Rt	Address/constant

Operation: if Rs > 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)
else PC ← PC + 1
Number of Cycles: 3
Sequence Controller States: 0, 1, and 12.
Comments: If condition is met, branch to PC + 1 + 4*offset.

6. Immediate addition: addi instruction

I-Type

001000	Rs	Rt	Address/constant

Operation: Rt ← Rs + (sign extended I[15:0])
PC ← PC +1
Number of Cycles: 4
Sequence Controller States: 0, 1, 7, and 8.
Comments: In the case of an overflow, an exception is created.

7. Immediate addition unsigned: addiu instruction

I-Type

001001	Rs	Rt	Address/constant

Operation: Rt ← Rs + (sign extended I[15:0])
PC ← PC +1
Number of Cycles: 4
Sequence Controller States: 0, 1, 29, and 8.
Comments: Same as immediate add instruction, but overflow is ignored.

8. Immediate set-if-less-than unsigned: sltiu instruction

I-Type

001011	Rs	Rt	Address/constant

Operation: if Rs < (sign extended I[15:0]) then Rt ← 1
else Rt ← 0
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 28, and 8.
Comments: None.

9. Immediate set-if-less-than: slti instruction

I-Type

001010	Rs	Rt	Address/constant

Operation: if Rs < (sign extended I[15:0]) then Rt ← 1
else Rt ← 0
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 6, and 8.
Comments: None.

10. Immediate logic AND: andi instruction

I-Type

001100	Rs	Rt	Address/constant

Operation: Rt ← Rs AND (sign extended I[15:0])
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 14, and 8.
Comments: None.

11. Immediate logic OR: ori instruction

I-Type

001101	Rs	Rt	Address/constant

Operation: Rt ← Rs OR (sign extended I[15:0])
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 15, and 8.
Comments: None.

12. Immediate logic XOR: xori instruction

I-Type

001110	Rs	Rt	Address/constant

Operation: Rt ← Rs XOR (sign extended I[15:0])
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 16, and 8.
Comments: None.

13. Load Word: lw instruction

I-Type

100011	Rs	Rt	Address/constant

Operation: Rt ← M[Rs + (sign extended I[15:0])]
PC ← PC + 1
Number of Cycles: 5
Sequence Controller States: 0, 1, 7, 18 and 19.
Comments: ALU output (i.e. Rs + sign extended I[15:0]) determines the address of the word loaded from memory into the register file at the location specified by Rt.

14. Load Unsigned Byte: lbu instruction

I-Type

100100	Rs	Rt	Address/constant

Operation: Rt ← sign extended with 0's (M[Rs + (sign extended I[15:0])])
PC ← PC + 1
Number of Cycles: 5
Sequence Controller States: 0, 1, 7, 22 and 19.
Comments: The first byte is signed extended with 0's and loaded from memory into the Register File at the location specified by Rt.

15. Load Byte: lb instruction

I-Type

100000	Rs	Rt	Address/constant

Operation: Rt ← sign extended with the leftmost bit (M[Rs + (sign extended I[15:0])])
PC ← PC + 1
Number of Cycles: 5
Sequence Controller States: 0, 1, 7, 23 and 19.
Comments: The first byte is signed extended with the most significant bit and loaded from memory into the Register File at the location specified by Rt.

16. Load Unsigned Half Word: lhu instruction

I-Type

100101	Rs	Rt	Address/constant

Operation: Rt ← sign extended with 0's (M[Rs + (sign extended I[15:0])])
PC ← PC + 1
Number of Cycles: 5
Sequence Controller States: 0, 1, 7, 24 and 19.
Comments: The first half of the word is signed extended with 0's and loaded from memory into the Register File at the location specified by Rt.

17. Load Half Word: lh instruction

I-Type

100001	Rs	Rt	Address/constant

Operation: Rt ← sign extended with the leftmost bit (M[Rs + (sign extended I[15:0])])
PC ← PC + 1
Number of Cycles: 5
Sequence Controller States: 0, 1, 7, 25 and 19.
Comments: The first half word is signed extended with the most significant bit and loaded from memory into the Register File at the location specified by Rt.

18. Store Byte: sb instruction

I-Type

101000	Rs	Rt	Address/constant

Operation: M[Rs + (sign extended I[15:0])] ← Rt[7:0]
PC ← PC + 1
Number of Cycles: 4
Sequence Controller States: 0, 1, 7, 26 and 34.
Comments: Rt[7:0] is sign extended with 0's and stored in memory at the address determined by the ALU output (i.e. Rs + sign extended I[15:0]).

19. Store Byte: sh instruction

I-Type

101001	Rs	Rt	Address/constant

Operation: M[Rs + (sign extended I[15:0])] ← Rt[15:0]
PC ← PC + 1

Number of Cycles: 4

Sequence Controller States: 0, 1, 7, 27 and 34.

Comments: Rt[15:0] is sign extended with 0's and stored in memory at the address determined by the ALU output (i.e. Rs + sign extended I[15:0]).

20. Store Word: sw instruction

I-Type

101011	Rs	Rt	Address/constant

Operation: M[Rs + (sign extended I[15:0])] ← Rt
PC ← PC + 1

Number of Cycles: 4

Sequence Controller States: 0, 1, 7, 17 and 34.

Comments: Rt[31:0] is stored in memory at the address determined by the ALU output (i.e. Rs + sign extended I[15:0]).

4 COMPONENTS AND OPERATION

The MIPS processor design uses a total of 10 different major components. Some components appear in more than one instance and in different variations like the multiplexor. The major components and quantities used are as follows:

- Register (8)
- Multiplexor (10)
- Random Access Memory (1)
- Sign Extend Module (8)
- Register File (1)
- Shift Module (1)
- Concatenate Module (1)
- Arithmetic Logic Unit (1)
- Arithmetic Logic Unit Controller (1)
- Sequence Controller (1)

The top level design also uses various logic gates, namely, two OR gates, five AND gates, and two inverters. The logic gates are used to accomplish program counter bunching operations as will be discussed in later sections. This chapter focuses on each component of the processor individually and describes their functionality as used in the processor design.

4.1 Multiplexor

The multiplexor, also known as a Mux, is used to enable certain parts of the processor data path at a given time and is controlled by the Sequence Controller. A general illustration of the multiplexor is shown in Figure 2.

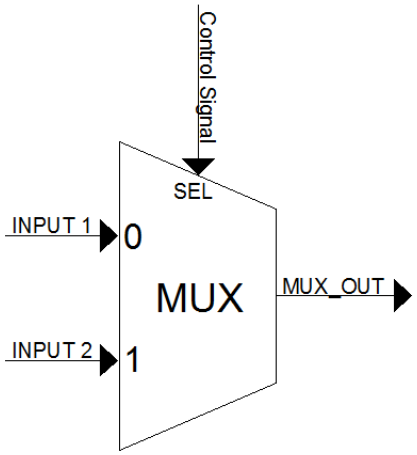


Figure 2. 2x1 Mux

There are 10 instances and three different variations of the multiplexor in the project's processor design. The three variations include 2x1, 4x1, and 8x1 multiplexor components. Each multiplexor component has its own reference designator and performs a specific function. The input select signal, SEL, of all the multiplexor instances originates from the Sequence Controller. The functionality of each multiplexor used in the processor design is described in Table 1.

Table 1. Mux Functionality

Component	Size	Control Signal	Functionality
Mux1	2x1	IorD	Determines the address of the next induction to be read from or written to RAM. Mux1 is controlled by the IorD signal. • When IorD = 0, the RAM address comes from the program counter. • When IorD = 1, the address comes from the ALU Register, which holds the value of the last ALU execution output.
Mux2	4x1	MEM_DATA_SEL	Determines the RAM write data source exclusively for Store instructions. • When MEM_DATA_SEL = 00, the RAM data source is the complete 32-bit content of Reg2. This is used in the Store Word (SW) instruction execution. • When MEM_DATA_SEL = 01, the RAM data source is the sign extended least significant byte of Reg2. This is used in the Store Byte (SB) instruction execution. • When MEM_DATA_SEL = 10, the RAM data source is the sign extended least significant half word of Reg2. This is used in the Store Half (SH) instruction execution. • Mux2 input '11' is not connected to anything and therefore not used.
Mux3	8x1	REG_DATA_SEL	Determines the data source of the data written to the Memory Instruction Register (IR) and used to facilitate the different Load instructions. • When REG_DATA_SEL = 000, the IR data source is the complete 32-bit content of the RAM data output. This is set during the Fetch cycle of every instruction and in the Memory cycle the Load Word (LW) instruction. • When REG_DATA_SEL = 001, the IR data source is the zero sign extended least significant byte of the RAM data output. This is used in the Load Byte Unsigned (LBu) instruction execution.

			• When REG_DATA_SEL = 010, the IR data source is the sign extended least significant byte of the RAM data output. This is used in the Load Byte (LB) instruction execution. • When REG_DATA_SEL = 011, the IR data source is the zero sign extended least significant half word of the RAM data output. This is used in the Load Half Unsigned (LHu) instruction execution. • When REG_DATA_SEL = 100, the IR data source is the sign extended least significant half word of the RAM data output. This is used in the Load Half (LH) instruction execution. • Mux3 inputs '101', '110', and '111' are not connected to anything and therefore not used.
Mux4	4x1	Reg_Dest	Determines the write address of data written to the Register File. • When Reg_Dest = 00, the address source is the Rt field of the I-type instructions derived from bits [20:16] of the instruction. This is set during the write back cycle of I-type instructions. • When Reg_Dest = 01, the address source is the Rd field of the R-type instructions derived from bits [15:11] of the instruction. This is set during the write back cycle of R-type instructions. • When Reg_Dest = 10, the address is a constant 31, binary 11111, which is the address location of the Return Address Register ($ra) in the Register File. This is set during the write back cycle of J-type instructions, particularly the Jump and Link (JAL) instruction. • Mux4 input '11' is not connected to anything and therefore not used.
Mux5	8x1	MEMtoREG	Determines the data source of data written to the Register File. • When MEMtoREG = 000, the Register File data source is the 32-bit output of the ALU output. This is used in all instructions were the Register file data is calculated by the ALU. • When MEMtoREG = 001, the Register File data source is the complete 32-bit output of IR. This is used in all Load instructions were the Register file data derived from the immediate field of the instruction bits [15:0]. Note that the immediate field is sign extended in accordance with the

			specific load instruction being executed as controlled by MUX3. • When MEMtoREG = 010, the Register File data source is the EPC register. This is set during the execution of processor exception instructions. • When MEMtoREG = 011, the Register File data source is the Cause register. This is set during the execution of processor exception instructions. • When MEMtoREG = 100, the Register File data source is the output of MUX3. This data path is used for Load operations, where data is read from RAM and stored into the Register File, bypassing the MIR to avoid adding an extra clock cycle. • When MEMtoREG = 101, the Register File data source is the output of the Program Counter. It is used for storing the Program Counter value of the next instruction (PC + 1) in JAL and JALR operations. • Mux5 inputs '110' and '111' are not connected to anything and therefore not used.
Mux6	2x1	SIGNEXT_SEL	Determines whether to sign extend the instruction immediate field with either the most significant bit for signed instructions, or zeros for unsigned instructions. • When SIGNEXT_SEL = 0, the immediate value is sign extended with the most significant bit. This is used in signed I-type instructions where the sign extended immediate value is used as the second operand of the ALU. • When SIGNEXT_SEL = 1, the immediate value is sign extended with zeros. This is used in unsigned I-type instructions where the zero sign extended immediate value is used as the second operand of the ALU.
Mux7	2x1	ALU_SEL1	Determines the source of the first operand of the ALU. • When ALU_SEL1 = 0, the 32-bit content of the current Program Counter (PC) is selected as the first operand of the ALU. This is used when incrementing the PC. • When ALU_SEL1 = 1, the content of Reg1 is selected as the first operand of the ALU. Note that Reg1 holds the Register File content at the location in the Register file specified by the Rs field, bits [25:21] of an R-type or I-type instructions.
Mux8	8x1	ALU_SEL2	Determines the source of the second operand of the ALU.

			• When ALU_SEL2 = 000, the content of Reg2 is selected as the second operand of the ALU. Note that Reg2 holds the Register File content at the location in the Register file specified by the Rt field, bits [20:16] of an R-type or I-type instructions. • When ALU_SEL2 = 001, the second operand of the ALU is a constant 1. This is used for incrementing the Program Counter. • When ALU_SEL2 = 010, the second operand of the ALU is the sign extended immediate field of the instruction. The immediate value is either sign extended with zeros or the most significant bit as determined by Mux6. • When ALU_SEL2 = 011, the second operand of the ALU is the sign extended immediate field of the instruction shifted left by two bits. • When ALU_SEL2 = 100, the second operand of the ALU is a constant 0. This is used in several branch instructions where only the first operand determines the branch target and the second ALU operand is not needed. For more information, see section 3 regarding the BLEZ, BGTZ, and BLTZ MIPS instructions. • Mux8 inputs '101', '110', and '111' are not connected to anything and therefore not used.
Mux9	2x1	CAUSE_SEL	Determines the input data value of the Cause register. • When CAUSE_SEL = 0, the data value 0 is passed to the Cause register. This indicates an arithmetic overflow processor exception. See section 5 for more detail. • When CAUSE_SEL = 1, the data value 1 is passed to the Cause register. This indicates an invalid instruction processor exception. See section 5 for more detail.
Mux10	8x1	PC_SEL	Determines the next value to be loaded into the Program Counter. • When PC_SEL = 000, the Program Counter input data is derived directly from the ALU output. This is used when incrementing the Program Counter during the Fetch cycle. • When PC_SEL = 001, the Program Counter data input is derived from the ALU Register. This is used when executing Branch instructions. • When PC_SEL = 010, the Program Counter data input is derived from the 26-bit address field and

			the 4 most significant bits of the Program Counter. The 26-bit address field is actually shifted left by 2 bits and that 28-bit value is concatenated with the 4 Program Counter bits to generate the 32-bit value. This value is written to the Program Counter when executing J-type instructions. • When PC_SEL = 011, the Program Counter data input is derived from the Reg1 which holds the value of the first ALU operand. This is used when executing jump register instructions where the content of the Register File specified by the Rs field of the instruction is the next Program Counter value. • When PC_SEL = 100, the Program Counter data input is set to location 0. This is used for Exception handling where location 0 contains the exception handling routine. See section 5 for more detail. • Mux10 inputs '101', '110', and '111' are not connected to anything and therefore not used.

4.2 ALU with Control Module

The Arithmetic Logic Unit (ALU) performs all of the major arithmetic and logical operations in the processor. Additionally, the ALU performs all of the shift operation with exception to a few 2-bit shift registers outside the ALU. The ALU operation is controlled by 4 control signals generated by the ALU controller. Figure 3 shows the ALU and ALU controller.

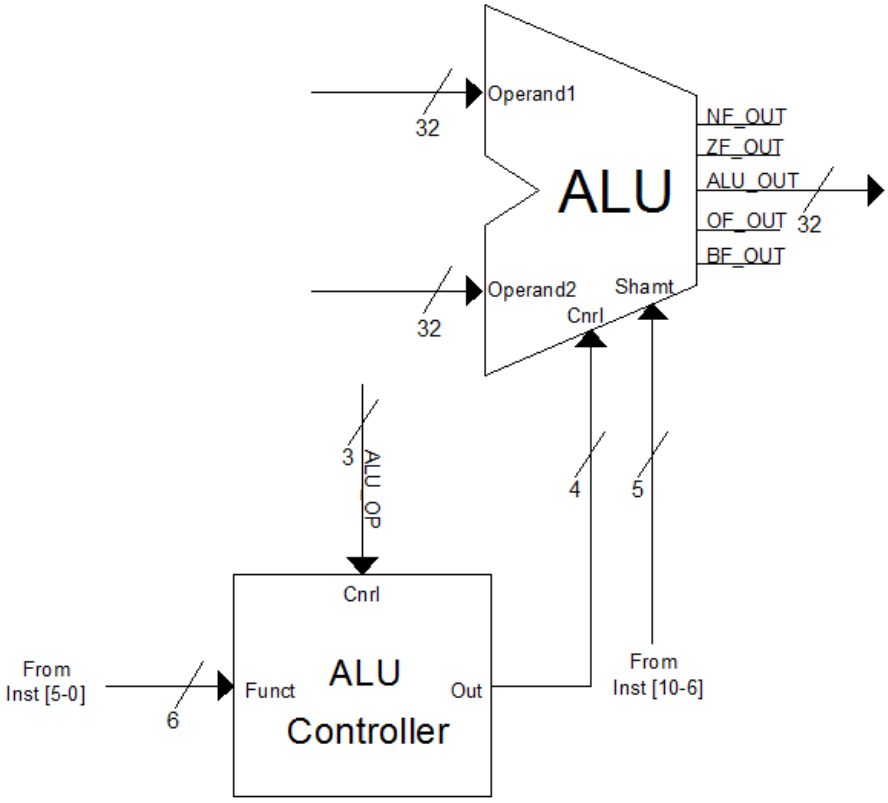


Figure 3. ALU with Controller

The 4 ALU input control signals dictate the ALU operation to be performed. This ALU design can perform a total of 14 operations. The ALU operations are shown in Table 2 along with their corresponding ALU control input combination.

Table 2. ALU Operations

ALU Control Input	ALU Operation
0000	AND
0001	OR
0010	add
0011	XOR
0100	NOR
0101	Set-if-less-than unsigned
0110	subtract
0111	Set-if-less-than
1000	Shift left logic
1001	Shift left logic by variable
1010	Shift right logic
1011	Shift right logic by variable
1100	Shift right arithmetic
1101	Shift right arith by variable

The ALU control signals are dependent on the 6-bit Function and 3-bit ALU_OP inputs of the ALU controller. The Function input is connected to the function field of an R-type instruction (bits [5:0]) while the ALU_OP inputs are derived from the sequence controller. The ALU Controller first decodes the ALU_OP signals to determine the instruction opcode. If the instruction is an R-type instruction, the ALU controller uses the Function input to determine the ALU operation (i.e. ALU Control output) to be performed. The instruction is not an R-type instruction, the Function input is not used and the ALU Controller determines the ALU operation based on the ALU_OP inputs alone. Table 3 shows the correlation between all ALU control signals and the resulting ALU operation for all of the supported instruction. Note that x's signify 'don't cares' and used when the Function field is not used.

Table 3. ALU Controller Logic

Opcode	ALU_OP	Operation	Function	ALU Operation	ALU Control
lw	000	Load word	XXXXXX	add	0010
lb	000	Load byte	XXXXXX	add	0010
lbu	000	Load byte unsigned	XXXXXX	add	0010
lh	000	Load half word	XXXXXX	add	0010
lhu	000	Load half word unsigned	XXXXXX	add	0010
sw	000	Store word	XXXXXX	add	0010
sb	000	Store byte	XXXXXX	add	0010
sh	000	Store half word	XXXXXX	add	0010
addi	000	Add immediate	XXXXXX	add	0010
addiu	000	Add immediate unsigned	XXXXXX	add	0010
andi	100	AND immediate	XXXXXX	AND	0000
ori	101	OR immediate	XXXXXX	OR	0001
xori	110	XOR immediate	XXXXXX	XOR	0011
j	000	jump	XXXXXX	add	0010
jal	000	jump and link	XXXXXX	add	0010
beq	001	Branch equal	XXXXXX	subtract	0110
bne	001	Branch not equal	XXXXXX	subtract	0110
Blez	001	Branch if less than or equal to zero	XXXXXX	subtract	0110
bgtz	001	Branch if greater than zero	XXXXXX	subtract	0110
bltz	001	Branch if less than zero	XXXXXX	subtract	0110
slti	011	Set-if-less-than immediate	XXXXXX	Set-if-less-than	0111
sltiu	011	Set-if-less-than immediate unsigned	XXXXXX	Set-if-less-than	0111
R-type	010	Jump register	001000	Don’t Care	XXXX
		Jump and link reg	001001	add	0010
		add	100000	add	0010
		addu	100001	add unsigned	0010
		sub	100010	subtract	0110
		subu	100011	subtract unsigned	0110
		Logical AND	100100	AND	0000
		Logical OR	100101	OR	0001
		Logical XOR	100110	XOR	0011
		Logical NOR	100111	NOR	0100
		sll	000000	Shift left logic	1000

		sllv	000100	Shift left logic by variable	1001
		srl	000010	Shift right logic	1010
		srlv	000110	Shift right logic by variable	1011
		sra	000011	Shift right arithmetic	1100
		srav	000111	Shift right arith by variable	1101
		Set-if-less-than unsigned	101001	Set-if-less-than unsigned	0101
		Set-if-less-than	101010	Set-if-less-than	0111

The ALU also has a 5-bit Shamt input, which is derived from bits [10:6] of the instruction being executed, and two 32-bit operands. The Shamt input determines the shift amount for each shift operation and is not used for any other operation.

The ALU has 5 output signals including a 32-bit ALU-OUT signal and 4 single bit outputs, namely the Negative Flag, Zero Flag, Overflow Flag, and Bad Instruction Flag. The ALU-OUT signal simply provides the ALU result when performing the specific ALU operation on the two operands.

The Negative Flag, or NF_OUT as shown in Figure 4, is set true when the ALU output is a negative number meaning the most significant bit is one. The Negative Flag is updated in both arithmetic and Boolean operations and is used as a trigger for various branch operations.

The Zero Flag, or ZF_OUT, is set true when the ALU output is zero. Like the Negative Flag, the Zero Flag is used to facilitate various branch instructions as described in section 3.

The Overflow Flag, or OF_OUT, is set true in the case where the ALU results produce an arithmetic overflow. An arithmetic overflow occurs when the ALU result is too large to be correctly captured by the 32 bit output. As an example, adding two positive numbers and getting a negative result is an indication of an overflow condition as the most significant bit was carried out of the 32 bit output. Likewise, adding two negative numbers and getting a positive result will result in an overflow.

The Bad Instruction Flag, or BF_OUT, is set when either the opcode or function code of the instruction being executed is invalid. Both Overflow and Bad Instruction flags are used to detect processor exceptions. See section 5 for more information on processor exceptions.

4.3 Program Counter with Control Circuitry

The Program Counter (PC) used in the project's processor design is actually a 32-bit register used to hold the address of the next instruction to be executed by the processor. The program counter is not an actual counter and cannot increment itself but is rather loaded with the content of the Cin input, at the rising edge of the system clock, whenever the PC_Load signal is set true by the Sequence Controller. If PC_Load remains low, the Program counter output does not change. The Program Counter uses a Combinational Block circuit to facilitate Branch operations. Both Program Counter and Combinational Block are shown in Figure 4.

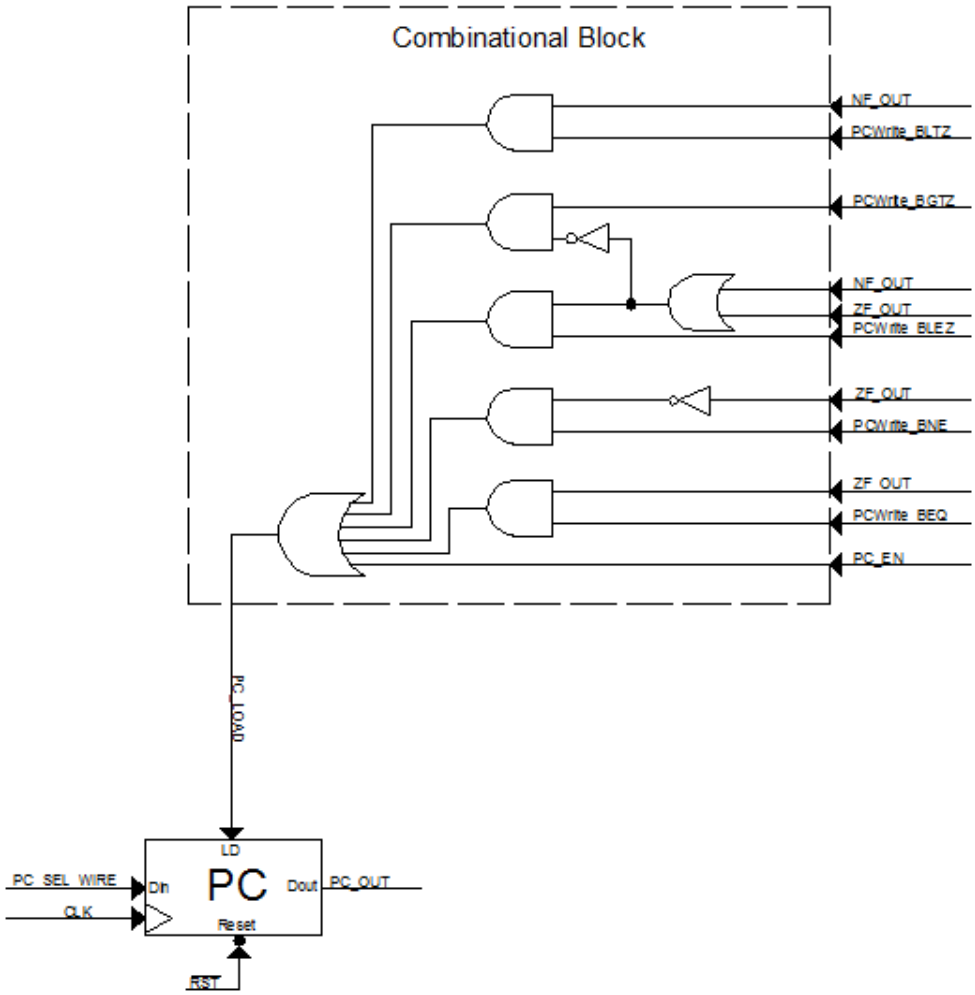


Figure 4. Program Counter with Combinational Control Logic

As shown in Figure 4, the PC_Load signal can be set true in several different scenarios. The most obvious scenario is when the PC_EN signal is set true by the Sequence Controller. This occurs when the Program Counter is loaded with the next instruction address, which is determined by the output of Mux10 (see section 4.1 for more detail). The following bullets show

all other scenarios in which the PC_LOAD signal is set true, enabling input loading of the Program Counter on the rising edge of clock.

- When both PCWrite_BEQ and ZF_OUT are both high. This is used in the ‘Branch if Equal’ operation (BEQ). The PCWrite_BEQ signal is set high by the Sequence Controller during the execution of the BEQ operation. The ZF_OUT signal is controlled by the ALU and is set high whenever the ALU output is zero. During the BEQ operation the ALU performs subtraction on its two operands. If the ALU operation results in a zero output, the branch condition is met and the Program Counter is loaded with the branch target.
- When PCWrite_BNE is set high and ZF_OUT is set low. This is used in the ‘Branch if Not Equal’ operation (BNE). The PCWrite_BNE signal is set high by the Sequence Controller during the execution of the BNE operation. During the BNE operation, the ALU performs subtraction on its two operands. If the ALU operation does not result in a zero output, the branch condition is met and the Program Counter is loaded with the branch target.
- When PCWrite_BLEZ and either NF_OUT or ZF_OUT are set high. This is used in the ‘Branch if less than or equal to zero’ operation (BLEZ). The PCWrite_BLEZ signal is set high by the Sequence Controller during the execution of the BLEZ operation. Similar to the ZF_OUT signal, NF_OUT is controlled by the ALU. NF_OUT is set high whenever the ALU output is a negative number. During the BLEZ operation, the first operand of the ALU is subtracted by zero, essentially being passed through the ALU. If the value is negative or zero, as determined by the corresponding ALU flags, the branch condition is met and the Program Counter is loaded with the branch target.
- When PCWrite_BGTZ is set high and both NF_OUT and ZF_OUT are set low. This is used in the ‘Branch if greater than zero’ operation (BGTZ). The PCWrite_BGTZ signal is set high by the Sequence Controller during the execution of the BGTZ operation. During the BGTZ operation, the first operand of the ALU is subtracted by zero. If the ALU result is neither negative nor zero, the branch condition is met and the Program Counter is loaded with the branch target.
- When both PCWrite_BLTZ and NF_OUT are set high. This is used in the ‘Branch if Less than zero’ operation (BLTZ). The PCWrite_BLTZ signal is set high by the Sequence Controller during the execution of the BLTZ operation. During the BLTZ operation, the first operand of the ALU is subtracted by zero. If the ALU result is a negative value, the branch condition is met and the Program Counter is loaded with the branch target.

4.4 Sequence Controller with State Register

The Sequence Controller, shown in Figure 5, is responsible for enabling and disabling all of the control signals in the processor design.

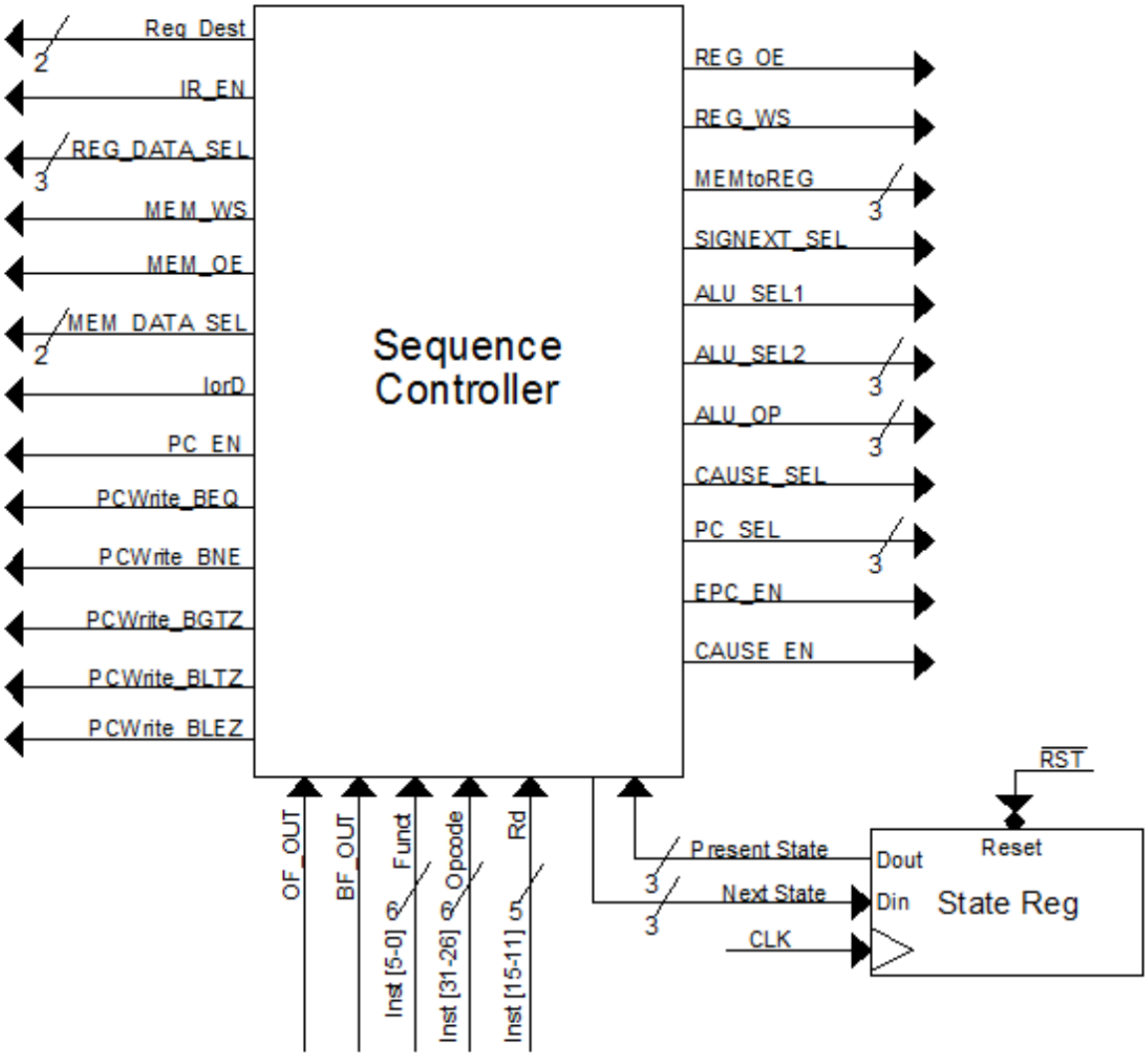


Figure 5. Sequence Controller with State Register

The Sequence controller has a total of 26 control signal outputs and 6 inputs. Table 4 lists all of the control signal outputs and describes their individual function.

Table 4. Sequence Controller Outputs

Control Signal	Function
PC_EN	Enables writing to the Program Counter
PC_SEL	Determines the data source for the Program Counter
IorD	Determines the address source for memory
MEM_DATA_SEL	Determines the source of data to be written to memory
MEM_OE	Enables reads from memory
MEM_WS	Enables writes to memory
REG_DATA_SEL	Determines the source of data to be written to the Instruction Register (IR)
IR_EN	Enables writes to the IR
Reg_Dest	Determines the address of a register in Register File to be written to
MEMtoREG	Determines the source of data to be written to the Register File
REG_OE	Enables reads from the Resister File
REG_WS	Enables writes to the register file
SIGNEXT_SEL	Selects between immediate data sign extended with 0's, in the case of an unsigned operation, and data sign extended with the MSB, which is used for signed operations
ALU_SEL1	Selects the ALU data source for the first operand
ALU_SEL2	Selects the ALU data source for the second operand
ALU_OP	Determines the operation to be performed by the ALU
EPC_EN	Enables writes to the EPC register
CAUSE_SEL	Determines whether the Cause register will be loaded with 0 or 1
CAUSE_EN	Enables writes to the Cause register
Next_State	Presents the next state machine state to the State Register.
PCWrite_BEQ	Enables 'branch if equal' operation
PCWrite_BNE	Enables 'branch if not equal' operation
PCWrite_BGTZ	Enables 'branch if greater than zero' operation
PCWrite_BLTZ	Enables 'branch if less than zero' operation
PCWrite_BLEZ	Enables 'branch if less than or equal to zero' operation

The Sequence Controller inputs are 'OF_OUT', 'BF_OUT', 'Opcode', 'Funct[0]', 'Rd', and 'Next State'. The Opcode input determines the instruction type once it is fetched and is vital in dictating the Sequence Controller outputs values throughout the execution of the instruction. 'OF_OUT' and 'BF_OUT' are direct outputs of the ALU and indicate when a processor exception has occurred. OF_OUT is set true when an arithmetic overflow has occurred. BF_OUT is set true when an invalid operation is detected. The 'Opcode', 'Funct[0]' and 'Rd' inputs are derived from the instruction being executed. The 'Funct[0]' is used to determine whether or not to ignore the overflow flag in order to avoid unwanted processor exception. Consider the following example. Both the Add and Subtract operations have unsigned counterparts. The operation and function codes are shown in Table 5:

Table 5. Function Field in Signed/Unsigned Operations

Operation	Function Code
add	100000
addu	100001
sub	100010
subu	100011

Performing an unsigned operation does not stop the ALU from setting the Overflow flag (OF_OUT) if an overflow occurs. This is not desired because in unsigned operations, the overflow is ignored and thus should not raise a processor exception. Going back to Table 5, it demonstrates that the LSB of the function field (Funct[0]) is always low for signed operations and high for unsigned operations. This allows us to use Funct[0] to determine whether to ignore false exceptions as they are detected by 'OF_OUT'.

The Rd input of the Sequence Controller is a 5-bit signal extracted directly from the instruction being decoded and is used to facilitate the MFC0 instruction. The MFC0 instruction is used to transfer the contents of either the EPC or Cause registers to the Register File. During the execution of the MFC0 instruction, if Rd = '01110', the content of the EPC register is saved; however, if Rd = '01101', the content of the Cause register is saved.

The Sequence Controller is designed as a finite state machine (FSM). The control signal outputs are dependent on the instruction being performed and the present state of the state machine. There are 5 phases in which an instruction is processed. The five phases are as follows:

- Fetch – The instruction is fetched from memory.
- Decode – The instruction is decoded in order to determine the necessary operations.
- Execute – The ALU operation is performed (if necessary).
- Memory Access – Data is read from or written to memory (if necessary).
- Write Back – Data is written to the Register File (if necessary).

Not every phase is utilized for every instruction. In fact, only the Load and Store operations utilize all 5 phases. The State Register is used to keep track of the phases being performed by the Sequence Controller. Each phase has its own 3-bit binary representation, hence the reason why the State Register is 3-bits wide. The State Register takes in the Next_State signal from the Sequence Controller and passes the value to the Present_State output at the rising edge of the system clock.

The Sequence Controller FSM diagram is shown in Figure 6. Note that Figure 6 shows the expanded FSM in more than 5 states. This is done for legibility. Figure 6 was generated using AutoCAD. See separate supporting documents for a more legible version.

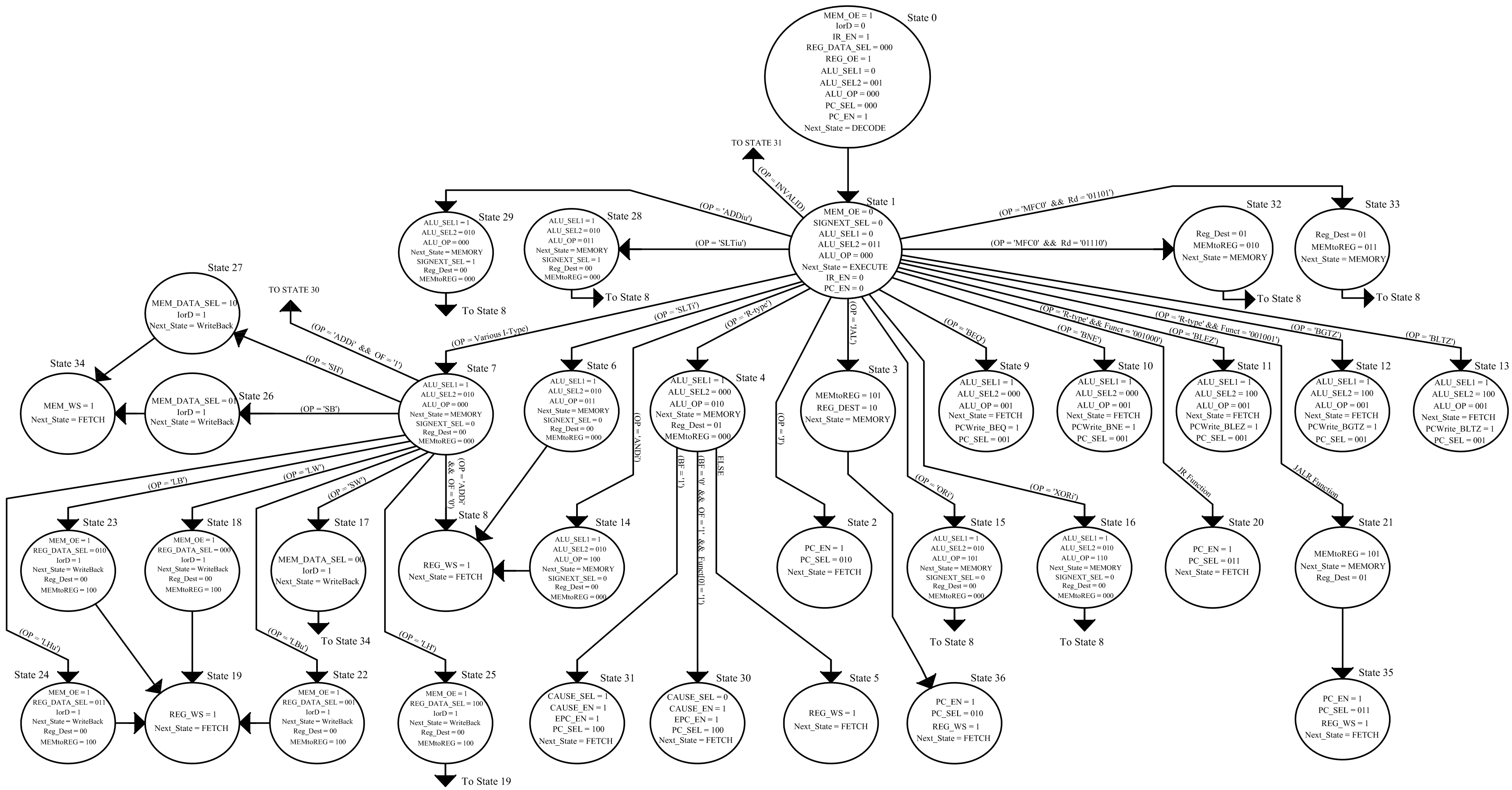


Figure 6. Sequence Controller FSM

Here is an explanation of each Sequence Controller FSM state corresponding with Figure 6:

- State 0 (Fetch Instruction) -
    - The instruction in the address location indicated by the current PC value is read from memory and set up to be written into the instruction register on the next clock cycle.
    - The current PC is incremented by 4 on the next clock cycle (Clock Cycle #2) and loaded into the PC register on clock cycle #3.
- State 1 (Decode) –
    - ALU registers are loaded with operands from Register File.
    - Branch target is calculated by the ALU.
    - Jump address is derived by shifting Instruction [25-0] left by two bits and concatenating with PC[31-28] to generate 32 bits address.
    - Sequence Generator decodes instruction.
- State 2 (Execute Jump) –
    - PC is loaded with the jump address derived in State 1.
- State 3 (Execute Jump and link) –
    - PC + 1 is brought to the Register File awaiting to be stored at the return address register $Ra
- State 4 (Execute R-type) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 and Reg2 content is derived directly from the instruction.
    - The ALU executes the operation determined by the function (bits [5-0]) and shamt (bits[10:6]) fields of the instruction as the ALU_OP signal is set to '10'.
- State 5 (Completion of R-type) –
    - ALU results are stored in the Register File at the location determined by the register destination field Rd (bits [15-11]) of the instruction.
- State 6 (Execution of SLTi) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the sign extended immediate value.
    - The ALU executes a "set-if-less-than" operation.
- State 7 (Execution of various I-type) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the sign extended immediate value.
    - The ALU executes an "add" operation.
- State 8 (Completion of various I-type and MFC0) –
    - ALU results are stored in the Register File at the location determined by the Rt field (bits [20-16]) of the instruction.
    - Memory stores for EPC and Cause registers occur in the case of MFC0.

- State 9 (Completion of BEQ) –
    - ALU executes the branch condition
    - If branch condition is met (i.e. Rs = Rt), the PC is loaded with the branch target calculated in State 1.
- State 10 (Completion of BNE) –
    - ALU executes the branch condition
    - If branch condition is met (i.e. Rs ≠ Rt), the PC is loaded with the branch target calculated in State 1.
- State 11 (Completion of BLEZ) –
    - ALU executes the branch condition
    - If branch condition is met (i.e. Rs ≤ 0), the PC is loaded with the branch target calculated in State 1.
- State 12 (Completion of BGTZ) –
    - ALU executes the branch condition
    - If branch condition is met (i.e. Rs > 0), the PC is loaded with the branch target calculated in State 1.
- State 13 (Completion of BLTZ) –
    - ALU executes the branch condition
    - If branch condition is met (i.e. Rs < 0), the PC is loaded with the branch target calculated in State 1.
- State 14 (Execution of ANDi) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the sign extended immediate value.
    - The ALU executes an "AND" operation.
- State 15 (Execution of ORi) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the sign extended immediate value.
    - The ALU executes an "OR" operation.
- State 16 (Execution of XORi) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the sign extended immediate value.
    - The ALU executes an "XOR" operation.
- State 17 (Memory write for SW) –
    - Contents of Reg2 are brought to RAM_data input to be stored at the address specified by the ALU output calculated in State 7.
- State 18 (Memory access for LW) –
    - Memory location specified by the ALU output is read and written into the instruction register.

- State 19 (Memory write for LW) –
    - Memory data from instruction register is loaded into the register file at the location specified by the Rt field of the extracted instruction.
- State 20 (Execute jump register (JR)) –
    - PC is loaded with register file content at location specified by Rs field of the instruction.
- State 21 (Execute jump and link register (JALR)) –
    - Register file content at location specified by Rs field of the instruction is extracted.
    - PC + 1 is brought to the Register File Write Data input to be stored in the return address register $Ra specified by the Rd field of the instruction.
- State 22 (Memory access for LBu) –
    - Memory location specified by the ALU output is read, and the unsigned least significant byte is written into the instruction register.
- State 23 (Memory access for LB) –
    - Memory location specified by the ALU output is read, and the least significant byte is written into the instruction register.
- State 24 (Memory access for LHu) –
    - Memory location specified by the ALU output is read, and the unsigned least significant haft word is written into the instruction register.
- State 25 (Memory access for LH) –
    - Memory location specified by the ALU output is read, and the least significant half word is written into the instruction register.
- State 26 (Memory write for SB) –
    - Least significant byte of Register 2 is sign extended with 0's and brought to the RAM Data input to be stored at the address specified by the ALU output calculated in State 7.
- State 27 (Memory write for SH) –
    - Least significant half word of Register 2 is sign extended with 0's and brought to the RAM Data input to be stored at the address specified by the ALU output calculated in State 7.
- State 28 (Execution of SLTiu) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the zero sign extended immediate value.
    - The ALU executes a "set-if-less-than" operation.
- State 29 (Execution of ADDiu) –
    - Operands from Reg1 and Reg2 are loaded into the ALU. Reg1 is derived directly from the instruction while Reg2 has the zero sign extended immediate value.
    - The ALU executes an "add" operation.

- State 30 (Exception handling for arithmetic overflow) –
    - The Cause register is loaded with ‘0’ to identify an arithmetic overflow exception.
    - The EPC register is loaded with the address of the next instruction (i.e. PC + 1).
    - The PC is loaded with the hardwired address of the exception handling routine, which is 0x0000.
- State 31 (Exception handling for invalid instruction) –
    - The Cause register is loaded with ‘1’ to identify an invalid instruction exception.
    - The EPC register is loaded with the address of the next instruction (i.e. PC + 1).
    - The PC is loaded with the hardwired address of the exception handling routine which is 0x0000.
- State 32 (Register write for EPC) –
    - Content of the EPC register are brought to the Register File location $14 to be stored in state 8.
- State 33 (Register write for Cause) –
    - Content of the Cause register are brought to the Register File location $13 to be stored in state 8.
- State 34 (Memory write Store operations) –
    - Content is written to RAM.
- State 35 (Memory write for jump and link register (JALR)) –
    - PC + 1 is stored in the Register File.
    - Calculated jump address is stored in the PC.
- State 36 (Memory write for jump and link (JAL)) –
    - PC + 1 is stored in the Register File.
    - PC is loaded with the jump address derived in State 1.

4.5 Memory and Register File

This processor design uses a Random Access Memory (RAM) and Register File as the two main storage devices. The RAM module is used to store the MIPS Instruction programs that get processed. In normal computer operations, the RAM is frequently updated with pages of instruction from a slower, larger storage device, such as a hard drive. This project does not go into detail about memory system architecture.

The RAM module, shown in Figure 7, has 4 inputs and 1 output. All data and address inputs are 32-bits wide in order to support the rest of the processor architecture. The RAM depth can be any arbitrary size up to 2^32 locations as the Address bus is 32-bits wide. There are also two single bit control signals, namely, MEM_OE and MEM_WS. Both control signals are controlled and generated by the Sequence Controller. On the rising edge of MEM_OE, the Register File content at the location specified by the Address (Addr) bus is placed on the output bus. The RAM module writes the content of the Data input into the Register File at the location specified by the Address bus on the rising edge of MEM_WS.

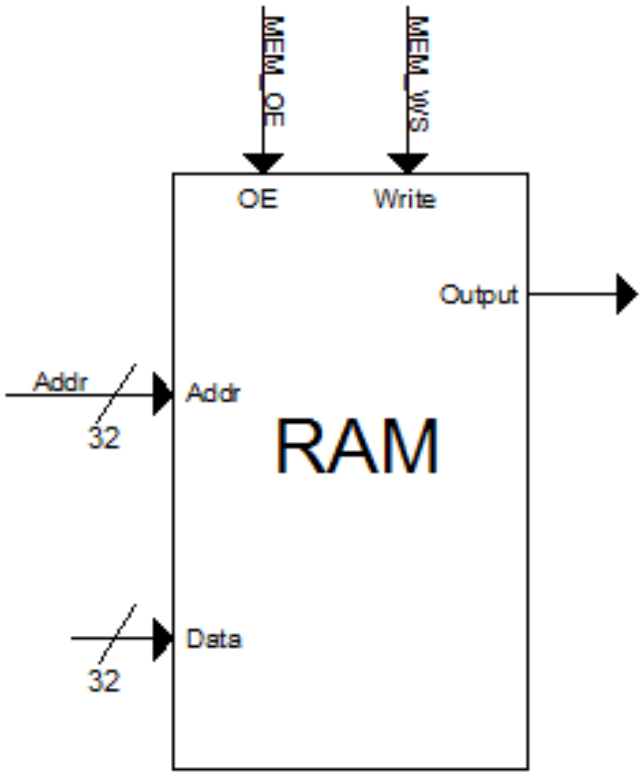


Figure 7. RAM Module

The Register File is a set of 32 registers, which are 32-bits wide used as the general purpose registers. Each register is mapped a certain way and has certain functionality. The Register File Mapping and register usage descriptions are shown in Table 6.

Table 6. Register File Mapping

Name	Reg #	Usage
$zero	0	Constant 0 value
$v0 - $v1	2-3	Values for results and expression evaluation
$a0 - $a3	4-7	arguments
$t0 - $t7	8-15	Temporary registers
$s0 - $s7	16-23	Saved registers
$t8 - $t9	24-25	Temporary registers
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

The Register File is similar to the RAM module, only it has dual ports while the RAM module only has a single port. Additionally, in reality the Register File is made from flip flops while RAM is made from DRAM or SRAM cells. This means that the Register File consumes more power and will take up more area on a real chip but will also be much faster than RAM. Using registers for faster operation is one of the key features of a MIPS processor. Since this project's processor will be implemented on an FPGA, the difference in composite material is negligible.

The Register File, shown in Figure 8, has a total of 6 inputs and 2 outputs. All data inputs and outputs are 32-bits wide while the Address inputs are 5-bits wide. There are also two single bit control signals, namely, REG_OE and REG_WS. At the rising edge of REG_OE, the content of the Register File at the location specified by the Reg1 and Reg2 inputs are placed on the Reg1_Data and Reg2_data outputs, respectively. The Reg1 and Reg2 inputs are derived directly from the Rs (Inst[25-21]) and Rt (Inst[20-16]) fields of the instruction being executed. At the rising edge of REG_WS, the content of the Write_Data input is written to the Register File at the location specified by the Write_Reg input.

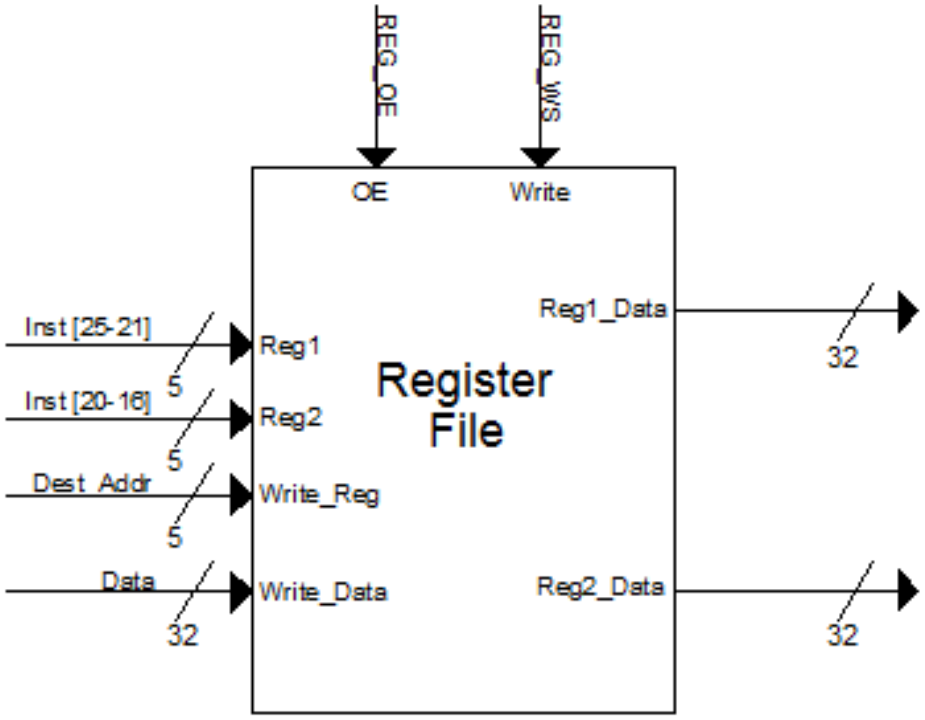


Figure 8. Register File

4.6 Registers and Miscellaneous Components

The processor design contains a total of 8 synchronous registers. Each register is 32-bits wide with exception to the 3-bit State Register, which is used to synchronize the Sequence Controller with the rest of the processor architecture. The registers are the only components in the design synchronous to the system clock. Table 7 lists all registers and describes their functionality.

Table 7. Processor Registers

Register Name	Functionality
PC	Holds the memory location of instructions being executed
IR	Holds the Instruction being executed after it is fetched from memory
Reg1	Holds the register content from the Register File location specified by the Rs field of the Instruction
Reg2	Holds the register content from the Register File location specified by the Rt field of the Instruction
ALU Register	Holds the ALU output content
EPC	Holds the current PC value in the case of a processor exception
Cause	Holds the exception cause value
State Reg	Holds and manages the Sequence Controller FSM states

The processor design contains Sign Extend, Shift, and Concatenate components. The Sign Extend components are used to adapt signals which are smaller than 32-bits into a 32-bit port. There are 4 different types of Sign Extend components. The "*Sign Extend Byte"* component takes in 8-bits and sign extends the most significant bit (MSB) to create a 32-bit output. The "*Sign Extend Byte with 0's"* component also takes in 8-bits and outputs a 32-bit signal. The difference between the two components discussed above is that the "*Sign Extend Byte with 0's"* component sign extends the 8 input bits with 0's in order to represent an unsigned number which is used in unsigned operations. The "*Sign Extend Half Word"* and "*Sign Extend Half Word with 0's"* components work similarly to the Sign Extend components discussed above only they take in 16-bits and output 32-bits.

Shift components are used to calculate branch and jump target addresses. Shift components, or "*Shift Left 2*" components to be exact, operate by shifting the input two bits to the left and filling the least significant bit (LSB) locations with 0's. This essentially multiplies the input by 4. The only other component in the processor capable of shifting is the ALU but it is not practical to have the ALU busy with such small tasks.

The Concatenate component is used strictly to support calculating Jump addresses. During the Jump and Link (JAL) operation the processor jumps to the memory address location specified by concatenating the shifted Address field of the instruction with the 4 most significant bits of the Program Counter. This is the sole function of the Concatenate component.

5 PROCESSOR EXCEPTIONS

Exceptions are unexpected events from within the processor that interrupt the normal executions of an instruction. When an exception occurs, the processor performs a set of instructions outside of the program or set of instructions being executed in response to the exception.

There are two kinds of events that may trigger an exception in this project's processor design. The two events are *arithmetic overflow* and *undefined instruction*. An *arithmetic overflow* occurs when the ALU 'OF_OUT' flag is set true as a result of an overflow detected during the execution of a add, addi, or sub instruction. Note that the overflow flag is ignored when the instruction indicates an unsigned operation such as addu, addiu, or subu. An *undefined instruction* occurs when the instruction being executed has an invalid opcode or function code in the case of R-type instructions.

There are two registers used in the processor that help facilitate exception handling. The two registers are EPC and the Cause registers. When an exception occurs, the current Program Counter is saved in the EPC register and the Cause register is loaded with either a '1' or '0' depending on the exception type. The Cause register is 32-bits wide, but only one bit is needed in this design. This is done to support regularity as all registers in the processor are 32-bits wide. The Cause register being set to 0 is an indication that the current exception type is an *arithmetic overflow*, while a value of '1' indicates an *undefined instruction* exception.

Both the Cause and EPC registers need to be actively available to the processor. In order to do so, they need to be transferred to the Register File. The MFC0 instruction allows the processor to transfer the content of each exception register to the Register File. See section 3 for more detail.

When an exception occurs, the following events are executed:

- The EPC register is loaded with the Program Counter value.
- The Cause register is loaded with the correct value to specify the exception type.
- The processor jumps to a certain address that checks the cause of the exception and branches to the correct exception handling routine for that exception. For this design, hex 0x0000 was selected as the jump address (Bistriceabu Lab 7).

The project's processor uses *polled exception handling*, which means that every exception, two in total, has its own exception handling routine. For the purpose of this study, exception handling routines or the code that selects the routines located in memory location 0x0000 will not be looked into.

6 COMPONENT LEVEL DESIGN AND TEST

To implement the design, Verilog HDL and the Synopsys VCS Compiler/Simulator were used. Verilog was chosen over VHDL because of the flexibility of the language and because the Synopsys tool was available for use through university resources. As for this design approach, Verilog behavioral modeling was first used to code each individual component of the processor and tested each component by generating a Verilog test bench program. By doing so, there was an assurance that each component was working as expected before generating the top level design. Due to previous experience, it was noted that troubleshooting at the top design level is much harder than working out a few issues at the component level. Once each component was designed and tested, the top level processor design was created using structural Verilog as described in section 7.

Section 6 contains the Verilog code, test bench program, and test results for each component of the RISC Processor. Note that some of the component code programs span multiple figures as they are too large to fit in one figure.

6.1 ALU Design and Test

Design Code:

The ALU Verilog design code is shown in Figures 9 through 11. Local parameters were used on each ALU operation to make code more legible. Both signed and unsigned variables were used to cover all required operations. Subtraction was performed similar to addition only with the 2's compliment taken on input B prior to execution. This made it easier to implement the overflow flag function in subtraction operations.

```
1  /*************************************************************************
2  *** Filename: alu.v Created by Orr Chakon                              ***
3  *************************************************************************/
4
5  `timescale 1ns/1ns
6  module alu(A, B, CNRL, SHAMT, ALU_OUT, NF_OUT, ZF_OUT, OF_OUT, BF_OUT);
7
8  //OPCODE Parameters
9  localparam  CNRL_AND = 4'b0000,
10             CNRL_OR = 4'b0001,
11             CNRL_ADD = 4'b0010,
12             CNRL_XOR = 4'b0011,
13             CNRL_NOR = 4'b0100,
14             CNRL_SLTU = 4'b0101,
15             CNRL_SUB = 4'b0110,
16             CNRL_SLT = 4'b0111,
17             CNRL_SLL = 4'b1000,
18             CNRL_SLL_VAR = 4'b1001,
19             CNRL_SRL = 4'b1010,
20             CNRL_SRL_VAR = 4'b1011,
21             CNRL_SRA = 4'b1100,
22             CNRL_SRA_VAR = 4'b1101;
23 output signed [31:0] ALU_OUT;              //alu output
24 output NF_OUT, ZF_OUT, OF_OUT, BF_OUT;     //alu flag outputs
25 input signed [31:0] A;                     //operands
26 input signed [31:0] B;
27 input [3:0] CNRL;                         //control input
28 input [4:0] SHAMT;                        //shift amount input
29
30 //register declarations
31 reg signed [31:0] ALU_OUT;
32 reg NF_OUT, ZF_OUT, OF_OUT, BF_OUT;
33 reg [31:0] B_TEMP;
34 wire [31:0] unsigned_A;     //unsigned input A
35 wire [31:0] unsigned_B;     //unsigned input B
36
37 assign   unsigned_A = A;
38 assign   unsigned_B = B;
39
40 always@ (A, B, CNRL, SHAMT) begin
41
42         case (CNRL)
43
44             CNRL_AND: begin
45                 ALU_OUT = A & B;    //and
46                 BF_OUT = 1'b0;      //disable OPCODE invalid flag
47                 OF_OUT = 1'b0;      //disable overflow flag
48                 end
49
50             CNRL_OR:    begin
51                 ALU_OUT = A | B;    //or
52                 BF_OUT = 1'b0;      //disable OPCODE invalid flag
53                 OF_OUT = 1'b0;      //disable overflow flag
```

Verilog file | length : 4,777 lines : 153 | Ln : 53 Col : 60 Sel : 0 | 0 | Windows (CR L

Figure 9. ALU Design Code Part 1

```
49
50          CNRL_OR:    begin
51              ALU_OUT = A | B;    //or
52              BF_OUT = 1'b0;      //disable OPCODE invalid flag
53              OF_OUT = 1'b0;      //disable overflow flag
54              end
55
56          CNRL_ADD: begin
57              ALU_OUT = A + B;    //add
58              BF_OUT = 1'b0;      //disable OPCODE invalid flag
59              //produce overflow flag
60              if ((A[31] && B[31]) && !ALU_OUT[31]) OF_OUT = 1'b1;           //set overflow flag if inputs are positive and output is negative
61              else if ((!A[31] && !B[31]) && ALU_OUT[31]) OF_OUT = 1'b1;     //set overflow flag if inputs are negative and output is positive
62              else OF_OUT = 1'b0;  //disable overflow flag
63              end
64
65          CNRL_XOR: begin
66              ALU_OUT = A ^ B;    //xor
67              BF_OUT = 1'b0;      //disable OPCODE invalid flag
68              OF_OUT = 1'b0;      //disable overflow flag
69              end
70
71          CNRL_NOR: begin
72              ALU_OUT = ~(A | B); //nor
73              BF_OUT = 1'b0;      //disable OPCODE invalid flag
74              OF_OUT = 1'b0;      //disable overflow flag
75              end
76
77          CNRL_SLTU: begin
78              if (unsigned_A < unsigned_B) ALU_OUT = 32'b1;   //Set if less than (unsigned)
79              else ALU_OUT = 32'b0;
80              BF_OUT = 1'b0;      //disable OPCODE invalid flag
81              OF_OUT = 1'b0;      //disable overflow flag
82              end
83
84          CNRL_SUB:   begin           //subtraction
85              BF_OUT = 1'b0;          //disable OPCODE invalid flag
86              B_TEMP = ~B + 1;        //2's compliment of input B used to facilitate subtraction with overflow detection
87              ALU_OUT = A + B_TEMP;   //same as A - B
88              //produce overflow flag
89              if ((A[31] && B_TEMP[31]) && !ALU_OUT[31]) OF_OUT = 1'b1;
90              else if ((!A[31] && !B_TEMP[31]) && ALU_OUT[31]) OF_OUT = 1'b1;
91              else OF_OUT = 1'b0;
92              end
93
94          CNRL_SLT: begin
95              if (A < B) ALU_OUT = 32'b1; //Set if less than
96              else ALU_OUT = 32'b0;
97              BF_OUT = 1'b0;      //disable OPCODE invalid flag
98              OF_OUT = 1'b0;      //disable overflow flag
99              end
100
101         CNRL_SLL: begin
```

erilog file | length : 4,777 lines : 153 | Ln : 53 Col : 60 Sel : 0 | 0 | Windows (CR LF

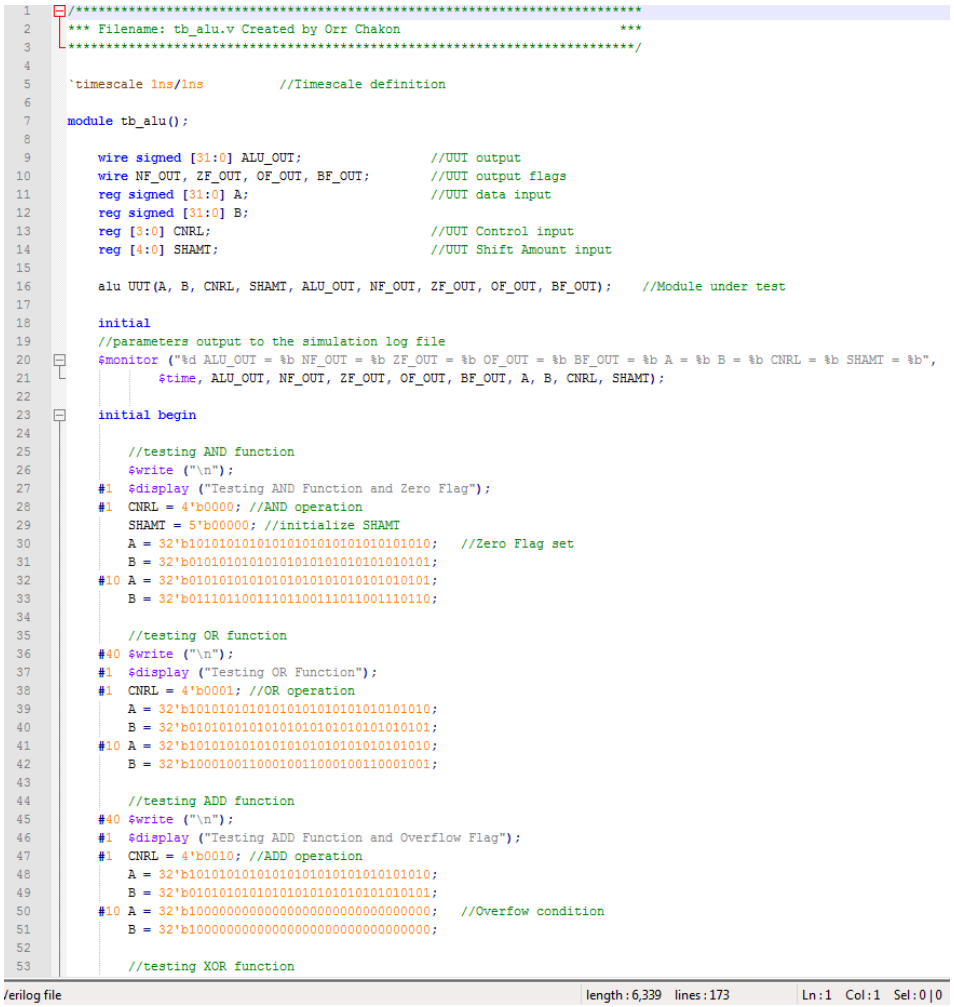
Figure 10. ALU Design Code Part 2

```
            CNRL_SLL: begin
                ALU_OUT = B << SHAMT;   //Shift left logic
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            CNRL_SLL_VAR: begin
                ALU_OUT = B << A;   //Shift left logic by variable
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            CNRL_SRL: begin
                ALU_OUT = B >> SHAMT;  //Shift right logic
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            CNRL_SRL_VAR: begin
                ALU_OUT = B >> A;   //Shift right logic by variable
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            CNRL_SRA: begin
                ALU_OUT = B >>> SHAMT;  //shift right arithmetic
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            CNRL_SRA_VAR: begin
                ALU_OUT = B >>> A;  //shift right arithmetic by variable
                BF_OUT = 1'b0;      //disable OPCODE invalid flag
                OF_OUT = 1'b0;      //disable overflow flag
                end

            default:
                begin   BF_OUT = 1'b1;  //set flag if OPCODE in invalid
                end
        endcase

    end

    always@ (ALU_OUT) begin
        if (!ALU_OUT) ZF_OUT = 1'b1;    //produce zero flag in all operations
        else ZF_OUT = 1'b0;

        if (ALU_OUT[31]) NF_OUT = 1'b1; //produce negative flag in all operations
        else NF_OUT = 1'b0;
    end

    endmodule

Verilog file   length : 4,777   lines : 153   Ln : 98   Col : 60   Sel : 0 | 0   Windows (CR
```

Figure 11. ALU Design Code Part 3

Test Bench:

The Test Bench program for the ALU design is shown in Figures 12 through 15. In order to test the ALU operation, the test script was designed to cover all functions of the ALU component, including ALU output and flag generation.



```
/*********************************************************************
*** Filename: tb_alu.v Created by Orr Chakon                      ***
*********************************************************************/

`timescale 1ns/1ns          //Timescale definition

module tb_alu();

    wire signed [31:0] ALU_OUT;                 //UUT output
    wire NF_OUT, ZF_OUT, OF_OUT, BF_OUT;        //UUT output flags
    reg signed [31:0] A;                        //UUT data input
    reg signed [31:0] B;
    reg [3:0] CNRL;                             //UUT Control input
    reg [4:0] SHAMT;                            //UUT Shift Amount input

    alu UUT(A, B, CNRL, SHAMT, ALU_OUT, NF_OUT, ZF_OUT, OF_OUT, BF_OUT);    //Module under test

    initial
    //parameters output to the simulation log file
    $monitor ("%d ALU_OUT = %b NF_OUT = %b ZF_OUT = %b OF_OUT = %b BF_OUT = %b A = %b B = %b CNRL = %b SHAMT = %b",
            $time, ALU_OUT, NF_OUT, ZF_OUT, OF_OUT, BF_OUT, A, B, CNRL, SHAMT);

    initial begin

        //testing AND function
        $write ("\n");
    #1  $display ("Testing AND Function and Zero Flag");
    #1  CNRL = 4'b0000; //AND operation
        SHAMT = 5'b00000; //initialize SHAMT
        A = 32'b10101010101010101010101010101010;   //Zero Flag set
        B = 32'b01010101010101010101010101010101;
    #10 A = 32'b01010101010101010101010101010101;
        B = 32'b01110111001110110011101100111011O;

        //testing OR function
    #40 $write ("\n");
    #1  $display ("Testing OR Function");
    #1  CNRL = 4'b0001; //OR operation
        A = 32'b10101010101010101010101010101010;
        B = 32'b01010101010101010101010101010101;
    #10 A = 32'b10101010101010101010101010101010;
        B = 32'b10001001100010011000100110001001;

        //testing ADD function
    #40 $write ("\n");
    #1  $display ("Testing ADD Function and Overflow Flag");
    #1  CNRL = 4'b0010; //ADD operation
        A = 32'b10101010101010101010101010101010;
        B = 32'b01010101010101010101010101010101;
    #10 A = 32'b10000000000000000000000000000000;   //Overfow condition
        B = 32'b10000000000000000000000000000000;

        //testing XOR function
```

/erilog file length : 6,339 lines : 173 Ln : 1 Col : 1 Sel : 0 | 0

Figure 12. ALU Test Bench Part 1

```
53          //testing XOR function
54        #40 $write ("\n");
55        #1  $display ("Testing XOR Function");
56        #1  CNRL = 4'b0011; //XOR operation
57            A = 32'b10101010101010101010101010101010;
58            B = 32'b01010101010101010101010101010101;
59        #10 A = 32'b11010011110100111101001111010011;
60            B = 32'b10001001100010011000100110001001;
61
62            //testing NOR function
63        #40 $write ("\n");
64        #1  $display ("Testing NOR Function");
65        #1  CNRL = 4'b0100; //NOR operation
66            A = 32'b00000000000000000000000000000000;
67            B = 32'b01010101010101010101010101010101;
68        #10 A = 32'b11111111111111111111111111111111;
69            B = 32'b10101010101010101010101010101010;
70
71            //testing SLTU function
72        #40 $write ("\n");
73        #1  $display ("Testing SLTU Function");
74        #1  CNRL = 4'b0101; //SLTU operation
75            A = 32'b00000000000000000000000000000000;
76            B = 32'b11111111111111111111111111111111; //verifiying operation is unsigned
77        #10 A = 32'b11111111111111111111111111111111;
78            B = 32'b10101010101010101010101010101010;
79
80            //testing SUB function
81        #40 $write ("\n");
82        #1  $display ("Testing SUB Function and Negative Flag");
83        #1  CNRL = 4'b0110; //SUB operation
84            A = 32'b00000000000000000000000000001001;
85            B = 32'b00000000000000000000000000100000; //Negative Flag set
86        #10 A = 32'b11111111111111111111111111111111;
87            B = 32'b00000000000000000000000000000001; //Overflow condition
88
89            //testing SLT function
```

Figure 13. ALU Test Bench Part 2

```
89          //testing SLT function
90      #40 $write ("\n");
91      #1  $display ("Testing SLT Function");
92      #1  CNRL = 4'b0111; //SLT operation
93          A = 32'b00000000000000000000000000000000;
94          B = 32'b11111111111111111111111111111111; //verifiying operation is signed
95      #10 A = 32'b00000000000000000000000000000001;
96          B = 32'b00000000000000000000000001010101;
97
98          //testing SLL function
99      #40 $write ("\n");
100     #1  $display ("Testing SLL Function");
101     #1  CNRL = 4'b1000; //SLL operation
102         SHAMT = 5'b00011; //Shift left by 3
103         A = 32'b00000000000000000000000000000000;
104         B = 32'b11111111111111111111111111111111;
105     #10 SHAMT = 5'b10000; //Shift left by 16
106         A = 32'b11111111111111111111111111111111;
107         B = 32'b11111111111111111111111111111111;
108
109         //testing SLL by Variable function
110     #40 $write ("\n");
111     #1  $display ("Testing SLL by Variable Function");
112     #1  CNRL = 4'b1001; //SLL_VAR operation
113         SHAMT = 5'b00111; //Shift input ignored
114         A = 32'b00000000000000000000000000000101;   //shift left by 5
115         B = 32'b11111111111111111111111111111111;
116     #10 SHAMT = 5'b11111; //Shift input ignored
117         A = 32'b00000000000000000000000000010100;   //shift left by 20
118         B = 32'b11111111111111111111111111111111;
119
120             //testing SRL function
121     #40 $write ("\n");
122     #1  $display ("Testing SRL Function");
123     #1  CNRL = 4'b1010; //SRL operation
124         SHAMT = 5'b01001; //Shift right by 9
125         A = 32'b00000000000000000000000000000000;
126         B = 32'b11111111111111111111111111111111;
127     #10 SHAMT = 5'b11011; //Shift right by 27
128         A = 32'b11111111111111111111111111111111;
129         B = 32'b11111111111111111111111111111111;
130
131             //testing SRL by variable function
```

Figure 14. ALU Test Bench Part 3

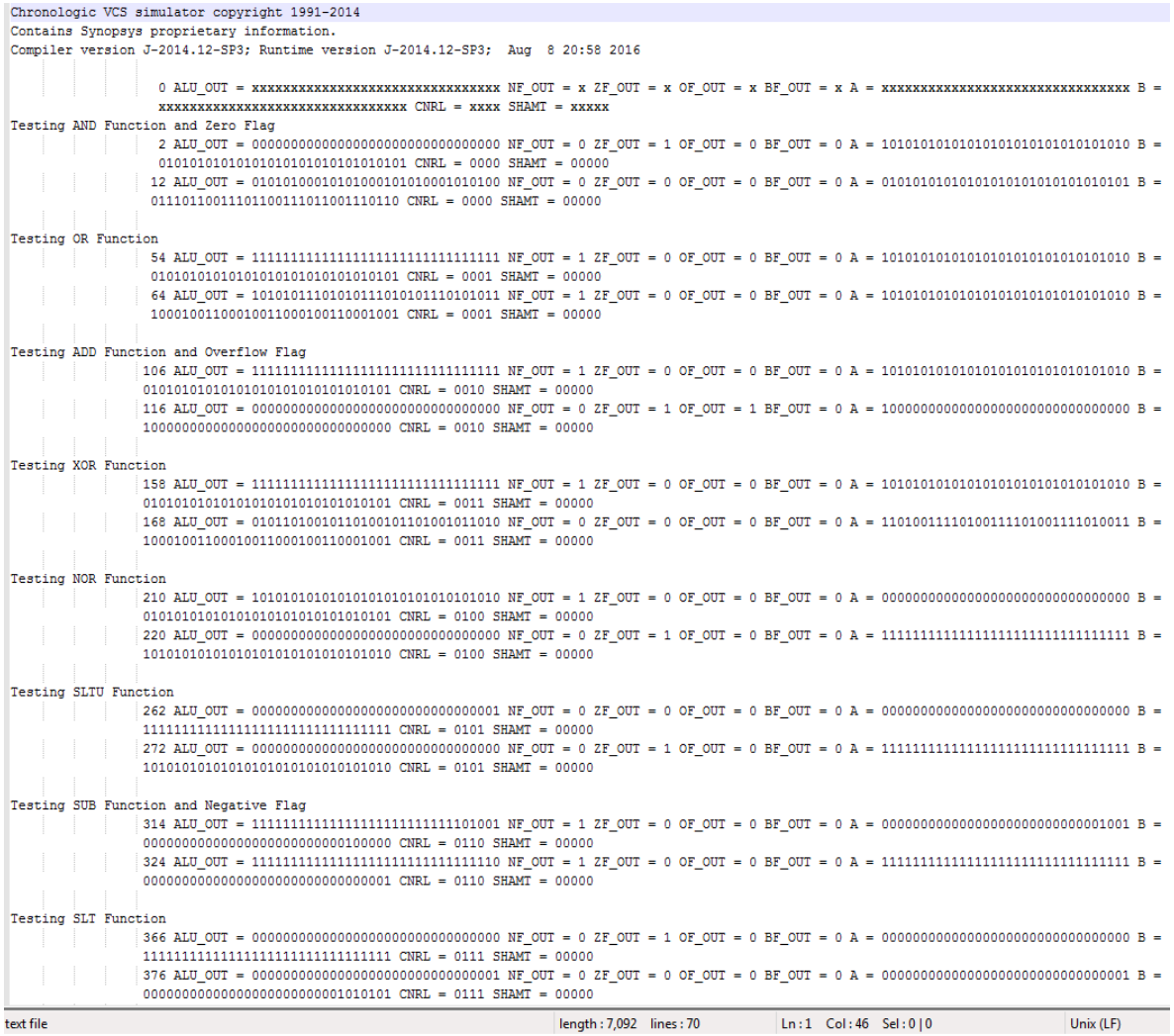
```
131             //testing SRL by variable function
132     #40 $write ("\n");
133     #1  $display ("Testing SRL by Variable Function");
134     #1  CNRL = 4'b1011; //SRL_VAR operation
135         SHAMT = 5'b10111; //Shift input ignored
136         A = 32'b00000000000000000000000000000110;   //shift right by 6
137         B = 32'b11111111111111111111111111111111;
138     #10 SHAMT = 5'b11101; //Shift input ignored
139         A = 32'b00000000000000000000000000010010;   //shift right by 18
140         B = 32'b11111111111111111111111111111111;
141
142                 //testing SRA function
143     #40 $write ("\n");
144     #1  $display ("Testing SRA Function");
145     #1  CNRL = 4'b1100; //SRA operation
146         SHAMT = 5'b01010; //Shift right by 10
147         A = 32'b00000000000000000000000000000000;
148         B = 32'b10000000000000000000000000000000;
149     #10 SHAMT = 5'b01000; //Shift right by 8
150         A = 32'b11111111111111111111111111111111;
151         B = 32'b01111111111111111111111111111111;
152
153                     //testing SRA by variable function
154     #40 $write ("\n");
155     #1  $display ("Testing SRA by Variable Function");
156     #1  CNRL = 4'b1101; //SRA_VAR operation
157         SHAMT = 5'b01010; //Shift input ignored
158         A = 32'b00000000000000000000000000000111;   //shift right by 7
159         B = 32'b10000000000000000000000000000000;
160     #10 SHAMT = 5'b01000; //Shift input ignored
161         A = 32'b00000000000000000000000000001011;   //shift right by 11
162         B = 32'b01111111111111111111111111111111;
163
164                         //testing invalid opcode (BF_OUT)
165     #40 $write ("\n");
166     #1  $display ("Testing Invalid Opcode AKA BF_OUT");
167     #1  CNRL = 4'b1111; //invalid opcode
168
169
170     #100    $finish;            //Terminate simulation after 100ns
171     end                         //End of simulation
172 endmodule
173
```

Verilog file | length : 6,339 lines : 173 | Ln : 122 Col : 43 Sel : 0 | 0

Figure 15. ALU Test Bench Part 4

Test Results:

Simulation results for the ALU component are shown in Figures 16 and 17.



```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Aug  8 20:58 2016

                   0 ALU_OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx NF_OUT = x ZF_OUT = x OF_OUT = x BF_OUT = x A = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx B =
                   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx CNRL = xxxx SHAMT = xxxxx
Testing AND Function and Zero Flag
                   2 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 0 BF_OUT = 0 A = 10101010101010101010101010101010 B =
                   01010101010101010101010101010101 CNRL = 0000 SHAMT = 00000
                  12 ALU_OUT = 01010100010101000101010001010100 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 01010101010101010101010101010101 B =
                  01110110011101100111011001110110 CNRL = 0000 SHAMT = 00000

Testing OR Function
                  54 ALU_OUT = 11111111111111111111111111111111 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 10101010101010101010101010101010 B =
                   01010101010101010101010101010101 CNRL = 0001 SHAMT = 00000
                  64 ALU_OUT = 10101011101010111010101110101011 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 10101010101010101010101010101010 B =
                  10001001100010011000100110001001 CNRL = 0001 SHAMT = 00000

Testing ADD Function and Overflow Flag
                 106 ALU_OUT = 11111111111111111111111111111111 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 10101010101010101010101010101010 B =
                 01010101010101010101010101010101 CNRL = 0010 SHAMT = 00000
                 116 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 1 BF_OUT = 0 A = 10000000000000000000000000000000 B =
                 10000000000000000000000000000000 CNRL = 0010 SHAMT = 00000

Testing XOR Function
                 158 ALU_OUT = 11111111111111111111111111111111 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 10101010101010101010101010101010 B =
                 01010101010101010101010101010101 CNRL = 0011 SHAMT = 00000
                 168 ALU_OUT = 01011010010110100101101001011010 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 11010011110100111101001111010011 B =
                 10001001100010011000100110001001 CNRL = 0011 SHAMT = 00000

Testing NOR Function
                 210 ALU_OUT = 10101010101010101010101010101010 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                 01010101010101010101010101010101 CNRL = 0100 SHAMT = 00000
                 220 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                 10101010101010101010101010101010 CNRL = 0100 SHAMT = 00000

Testing SLTU Function
                 262 ALU_OUT = 00000000000000000000000000000001 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                 11111111111111111111111111111111 CNRL = 0101 SHAMT = 00000
                 272 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                 10101010101010101010101010101010 CNRL = 0101 SHAMT = 00000

Testing SUB Function and Negative Flag
                 314 ALU_OUT = 11111111111111111111111111101001 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000001001 B =
                 00000000000000000000000000100000 CNRL = 0110 SHAMT = 00000
                 324 ALU_OUT = 11111111111111111111111111111110 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                 00000000000000000000000000000001 CNRL = 0110 SHAMT = 00000

Testing SLT Function
                 366 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                 11111111111111111111111111111111 CNRL = 0111 SHAMT = 00000
                 376 ALU_OUT = 00000000000000000000000000000001 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000001 B =
                 00000000000000000000000001010101 CNRL = 0111 SHAMT = 00000
```

text file | length : 7,092 | lines : 70 | Ln : 1 | Col : 46 | Sel : 0 | 0 | Unix (LF)

Figure 16. ALU Simulation Results Part 1

```
Testing SLT Function
                366 ALU_OUT = 00000000000000000000000000000000 NF_OUT = 0 ZF_OUT = 1 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                11111111111111111111111111111111 CNRL = 0111 SHAMT = 00000
                376 ALU_OUT = 00000000000000000000000000000001 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000001 B =
                00000000000000000000000001010101 CNRL = 0111 SHAMT = 00000

Testing SLL Function
                418 ALU_OUT = 11111111111111111111111111111000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                11111111111111111111111111111111 CNRL = 1000 SHAMT = 00011
                428 ALU_OUT = 11111111111111110000000000000000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                11111111111111111111111111111111 CNRL = 1000 SHAMT = 10000

Testing SLL by Variable Function
                470 ALU_OUT = 11111111111111111111111111100000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000101 B =
                11111111111111111111111111111111 CNRL = 1001 SHAMT = 00111
                480 ALU_OUT = 11111111111100000000000000000000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000010100 B =
                11111111111111111111111111111111 CNRL = 1001 SHAMT = 11111

Testing SRL Function
                522 ALU_OUT = 00000000011111111111111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                11111111111111111111111111111111 CNRL = 1010 SHAMT = 01001
                532 ALU_OUT = 00000000000000000000000000011111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                11111111111111111111111111111111 CNRL = 1010 SHAMT = 11011

Testing SRL by Variable Function
                574 ALU_OUT = 00000011111111111111111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000110 B =
                11111111111111111111111111111111 CNRL = 1011 SHAMT = 10111
                584 ALU_OUT = 00000000000000000011111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000010010 B =
                11111111111111111111111111111111 CNRL = 1011 SHAMT = 11101

Testing SRA Function
                626 ALU_OUT = 11111111111000000000000000000000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000000 B =
                10000000000000000000000000000000 CNRL = 1100 SHAMT = 01010
                636 ALU_OUT = 00000000011111111111111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 11111111111111111111111111111111 B =
                01111111111111111111111111111111 CNRL = 1100 SHAMT = 01000

Testing SRA by Variable Function
                678 ALU_OUT = 11111111000000000000000000000000 NF_OUT = 1 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000000111 B =
                10000000000000000000000000000000 CNRL = 1101 SHAMT = 01010
                688 ALU_OUT = 00000000000001111111111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 0 A = 00000000000000000000000000001011 B =
                01111111111111111111111111111111 CNRL = 1101 SHAMT = 01000

Testing Invalid Opcode AKA BF_OUT
                730 ALU_OUT = 00000000000011111111111111111111 NF_OUT = 0 ZF_OUT = 0 OF_OUT = 0 BF_OUT = 1 A = 00000000000000000000000000001011 B =
                01111111111111111111111111111111 CNRL = 1111 SHAMT = 01000
$finish called from file "tb_alu.v", line 175.
$finish at simulation time                830
         V C S   S i m u l a t i o n   R e p o r t
Time: 830 ns
CPU Time:      0.490 seconds;       Data structure size:   0.0Mb
Mon Aug  8 20:58:55 2016
```

text file | length : 7,092 lines : 70 | Ln : 1 Col : 46 Sel : 0 | 0 | Unix (LF)

Figure 17. ALU Simulation Results Part 2

6.2 ALU Controller Design and Test

Design Code:

The ALU Controller design code is shown in Figure 18. A case statement was used to select between each Control (CNRL) input. Additionally, another case statement was used when CNRL = ‘010’ to select between each Function input (FUNCT) for R-type instructions.

```
/*********************************************************************
*** Filename: alu_cont.v Created by Orr Chakon                     ***
*********************************************************************/

`timescale 1 ns / 1 ns

module alu_cont(FUNCT, CNRL, OUT);

// Port declaration
   output [3:0] OUT;
   input [5:0] FUNCT;
   input [2:0] CNRL;

// Internal variable declarations
   reg [3:0] OUT;

// Model of combinational ALU controller
   always @(FUNCT, CNRL)    //triggered off input change
    begin
        case(CNRL)
        3'b000:OUT=4'b0010;     //invoke ALU ADD operation
        3'b001:OUT=4'b0110;     //invoke ALU SUB operation
        3'b010:                 //R-Type operations
            case(FUNCT)
            6'b001001: OUT=4'b0010; //invoke ALU ADD operation
            6'b100000: OUT=4'b0010; //invoke ALU ADD operation
            6'b100001: OUT=4'b0010; //invoke ALU ADD operation
            6'b100010: OUT=4'b0110; //invoke ALU SUB operation
            6'b100011: OUT=4'b0110; //invoke ALU SUB operation
            6'b100100: OUT=4'b0000; //invoke ALU AND operation
            6'b100101: OUT=4'b0001; //invoke ALU OR operation
            6'b100110: OUT=4'b0011; //invoke ALU XOR operation
            6'b100111: OUT=4'b0100; //invoke ALU NOR operation
            6'b000000: OUT=4'b1000; //invoke ALU Shift-Left-Logic operation
            6'b000100: OUT=4'b1001; //invoke ALU Shift-Left-Logic by variable operation
            6'b000010: OUT=4'b1010; //invoke ALU Shift-Right-Logic operation
            6'b000110: OUT=4'b1011; //invoke ALU Shift-Right-Logic by variable operation
            6'b000011: OUT=4'b1100; //invoke ALU Shift Right Arithmetic operation
            6'b000111: OUT=4'b1101; //invoke ALU Shift Right Arithmetic by variable operation
            6'b101001: OUT=4'b0101; //invoke ALU Set-If-Less-Than Unsigned operation
            6'b101010: OUT=4'b0111; //invoke ALU Set-If-Less-Than operation
            default:OUT=4'bxxxx;
            endcase
        3'b011:OUT=4'b0111;     //invoke ALU Set-If-Less-Than operation
        3'b100:OUT=4'b0000;     //invoke ALU AND operation
        3'b101:OUT=4'b0001;     //invoke ALU OR operation
        3'b110:OUT=4'b0011;     //invoke ALU XOR operation
        default:OUT=4'bxxxx;
        endcase
    end

endmodule

Verilog file                                  length : 2,059   lines : 53   Lr
```

Figure 18. ALU Controller Design Code

Test Bench:

The Test Bench program for the ALU Controller is shown in Figures 19 through 21. In order to test the ALU Controller design, there was a cycle between all CNRL and FUNCT inputs.

```
/*********************************************************************
*** Filename: tb_alu_cont.v Created by Orr Chakon                  ***
*********************************************************************/

`timescale 1 ns /1 ns
module tb_alu_cont();

// Port declaration
   reg [2:0] CNRL;
   reg [5:0] FUNCT;
   wire [3:0] OUT;

// UUT instantiation

   alu_cont UUT (FUNCT, CNRL, OUT);


initial begin

  #10 $write("\n"); //new line used for clarity
      CNRL=3'b000;
      FUNCT=6'bxxxxxx;      //testing LOAD/STORE operation. Output should be 0010 (ADD).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b010;
      FUNCT=6'b001000;      //testing Jump Register (R-Type) operation. Output should be xxxx.
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b100;      //testing AND Immediate operation. Output should be 0000 (AND).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b010;
      FUNCT=6'b001001;      //testing Jump and Link Register (R-Type) operation. Output should be 0010 (ADD).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b101;  //testing OR Immediate operation. Output should be 0001 (OR).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b010;
      FUNCT=6'b100000;      //testing ADD (R-type) operation. Output should be 0010 (ADD).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b110;  //testing XOR Immediate operation. Output should be 0011 (XOR).
      $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);

  #10 $write("\n");
      CNRL=3'b010;
```

Verilog file | length : 6,165 lines : 139 | Ln : 1 Col : 77 Sel : 0 | 0

Figure 19. ALU Controller Test Bench Part 1

```
51
52      #10 $write("\n");
53          CNRL=3'b010;
54          FUNCT=6'b100001;      //testing Add Unsigned (R-Type) operation. Output should be 0010 (ADD).
55          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
56
57      #10 $write("\n");
58          CNRL=3'b001;      //testing Branch operation. Output should be 0110 (subtract).
59          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
60
61      #10 $write("\n");
62          CNRL=3'b010;
63          FUNCT=6'b100100;      //testing AND (R-Type) operation. Output should be 0000 (AND).
64          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
65
66      #10 $write("\n");
67          CNRL=3'b011;      //testing Set-If-Less-Than immediate operation. Output should be 0111 (Set-If-Less-Than immediate).
68          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
69
70      #10 $write("\n");
71          CNRL=3'b010;
72          FUNCT=6'b100010;      //testing Subtract (R-Type) operation. Output should be 0110 (Subtract).
73          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
74
75      #10 $write("\n");
76          CNRL=3'b111;      //testing illegal operation. Output should be xxxx.
77          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
78
79      #10 $write("\n");
80          CNRL=3'b010;
81          FUNCT=6'b100011;      //testing Subtract Unsigned (R-Type) operation. Output should be 0110 (Subtract).
82          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
83
84      #10 $write("\n");
85          FUNCT=6'b100101;      //testing OR (R-Type) operation. Output should be 0001 (OR).
86          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
87
88      #10 $write("\n");
89          FUNCT=6'b100110;      //testing XOR (R-Type) operation. Output should be 0011 (XOR).
90          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
91
92      #10 $write("\n");
93          FUNCT=6'b100111;      //testing NOR (R-Type) operation. Output should be 0100 (NOR).
94          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
95
96      #10 $write("\n");
97          FUNCT=6'b000000;      //testing Shift-Left-Logic (R-Type) operation. Output should be 1000 (SLL).
98          $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
99
100     #10 $write("\n");
101         FUNCT=6'b000100;      //testing Shift-Left-Logic by Variable (R-Type) operation. Output should be 1001 (SLLV).
102         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
103
erilog file    length : 6,165   lines : 139    Ln : 1   Col : 77   Sel : 0 | 0    U
```

Figure 20. ALU Controller Test Bench Part 2

```
102         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
103
104     #10 $write("\n");
105         FUNCT=6'b000010;      //testing Shift-Right-Logic (R-Type) operation. Output should be 1010 (SRL).
106         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
107
108     #10 $write("\n");
109         FUNCT=6'b000110;      //testing Shift-Right-Logic by Variable (R-Type) operation. Output should be 1011 (SRLV).
110         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
111
112     #10 $write("\n");
113         FUNCT=6'b000011;      //testing Shift-Right-Arithmetic (R-Type) operation. Output should be 1100 (SRA).
114         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
115
116     #10 $write("\n");
117         FUNCT=6'b000111;      //testing Shift-Right-Arithmetic by Variable (R-Type) operation. Output should be 1101 (SRAV).
118         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
119
120     #10 $write("\n");
121         FUNCT=6'b101001;      //testing Set-If-Less-Than Unsigned (R-Type) operation. Output should be 0101 (Set-If-Less-Than Unsigned).
122         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
123
124     #10 $write("\n");
125         FUNCT=6'b101010;      //testing Set-If-Less-Than (R-Type) operation. Output should be 0111 (Set-If-Less-Than).
126         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
127
128     #10 $write("\n");
129         FUNCT=6'b111111;      //testing illegal (R-Type) operation. Output should be xxxx.
130         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
131
132     #10 $write("\n");
133         FUNCT=6'bxxxxxx;      //testing illegal (R-Type) operation. Output should be xxxx.
134         $strobe("%d UUT: CNRL = %b FUNCT = %b OUT = %b" , $time, CNRL, FUNCT, OUT);
135
136     #20 $finish;
137    end
138 endmodule
139
```

Verilog file | length : 6,165 lines : 139 | Ln : 1 Col : 77 Sel : 0 | 0 | U

Figure 21. ALU Controller Test Bench Part 3

Test Results:

Simulation results for the ALU Controller design are shown in Figure 22.

```
 10 UUT: CNRL = 000 FUNCT = xxxxxx OUT = 0010

 20 UUT: CNRL = 010 FUNCT = 001000 OUT = xxxx

 30 UUT: CNRL = 100 FUNCT = 001000 OUT = 0000

 40 UUT: CNRL = 010 FUNCT = 001001 OUT = 0010

 50 UUT: CNRL = 101 FUNCT = 001001 OUT = 0001

 60 UUT: CNRL = 010 FUNCT = 100000 OUT = 0010

 70 UUT: CNRL = 110 FUNCT = 100000 OUT = 0011

 80 UUT: CNRL = 010 FUNCT = 100001 OUT = 0010

 90 UUT: CNRL = 001 FUNCT = 100001 OUT = 0110

100 UUT: CNRL = 010 FUNCT = 100100 OUT = 0000

110 UUT: CNRL = 011 FUNCT = 100100 OUT = 0111

120 UUT: CNRL = 010 FUNCT = 100010 OUT = 0110

130 UUT: CNRL = 111 FUNCT = 100010 OUT = xxxx

140 UUT: CNRL = 010 FUNCT = 100011 OUT = 0110

150 UUT: CNRL = 010 FUNCT = 100101 OUT = 0001

160 UUT: CNRL = 010 FUNCT = 100110 OUT = 0011

170 UUT: CNRL = 010 FUNCT = 100111 OUT = 0100

180 UUT: CNRL = 010 FUNCT = 000000 OUT = 1000

190 UUT: CNRL = 010 FUNCT = 000100 OUT = 1001

200 UUT: CNRL = 010 FUNCT = 000010 OUT = 1010

210 UUT: CNRL = 010 FUNCT = 000110 OUT = 1011

220 UUT: CNRL = 010 FUNCT = 000011 OUT = 1100

230 UUT: CNRL = 010 FUNCT = 000111 OUT = 1101

240 UUT: CNRL = 010 FUNCT = 101001 OUT = 0101

250 UUT: CNRL = 010 FUNCT = 101010 OUT = 0111

260 UUT: CNRL = 010 FUNCT = 111111 OUT = xxxx

270 UUT: CNRL = 010 FUNCT = xxxxxx OUT = xxxx
```

Figure 22. ALU Controller Simulation Results

6.3 Combinational Block Design and Test

Design Code:

Design code for the Combinational Block is shown in Figure 23.

```
/*********************************************************************************
*** Filename: comb.v Created by Orr Chakon                                     ***
*********************************************************************************/

`timescale 1 ns / 1 ns

module comb(NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN, OUT);

// Port declaration
   output OUT;
   input NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN;

// Internal variable declarations
   reg OUT;

//behavioral model of the Combinational Block circuit
always@ (NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN) begin
OUT = (NF & BLTZ) | (BGTZ & ~(NF | ZF)) | (BLEZ & (NF | ZF)) | (~ZF & BNE) | (ZF & BEQ) | PC_EN;
end

endmodule

```

Figure 23. Combinational Block Design Code

Test Bench:

The Combinational Test Bench Program is shown in Figures 24 through 26.

```
/*********************************************************************************
*** Filename: tb_comb.v Created by Orr Chakon                                  ***
*********************************************************************************/

`timescale 1 ns /1 ns
module tb_comb();

// Port declaration
   reg NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN;
   wire OUT;

// UUT instantiation

   comb UUT (NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN, OUT);

 initial
   $monitor ("%d NF = %b ZF = %b BLTZ = %b BGTZ = %b BLEZ = %b BNE = %b BEQ = %b PC_EN = %b OUT = %b",
             $time, NF, ZF, BLTZ, BGTZ, BLEZ, BNE, BEQ, PC_EN, OUT);

initial begin

  #10 $write("\n"); //new line used for clarity
      NF=1'b0;
      ZF=1'b0;
      BLTZ=1'b0;
      BGTZ=1'b0;
      BLEZ=1'b0;
      BNE=1'b0;
      BEQ=1'b0;
      PC_EN=1'b0;

  #10 $write("\n"); //new line used for clarity
      NF=1'b1;
      ZF=1'b1;
      BLTZ=1'b1;
      BGTZ=1'b1;
      BLEZ=1'b1;
      BNE=1'b1;
      BEQ=1'b1;
      PC_EN=1'b1;

  #10 $write("\n"); //new line used for clarity
      NF=1'b0;
      ZF=1'b0;
      BLTZ=1'b1;
      BGTZ=1'b1;
      BLEZ=1'b1;
      BNE=1'b0;
      BEQ=1'b1;
      PC_EN=1'b0;

   #10 $write("\n");    //new line used for clarity
      NF=1'b1;
```

Verilog file | length : 2,923 lines : 155 | Ln : 1 Col : 7

Figure 24. Combinational Block Test Bench Part 1

```
52      #10 $write("\n");    //new line used for clarity
53          NF=1'b1;
54          ZF=1'b0;
55          BLTZ=1'b0;
56          BGTZ=1'b1;
57          BLEZ=1'b0;
58          BNE=1'b0;
59          BEQ=1'b0;
60          PC_EN=1'b0;
61
62      #10 $write("\n"); //new line used for clarity
63          NF=1'b1;
64          ZF=1'b1;
65          BLTZ=1'b0;
66          BGTZ=1'b0;
67          BLEZ=1'b0;
68          BNE=1'b0;
69          BEQ=1'b0;
70          PC_EN=1'b0;
71
72      #10 $write("\n"); //new line used for clarity
73          NF=1'b0;
74          ZF=1'b0;
75          BLTZ=1'b0;
76          BGTZ=1'b1;
77          BLEZ=1'b0;
78          BNE=1'b0;
79          BEQ=1'b0;
80          PC_EN=1'b0;
81
82      #10 $write("\n"); //new line used for clarity
83          NF=1'b0;
84          ZF=1'b0;
85          BLTZ=1'b0;
86          BGTZ=1'b0;
87          BLEZ=1'b1;
88          BNE=1'b0;
89          BEQ=1'b0;
90          PC_EN=1'b0;
91
92      #10 $write("\n"); //new line used for clarity
93          NF=1'b1;
94          ZF=1'b0;
95          BLTZ=1'b0;
96          BGTZ=1'b0;
97          BLEZ=1'b1;
98          BNE=1'b0;
99          BEQ=1'b0;
100         PC_EN=1'b0;
101
102     #10 $write("\n"); //new line used for clarity
103         NF=1'b0;
104         ZF=1'b1;

Verilog file    length : 2,923   lines : 155    Ln : 17   Col : 104
```

Figure 25. Combinational Block Test Bench Part 2

```
102     #10 $write("\n"); //new line used for clarity
103         NF=1'b0;
104         ZF=1'b1;
105         BLTZ=1'b0;
106         BGTZ=1'b0;
107         BLEZ=1'b0;
108         BNE=1'b1;
109         BEQ=1'b0;
110         PC_EN=1'b0;
111
112     #10 $write("\n"); //new line used for clarity
113         NF=1'b0;
114         ZF=1'b0;
115         BLTZ=1'b0;
116         BGTZ=1'b0;
117         BLEZ=1'b0;
118         BNE=1'b1;
119         BEQ=1'b0;
120         PC_EN=1'b0;
121
122     #10 $write("\n"); //new line used for clarity
123         NF=1'b0;
124         ZF=1'b1;
125         BLTZ=1'b0;
126         BGTZ=1'b1;
127         BLEZ=1'b0;
128         BNE=1'b0;
129         BEQ=1'b0;
130         PC_EN=1'b0;
131
132     #10 $write("\n"); //new line used for clarity
133         NF=1'b0;
134         ZF=1'b1;
135         BLTZ=1'b0;
136         BGTZ=1'b0;
137         BLEZ=1'b0;
138         BNE=1'b0;
139         BEQ=1'b1;
140         PC_EN=1'b0;
141
142     #10 $write("\n"); //new line used for clarity
143         NF=1'b0;
144         ZF=1'b0;
145         BLTZ=1'b0;
146         BGTZ=1'b0;
147         BLEZ=1'b0;
148         BNE=1'b0;
149         BEQ=1'b0;
150         PC_EN=1'b1;
151
152     #20 $finish;
153      end
154 endmodule
Verilog file                                  length : 2,923   lines : 155   Ln : 16   Col : 12
```

Figure 26. Combinational Block Test Bench Part 3

Test Results:

Simulation results for the Combinational Block Circuit are shown in Figure 27.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Sep 22 22:02 2016
                   0 NF = x ZF = x BLTZ = x BGTZ = x BLEZ = x BNE = x BEQ = x PC_EN = x OUT = x

                  10 NF = 0 ZF = 0 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 0

                  20 NF = 1 ZF = 1 BLTZ = 1 BGTZ = 1 BLEZ = 1 BNE = 1 BEQ = 1 PC_EN = 1 OUT = 1

                  30 NF = 0 ZF = 0 BLTZ = 1 BGTZ = 1 BLEZ = 1 BNE = 0 BEQ = 1 PC_EN = 0 OUT = 1

                  40 NF = 1 ZF = 0 BLTZ = 0 BGTZ = 1 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 0

                  50 NF = 1 ZF = 1 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 0

                  60 NF = 0 ZF = 0 BLTZ = 0 BGTZ = 1 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 1

                  70 NF = 0 ZF = 0 BLTZ = 0 BGTZ = 0 BLEZ = 1 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 0

                  80 NF = 1 ZF = 0 BLTZ = 0 BGTZ = 0 BLEZ = 1 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 1

                  90 NF = 0 ZF = 1 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 1 BEQ = 0 PC_EN = 0 OUT = 0

                 100 NF = 0 ZF = 0 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 1 BEQ = 0 PC_EN = 0 OUT = 1

                 110 NF = 0 ZF = 1 BLTZ = 0 BGTZ = 1 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 0 OUT = 0

                 120 NF = 0 ZF = 1 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 0 BEQ = 1 PC_EN = 0 OUT = 1

                 130 NF = 0 ZF = 0 BLTZ = 0 BGTZ = 0 BLEZ = 0 BNE = 0 BEQ = 0 PC_EN = 1 OUT = 1
$finish called from file "tb_comb.v", line 155.
$finish at simulation time                 150
           V C S   S i m u l a t i o n   R e p o r t
Time: 150 ns
CPU Time:      0.500 seconds;       Data structure size:   0.0Mb
Thu Sep 22 22:02:49 2016

```

Figure 27. Combinational Block Simulation Results

6.4 Concatenate Module Design and Test

Design Code:

Design code for the Concatenate Module is shown in Figure 28. The Concatenate Module takes in a 26-bit and 4-bit input and concatenates them with ‘00’ to generate the output.

```
1  /*************************************************************************
2  *** Filename: concatinate.v Created by Orr Chakon
3  *************************************************************************
4
5  `timescale 1 ns / 1 ns
6
7  module concatinate(CIN26, CIN4, OUT);
8
9  // Port declaration
10    output [31:0] OUT;
11    input [25:0] CIN26;
12    input [3:0] CIN4;
13
14 // Internal variable declarations
15    reg [31:0] OUT;
16
17 // OUT = CIN4 || CIN26 || 00
18    always @(CIN26, CIN4)       //triggered off input change
19    begin
20     OUT[1:0] = 2'b00;       //First two bits are a constant '00'
21     OUT[27:2] = CIN26;
22     OUT[31:28] = CIN4;
23     end
24 endmodule
25
```

Figure 28. Concatenate Module Design Code

Test Bench:

The Test Bench Program for the Concatenate Module is shown in Figure 29.

```
/*********************************************************************************
*** Filename: tb_concatinate.v Created by Orr Chakon                          ***
*********************************************************************************/

`timescale 1 ns /1 ns
module tb_concatinate();

// Port declaration
   reg [25:0] CIN26;
   reg [3:0] CIN4;
   wire [31:0] OUT;

// UUT instantiation
   concatinate UUT (CIN26, CIN4, OUT);

initial
//variables output to the simulation log file
$monitor ("%d CIN26 = %b CIN4 = %b OUT = %b", $time, CIN26, CIN4, OUT);

initial begin

  #10 $write("\n"); //new line used for clarity
        CIN26 = 26'b11111111111111111111111111;
        CIN4 = 4'b1111;

  #10 $write("\n"); //new line used for clarity
        CIN26 = 26'b00000000000000000000000000;
        CIN4 = 4'b0000;

  #10 $write("\n"); //new line used for clarity
        CIN26 = 26'b10000000000000000000000001;
        CIN4 = 4'b1001;


  #20 $finish;
   end
endmodule
```

Figure 29. Concatenate Module Test Bench

Test Results:

Simulation results for the Concatenate Module are shown in Figure 30.

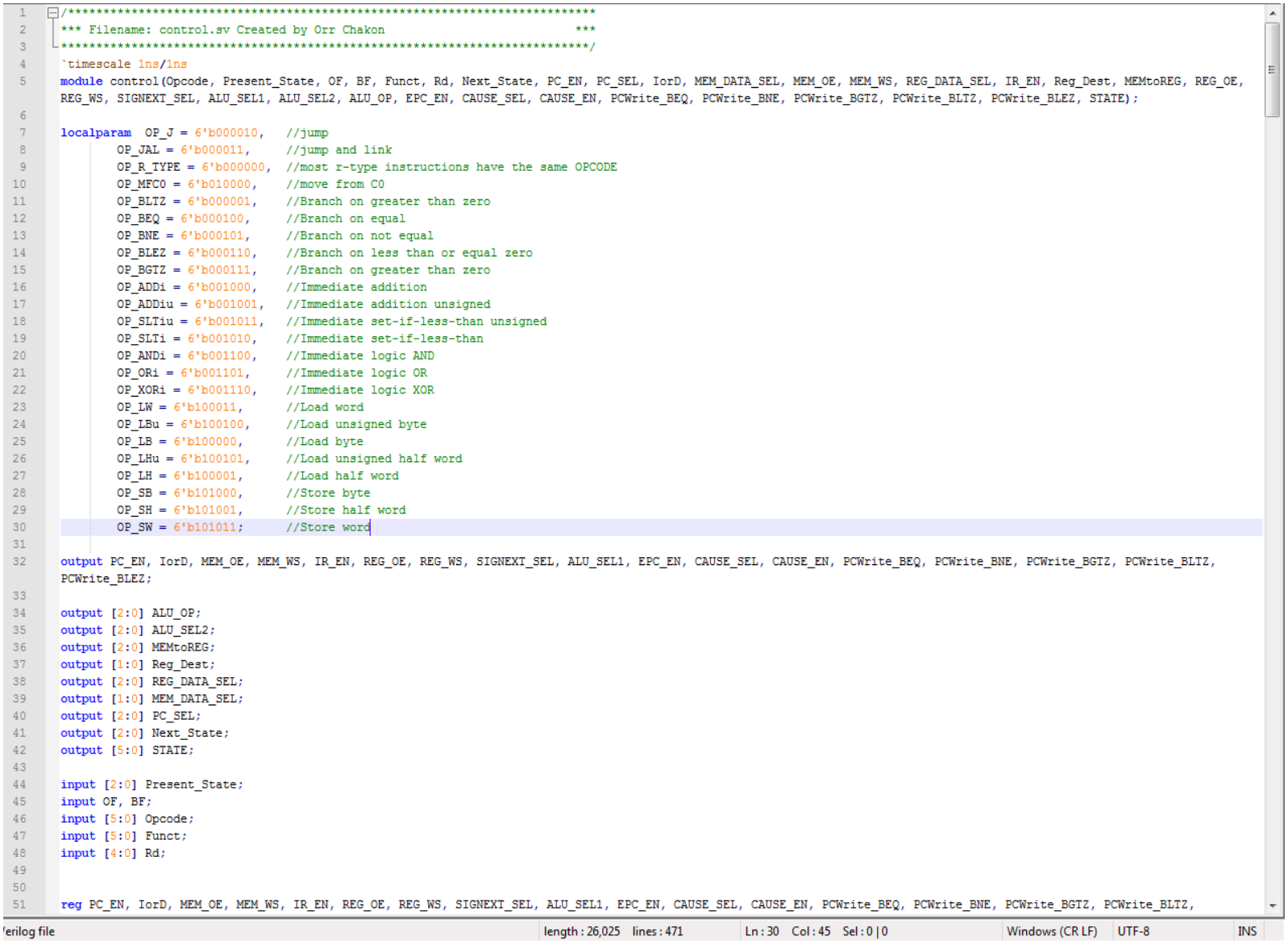
```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct 15 15:20 2016
                   0 CIN26 = xxxxxxxxxxxxxxxxxxxxxxxxxx CIN4 = xxxx OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                  10 CIN26 = 11111111111111111111111111 CIN4 = 1111 OUT = 11111111111111111111111111111100

                  20 CIN26 = 00000000000000000000000000 CIN4 = 0000 OUT = 00000000000000000000000000000000

                  30 CIN26 = 10000000000000000000000001 CIN4 = 1001 OUT = 10011000000000000000000000000100
$finish called from file "tb_concatinate.v", line 39.
$finish at simulation time                   50
          V C S   S i m u l a t i o n   R e p o r t
Time: 50 ns
CPU Time:      0.530 seconds;       Data structure size:   0.0Mb
Sat Oct 15 15:20:15 2016
```

Figure 30. Concatenate Module Simulation Results

6.5 Sequence Controller Design and Test

Design Code:

The Sequence Controller design code is shown in Figures 31 through 40. The design is an implementation of the state machine shown in Figure 6 with the only deviation occurring in State 0 where various control signals are being initialized. The STATE signal was used for debug purposes only and has no functional use in the processor design. It was vital for troubleshooting at the top level as it outputs the actual states executed, which helps resolve control logic related problems.



```
/*********************************************************************
*** Filename: control.sv Created by Orr Chakon                     ***
*********************************************************************/
`timescale 1ns/1ns
module control(Opcode, Present_State, OF, BF, Funct, Rd, Next_State, PC_EN, PC_SEL, IorD, MEM_DATA_SEL, MEM_OE, MEM_WS, REG_DATA_SEL, IR_EN, Reg_Dest, MEMtoREG, REG_OE,
REG_WS, SIGNEXT_SEL, ALU_SEL1, ALU_SEL2, ALU_OP, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ, STATE);

localparam  OP_J = 6'b000010,   //jump
        OP_JAL = 6'b000011,     //jump and link
        OP_R_TYPE = 6'b000000,  //most r-type instructions have the same OPCODE
        OP_MFC0 = 6'b010000,    //move from C0
        OP_BLTZ = 6'b000001,    //Branch on greater than zero
        OP_BEQ = 6'b000100,     //Branch on equal
        OP_BNE = 6'b000101,     //Branch on not equal
        OP_BLEZ = 6'b000110,    //Branch on less than or equal zero
        OP_BGTZ = 6'b000111,    //Branch on greater than zero
        OP_ADDi = 6'b001000,    //Immediate addition
        OP_ADDiu = 6'b001001,   //Immediate addition unsigned
        OP_SLTiu = 6'b001011,   //Immediate set-if-less-than unsigned
        OP_SLTi = 6'b001010,    //Immediate set-if-less-than
        OP_ANDi = 6'b001100,    //Immediate logic AND
        OP_ORi = 6'b001101,     //Immediate logic OR
        OP_XORi = 6'b001110,    //Immediate logic XOR
        OP_LW = 6'b100011,      //Load word
        OP_LBu = 6'b100100,     //Load unsigned byte
        OP_LB = 6'b100000,      //Load byte
        OP_LHu = 6'b100101,     //Load unsigned half word
        OP_LH = 6'b100001,      //Load half word
        OP_SB = 6'b101000,      //Store byte
        OP_SH = 6'b101001,      //Store half word
        OP_SW = 6'b101011;      //Store word

output PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ,
PCWrite_BLEZ;

output [2:0] ALU_OP;
output [2:0] ALU_SEL2;
output [2:0] MEMtoREG;
output [1:0] Reg_Dest;
output [2:0] REG_DATA_SEL;
output [1:0] MEM_DATA_SEL;
output [2:0] PC_SEL;
output [2:0] Next_State;
output [5:0] STATE;

input [2:0] Present_State;
input OF, BF;
input [5:0] Opcode;
input [5:0] Funct;
input [4:0] Rd;


reg PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ,
```


Figure 31. Sequence Controller Design Code Part 1

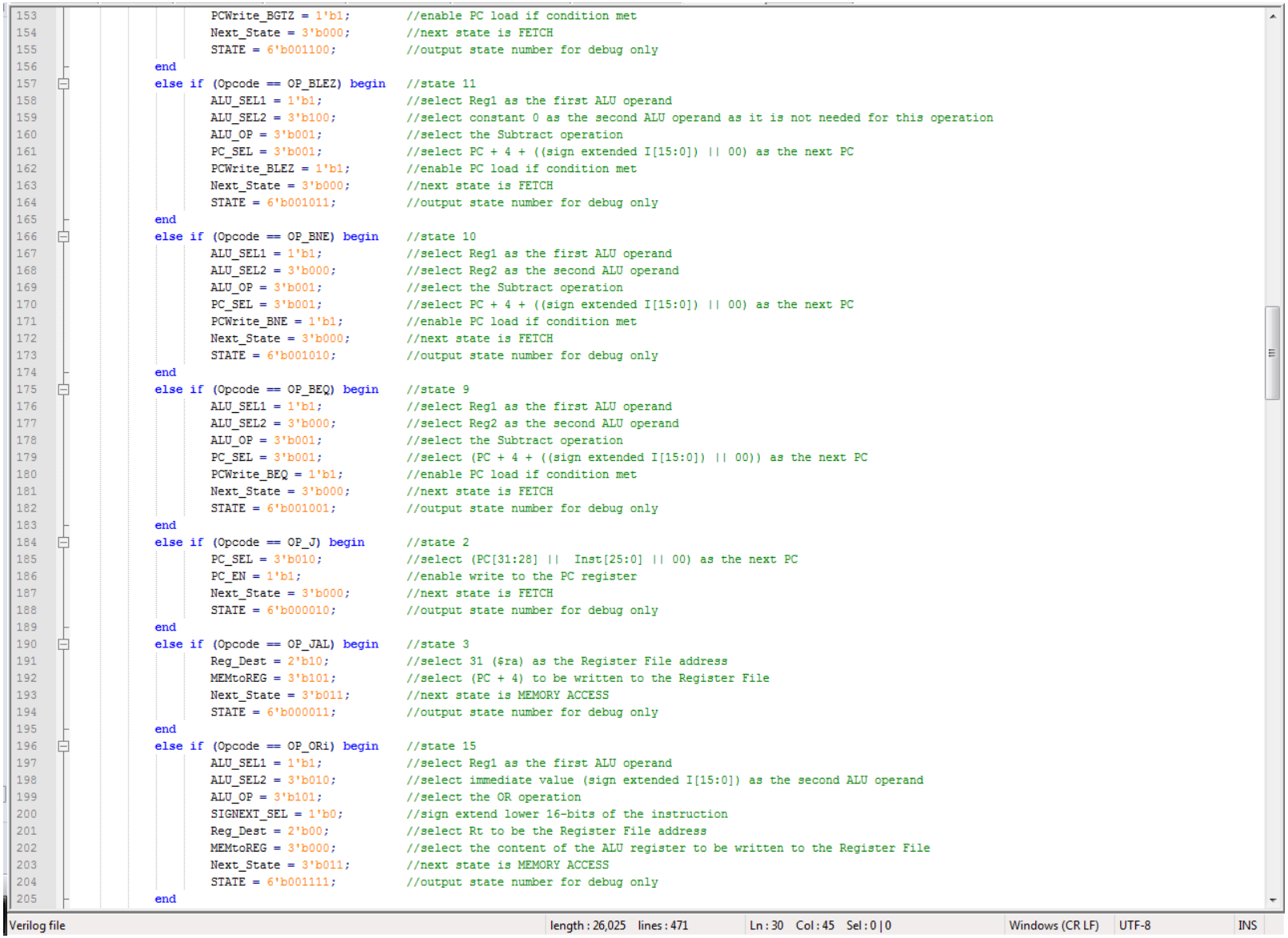
```
51  reg PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ,
    PCWrite_BLEZ;
52
53  reg [2:0] ALU_OP;
54  reg [2:0] ALU_SEL2;
55  reg [2:0] MEMtoREG;
56  reg [1:0] Reg_Dest;
57  reg [2:0] REG_DATA_SEL;
58  reg [1:0] MEM_DATA_SEL;
59  reg [2:0] PC_SEL;
60  reg [2:0] Next_State;
61  reg [5:0] STATE;
62
63  wire [5:0] Opcode;
64  wire [5:0] Funct;
65  wire [4:0] Rd;
66  wire OF, BF;
67
68
69
70  always@ (Present_State, Opcode) begin
71      case (Present_State)
72          0: begin                    //state 0
73
74              IorD = 1'b0;                        //Select the PC as the address to RAM
75              MEM_OE = 1'b1;                      //enable reading the next instruction from RAM
76              REG_DATA_SEL = 3'b000;              //select the instruction from RAM to be written into the IR
77              IR_EN = 1'b1;                       //enable write to IR
78              REG_OE = 1'b1;                      //enables reading instruction operands from Register File
79              ALU_SEL1 = 1'b0;                    //select PC as the first ALU operand
80              ALU_SEL2 = 3'b001;                  //select constant 1 as the second ALU operand
81              ALU_OP = 3'b000;                    //select the ADD operation in order to increment PC by 1
82              PC_SEL = 3'b000;                    //select the incremented PC to be the next PC value
83              PC_EN = 1'b1;                       //write the incremented PC value into the PC register
84              Next_State = 3'b001;                //next state is DECODE
85              STATE = 6'b000000;                  //output state number for debug only
86
87              //initialize all other outputs to zero.
88              MEM_WS = 1'b0;
89              REG_WS = 1'b0;
90              EPC_EN = 1'b0;
91              CAUSE_EN = 1'b0;
92              PCWrite_BEQ = 1'b0;
93              PCWrite_BNE = 1'b0;
94              PCWrite_BGTZ = 1'b0;
95              PCWrite_BLTZ = 1'b0;
96              PCWrite_BLEZ = 1'b0;
97              SIGNEXT_SEL = 1'b0;
98              CAUSE_SEL = 1'b0;
99              MEMtoREG = 3'b000;
100             Reg_Dest = 2'b00;
101             MEM_DATA_SEL = 2'b00;
102         end
```

erilog file | length : 26,025 lines : 471 | Ln : 30 Col : 45 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

Figure 32. Sequence Controller Design Code Part 2

```
104         1: begin                        //state 1
105
106             SIGNEXT_SEL = 1'b0;             //sign extend lower 16-bits of the instruction
107             ALU_SEL1 = 1'b0;                //select PC as the first ALU operand
108             ALU_SEL2 = 3'b011;              //select sign extend lower 16-bits (shifted left by 2)
109             ALU_OP = 3'b000;                //select the ADD operation
110             Next_State = 3'b010;            //next state is EXECUTE
111             IR_EN = 1'b0;                   //disable write to IR
112             MEM_OE = 1'b0;                  //disable reading from RAM
113             PC_EN = 1'b0;                   //disable writes to PC register
114             STATE = 6'b000001;              //output state number for debug only
115         end
116         2: begin
117             if (Opcode == OP_MFC0) begin
118                 if (Rd == 5'b01101) begin       //state 33
119                     Reg_Dest = 2'b01;           //select Rd to be the Register File address
120                     MEMtoREG = 3'b011;          //select the content of the Cause register to be written to the Register File
121                     Next_State = 3'b011;        //next state is MEMORY ACCESS
122                     STATE = 6'b100001;          //output state number for debug only
123                 end
124                 else if (Rd == 5'b01110) begin  //state 32
125                     Reg_Dest = 2'b01;           //select Rd to be the Register File address
126                     MEMtoREG = 3'b010;          //select the content of the EPC register to be written to the Register File
127                     Next_State = 3'b011;        //next state is MEMORY ACCESS
128                     STATE = 6'b100000;          //output state number for debug only
129                 end
130                 else begin                      //go to invalid instruction state 31
131                     CAUSE_SEL = 1'b1;           //select '1' to be written to the Cause register indicating an invalid operation
132                     CAUSE_EN = 1'b1;            //enable writing to the Cause register
133                     EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
134                     PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
135                     Next_State = 3'b000;        //next state is FETCH
136                     STATE = 6'b011111;          //output state number for debug only
137                 end
138             end
139             else if (Opcode == OP_BLTZ) begin   //state 13
140                     ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
141                     ALU_SEL2 = 3'b100;          //select constant 0 as the second ALU operand as it is not needed for this operation
142                     ALU_OP = 3'b001;            //select the Subtract operation
143                     PC_SEL = 3'b001;            //select PC + 4 + ((sign extended I[15:0]) || 00) as the next PC
144                     PCWrite_BLTZ = 1'b1;        //enable PC load if condition met
145                     Next_State = 3'b000;        //next state is FETCH
146                     STATE = 6'b001101;          //output state number for debug only
147             end
148             else if (Opcode == OP_BGTZ) begin   //state 12
149                     ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
150                     ALU_SEL2 = 3'b100;          //select constant 0 as the second ALU operand as it is not needed for this operation
151                     ALU_OP = 3'b001;            //select the Subtract operation
152                     PC_SEL = 3'b001;            //select PC + 4 + ((sign extended I[15:0]) || 00) as the next PC
153                     PCWrite_BGTZ = 1'b1;        //enable PC load if condition met
154                     Next_State = 3'b000;        //next state is FETCH
155                     STATE = 6'b001100;          //output state number for debug only
156             end
```


Figure 33. Sequence Controller Design Code Part 3

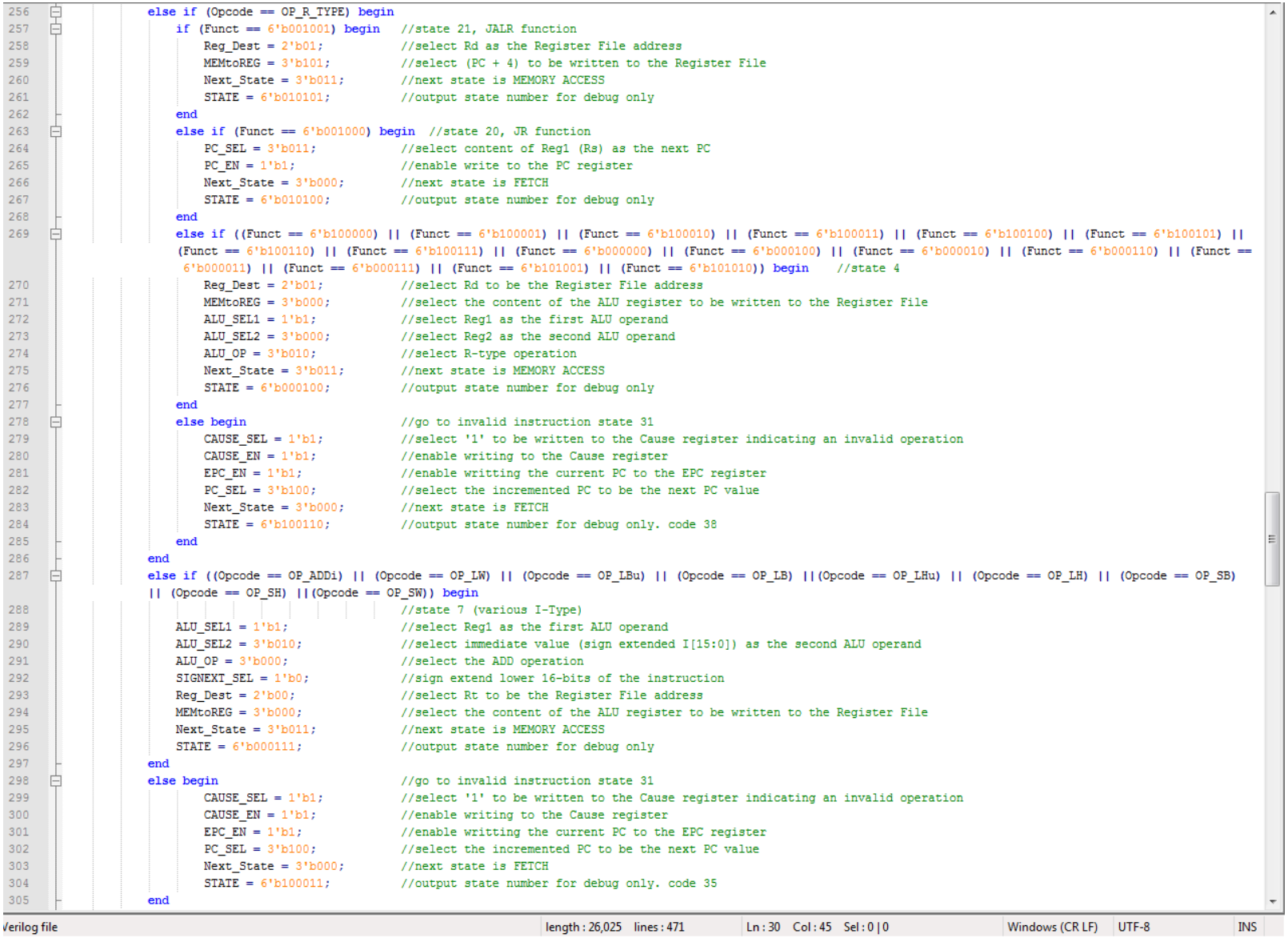


```
153                 PCWrite_BGTZ = 1'b1;        //enable PC load if condition met
154                 Next_State = 3'b000;        //next state is FETCH
155                 STATE = 6'b001100;          //output state number for debug only
156             end
157             else if (Opcode == OP_BLEZ) begin   //state 11
158                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
159                 ALU_SEL2 = 3'b100;          //select constant 0 as the second ALU operand as it is not needed for this operation
160                 ALU_OP = 3'b001;            //select the Subtract operation
161                 PC_SEL = 3'b001;            //select PC + 4 + ((sign extended I[15:0]) || 00) as the next PC
162                 PCWrite_BLEZ = 1'b1;        //enable PC load if condition met
163                 Next_State = 3'b000;        //next state is FETCH
164                 STATE = 6'b001011;          //output state number for debug only
165             end
166             else if (Opcode == OP_BNE) begin    //state 10
167                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
168                 ALU_SEL2 = 3'b000;          //select Reg2 as the second ALU operand
169                 ALU_OP = 3'b001;            //select the Subtract operation
170                 PC_SEL = 3'b001;            //select PC + 4 + ((sign extended I[15:0]) || 00) as the next PC
171                 PCWrite_BNE = 1'b1;         //enable PC load if condition met
172                 Next_State = 3'b000;        //next state is FETCH
173                 STATE = 6'b001010;          //output state number for debug only
174             end
175             else if (Opcode == OP_BEQ) begin    //state 9
176                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
177                 ALU_SEL2 = 3'b000;          //select Reg2 as the second ALU operand
178                 ALU_OP = 3'b001;            //select the Subtract operation
179                 PC_SEL = 3'b001;            //select (PC + 4 + ((sign extended I[15:0]) || 00)) as the next PC
180                 PCWrite_BEQ = 1'b1;         //enable PC load if condition met
181                 Next_State = 3'b000;        //next state is FETCH
182                 STATE = 6'b001001;          //output state number for debug only
183             end
184             else if (Opcode == OP_J) begin      //state 2
185                 PC_SEL = 3'b010;            //select (PC[31:28] ||  Inst[25:0] || 00) as the next PC
186                 PC_EN = 1'b1;               //enable write to the PC register
187                 Next_State = 3'b000;        //next state is FETCH
188                 STATE = 6'b000010;          //output state number for debug only
189             end
190             else if (Opcode == OP_JAL) begin    //state 3
191                 Reg_Dest = 2'b10;           //select 31 ($ra) as the Register File address
192                 MEMtoREG = 3'b101;          //select (PC + 4) to be written to the Register File
193                 Next_State = 3'b011;        //next state is MEMORY ACCESS
194                 STATE = 6'b000011;          //output state number for debug only
195             end
196             else if (Opcode == OP_ORi) begin    //state 15
197                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
198                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
199                 ALU_OP = 3'b101;            //select the OR operation
200                 SIGNEXT_SEL = 1'b0;         //sign extend lower 16-bits of the instruction
201                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
202                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
203                 Next_State = 3'b011;        //next state is MEMORY ACCESS
204                 STATE = 6'b001111;          //output state number for debug only
205             end
```


Figure 34. Sequence Controller Design Code Part 4

```
203                 Next_State = 3'b011;        //next state is MEMORY ACCESS
204                 STATE = 6'b001111;          //output state number for debug only
205             end
206             else if (Opcode == OP_XORi) begin   //state 16
207                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
208                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
209                 ALU_OP = 3'b110;            //select the XOR operation
210                 SIGNEXT_SEL = 1'b0;         //sign extend lower 16-bits of the instruction
211                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
212                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
213                 Next_State = 3'b011;        //next state is MEMORY ACCESS
214                 STATE = 6'b010000;          //output state number for debug only
215             end
216             else if (Opcode == OP_ANDi) begin   //state 14
217                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
218                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
219                 ALU_OP = 3'b100;            //select the AND operation
220                 SIGNEXT_SEL = 1'b0;         //sign extend lower 16-bits of the instruction
221                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
222                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
223                 Next_State = 3'b011;        //next state is MEMORY ACCESS
224                 STATE = 6'b001110;          //output state number for debug only
225             end
226             else if (Opcode == OP_SLTi) begin   //state 6
227                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
228                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
229                 ALU_OP = 3'b011;            //select the SLT operation
230                 SIGNEXT_SEL = 1'b0;         //sign extend lower 16-bits of the instruction
231                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
232                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
233                 Next_State = 3'b011;        //next state is MEMORY ACCESS
234                 STATE = 6'b000110;          //output state number for debug only
235             end
236             else if (Opcode == OP_SLTiu) begin  //state 28
237                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
238                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
239                 ALU_OP = 3'b011;            //select the SLT operation
240                 SIGNEXT_SEL = 1'b1;         //sign extend lower 16-bits of the instruction with zero's
241                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
242                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
243                 Next_State = 3'b011;        //next state is MEMORY ACCESS
244                 STATE = 6'b011100;          //output state number for debug only
245             end
246             else if (Opcode == OP_ADDiu) begin  //state 29
247                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
248                 ALU_SEL2 = 3'b010;          //select immediate value (sign extended I[15:0]) as the second ALU operand
249                 ALU_OP = 3'b000;            //select the ADD operation
250                 SIGNEXT_SEL = 1'b1;         //sign extend lower 16-bits of the instruction with zero's
251                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
252                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
253                 Next_State = 3'b011;        //next state is MEMORY ACCESS
254                 STATE = 6'b011101;          //output state number for debug only
255             end
```


Figure 35. Sequence Controller Design Code Part 5



```
256         else if (Opcode == OP_R_TYPE) begin
257             if (Funct == 6'b001001) begin   //state 21, JALR function
258                 Reg_Dest = 2'b01;           //select Rd as the Register File address
259                 MEMtoREG = 3'b101;          //select (PC + 4) to be written to the Register File
260                 Next_State = 3'b011;        //next state is MEMORY ACCESS
261                 STATE = 6'b010101;          //output state number for debug only
262             end
263             else if (Funct == 6'b001000) begin  //state 20, JR function
264                 PC_SEL = 3'b011;            //select content of Reg1 (Rs) as the next PC
265                 PC_EN = 1'b1;               //enable write to the PC register
266                 Next_State = 3'b000;        //next state is FETCH
267                 STATE = 6'b010100;          //output state number for debug only
268             end
269             else if ((Funct == 6'b100000) || (Funct == 6'b100001) || (Funct == 6'b100010) || (Funct == 6'b100011) || (Funct == 6'b100100) || (Funct == 6'b100101) ||
                (Funct == 6'b100110) || (Funct == 6'b100111) || (Funct == 6'b000000) || (Funct == 6'b000100) || (Funct == 6'b000010) || (Funct == 6'b000110) || (Funct ==
                 6'b000011) || (Funct == 6'b000111) || (Funct == 6'b101001) || (Funct == 6'b101010)) begin    //state 4
270                 Reg_Dest = 2'b01;           //select Rd to be the Register File address
271                 MEMtoREG = 3'b000;          //select the content of the ALU register to be written to the Register File
272                 ALU_SEL1 = 1'b1;            //select Reg1 as the first ALU operand
273                 ALU_SEL2 = 3'b000;          //select Reg2 as the second ALU operand
274                 ALU_OP = 3'b010;            //select R-type operation
275                 Next_State = 3'b011;        //next state is MEMORY ACCESS
276                 STATE = 6'b000100;          //output state number for debug only
277             end
278             else begin                      //go to invalid instruction state 31
279                 CAUSE_SEL = 1'b1;           //select '1' to be written to the Cause register indicating an invalid operation
280                 CAUSE_EN = 1'b1;            //enable writing to the Cause register
281                 EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
282                 PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
283                 Next_State = 3'b000;        //next state is FETCH
284                 STATE = 6'b100110;          //output state number for debug only. code 38
285             end
286         end
287         else if ((Opcode == OP_ADDi) || (Opcode == OP_LW) || (Opcode == OP_LBu) || (Opcode == OP_LB) ||(Opcode == OP_LHu) || (Opcode == OP_LH) || (Opcode == OP_SB)
            || (Opcode == OP_SH) ||(Opcode == OP_SW)) begin
288                                             //state 7 (various I-Type)
289             ALU_SEL1 = 1'b1;                //select Reg1 as the first ALU operand
290             ALU_SEL2 = 3'b010;              //select immediate value (sign extended I[15:0]) as the second ALU operand
291             ALU_OP = 3'b000;                //select the ADD operation
292             SIGNEXT_SEL = 1'b0;             //sign extend lower 16-bits of the instruction
293             Reg_Dest = 2'b00;               //select Rt to be the Register File address
294             MEMtoREG = 3'b000;              //select the content of the ALU register to be written to the Register File
295             Next_State = 3'b011;            //next state is MEMORY ACCESS
296             STATE = 6'b000111;              //output state number for debug only
297         end
298         else begin                          //go to invalid instruction state 31
299                 CAUSE_SEL = 1'b1;           //select '1' to be written to the Cause register indicating an invalid operation
300                 CAUSE_EN = 1'b1;            //enable writing to the Cause register
301                 EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
302                 PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
303                 Next_State = 3'b000;        //next state is FETCH
304                 STATE = 6'b100011;          //output state number for debug only. code 35
305         end
```


Figure 36. Sequence Controller Design Code Part 6

```
305             end
306         end
307         3: begin
308             if ((Opcode == OP_ORi) || (Opcode == OP_XORi) || (Opcode == OP_ANDi) || (Opcode == OP_SLTi) ||(Opcode == OP_ADDiu) || (Opcode == OP_SLTiu) || ((Opcode ==
            OP_MFC0) && (Rd == 5'b01110)) || ((Opcode == OP_MFC0) && (Rd == 5'b01101))) begin //state 8
309             REG_WS = 1'b1;                  //enable writing to the Register File
310             Next_State = 3'b000;            //next state is FETCH
311             STATE = 6'b001000;              //output state number for debug only
312         end
313         else if ((Opcode == OP_R_TYPE) && (Funct == 6'b001001)) begin  //State 35
314             PC_SEL = 3'b011;            //select content of Reg1 (Rs) as the next PC
315             PC_EN = 1'b1;               //enable write to the PC register
316             REG_WS = 1'b1;                  //enable writing to the Register File
317             Next_State = 3'b000;            //next state is FETCH
318             STATE = 6'b101101;              //output state number for debug only. code 45
319         end
320         else if (Opcode == OP_JAL) begin    //State 36
321             PC_SEL = 3'b010;            //select (PC[31:28] ||  Inst[25:0] || 00) as the next PC
322             PC_EN = 1'b1;               //enable write to the PC register
323             REG_WS = 1'b1;                  //enable writing to the Register File
324             Next_State = 3'b000;            //next state is FETCH
325             STATE = 6'b101110;              //output state number for debug only. code 46
326         end
327         else if (Opcode == OP_R_TYPE) begin
328             if (BF == 1'b1) begin           //state 31
329                 CAUSE_SEL = 1'b1;           //select '1' to be written to the Cause register indicating an invalid operation
330                 CAUSE_EN = 1'b1;            //enable writing to the Cause register
331                 EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
332                 PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
333                 Next_State = 3'b000;        //next state is FETCH
334                 STATE = 6'b100100;          //output state number for debug only. code 36
335             end
336             else if ((BF == 1'b0) && (OF == 1'b1)   && (Funct[0] == 1'b1)) begin        //state 30
337                 CAUSE_SEL = 1'b0;           //select '1' to be written to the Cause register indicating an arithmetic overflow
338                 CAUSE_EN = 1'b1;            //enable writing to the Cause register
339                 EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
340                 PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
341                 Next_State = 3'b000;        //next state is FETCH
342                 STATE = 6'b011110;          //output state number for debug only
343             end
344             else begin                      //state 5
345                 REG_WS = 1'b1;              //enable writing to the Register File
346                 Next_State = 3'b000;        //next state is FETCH
347                 STATE = 6'b000101;          //output state number for debug only
348             end
349         end
350         else if ((Opcode == OP_ADDi) || (Opcode == OP_LW) || (Opcode == OP_LBu) || (Opcode == OP_LB) ||(Opcode == OP_LHu) || (Opcode == OP_LH) || (Opcode == OP_SB)
```

Figure 37. Sequence Controller Design Code Part 7

```
350         else if ((Opcode == OP_ADDi) || (Opcode == OP_LW) || (Opcode == OP_LBu) || (Opcode == OP_LB) ||(Opcode == OP_LHu) || (Opcode == OP_LH) || (Opcode == OP_SB)
            || (Opcode == OP_SH) ||(Opcode == OP_SW)) begin     //various I-Type
351             if ((Opcode == OP_ADDi) && (OF == 1'b0)) begin  //state 8
352                 REG_WS = 1'b1;              //enable writing to the Register File
353                 Next_State = 3'b000;        //next state is FETCH
354                 STATE = 6'b001000;          //output state number for debug only
355             end
356             else if ((Opcode == OP_ADDi) && (OF == 1'b1)) begin //state 30
357                 CAUSE_SEL = 1'b0;           //select '1' to be written to the Cause register indicating an arithmetic overflow
358                 CAUSE_EN = 1'b1;            //enable writing to the Cause register
359                 EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
360                 PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
361                 Next_State = 3'b000;        //next state is FETCH
362                 STATE = 6'b011110;          //output state number for debug only
363             end
364             else if (Opcode == OP_LH) begin //state 25
365                 IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
366                 MEM_OE = 1'b1;              //enable reading from RAM
367                 REG_DATA_SEL = 3'b100;      //select the sign extended half word to be written into the IR
368                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
369                 MEMtoREG = 3'b100;          //select the content of the IR register to be written to the Register File
370                 Next_State = 3'b100;        //next state is WRITEBACK
371                 STATE = 6'b011001;          //output state number for debug only
372             end
373             else if (Opcode == OP_SW) begin //state 17
374                 IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
375                 MEM_DATA_SEL = 2'b00;       //select the content of Reg2 to be written to memory
376                 Next_State = 3'b100;        //next state is WRITEBACK
377                 STATE = 6'b010001;          //output state number for debug only
378             end
379             else if (Opcode == OP_LBu) begin    //state 22
380                 IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
381                 MEM_OE = 1'b1;              //enable reading from RAM
382                 REG_DATA_SEL = 3'b001;      //select the zero sign extended byte to be written into the IR
383                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
384                 MEMtoREG = 3'b100;          //select the content of the IR register to be written to the Register File
385                 Next_State = 3'b100;        //next state is WRITEBACK
386                 STATE = 6'b010110;          //output state number for debug only
387             end
388             else if (Opcode == OP_LW) begin //state 18
389                 IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
390                 MEM_OE = 1'b1;              //enable reading from RAM
391                 REG_DATA_SEL = 3'b000;      //select RAM data to be written into the IR
392                 Reg_Dest = 2'b00;           //select Rt to be the Register File address
393                 MEMtoREG = 3'b100;          //select the content of the IR register to be written to the Register File
394                 Next_State = 3'b100;        //next state is WRITEBACK
395                 STATE = 6'b010010;          //output state number for debug only
396             end
```

Figure 38. Sequence Controller Design Code Part 8

```
            else if (Opcode == OP_LB) begin //state 23
                IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
                MEM_OE = 1'b1;              //enable reading from RAM
                REG_DATA_SEL = 3'b010;      //select sign extended byte to be written into the IR
                Reg_Dest = 2'b00;           //select Rt to be the Register File address
                MEMtoREG = 3'b100;          //select the content of the IR register to be written to the Register File
                Next_State = 3'b100;        //next state is WRITEBACK
                STATE = 6'b010111;          //output state number for debug only
            end
            else if (Opcode == OP_LHu) begin    //state 24
                IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
                MEM_OE = 1'b1;              //enable reading from RAM
                REG_DATA_SEL = 3'b011;      //select the zero sign extended half word to be written into the IR
                Reg_Dest = 2'b00;           //select Rt to be the Register File address
                MEMtoREG = 3'b100;          //select the content of the IR register to be written to the Register File
                Next_State = 3'b100;        //next state is WRITEBACK
                STATE = 6'b011000;          //output state number for debug only
            end
            else if (Opcode == OP_SB) begin //state 26
                IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
                MEM_DATA_SEL = 2'b01;       //select the sign extended byte to be written to memory
                Next_State = 3'b100;        //next state is WRITEBACK
                STATE = 6'b011010;          //output state number for debug only
            end
            else if (Opcode == OP_SH) begin //state 27
                IorD = 1'b1;                //Select the content of the ALU register as the address to RAM
                MEM_DATA_SEL = 2'b10;       //select the sign extended half word to be written to memory
                Next_State = 3'b100;        //next state is WRITEBACK
                STATE = 6'b011011;          //output state number for debug only
            end
            else begin                          //go to invalid instruction state 31
                CAUSE_SEL = 1'b1;           //select '1' to be written to the Cause register indicating an invalid operation
                CAUSE_EN = 1'b1;            //enable writing to the Cause register
                EPC_EN = 1'b1;              //enable writting the current PC to the EPC register
                PC_SEL = 3'b100;            //select the incremented PC to be the next PC value
                Next_State = 3'b000;        //next state is FETCH
                STATE = 6'b011111;          //output state number for debug only
            end
        end
        else begin                          //go to invalid instruction state 31
            CAUSE_SEL = 1'b1;               //select '1' to be written to the Cause register indicating an invalid operation
            CAUSE_EN = 1'b1;                //enable writing to the Cause register
            EPC_EN = 1'b1;                  //enable writting the current PC to the EPC register
            PC_SEL = 3'b100;                //select the incremented PC to be the next PC value
            Next_State = 3'b000;            //next state is FETCH
            STATE = 6'b100101;              //output state number for debug only. code 37
        end
    end
```

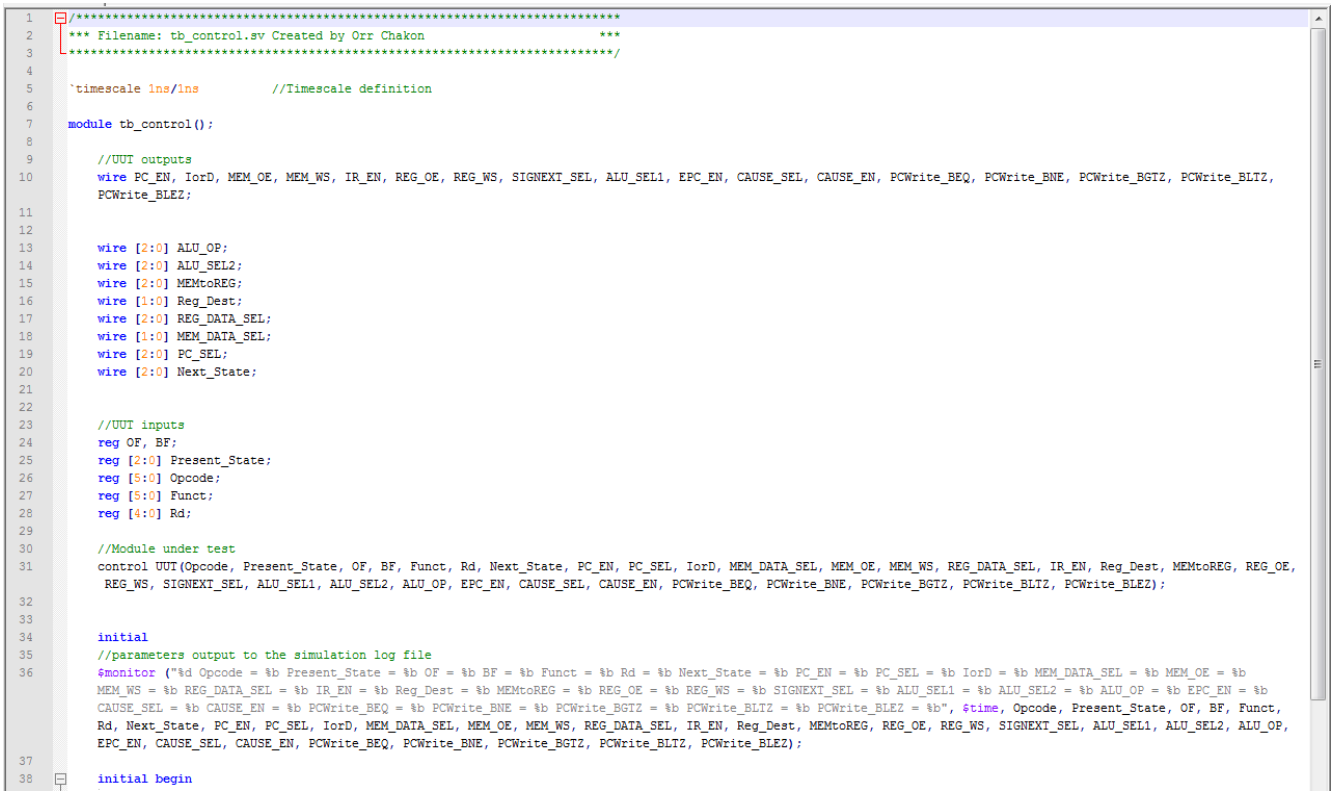
Figure 39. Sequence Controller Design Code Part 9

```
445         4: begin
446             if ((Opcode == OP_SW) || (Opcode == OP_SH) || (Opcode == OP_SB)) begin  //state 34
447                 MEM_WS = 1'b1;                  //enable write to memory
448                 Next_State = 3'b000;            //next state is FETCH
449                 STATE = 6'b100010;              //output state number for debug only. code 34
450             end
451             else begin                          //state 19
452                 REG_WS = 1'b1;                  //enable writing to the Register File
453                 Next_State = 3'b000;            //next state is FETCH
454                 STATE = 6'b010011;              //output state number for debug only
455             end
456         end
457         default: begin                          //go to invalid instruction state 31
458             CAUSE_SEL = 1'b1;                   //select '1' to be written to the Cause register indicating an invalid operation
459             CAUSE_EN = 1'b1;                    //enable writing to the Cause register
460             EPC_EN = 1'b1;                      //enable writting the current PC to the EPC register
461             PC_SEL = 3'b100;                    //select the incremented PC to be the next PC value
462             Next_State = 3'b000;                //next state is FETCH
463             STATE = 6'b100110;                  //output state number for debug only. code 38
464         end
465     endcase
466 end
467
468 endmodule
```

Figure 40. Sequence Controller Design Code Part 10

Test Bench:

The Test Bench program for the Sequence Controller is shown in Figures 41 and 42. Extensive testing of the Sequence Controller at the module level is rather pointless because of the need to verify the operation and timing of the control logic once it is connected to the rest of the components in order to verify complete functionality. As shown in Figures 41 and 42, testing at the module level consisted of a single test of the Load Word (LW) function. The LW function was chosen because it went through all 5 processor cycles (i.e. FETCH, DECODE, EXECUTE, MEMORY, and WRITEBACK), reaching every state of the Sequence Controller state machine. This test was done to make sure the state machine concept was operational. Extensive testing was done on the Sequence Controller at the top processor level. Several bugs where discovered and fixed to make the processor fully functional.



```
/*********************************************************************
 *** Filename: tb_control.sv Created by Orr Chakon                 ***
 *********************************************************************/

`timescale 1ns/1ns          //Timescale definition

module tb_control();

    //UUT outputs
    wire PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ,
    PCWrite_BLEZ;


    wire [2:0] ALU_OP;
    wire [2:0] ALU_SEL2;
    wire [2:0] MEMtoREG;
    wire [1:0] Reg_Dest;
    wire [2:0] REG_DATA_SEL;
    wire [1:0] MEM_DATA_SEL;
    wire [2:0] PC_SEL;
    wire [2:0] Next_State;


    //UUT inputs
    reg OF, BF;
    reg [2:0] Present_State;
    reg [5:0] Opcode;
    reg [5:0] Funct;
    reg [4:0] Rd;

    //Module under test
    control UUT(Opcode, Present_State, OF, BF, Funct, Rd, Next_State, PC_EN, PC_SEL, IorD, MEM_DATA_SEL, MEM_OE, MEM_WS, REG_DATA_SEL, IR_EN, Reg_Dest, MEMtoREG, REG_OE,
     REG_WS, SIGNEXT_SEL, ALU_SEL1, ALU_SEL2, ALU_OP, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ);


    initial
    //parameters output to the simulation log file
    $monitor ("%d Opcode = %b Present_State = %b OF = %b BF = %b Funct = %b Rd = %b Next_State = %b PC_EN = %b PC_SEL = %b IorD = %b MEM_DATA_SEL = %b MEM_OE = %b
    MEM_WS = %b REG_DATA_SEL = %b IR_EN = %b Reg_Dest = %b MEMtoREG = %b REG_OE = %b REG_WS = %b SIGNEXT_SEL = %b ALU_SEL1 = %b ALU_SEL2 = %b ALU_OP = %b EPC_EN = %b
    CAUSE_SEL = %b CAUSE_EN = %b PCWrite_BEQ = %b PCWrite_BNE = %b PCWrite_BGTZ = %b PCWrite_BLTZ = %b PCWrite_BLEZ = %b", $time, Opcode, Present_State, OF, BF, Funct,
    Rd, Next_State, PC_EN, PC_SEL, IorD, MEM_DATA_SEL, MEM_OE, MEM_WS, REG_DATA_SEL, IR_EN, Reg_Dest, MEMtoREG, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, ALU_SEL2, ALU_OP,
    EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ);

    initial begin
```

Figure 41. Sequence Controller Test Bench Part 1

```
    initial begin

        //testing Load Word function
        $write ("\n");
        Opcode = 6'b100011;         //Load Word function
        OF = 1'b0;                  //initialize OF
        BF = 1'b0;                  //initialize BF
        Funct = 6'b000000;          //initialize Funct
        Rd = 5'b00000;              //initialize Rd
        Present_State = 3'b000;     //initialize state variable
    #0  $display ("Testing Load Word (LW) Function");

    #50 $display ("Simulation of rising clock edge 1");
        Present_State = 3'b001;

    #50 $display ("Simulation of rising clock edge 2");
        Present_State = 3'b010;

    #50 $display ("Simulation of rising clock edge 3");
        Present_State = 3'b011;

    #50 $display ("Simulation of rising clock edge 4");
        Present_State = 3'b100;

    #100    $finish;            //Terminate simulation after 100ns
    end                     //End of simulation
endmodule

erilog file   length : 2,707   lines : 65   Ln : 22   Col : 5   Sel : 0 | 0   Unix (LF)   UTF-8   INS
```

Figure 42. Sequence Controller Test Bench Part 2

Test Results:

Simulation results for the Sequence Controller are shown in Figure 43.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct  9 20:02 2016

Testing Load Word (LW) Function
                    0 Opcode = 0100011 Present_State = 000 OF = 0 BF = 0 Funct = 0000000 Rd = 00000 Next_State = 001 PC_EN = 1 PC_SEL = 000 IorD = 0 MEM_DATA_SEL = 00 MEM_OE
                    = 1 MEM_WS = 0 REG_DATA_SEL = 000 IR_EN = 1 Reg_Dest = 00 MEMtoREG = 00 REG_OE = 1 REG_WS = 0 SIGNEXT_SEL = 0 ALU_SEL1 = 0 ALU_SEL2 = 001 ALU_REG_EN = 0
                    ALU_OP = 000 EPC_EN = 0 CAUSE_SEL = 0 CAUSE_EN = 0 PCWrite_BEQ = 0 PCWrite_BNE = 0 PCWrite_BGTZ = 0 PCWrite_BLTZ = 0 PCWrite_BLEZ = 0
Simulation of rising clock edge 1
                   50 Opcode = 0100011 Present_State = 001 OF = 0 BF = 0 Funct = 0000000 Rd = 00000 Next_State = 010 PC_EN = 1 PC_SEL = 000 IorD = 0 MEM_DATA_SEL = 00 MEM_OE
                   = 1 MEM_WS = 0 REG_DATA_SEL = 000 IR_EN = 1 Reg_Dest = 00 MEMtoREG = 00 REG_OE = 1 REG_WS = 0 SIGNEXT_SEL = 0 ALU_SEL1 = 0 ALU_SEL2 = 011 ALU_REG_EN = 0
                   ALU_OP = 000 EPC_EN = 0 CAUSE_SEL = 0 CAUSE_EN = 0 PCWrite_BEQ = 0 PCWrite_BNE = 0 PCWrite_BGTZ = 0 PCWrite_BLTZ = 0 PCWrite_BLEZ = 0
Simulation of rising clock edge 2
                  100 Opcode = 0100011 Present_State = 010 OF = 0 BF = 0 Funct = 0000000 Rd = 00000 Next_State = 011 PC_EN = 1 PC_SEL = 000 IorD = 0 MEM_DATA_SEL = 00 MEM_OE
                  = 1 MEM_WS = 0 REG_DATA_SEL = 000 IR_EN = 1 Reg_Dest = 00 MEMtoREG = 00 REG_OE = 1 REG_WS = 0 SIGNEXT_SEL = 0 ALU_SEL1 = 1 ALU_SEL2 = 010 ALU_REG_EN = 0
                  ALU_OP = 000 EPC_EN = 0 CAUSE_SEL = 0 CAUSE_EN = 0 PCWrite_BEQ = 0 PCWrite_BNE = 0 PCWrite_BGTZ = 0 PCWrite_BLTZ = 0 PCWrite_BLEZ = 0
Simulation of rising clock edge 3
                  150 Opcode = 0100011 Present_State = 011 OF = 0 BF = 0 Funct = 0000000 Rd = 00000 Next_State = 100 PC_EN = 1 PC_SEL = 000 IorD = 1 MEM_DATA_SEL = 00 MEM_OE
                  = 1 MEM_WS = 0 REG_DATA_SEL = 000 IR_EN = 1 Reg_Dest = 00 MEMtoREG = 00 REG_OE = 1 REG_WS = 0 SIGNEXT_SEL = 0 ALU_SEL1 = 1 ALU_SEL2 = 010 ALU_REG_EN = 0
                  ALU_OP = 000 EPC_EN = 0 CAUSE_SEL = 0 CAUSE_EN = 0 PCWrite_BEQ = 0 PCWrite_BNE = 0 PCWrite_BGTZ = 0 PCWrite_BLTZ = 0 PCWrite_BLEZ = 0
Simulation of rising clock edge 4
                  200 Opcode = 0100011 Present_State = 100 OF = 0 BF = 0 Funct = 0000000 Rd = 00000 Next_State = 000 PC_EN = 1 PC_SEL = 000 IorD = 1 MEM_DATA_SEL = 00 MEM_OE
                  = 1 MEM_WS = 0 REG_DATA_SEL = 000 IR_EN = 1 Reg_Dest = 00 MEMtoREG = 01 REG_OE = 1 REG_WS = 1 SIGNEXT_SEL = 0 ALU_SEL1 = 1 ALU_SEL2 = 010 ALU_REG_EN = 0
                  ALU_OP = 000 EPC_EN = 0 CAUSE_SEL = 0 CAUSE_EN = 0 PCWrite_BEQ = 0 PCWrite_BNE = 0 PCWrite_BGTZ = 0 PCWrite_BLTZ = 0 PCWrite_BLEZ = 0
$finish called from file "tb_control.sv", line 67.
$finish at simulation time                300
           V C S   S i m u l a t i o n   R e p o r t
Time: 300 ns
CPU Time:      0.480 seconds;       Data structure size:   0.0Mb
Sun Oct  9 20:02:20 2016

```

Figure 43. Sequence Controller Simulation Results

6.6 Multiplexor Design and Test

Design Code:

The multiplexor design code is shown in Figure 44. There are actually three different types of multiplexors in this project's processor design as described in section 4.1. Due to design similarity, I'm only showing documentation for the 8x1 multiplexor.

```
/*********************************************************************
*** Filename: mux8.v Created by Orr Chakon                        ***
*********************************************************************/

`timescale 1 ns / 1 ns

module mux8(A, B, C, D, E, F, G, H, SEL, OUT);

// Port declaration
   output [31:0] OUT;
   input [31:0] A, B, C, D, E, F, G, H;
   input [2:0] SEL;


// Internal variable declarations
   reg [31:0] OUT;

// Model of multiplexer
   always @(A, B, C, D, E, F, G, H, SEL)    //triggered off input change
    begin
        case(SEL)
        3'b000:OUT=A;
        3'b001:OUT=B;
        3'b010:OUT=C;
        3'b011:OUT=D;
        3'b100:OUT=E;
        3'b101:OUT=F;
        3'b110:OUT=G;
        3'b111:OUT=H;
        default:OUT=32'bxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
        endcase
    end

endmodule

```

Figure 44. Multiplexor Design Code

Test Bench:

The Test Bench Program from the multiplexor is shown in Figures 45 and 46.

```
/*********************************************************************
*** Filename: tb_mux8.v Created by Orr Chakon                     ***
*********************************************************************/
`timescale 1 ns /1 ns
module tb_mux8();

// Port declaration
   reg [31:0] A, B, C, D, E, F, G, H;
   reg [2:0] SEL;
   wire [31:0] OUT;

// UUT instantiation

   mux8 UUT (A, B, C, D, E, F, G, H, SEL, OUT);

initial begin

  #10 $write("\n"); //new line used for clarity
      A=32'b00000000000000000000000000000000;
      B=32'b11111111111111111111111111111111;
      C=32'b01010101010101010101010101010101;
      D=32'b10101010101010101010101010101010;
      E=32'b00000000111111110000000011111111;
      F=32'b11111111000000001111111100000000;
      G=32'b00000000000000001111111111111111;
      H=32'b11111111111111110000000000000000;
   #10 SEL=3'b000;
      $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);

  #10 $write("\n"); //new line used for clarity
      A=32'b00000000000000000000000000000000;
      B=32'b11111111111111111111111111111111;
      C=32'b01010101010101010101010101010101;
      D=32'b10101010101010101010101010101010;
      E=32'b00000000111111110000000011111111;
      F=32'b11111111000000001111111100000000;
      G=32'b00000000000000001111111111111111;
      H=32'b11111111111111110000000000000000;
   #10 SEL=3'b001;
      $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);

        #10 $write("\n");   //new line used for clarity
      A=32'b00000000000000000000000000000000;
      B=32'b11111111111111111111111111111111;
      C=32'b01010101010101010101010101010101;
      D=32'b10101010101010101010101010101010;
      E=32'b00000000111111110000000011111111;
      F=32'b11111111000000001111111100000000;
      G=32'b00000000000000001111111111111111;
      H=32'b11111111111111110000000000000000;
   #10 SEL=3'b010;
      $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);

erilog file    length : 3,916   lines : 93    Ln : 1   Col : 1   Sel : 0 | 0    Unix (LF)
```

Figure 45. Multiplexor Test Bench Part 1

```
53
54      #10 $write("\n"); //new line used for clarity
55          A=32'b00000000000000000000000000000000;
56          B=32'b11111111111111111111111111111111;
57          C=32'b01010101010101010101010101010101;
58          D=32'b10101010101010101010101010101010;
59          E=32'b00000000111111110000000011111111;
60          F=32'b11111111000000001111111100000000;
61          G=32'b00000000000000001111111111111111;
62          H=32'b11111111111111110000000000000000;
63       #10 SEL=3'b011;
64          $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);
65
66      #10 $write("\n"); //new line used for clarity
67          A=32'b00000000000000000000000000000000;
68          B=32'b11111111111111111111111111111111;
69          C=32'b01010101010101010101010101010101;
70          D=32'b10101010101010101010101010101010;
71          E=32'b00000000111111110000000011111111;
72          F=32'b11111111000000001111111100000000;
73          G=32'b00000000000000001111111111111111;
74          H=32'b11111111111111110000000000000000;
75       #10 SEL=3'b100;
76          $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);
77
78      #10 $write("\n"); //new line used for clarity
79          A=32'b00000000000000000000000000000000;
80          B=32'b11111111111111111111111111111111;
81          C=32'b01010101010101010101010101010101;
82          D=32'b10101010101010101010101010101010;
83          E=32'b00000000111111110000000011111111;
84          F=32'b11111111000000001111111100000000;
85          G=32'b00000000000000001111111111111111;
86          H=32'b11111111111111110000000000000000;
87       #10 SEL=3'b101;
88          $strobe("%d UUT: A = %b B = %b C = %b D = %b E = %b F = %b G = %b H = %b SEL = %b OUT = %b" , $time, A, B, C, D, E, F, G, H, SEL, OUT);
89
90      #20 $finish;
91       end
92  endmodule
93
```

erilog file | length : 3,916 lines : 93 | Ln : 1 Col : 1 Sel : 0 | 0 | Unix (LF)

Figure 46. Multiplexor Test Bench Part 2

Test Results:

The Simulation results for the multiplexor are shown in Figure 47.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct 28 22:16 2016

                20 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
                10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
                11111111111111110000000000000000 SEL = 000 OUT = 00000000000000000000000000000000

                40 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
                10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
                11111111111111110000000000000000 SEL = 001 OUT = 11111111111111111111111111111111

                60 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
                10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
                11111111111111110000000000000000 SEL = 010 OUT = 01010101010101010101010101010101

                80 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
                10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
                11111111111111110000000000000000 SEL = 011 OUT = 10101010101010101010101010101010

               100 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
               10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
               11111111111111110000000000000000 SEL = 100 OUT = 00000000111111110000000011111111

               120 UUT: A = 00000000000000000000000000000000 B = 11111111111111111111111111111111 C = 01010101010101010101010101010101 D =
               10101010101010101010101010101010 E = 00000000111111110000000011111111 F = 11111111000000001111111100000000 G = 00000000000000001111111111111111 H =
               11111111111111110000000000000000 SEL = 101 OUT = 11111111000000001111111100000000
$finish called from file "tb_mux8.v", line 90.
$finish at simulation time                140
         V C S   S i m u l a t i o n   R e p o r t
Time: 140 ns
CPU Time:      0.520 seconds;       Data structure size:   0.0Mb
Fri Oct 28 22:16:35 2016
```

Figure 47. Multiplexor Simulation Results

6.7 Program Counter Design and Test

Design Code:

The Program Counter (PC) design code is shown in Figure 48. As shown in the design code, the active low reset signal (RST_n) was priority over the load function. The RST_n function is also asynchronous while the load function is synchronized to the system clock (CLK).

```
/*****************************************************************************
*** Filename: pc.v Created by Orr Chakon                                   ***
*****************************************************************************/

`timescale 1 ns / 1 ns

module pc(Dout, RST_n, CLK, Din, LD);

// Port declaration
   output [31:0] Dout;
   input [31:0] Din;
   input RST_n, CLK, LD;


// Internal variable declarations
   reg [31:0] Dout;

   always @(posedge CLK or negedge RST_n)
   begin
    if(!RST_n)          //Reset has top priority
    Dout <= 32'b0;
    else if (LD)        //only update when LD is set true
    Dout <= Din;
    else
    Dout <= Dout;
   end

endmodule

```

Figure 48. Program Counter Design Code

Test Bench:

The Test Bench Program for the Program Counter is shown in Figure 49. In order to test the Program Counter design, all possible combinations of the Din input were loaded and the reset signal was toggled to make sure the Dout output remained low while the module was in reset.

```
/*********************************************************************************
*** Filename: pc.v Created by Orr Chakon                                      ***
*********************************************************************************/

`timescale 1 ns / 1 ns

module tb_pc();

    reg [31:0] Din;                    //regesters required to hold the inputs
    reg RST_n, CLK, LD;
    wire [31:0] Dout;                  // Wired used to capture the output

    pc UUT(Dout, RST_n, CLK, Din, LD);  //Module under test

    initial
    CLK = 1'b0;        // Initialize clock to 0

    always #5 CLK = ~CLK;  // 10ns clock period

    initial
    //parameters output to the simulation log file
    $monitor ("%d Dout = %b Din = %b LD = %b CLK = %b RST_n = %b", $time, Dout, Din, LD, CLK, RST_n);

    initial begin
        $vcdpluson;         //Enable graphical viewer

        LD = 1'b1;
        Din = 32'b111;
        RST_n = 1'b1;

    #15 Din = 32'b110;
    #15 Din = 32'b101;
    #15 Din = 32'b100;
    #15 Din = 32'b111;
        RST_n = 1'b0;
    #15 Din = 32'b110;      //output not affected as reset held low
    #15 Din = 32'b101;      //output not affected as reset held low
    #15 Din = 32'b100;      //output not affected as reset held low
    #15 Din = 32'b011;      //output not affected as reset held low
        RST_n = 1'b1;
    #15 Din = 32'b010;
    #15 Din = 32'b001;
    #15 Din = 32'b000;
    #15 Din = 32'b111;
        LD = 1'b0;          //output not affected as LD is disabled
    #15 Din = 32'b110;      //output not affected as LD is disabled
    #15 Din = 32'b101;      //output not affected as LD is disabled
    #15 Din = 32'b100;      //output not affected as LD is disabled
    #15 $finish;            //Terminate simulation after 15ns
    end                     //End of simulation
endmodule

/erilog file                                  length : 1,602   lines : 52      Ln : 1
```

Figure 49. Program Counter Test Bench

Test Results:

The simulation results for the Program Counter design are shown in Figure 50. The results show the output (Dout) staying low while the reset signal (RST_N) is low.

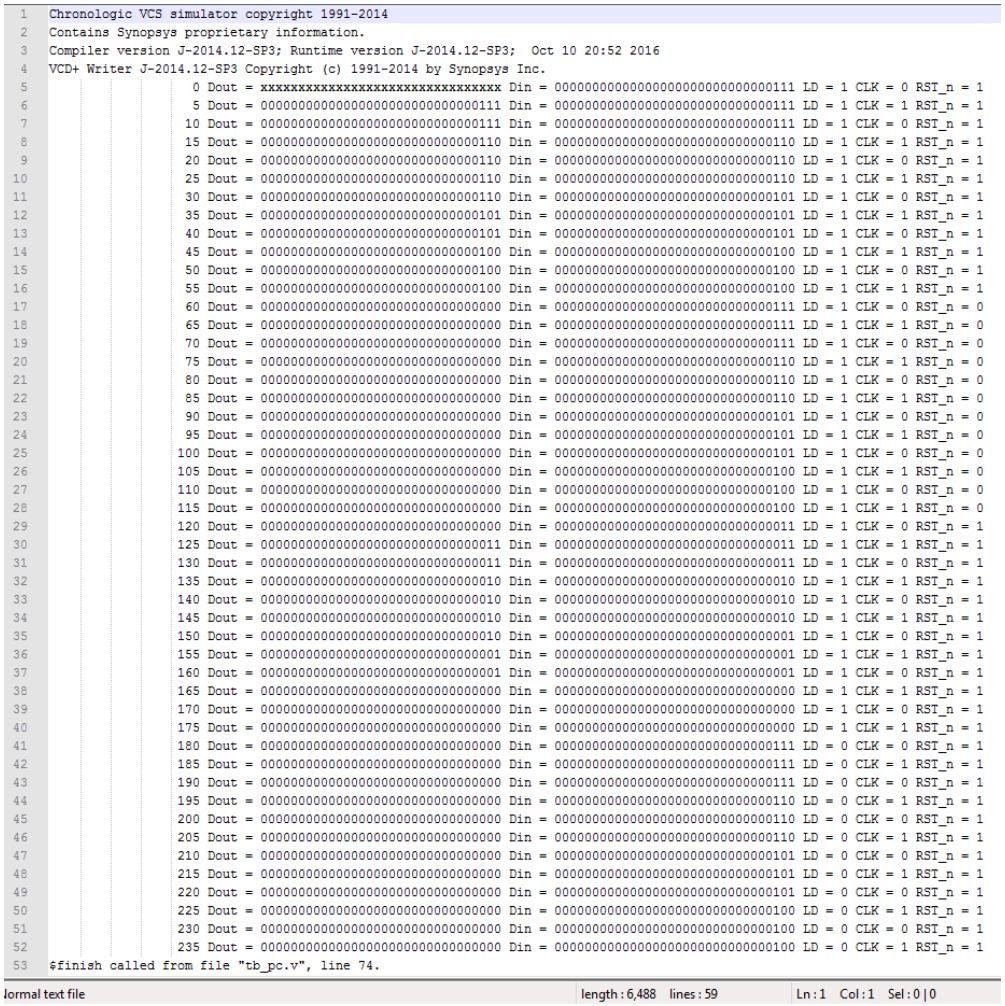


Figure 50. Program Counter Simulation Results

6.8 RAM Design and Test

Design Code:

The RAM module design code is shown in Figure 51. As shown in the design code, the project used a 256 x 32 memory array using the WS signal as an edge sensitive signal in order to create the flip flops. The system clock could have been used instead of WS, but that would have added a clock delay to each executed instruction. (See section 8 for more on processor timing.) A Read access from RAM is completely asynchronous and is controlled by the OE signal. Note that the Read function is sensitive to the address signal (Addr), allowing for block reads while OE is set true and the Addr signal is changing. For resource conservation purposes, the project only used 8 bits for the Addr line while the same signal is 32-bits wide on the top level processor architecture shown in Figure 1. This can easily be changed in the design code if a bigger memory block is needed for future use.

```
1  /*************************************************************************
2  *** Filename: RAM.v Created by Orr Chakon                              ***
3  *************************************************************************/
4
5  `timescale 1ns/1ns
6  module RAM(DataOut, Addr, WriteData, OE, WS);
7
8  //Port declaration
9     output [31:0] DataOut;   //data being read out of RAM
10    input [31:0] WriteData;  //data being written to RAM
11    input [7:0] Addr;        //RAM address
12    input OE, WS;            //OE = read enable, WS = write enable
13
14 //Internal wire declaration
15
16    reg [31:0]mem [0:255]; //256x32 memory block
17    reg [31:0]DataOut;   //used for data output
18
19 always@(OE, Addr) begin    //asynchronous read
20       if (OE)               //active high OE
21       DataOut = mem[Addr];  //work at location Addr goes to output
22       else
23       DataOut = DataOut;    //output says the same if OE not enabled
24    end
25
26
27 always@(posedge WS)    //write routine. creates flip-flops. Active high WS.
28    begin
29          mem[Addr] <= WriteData; //word written at location Addr
30    end
31 endmodule
32
33
```

Figure 51. RAM Design Code

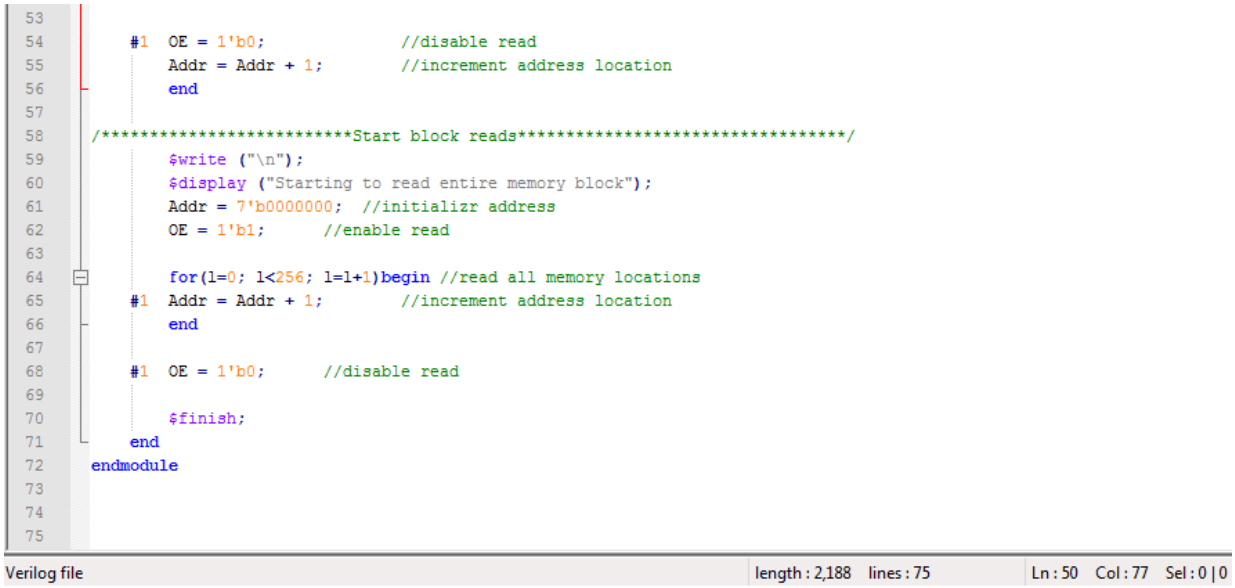
Test Bench:

The Test Bench Program for the RAM module is shown in Figures 52 and 53. In order to test the RAM module, a process that writes the current address to each corresponding location was first set up. The next step was to set up a process to read each memory location individually, followed by a process to read the entire memory block as a block read without toggling the OE signal.

```
1  /*********************************************************************
2  *** Filename: tb_RAM.v Created by Orr Chakon                       ***
3  *********************************************************************/
4
5  `timescale 1 ns / 1 ns          //Timescale definition
6
7  module tb_RAM();
8
9  //port declaration
10     reg OE, WS;
11     reg [7:0] Addr;
12     reg [31:0] WriteData;
13     wire [31:0] DataOut;
14     integer l;
15
16     RAM UUT(DataOut, Addr, WriteData, OE, WS);  //Module under test
17
18     initial
19     //parameters output to the simulation log file
20     $monitor ("%d DataOut = %b Addr = %b WriteData = %b OE = %b WS = %b", $time, DataOut, Addr, WriteData, OE, WS);
21
22     initial begin
23         $vcdpluson;         //Enable graphical viewer
24
25         $monitoroff;    //turn off $monitor for write process
26
27 /*************************Start write process*********************************/
28         OE = 1'b0;  //initialize OE to disable reading
29         Addr = 7'b0000000;  //initializr address
30         WriteData = 32'b00000000000000000000000000000000;   //initialize data input
31
32
33         for(l=0; l<256; l=l+1)begin //increment for each memory location.
34
35     #1  WS = 1'b1;  //write address to each memory location
36
37     #1  WS = 1'b0;  //reset WS
38
39         Addr = Addr + 1;            //increment address location
40         WriteData = WriteData + 1;  //increment Data input
41         end
42
43 /*************************Start read process*********************************/
44
45         WriteData = 32'b00000000000000000000000000000000;   //reset data input
46         Addr = 7'b0000000;  //initializr address
47         $monitoron;     //turn on $monitor for read process
48
49         $display ("Start to read all individual memory locations");
50         for(l=0; l<256; l=l+1)begin //read all memory locations individually
51
52     #1  OE = 1'b1;              //set OE to read data
53
```

Verilog file | length : 2,188 lines : 75 | Ln : 1 Col : 77 Sel : 0 | 0

Figure 52. RAM Test Bench Part 1



```
53
54          #1  OE = 1'b0;              //disable read
55              Addr = Addr + 1;        //increment address location
56              end
57
58      /*************************Start block reads*********************************/
59              $write ("\n");
60              $display ("Starting to read entire memory block");
61              Addr = 7'b0000000;  //initializr address
62              OE = 1'b1;      //enable read
63
64              for(l=0; l<256; l=l+1)begin //read all memory locations
65          #1  Addr = Addr + 1;        //increment address location
66              end
67
68          #1  OE = 1'b0;      //disable read
69
70              $finish;
71          end
72      endmodule
73
74
75
```

Verilog file | length : 2,188 lines : 75 | Ln : 50 Col : 77 Sel : 0 | 0

Figure 53. RAM Test Bench Part 1

Test Results:

The simulation results for the RAM module test are shown in Figures 54 and 55. Since the simulation results are extensive, this project will only show the results for the beginning of both individual and block read processes.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct 29 00:28 2016
VCD+ Writer J-2014.12-SP3 Copyright (c) 1991-2014 by Synopsys Inc.
Start to read all individual memory locations
                 512 DataOut = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Addr = 00000000 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 513 DataOut = 00000000000000000000000000000000 Addr = 00000000 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 514 DataOut = 00000000000000000000000000000000 Addr = 00000001 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 515 DataOut = 00000000000000000000000000000001 Addr = 00000001 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 516 DataOut = 00000000000000000000000000000001 Addr = 00000010 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 517 DataOut = 00000000000000000000000000000010 Addr = 00000010 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 518 DataOut = 00000000000000000000000000000010 Addr = 00000011 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 519 DataOut = 00000000000000000000000000000011 Addr = 00000011 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 520 DataOut = 00000000000000000000000000000011 Addr = 00000100 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 521 DataOut = 00000000000000000000000000000100 Addr = 00000100 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 522 DataOut = 00000000000000000000000000000100 Addr = 00000101 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 523 DataOut = 00000000000000000000000000000101 Addr = 00000101 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 524 DataOut = 00000000000000000000000000000101 Addr = 00000110 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 525 DataOut = 00000000000000000000000000000110 Addr = 00000110 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 526 DataOut = 00000000000000000000000000000110 Addr = 00000111 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 527 DataOut = 00000000000000000000000000000111 Addr = 00000111 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 528 DataOut = 00000000000000000000000000000111 Addr = 00001000 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 529 DataOut = 00000000000000000000000000001000 Addr = 00001000 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 530 DataOut = 00000000000000000000000000001000 Addr = 00001001 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 531 DataOut = 00000000000000000000000000001001 Addr = 00001001 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 532 DataOut = 00000000000000000000000000001001 Addr = 00001010 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 533 DataOut = 00000000000000000000000000001010 Addr = 00001010 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 534 DataOut = 00000000000000000000000000001010 Addr = 00001011 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 535 DataOut = 00000000000000000000000000001011 Addr = 00001011 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 536 DataOut = 00000000000000000000000000001011 Addr = 00001100 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 537 DataOut = 00000000000000000000000000001100 Addr = 00001100 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 538 DataOut = 00000000000000000000000000001100 Addr = 00001101 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 539 DataOut = 00000000000000000000000000001101 Addr = 00001101 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 540 DataOut = 00000000000000000000000000001101 Addr = 00001110 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 541 DataOut = 00000000000000000000000000001110 Addr = 00001110 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 542 DataOut = 00000000000000000000000000001110 Addr = 00001111 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
                 543 DataOut = 00000000000000000000000000001111 Addr = 00001111 WriteData = 00000000000000000000000000000000 OE = 1 WS = 0
                 544 DataOut = 00000000000000000000000000001111 Addr = 00010000 WriteData = 00000000000000000000000000000000 OE = 0 WS = 0
```

Figure 54. RAM Simulation Results Showing Individual Reads

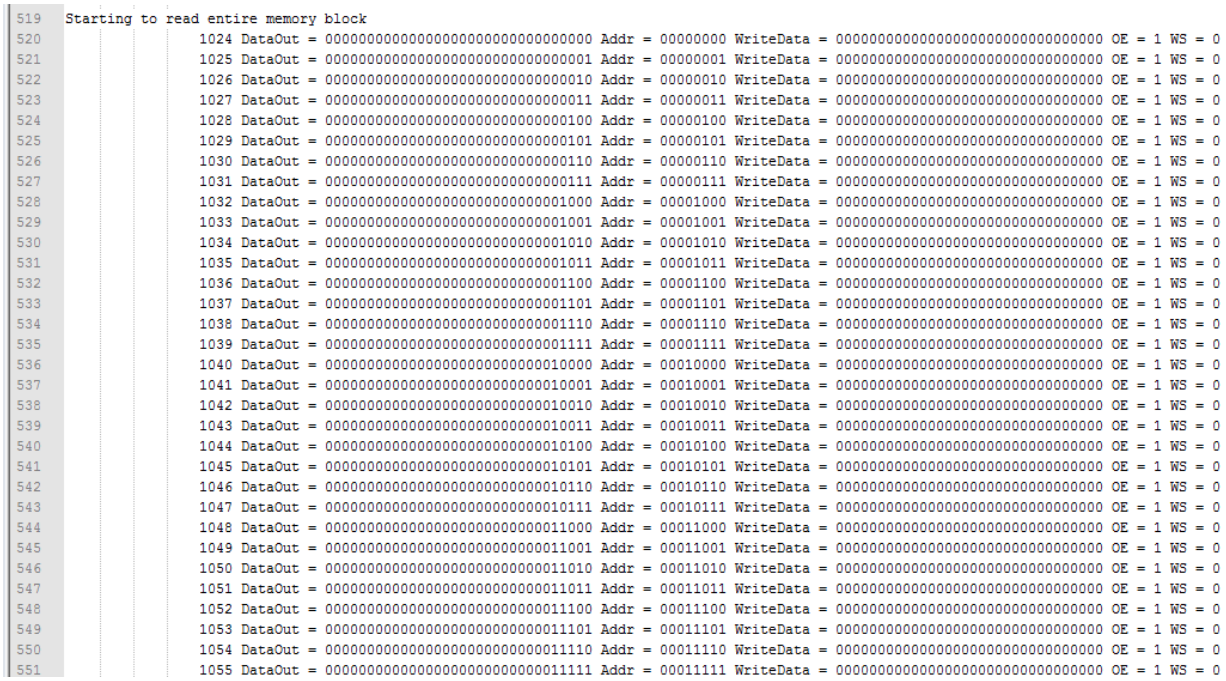


Figure 55. RAM Simulation Results Showing Block Read

6.9 Register File Design and Test

Design Code:

The Register File design code is shown in Figure 56. The Register File design is very similar to the RAM design, only it uses a 32 x 32 memory array and has two I/O ports to facilitate simultaneous reads from two separate addresses.

```
/*
*** Filename: REG_FILE.v Created by Orr Chakon                          ***
*************************************************************************/

`timescale 1ns/1ns
module REG_FILE(Reg1_data, Reg2_data, Reg1, Reg2, Write_Reg, Write_Data, OE, WS);

//Port declaration
   output [31:0] Reg1_data;     //data being read out of Reg1 port
   output [31:0] Reg2_data;     //data being read out of Reg2 port
   input [4:0] Reg1;            //Reg1 address
   input [4:0] Reg2;            //Reg2 address
   input [4:0] Write_Reg;       //address of write register
   input [31:0] Write_Data;     //write register data
   input OE, WS;                //OE = read enable, WS = write enable

//Internal wire declaration

   reg [31:0]mem [0:31]; //32x32 register file
   reg [31:0]Reg1_data; //used for Reg1_data output
   reg [31:0]Reg2_data; //used for Reg2_data output

always@(OE, Reg1, Reg2) begin  //asynchronous read
     if (OE) begin             //active high OE
     Reg1_data = mem[Reg1];
     Reg2_data = mem[Reg2];
     end
     else begin
     Reg1_data = Reg1_data;
     Reg2_data = Reg2_data;
     end
  end


always@(posedge WS)    //write routine. creates flip-flops. Active high WS.
   begin
         mem[Write_Reg] <= Write_Data;   //write to location specified by Write_Reg
   end
endmodule
```

Figure 56. Register File Design Code

Test Bench:

The test bench program for the Register File design is shown in Figures 57 and 58. The test program is functionally similar to the Test Bench used to test the RAM module only it reads from two locations at a time. Reg1 selects the even locations while Reg2 selects the odds.

```
/*********************************************************************
 *** Filename: tb_REG_FILE.v Created by Orr Chakon                 ***
 *********************************************************************/

`timescale 1 ns / 1 ns          //Timescale definition

module tb_REG_FILE();

//port declaration
    reg OE, WS;
    reg [4:0] Reg1;
    reg [4:0] Reg2;
    reg [4:0] Write_Reg;
    reg [31:0] Write_Data;
    wire [31:0] Reg1_data;
    wire [31:0] Reg2_data;

    integer l;

    REG_FILE UUT(Reg1_data, Reg2_data, Reg1, Reg2, Write_Reg, Write_Data, OE, WS);  //Module under test

    initial
    //parameters output to the simulation log file
    $monitor ("%d Reg1_data = %b Reg2_data = %b Reg1 = %b Reg2 = %b Write_Reg = %b Write_Data = %b OE = %b WS = %b", $time, Reg1_data, Reg2_data, Reg1, Reg2, Write_Reg,
    Write_Data, OE, WS);

    initial begin
        $vcdpluson;         //Enable graphical viewer

        $monitoroff;    //turn off $monitor for write process

/*************************Start write process*********************************/

        OE = 1'b0;      //initialize OE to disable reading
        Write_Reg = 5'b00000;   //initializr address
        Write_Data = 32'b00000000000000000000000000000000;  //initialize data input


        for(l=0; l<32; l=l+1)begin  //write address

    #1  WS = 1'b1;  //write data

    #1  WS = 1'b0;  //reset WS

        Write_Reg = Write_Reg + 1;      //increment address location
        Write_Data = Write_Data + 100;  //increment Data input
        end
```

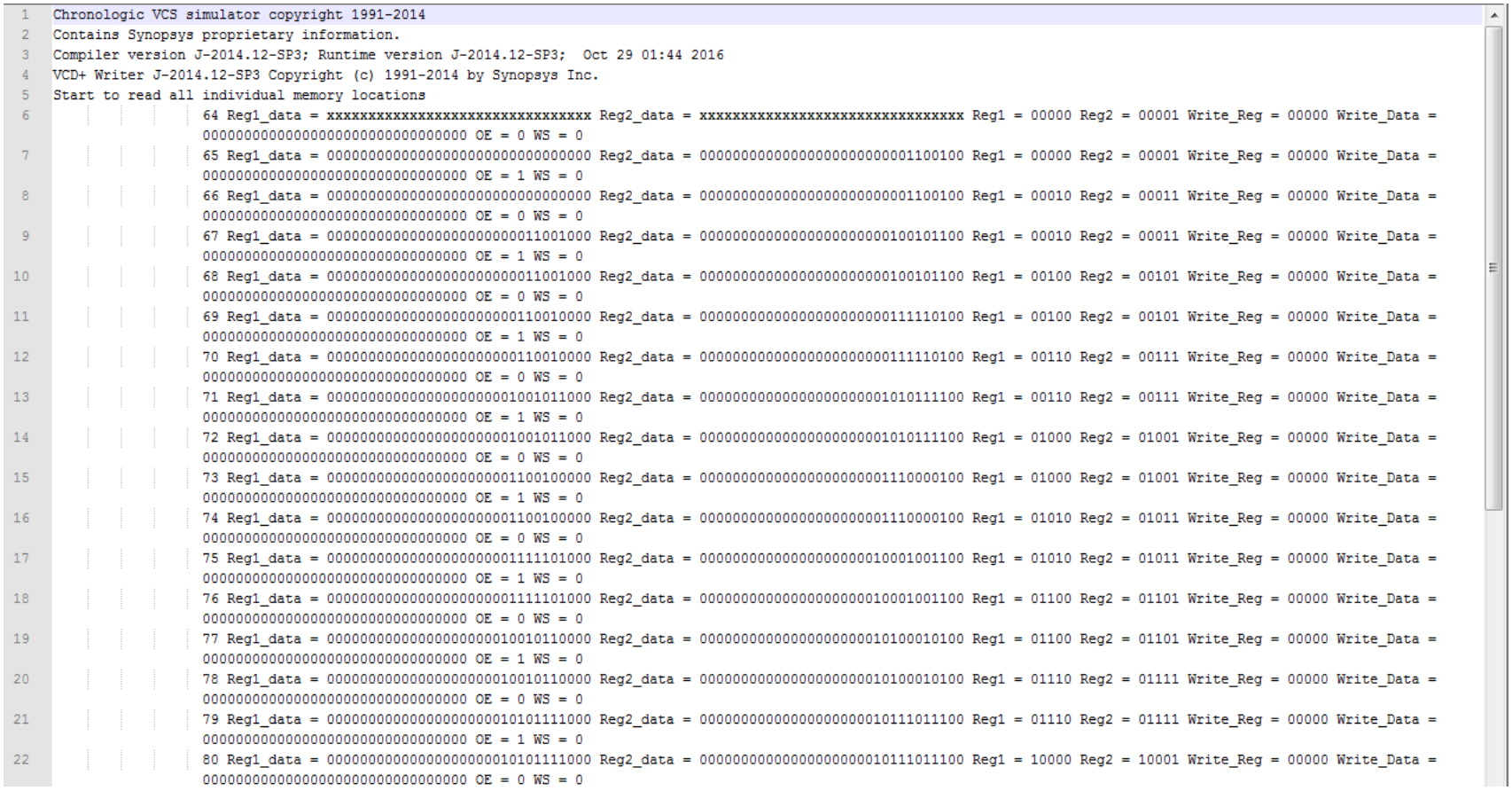
Figure 57. Register File Test Bench Part 1

```
48
49          Write_Data = 32'b00000000000000000000000000000000;  //reset data input
50          Reg1 = 5'b0000000;  //initialize reg1 address
51          Reg2 = 5'b0000001;  //initialize reg2 address
52
53          $monitoron;     //turn on $monitor for read process
54
55  /*************************Start read process*********************************/
56          $display ("Start to read all individual memory locations");
57          for(l=0; l<16; l=l+1)begin  //read all memory locations
58
59      #1  OE = 1'b1;  //set OE to read data
60
61      #1  OE = 1'b0;  //disable read
62          Reg1 = Reg1 + 2;        //increment Reg1 address location
63          Reg2 = Reg2 + 2;        //increment Reg2 address location
64          end
65
66  /*************************Start block reads*********************************/
67
68          $write ("\n");
69          $display ("Starting to read entire memory block");
70          Reg1 = 5'b0000000;  //initialize reg1 address
71          Reg2 = 5'b0000001;  //initialize reg2 address
72
73          OE = 1'b1;      //enable read
74
75          for(l=0; l<16; l=l+1)begin  //read all memory locations
76
77      #1  Reg1 = Reg1 + 2;        //increment Reg1 address location
78          Reg2 = Reg2 + 2;        //increment Reg2 address location
79          end
80
81      #1  OE = 1'b0;      //disable read
82
83          $finish;
84      end
85  endmodule
86
87
```

Figure 58. Register File Test Bench Part 2

Test Results:

Simulation results for the Register File are shown in Figures 59 and 60. Only simulation results for the beginning of each read process are shown.



```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct 29 01:44 2016
VCD+ Writer J-2014.12-SP3 Copyright (c) 1991-2014 by Synopsys Inc.
Start to read all individual memory locations
                64 Reg1_data = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Reg2_data = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Reg1 = 00000 Reg2 = 00001 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                65 Reg1_data = 00000000000000000000000000000000 Reg2_data = 00000000000000000000000001100100 Reg1 = 00000 Reg2 = 00001 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                66 Reg1_data = 00000000000000000000000000000000 Reg2_data = 00000000000000000000000001100100 Reg1 = 00010 Reg2 = 00011 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                67 Reg1_data = 00000000000000000000000011001000 Reg2_data = 00000000000000000000000100101100 Reg1 = 00010 Reg2 = 00011 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                68 Reg1_data = 00000000000000000000000011001000 Reg2_data = 00000000000000000000000100101100 Reg1 = 00100 Reg2 = 00101 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                69 Reg1_data = 00000000000000000000000110010000 Reg2_data = 00000000000000000000000111110100 Reg1 = 00100 Reg2 = 00101 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                70 Reg1_data = 00000000000000000000000110010000 Reg2_data = 00000000000000000000000111110100 Reg1 = 00110 Reg2 = 00111 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                71 Reg1_data = 00000000000000000000001001011000 Reg2_data = 00000000000000000000001010111100 Reg1 = 00110 Reg2 = 00111 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                72 Reg1_data = 00000000000000000000001001011000 Reg2_data = 00000000000000000000001010111100 Reg1 = 01000 Reg2 = 01001 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                73 Reg1_data = 00000000000000000000001100100000 Reg2_data = 00000000000000000000001110000100 Reg1 = 01000 Reg2 = 01001 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                74 Reg1_data = 00000000000000000000001100100000 Reg2_data = 00000000000000000000001110000100 Reg1 = 01010 Reg2 = 01011 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                75 Reg1_data = 00000000000000000000001111101000 Reg2_data = 00000000000000000000010001001100 Reg1 = 01010 Reg2 = 01011 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                76 Reg1_data = 00000000000000000000001111101000 Reg2_data = 00000000000000000000010001001100 Reg1 = 01100 Reg2 = 01101 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                77 Reg1_data = 00000000000000000000010010110000 Reg2_data = 00000000000000000000010100010100 Reg1 = 01100 Reg2 = 01101 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                78 Reg1_data = 00000000000000000000010010110000 Reg2_data = 00000000000000000000010100010100 Reg1 = 01110 Reg2 = 01111 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
                79 Reg1_data = 00000000000000000000010101111000 Reg2_data = 00000000000000000000010111011100 Reg1 = 01110 Reg2 = 01111 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 1 WS = 0
                80 Reg1_data = 00000000000000000000010101111000 Reg2_data = 00000000000000000000010111011100 Reg1 = 10000 Reg2 = 10001 Write_Reg = 00000 Write_Data =
                00000000000000000000000000000000 OE = 0 WS = 0
```

Figure 59. Register File Simulation Results Showing Individual Reads

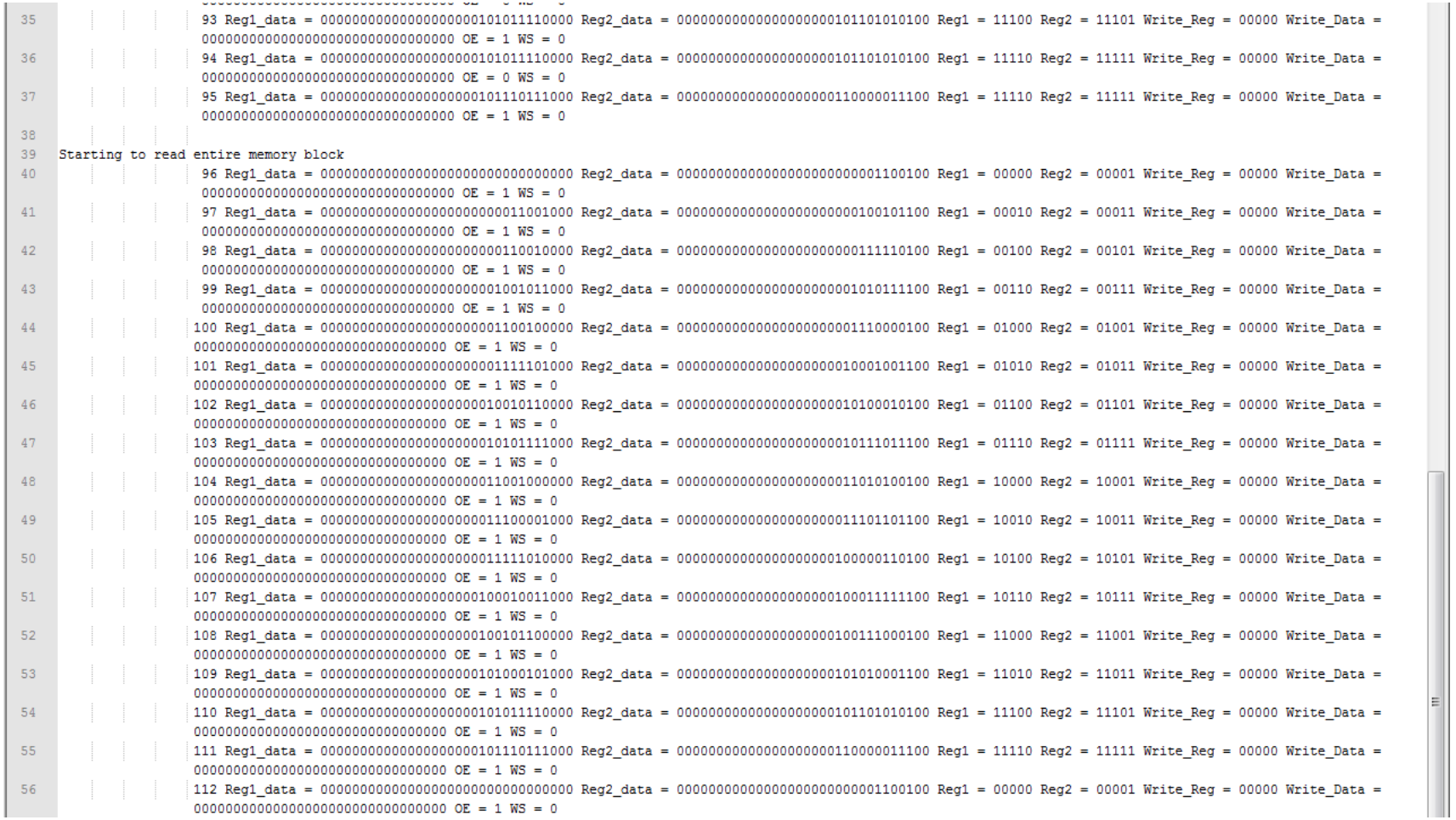


Figure 60. Register File Simulation Results Showing Block Reads

6.11 Scalable Register Design and Test

Design Code:

The Scalable Register Design Code is shown in Figure 61. The Processor design uses several registers of different sizes. Making the inputs and outputs of the Register design scalable allowed me to use one component, instantiated at different sizes, to meet the particular Register requirement. The Register shown in Figure 61 has an enable (EN) input while some of the Registers in the project's processor design do not. For those instances where an enable signal is not desired, a separate Register design with no enable was created, meaning they take in a new value at every rising edge of System Clock. To avoid redundancy, this project will only show the design code, Test Bench, and test results of the scalable Register with enable signal.

```
1  /*************************************************************************
2  *** Filename: scale_reg_en.v Created by Orr Chakon                     ***
3  *************************************************************************/
4
5  `timescale 1 ns / 1 ns
6
7  module scale_reg_en(DATA, EN, CLK, OUT);
8  parameter REG_SIZE = 32;    //scalable register size. Defualt width is 32 bits
9
10 // Port declaration
11    output [REG_SIZE-1:0] OUT;
12    input [REG_SIZE-1:0] DATA;
13    input EN, CLK;
14
15
16 // Internal variable declarations
17    reg [REG_SIZE-1:0] OUT;
18
19 // Model of register
20    always @(posedge CLK) begin
21     if (EN) OUT <= DATA;        //Only store new value when EN is true
22     else OUT <= OUT;
23    end
24 endmodule
25
```

Figure 61. Scalable Register Design Code

Test Bench:

The Test Bench program for the Scalable Register design is shown in Figure 62. In order to test the unit under test (UUT), several values were passed through and the EN signal was toggled, making sure the values only got latched at the rising edge of Clock while EN was set true.

```
/*********************************************************************************
*** Filename: tb_scale_reg_en.v Created by Orr Chakon                          ***
*********************************************************************************/

`timescale 1 ns /1 ns
module tb_scale_reg_en();

// Port declaration
   parameter SIZE = 32;
   reg [SIZE-1:0]DATA;
   reg EN, CLK;
   wire [SIZE-1:0]OUT;

// UUT instantiations

scale_reg_en UUT(DATA, EN, CLK, OUT);

initial CLK = 1'b0;
always #25 CLK = ~CLK;  //clock generator

initial
//variables output to the simulation log file
$monitor ("%d CLK = %b EN = %b DATA = %b OUT = %b", $time, CLK, EN, DATA, OUT);

initial begin

    EN = 1'b0; DATA = 32'b10101010101010101010101010101010;
#10 EN = 1'b1;
#50 DATA = 32'b01010101010101010101010101010101;
#50 DATA = 32'b00000000000000000000000000000000;
#50 DATA = 32'b11111111111111111111111111111111;
#50 DATA = 32'b00000000000000000000000000000000; EN = 1'b0;
#50 DATA = 32'b10101010101010101010101010101010;
#50 DATA = 32'b01010101010101010101010101010101; EN = 1'b1;

  #100 $finish;
   end
endmodule
```

Figure 62. Scalable Register Test Bench

Test Results:

The simulation results for the Scalable Register design are shown in Figure 63. As shown in Figure 63, the Register input is only latched at rising edge of Clock when EN is set true. When EN is set low, the Register output remains the same.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Jul  7 23:26 2016
                   0 CLK = 0 EN = 0 DATA = 10101010101010101010101010101010 OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                  10 CLK = 0 EN = 1 DATA = 10101010101010101010101010101010 OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                  25 CLK = 1 EN = 1 DATA = 10101010101010101010101010101010 OUT = 10101010101010101010101010101010
                  50 CLK = 0 EN = 1 DATA = 10101010101010101010101010101010 OUT = 10101010101010101010101010101010
                  60 CLK = 0 EN = 1 DATA = 01010101010101010101010101010101 OUT = 10101010101010101010101010101010
                  75 CLK = 1 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
                 100 CLK = 0 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
                 110 CLK = 0 EN = 1 DATA = 00000000000000000000000000000000 OUT = 01010101010101010101010101010101
                 125 CLK = 1 EN = 1 DATA = 00000000000000000000000000000000 OUT = 00000000000000000000000000000000
                 150 CLK = 0 EN = 1 DATA = 00000000000000000000000000000000 OUT = 00000000000000000000000000000000
                 160 CLK = 0 EN = 1 DATA = 11111111111111111111111111111111 OUT = 00000000000000000000000000000000
                 175 CLK = 1 EN = 1 DATA = 11111111111111111111111111111111 OUT = 11111111111111111111111111111111
                 200 CLK = 0 EN = 1 DATA = 11111111111111111111111111111111 OUT = 11111111111111111111111111111111
                 210 CLK = 0 EN = 0 DATA = 00000000000000000000000000000000 OUT = 11111111111111111111111111111111
                 225 CLK = 1 EN = 0 DATA = 00000000000000000000000000000000 OUT = 11111111111111111111111111111111
                 250 CLK = 0 EN = 0 DATA = 00000000000000000000000000000000 OUT = 11111111111111111111111111111111
                 260 CLK = 0 EN = 0 DATA = 10101010101010101010101010101010 OUT = 11111111111111111111111111111111
                 275 CLK = 1 EN = 0 DATA = 10101010101010101010101010101010 OUT = 11111111111111111111111111111111
                 300 CLK = 0 EN = 0 DATA = 10101010101010101010101010101010 OUT = 11111111111111111111111111111111
                 310 CLK = 0 EN = 1 DATA = 01010101010101010101010101010101 OUT = 11111111111111111111111111111111
                 325 CLK = 1 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
                 350 CLK = 0 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
                 375 CLK = 1 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
                 400 CLK = 0 EN = 1 DATA = 01010101010101010101010101010101 OUT = 01010101010101010101010101010101
$finish called from file "tb_scale_reg_en.v", line 46.
$finish at simulation time                 410
           V C S   S i m u l a t i o n   R e p o r t
Time: 410 ns
CPU Time:      0.600 seconds;       Data structure size:   0.0Mb
Thu Jul  7 23:26:07 2016
```

Figure 63. Scalable Register Simulation Results

6.12 Shift-By-Two Module Design and Test

Design Code:

The design code for the Shift Module is shown in Figure 64. The Shift Module is not a register as it shifts the input asynchronously by two, which essentially acts to multiply the input by 4. Rather than going through this module, there was an option to simply handle the shifting by setting the two least significant bits to a constant '0' and wire the rest accordingly, but this module helped make the top level design more clean and easy to read.

```
1  /*************************************************************************
2  *** Filename: shift_left2.v Created by Orr Chakon                      ***
3  *************************************************************************/
4
5  `timescale 1 ns / 1 ns
6
7  module shift_left2(SHIFT_IN, SHIFT_OUT);
8
9  // Port declaration
10    output [31:0] SHIFT_OUT;
11    input [31:0] SHIFT_IN;
12
13 // Internal variable declarations
14    reg [31:0] SHIFT_OUT;
15
16 // Model of shift register
17    always @(SHIFT_IN)      //triggered off input change
18    begin
19     SHIFT_OUT[1:0] = 2'b00;     //output two LSBs get filled with zeros
20     SHIFT_OUT[31:2] = SHIFT_IN[29:0];   //shift left 2
21     end
22 endmodule
23
```

Figure 64. Shift Module Design Code

Test Bench:

The Test Bench program for the Shift Module is shown in Figure 65. In order to test the UUT, input values were passed and then the proper shifting was verified.

```
/*********************************************************************************
*** Filename: tb_shift_left2.v Created by Orr Chakon                          ***
*********************************************************************************/

`timescale 1 ns /1 ns
module tb_shift_left2();

// Port declaration
   reg [31:0] SHIFT_IN;
   wire [31:0] SHIFT_OUT;

// UUT instantiation
   shift_left2 UUT (SHIFT_IN, SHIFT_OUT);   //instantiation of sign extend byte module

initial
//variables output to the simulation log file
$monitor ("%d SHIFT_IN = %b SHIFT_OUT = %b", $time, SHIFT_IN, SHIFT_OUT);

initial begin

  #10 $write("\n"); //new line used for clarity
      SHIFT_IN=32'b11111111111111111111111111111111;
  #10 SHIFT_IN=32'b00000000000000000000000000000000;
  #10 SHIFT_IN=32'b10101010101010101010101010101010;
  #10 SHIFT_IN=32'b01010101010101010101010101010101;
  #10 SHIFT_IN=32'b11111111111111111111111111111110;
  #10 SHIFT_IN=32'b00000000000000000000000000000001;
  #10 SHIFT_IN=32'b00100000000000000000000000000001;

  #20 $finish;
   end
endmodule
```

Figure 65. Shift Module Test Bench

Test Results:

The Simulation Results for the Shift Module are shown in Figure 66.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Jul  8 17:30 2016
                   0 SHIFT_IN = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx SHIFT_OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                  10 SHIFT_IN = 11111111111111111111111111111111 SHIFT_OUT = 11111111111111111111111111111100
                  20 SHIFT_IN = 00000000000000000000000000000000 SHIFT_OUT = 00000000000000000000000000000000
                  30 SHIFT_IN = 10101010101010101010101010101010 SHIFT_OUT = 10101010101010101010101010101000
                  40 SHIFT_IN = 01010101010101010101010101010101 SHIFT_OUT = 01010101010101010101010101010100
                  50 SHIFT_IN = 11111111111111111111111111111110 SHIFT_OUT = 11111111111111111111111111111000
                  60 SHIFT_IN = 00000000000000000000000000000001 SHIFT_OUT = 00000000000000000000000000000100
                  70 SHIFT_IN = 00100000000000000000000000000001 SHIFT_OUT = 10000000000000000000000000000100
$finish called from file "tb_shift_left2.v", line 34.
$finish at simulation time                   90
           V C S   S i m u l a t i o n   R e p o r t
Time: 90 ns
CPU Time:      0.520 seconds;       Data structure size:   0.0Mb
Fri Jul  8 17:30:16 2016
```

Figure 66. Shift Module Simulation Results

6.13 Sign Extend Module Design and Test

Design Code:

The design code for the Sign Extend Module is shown in Figure 67. The Sign Extend Module is completely asynchronous by design. This processor design used four different Sign Extend Modules to extend both signed and unsigned bytes and half words to a full 32-bit word. The signed modules extend the most significant bit while the unsigned modules extend with zeros. Like a few other modules in the processor design, the Sign Extend Modules are used to make the top level design simpler and could have been replaced with simple wiring. For the purpose of this report, only documentation for the Sign-Extend-by-16 Module will be shown as the other three modules are similar by design.

```
1  /*********************************************************************
2  *** Filename: sign_extend16.v Created by Orr Chakon                ***
3  *********************************************************************/
4
5  `timescale 1 ns / 1 ns
6
7  module sign_extend16(BYTE_IN, OUT);
8
9  // Port declaration
10    output [31:0] OUT;
11    input [15:0] BYTE_IN;
12
13 // Internal variable declarations
14    reg [31:0] OUT;
15
16 // Model of sign extender
17    always @(BYTE_IN)       //triggered off input change
18    begin
19     OUT[15:0] = BYTE_IN[15:0];      //first 16 bits stays the same
20     OUT[31:16] = {16{BYTE_IN[15]}}; //sign extend remaining 16 bits
21     end
22 endmodule
23
```

Figure 67. Sign Extend Module Design Code

Test Bench:

The Sign Extend Module Test Bench Program is shown in Figure 68. In order to test the UUT, input values were passed and then the sign extended output was verified.

```
/*********************************************************************************
*** Filename: tb_sign_extend16.v Created by Orr Chakon                        ***
*********************************************************************************/

`timescale 1 ns /1 ns
module tb_sign_extend16();

// Port declaration
   reg [15:0] BYTE_IN;
   wire [31:0] OUT;

// UUT instantiation
   sign_extend16 UUT (BYTE_IN, OUT);

initial
//variables output to the simulation log file
$monitor ("%d BYTE_IN = %b OUT = %b", $time, BYTE_IN, OUT);

initial begin

  #10 $write("\n"); //new line used for clarity
      BYTE_IN=16'b1111111111111111;
  #10 BYTE_IN=16'b0000000000000000;
  #10 BYTE_IN=16'b1010101010101010;
  #10 BYTE_IN=16'b0101010101010101;
  #10 BYTE_IN=16'b1111111111111110;
  #10 BYTE_IN=16'b0000000000000001;

  #20 $finish;
   end
endmodule
```

Figure 68. Sign Extend Module Test Bench

Test Results:

The Simulation Results for the Sign Extend Module are shown in Figure 69.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Jul  8 16:28 2016
                   0 BYTE_IN = xxxxxxxxxxxxxxxx OUT = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                  10 BYTE_IN = 1111111111111111 OUT = 11111111111111111111111111111111
                  20 BYTE_IN = 0000000000000000 OUT = 00000000000000000000000000000000
                  30 BYTE_IN = 1010101010101010 OUT = 11111111111111111010101010101010
                  40 BYTE_IN = 0101010101010101 OUT = 00000000000000000101010101010101
                  50 BYTE_IN = 1111111111111110 OUT = 11111111111111111111111111111110
                  60 BYTE_IN = 0000000000000001 OUT = 00000000000000000000000000000001
$finish called from file "tb_sign_extend16.v", line 33.
$finish at simulation time                   80
           V C S   S i m u l a t i o n   R e p o r t
Time: 80 ns
CPU Time:      0.540 seconds;       Data structure size:   0.0Mb
Fri Jul  8 16:28:51 2016
```

Figure 69. Sign Extend Module Simulation Results

6.14 State Register Design and Test

Design Code:

The Design Code for the State Register is shown in Figure 70. The State Register is a synchronous module that passes the input directly to the output on rising edge of system clock when Reset is disabled.

```
1  /*************************************************************************
2  *** Filename: StateReg.v Created by Orr Chakon                        ***
3  *************************************************************************/
4
5  `timescale 1 ns / 1 ns
6
7  module StateReg(Present_State, Next_State, CLK, RST_n);
8
9  // Port declaration
10    output [2:0] Present_State;
11    input [2:0] Next_State;
12    input CLK, RST_n;
13
14
15 // Internal variable declarations
16    reg [2:0] Present_State;
17
18    always @(posedge CLK or negedge RST_n)
19    begin
20     if(!RST_n)          //Reset has top priority. Active low.
21     Present_State <= 3'b000;
22     else
23     Present_State <= Next_State;
24    end
25
26 endmodule
```

Figure 70. State Register Design Code

Test Bench:

The State Register Test Bench Program is shown in Figure 71. In order to test the UUT, all the possible input values were toggled through and then the output results were verified.

```
/*********************************************************************
*** Filename: tb_StateReg.v Created by Orr Chakon                  ***
*********************************************************************/

`timescale 1 ns / 1 ns

module tb_StateReg();

    reg [2:0] Next_State;                   //regesters required to hold the inputs
    reg CLK, RST_n;
    wire [2:0] Present_State;               // Wired used to capture the output

    StateReg UUT(Present_State, Next_State, CLK, RST_n);    //Module under test

    initial
    CLK = 1'b0;        // Initialize clock to 0

    always #5 CLK = ~CLK;  // 10ns clock period

    initial
    //parameters output to the simulation log file
    $monitor ("%d Present_State = %d Next_State = %d CLK = %b RST_n = %b", $time, Present_State, Next_State, CLK, RST_n);

    initial begin
        $vcdpluson;         //Enable graphical viewer


        Next_State = 3'b111;
        RST_n = 1'b1;

    #15 Next_State = 3'b110;
    #15 Next_State = 3'b101;
    #15 Next_State = 3'b100;
    #15 Next_State = 3'b111;
        RST_n = 1'b0;
    #15 Next_State = 3'b110;    //output not affected as reset held low
    #15 Next_State = 3'b101;
    #15 Next_State = 3'b100;
    #15 Next_State = 3'b011;
        RST_n = 1'b1;
    #15 Next_State = 3'b010;
    #15 Next_State = 3'b001;
    #15 Next_State = 3'b000;

    #15 $finish;            //Terminate simulation after 15ns
    end                     //End of simulation
endmodule
```

Figure 71. State Register Test Bench

Test Results:

The Simulation Results for the State Register design are shown in Figure 72.

```
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct  9 16:27 2016
VCD+ Writer J-2014.12-SP3 Copyright (c) 1991-2014 by Synopsys Inc.
                   0 Present_State = x Next_State = 7 CLK = 0 RST_n = 1
                   5 Present_State = 7 Next_State = 7 CLK = 1 RST_n = 1
                  10 Present_State = 7 Next_State = 7 CLK = 0 RST_n = 1
                  15 Present_State = 6 Next_State = 6 CLK = 1 RST_n = 1
                  20 Present_State = 6 Next_State = 6 CLK = 0 RST_n = 1
                  25 Present_State = 6 Next_State = 6 CLK = 1 RST_n = 1
                  30 Present_State = 6 Next_State = 5 CLK = 0 RST_n = 1
                  35 Present_State = 5 Next_State = 5 CLK = 1 RST_n = 1
                  40 Present_State = 5 Next_State = 5 CLK = 0 RST_n = 1
                  45 Present_State = 4 Next_State = 4 CLK = 1 RST_n = 1
                  50 Present_State = 4 Next_State = 4 CLK = 0 RST_n = 1
                  55 Present_State = 4 Next_State = 4 CLK = 1 RST_n = 1
                  60 Present_State = 0 Next_State = 7 CLK = 0 RST_n = 0
                  65 Present_State = 0 Next_State = 7 CLK = 1 RST_n = 0
                  70 Present_State = 0 Next_State = 7 CLK = 0 RST_n = 0
                  75 Present_State = 0 Next_State = 6 CLK = 1 RST_n = 0
                  80 Present_State = 0 Next_State = 6 CLK = 0 RST_n = 0
                  85 Present_State = 0 Next_State = 6 CLK = 1 RST_n = 0
                  90 Present_State = 0 Next_State = 5 CLK = 0 RST_n = 0
                  95 Present_State = 0 Next_State = 5 CLK = 1 RST_n = 0
                 100 Present_State = 0 Next_State = 5 CLK = 0 RST_n = 0
                 105 Present_State = 0 Next_State = 4 CLK = 1 RST_n = 0
                 110 Present_State = 0 Next_State = 4 CLK = 0 RST_n = 0
                 115 Present_State = 0 Next_State = 4 CLK = 1 RST_n = 0
                 120 Present_State = 0 Next_State = 3 CLK = 0 RST_n = 1
                 125 Present_State = 3 Next_State = 3 CLK = 1 RST_n = 1
                 130 Present_State = 3 Next_State = 3 CLK = 0 RST_n = 1
                 135 Present_State = 2 Next_State = 2 CLK = 1 RST_n = 1
                 140 Present_State = 2 Next_State = 2 CLK = 0 RST_n = 1
                 145 Present_State = 2 Next_State = 2 CLK = 1 RST_n = 1
                 150 Present_State = 2 Next_State = 1 CLK = 0 RST_n = 1
                 155 Present_State = 1 Next_State = 1 CLK = 1 RST_n = 1
                 160 Present_State = 1 Next_State = 1 CLK = 0 RST_n = 1
                 165 Present_State = 0 Next_State = 0 CLK = 1 RST_n = 1
                 170 Present_State = 0 Next_State = 0 CLK = 0 RST_n = 1
                 175 Present_State = 0 Next_State = 0 CLK = 1 RST_n = 1
$finish called from file "tb_StateReg.v", line 65.
$finish at simulation time                  180
          V C S   S i m u l a t i o n   R e p o r t
Time: 180 ns
CPU Time:      0.530 seconds;       Data structure size:   0.0Mb
Sun Oct  9 16:27:06 2016
```

Figure 72. State Register Simulation Results

7 RISC PROCESSOR OPERATION

In this section, the top level RISC Processor design code, test methods and results are documented.

7.1 RISC Processor Design Code

The RISC Processor Design is comprised of the sub-modules described in sections 4 and 6. The Processor design code, shown in Figures 73 and 74, is a structural Verilog implementation connecting all sub-modules in accordance with the Processor drawing shown in Figure 1. Note that there are some sub-module ports connected to a wire labeled NC. All ports connected to wire NC are not used and are essentially not connected in the processor design.

```
/**********************************************************************
*** Filename: RISC.sv Created by Orr Chakon                        ***
**********************************************************************/

`timescale 1ns/1ns
module risc(CLK, RST_n);
//Port delcaration
   input CLK, RST_n;

//Internal port declaration

    wire OF_OUT, BF_OUT, NF_OUT, ZF_OUT, PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE
    , PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ, PC_LOAD;
    wire [2:0] present_state, next_state, PC_SEL, REG_DATA_SEL, ALU_SEL2, ALU_OP, MEMtoREG;
    wire [1:0] MEM_DATA_SEL, Reg_Dest;
    wire [31:0] Inst, PC_OUT, PC_SEL_WIRE, ALU_REG_OUT, Addr, DATA, WriteData, Reg2_Out, EB1_Out, EW1_Out, EBZ1_Out, EB2_Out, EWZ1_Out, EW2_Out, DATA_WIRE, EPC_OUT,
    CAUSE_OUT, Reg_Write_Data, Reg1_Data, Reg2_Data, EW3_Out, EWZ2_Out, Immediate_Out, Immediate_SL, Reg1_Out, ALU_OPR1, ALU_OPR2, ALU_OUT, CAUSE_DATA, Inst_SL, CAT_OUT,
     NC;
    wire [4:0] Dest_Addr;
    wire [3:0] ALU_CONTROL;


// The netlist

control CTL(Inst[31:26], present_state[2:0], OF_OUT, BF_OUT, Inst[5:0], Inst[15:11], next_state[2:0], PC_EN, PC_SEL[2:0], IorD, MEM_DATA_SEL[1:0], MEM_OE, MEM_WS,
REG_DATA_SEL[2:0], IR_EN, Reg_Dest[1:0], MEMtoREG[2:0], REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, ALU_SEL2[2:0], ALU_OP[2:0], EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ,
PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ); //Sequence Controller

StateReg SRG(present_state[2:0], next_state[2:0], CLK, RST_n); //Phase Generator

comb CMB(NF_OUT, ZF_OUT, PCWrite_BLTZ, PCWrite_BGTZ, PCWrite_BLEZ, PCWrite_BNE, PCWrite_BEQ, PC_EN, PC_LOAD); //Combinational Block

pc PCR(PC_OUT[31:0], RST_n, CLK, PC_SEL_WIRE[31:0], PC_LOAD); //Program Counter

mux2 MUX1(PC_OUT[31:0], ALU_REG_OUT[31:0], IorD, Addr[31:0]); //Mux 1. Selects RAM address source.

RAM MEM1(DATA[31:0], Addr[7:0], WriteData[31:0], MEM_OE, MEM_WS); //RAM module. Most segnificant 26 bits of Addr not used to conserve resources.

mux4 MUX2(Reg2_Out[31:0], EB1_Out[31:0], EW1_Out[31:0], NC[31:0], MEM_DATA_SEL[1:0], WriteData[31:0]); //Mux2. Input D is a No Connect (NC). Selects RAM data source.

sign_extend8 EB1(Reg2_Out[7:0], EB1_Out[31:0]); //Sign Extend Byte. Input listed first.

sign_extend16 EW1(Reg2_Out[15:0], EW1_Out[31:0]); //Sign Extend Half Word. Input listed first.

sign_extend8_zero EBZ1(DATA[7:0], EBZ1_Out[31:0]); //Sign Extend Unsigned Byte. Input listed first.

sign_extend8 EB2(DATA[7:0], EB2_Out[31:0]); //Sign Extend Byte. Input listed first.

sign_extend16_zero EWZ1(DATA[15:0], EWZ1_Out[31:0]); //Sign Extend Unsigned Half Word. Input listed first.

sign_extend16 EW2(DATA[15:0], EW2_Out[31:0]); //Sign Extend Half Word. Input listed first.

mux8 MUX3(DATA[31:0], EBZ1_Out[31:0], EB2_Out[31:0], EWZ1_Out[31:0], EW2_Out[31:0], NC[31:0], NC[31:0], NC[31:0], REG_DATA_SEL[2:0], DATA_WIRE[31:0]); //Mux 3 Selects
```

erilog file | length : 5,432 lines : 91 | Ln : 84 Col : 103 Sel : 0 | 0 | Unix (LF) | UTF-8 | INS

Figure 73. RISC Processor Design Code Part 1

```
MIR data source.

scale_reg_en #(32)MIR(DATA_WIRE[31:0], IR_EN, CLK, Inst[31:0]); //Memory Instruction Register

mux4 MUX4(Inst[20:16], Inst[15:11], 5'b11111, NC[31:0], Reg_Dest[1:0], Dest_Addr[4:0]); //Mux4. Only 5 I/O bits used. Selects Register File address source.

mux8 MUX5(ALU_OUT[31:0], Inst[31:0], EPC_OUT[31:0], CAUSE_OUT[31:0], DATA_WIRE[31:0], PC_OUT[31:0], NC[31:0], NC[31:0], MEMtoREG[2:0], Reg_Write_Data[31:0]); //Mux5.
Selects Register File data source.

REG_FILE MEM2(Reg1_Data[31:0], Reg2_Data[31:0], Inst[25:21], Inst[20:16], Dest_Addr[4:0], Reg_Write_Data[31:0], REG_OE, REG_WS); //Register File

sign_extend16 EW3(Inst[15:0], EW3_Out[31:0]); //Sign Extend Half Word. Input listed first.

sign_extend16_zero EWZ2(Inst[15:0], EWZ2_Out[31:0]); //Sign Extend Unsigned Half Word. Input listed first.

mux2 MUX6(EW3_Out[31:0], EWZ2_Out[31:0], SIGNEXT_SEL, Immediate_Out[31:0]); //Mux 6. Selects Immediate value to be used by the ALU.

shift_left2 SLL1(Immediate_Out[31:0], Immediate_SL[31:0]);  //Shift Immediate value left by two bits. Input listed first.

scale_reg #(32)REG1(Reg1_Data[31:0], CLK, Reg1_Out[31:0]); //ALU Reg1. Input listed first.

scale_reg #(32)REG2(Reg2_Data[31:0], CLK, Reg2_Out[31:0]); //ALU Reg2. Input listed first.

mux2 MUX7(PC_OUT[31:0], Reg1_Out[31:0], ALU_SEL1, ALU_OPR1[31:0]); //Mux 7. Selects ALU Operand1 source.

mux8 MUX8(Reg2_Out[31:0], 32'b1, Immediate_Out[31:0], Immediate_SL[31:0], 32'b0, NC[31:0], NC[31:0], NC[31:0], ALU_SEL2[2:0], ALU_OPR2[31:0]); //Mux 3 Selects MIR data
source.

alu_cont CONT(Inst[5:0], ALU_OP[2:0], ALU_CONTROL[3:0]); //ALU controller .Inputs listed first.

alu ALU(ALU_OPR1[31:0], ALU_OPR2[31:0], ALU_CONTROL[3:0], Inst[10:6], ALU_OUT[31:0], NF_OUT, ZF_OUT, OF_OUT, BF_OUT); //ALU. Inputs listed first.

scale_reg #(32)REGA(ALU_OUT[31:0], CLK, ALU_REG_OUT[31:0]); //ALU Register. Inputs listed first.

mux2 MUX9(32'b0, 32'b1, CAUSE_SEL, CAUSE_DATA[31:0]); //Mux 9. Determines content of Cuase register.

scale_reg_en #(32)REGC(CAUSE_DATA[31:0], CAUSE_EN, CLK, CAUSE_OUT[31:0]); //Cause Register. Inputs listed first.

scale_reg_en #(32)REGE(PC_OUT[31:0], EPC_EN, CLK, EPC_OUT[31:0]); //EPC Register. Inputs listed first.

concatinate CAT(Inst[25:0], PC_OUT[31:28], CAT_OUT[31:0]); //concatinate to calculate PC address for Jump function

mux8 MUX10(ALU_OUT[31:0], ALU_REG_OUT[31:0], CAT_OUT[31:0], Reg1_Out[31:0], 32'b0, NC[31:0], NC[31:0], NC[31:0], PC_SEL[2:0], PC_SEL_WIRE[31:0]); //Mux 10. Selects PC
value.

endmodule

erilog file   length : 5,432   lines : 91   Ln : 84   Col : 103   Sel : 0 | 0   Unix (LF)   UTF-8   INS
```

Figure 74. RISC Processor Design Code Part 2

7.2 RISC Processor Test Setup and Documentation

In order to test the RISC Processor design, a MIPS test script was created, which contained all supported instructions and verified the proper execution of each instruction. (See section 7.3 for MIPS test script and design verification.) The Test Bench Program for the RISC Processor design is shown in Figure 75.

```
/*********************************************************************
*** Filename: tb_RISC.v Created by Orr Chakon                      ***
*********************************************************************/

`timescale 1 ns / 1 ns          //Timescale definition

module tb_risc();

    //port declarations
    reg CLK, RST_n;

    risc UUT(CLK, RST_n);       //cpu instantiation

    initial
    //parameters output to the simulation log file
    $monitor ("%d CLK = %b RST = %b PC_out = %d RAM_Addr = %d RAM_Data = %d RAM_WS = %b RAM_OE = %b RAM_out = %b IR_out = %b REG_WS = %b REG_OE = %b RegData = %d RegAddr
    = %d A_Reg = %d B_Reg = %d ALU_OPR1 = %d ALU_OPR2 = %d ALU_OUT = %d NF = %b ZF = %b OF = %b BF = %b EPC_EN = %d CAUSE_EN = %d STATE = %d", $time, CLK, RST_n, UUT.PCR.
    Dout, UUT.MUX1.OUT, UUT.MUX2.OUT, UUT.CTL.MEM_WS, UUT.CTL.MEM_OE, UUT.MEM1.DataOut, UUT.MIR.OUT, UUT.CTL.REG_WS, UUT.CTL.REG_OE, UUT.MUX5.OUT, UUT.MUX4.OUT, UUT.REG1.
    OUT, UUT.REG2.OUT, UUT.MUX7.OUT, UUT.MUX8.OUT, UUT.ALU.ALU_OUT, UUT.ALU.NF_OUT, UUT.ALU.ZF_OUT, UUT.ALU.OF_OUT, UUT.ALU.BF_OUT, UUT.CTL.EPC_EN, UUT.CTL.CAUSE_EN, UUT.
    CTL.STATE);

    initial CLK = 1'b0;
    always begin
    $write("\n");   //new line used for clarity
    #25 CLK = ~CLK; //clock generator
    end

    initial begin
    $readmemh("RAM_Data.txt", UUT.MEM1.mem);    //initialize RAM with data from RAM_Data.txt
    end

    initial begin
    $readmemh("RegFile_Data.txt", UUT.MEM2.mem);    //initialize Register File with data from RegFile_Data.txt

    end


    initial begin
        $vcdpluson;         //Enable graphical viewer

        RST_n = 1'b1;
    #5  RST_n = 1'b0;

    #5 RST_n = 1'b1;

    #15000  $finish;        //give enough time for the processor to execute all instructions before terminating simulation
    end
endmodule

```

Verilog file | length : 1,718 lines : 45 | Ln : 1 Col : 77 Sel : 0 | 0 | Unix (LF) | ANSI | INS

Figure 75. RISC Processor Test Bench

The Test Bench program uses the ***$monitor*** Verilog system task to scope down on internal signals for troubleshooting. Each time one of the signals included in the ***$monitor*** function changes, the new value is output to the test results file. Since the Processor design is synchronous, signals are only updated at the rising edge of system clock (CLK). Table 8 provides a description of each of the scoped signals in the Test Bench.

Signal Name	Description
CLK	System clock
RST	Active low system reset
PC_out	Program Counter output
RAM_Addr	RAM input address derived from Mux1
RAM_Data	RAM input data derived from Mux2
RAM_WS	RAM WS signal generated by the Sequence Controller
RAM_OE	RAM OE signal generated by the Sequence Controller
RAM_out	RAM data output
IR_out	Memory Instruction Register output
REG_WS	Register File WS signal generated by the Sequence Controller
REG_OE	Register File OE signal generated by the Sequence Controller
RegData	Register File write data derived from Mux5
RegAddr	Register File write address derived from Mux4
A_Reg	ALU Reg1 output
B_Reg	ALU Reg2 output
ALU_OPR1	ALU first operand derived from Mux7
ALU_OPR2	ALU second operand derived from Mux8
ALU_OUT	ALU output
NF	ALU negative flag output
ZF	ALU zero flag output
OF	ALU overflow flag output
BF	ALU invalid operation flag output
EPC_IN	Input to the EPC register
CAUSE_IN	Input to the CAUSE register
STATE	Sequence Controller state output used for debug purpose only

Table 8. Scoped Signals Included in Simulation

The RISC Processor has two inputs, namely global Clock (CLK) and Reset (RST_n), and no outputs. The Processor contains a RAM module, internal to the design, which is where the MIPS instructions are stored and executed from. Since there is no way to actively access the RAM module from outside of the Processor, there is no synthesizable way to load the MIPS instructions to be executed. In order to test this project's design, ***$readmemh*** was used, which is a non-synthesizable Verilog system task that allows for loading of memory arrays for simulation purposes through a text file. In this case, the text file is stored in the directory from which the processor is compiled from and follows a certain format. The ***$readmemh*** command is issued in

the Test Bench Program and contains the text file name and module to be loaded. The text file (i.e. RAM_Data.txt) used to load the RAM module is shown in Figures 76 and 77.

```
@000        //ADD
221820
@001        //SUB
602022
@002        //SW
AFDB0046
@003        //LW
8FC50046
@004        //AND
BE3024
@005        //BEQ (No Branch)
10A20000
@006        //BEQ (Branch executed)
1004000A
@008        //OR
1095025
@009        //SH
A7C80047
@00A        //LB
83CB0047
@00B        //LBu
93CC0047
@00C        //XOR
16C6826
@00D        //SB
A3C90048
@00E        //JR
E00008
@028        //ANDi
31AE53AA
@029        //ORi
35CF8585
@02A        //XORi
39F0FF00
@02B        //LH
87D10047
@02C        //LHu
97D20047
@02D        //BLTZ (No Branch)
4200004
@02E        //BLTZ (Branch executed)
6200004
@02F        //ADDi
20C70016
@030        //BNE (No Branch)
14040001
@031        //BNE (Branch Executed)
14E60001
@036        //Jump to location 008
8000002
@03F        //NOR
2329827
@040        //ADDiu
Normal text file                                    length : 1
```

Figure 76. RAM_Data.txt Showing Initial RAM Data Part 1

```
25F4F0F0
@041        //SLTiu (No Set)
2E950008
@042        //SLTiu (Set)
2EB58001
@043        //SLTi (No Set)
29160008
@044        //SLTi (Set)
2AD68181
@045        //BLEZ (No Branch)
1AA00002
@046        //BLEZ (Branch executed)
1AC00002
@04F        //SLL
CBA00
@050        //SLLV
77C004
@051        //SRL
18C902
@052        //SRLV
39D006
@053        //SRA
8E403
@054        //SRAV
1CE807
@055        //BGTZ (No Branch)
1EC00002
@056        //BGTZ (Branch Executed)
1C400002
@05F        //SLTu (No Set)
22029
@060        //SLTu (Set)
822029
@061        //SLT (No Set)
8B282A
@062        //SLT (Set)
102282A
@063        //JAL
C00001E
@078        //ADDu
1423021
@079        //SUBu
1423823
@07A        //JALR
3E0A809

Normal text file                                    length : 1,
```

Figure 77. RAM_Data.txt Showing Initial RAM Data Part 2

As shown in the RAM_Data text file, the memory locations and data are presented in hexadecimal format. The '@' symbol is used before each memory location allocation. The hexadecimal representation following each address is the data to be stored at that particular address. The synthesizer loads each memory location, per the text file, before executing the Test Bench Program.

In addition to the RAM module, the Processor also contains an internal Register File, which is where processor variables are stored and accessed during the execution of MIPS instructions. Since a memory block's initial value is unknown (denoted by 'x' in simulation), the execution of any instruction, other than the Jump instruction, will result in an unknown condition. For example, if the Register File is not loaded with any initial values and an ADD instruction were to

be executed, the result will be unknown since the operands are unknown and unknown(x) + unknown(x) = unknown(x). In order to get past this issue, the Register File was loaded with some initial values. The register File was loaded using a separate ***$readmemh*** command and text file. The text file (i.e. RegFile_Data.txt) used to load the Register File memory array is shown in Figure 78.

```
1   @000        //value 2 stored in memory location 0
2   00000002
3
4   @001        //value 3 stored in memory location 1
5   00000003
6
7   @002        //value 1 stored in memory location 2
8   00000001
9
10  @008
11  AAAAAAAA    //10101010....10
12
13  @009
14  55555555    //01010101....01
15
16  @1B         //location 27
17  00000032    //50
18
19  @1E         //location 30
20  0000001E    //30
21
```

Figure 78. RegFile_Data.txt Showing Initial Register File Data

Note that in normal MIPS processor operation, the Register File is partitioned a certain way with each register serving a particular purpose. For the test setup, each of the 32 registers in the Register File were treated as a general purpose register and the formal partitioning was ignored as it added no value toward assuring the proper functionality of the design. Translating register names into binary Register File addresses is done at the compiler level and is beyond the scope of this project's design.

Once the RAM module is loaded with instructions and the Register File is loaded with its initial values, the reset signal is toggled and the processor proceeds to execute the MIPS code.

7.3 RISC Processor Test Results and Validation

This section outlines the MIPS instruction test script and provides a detailed analysis of the test results. The MIPS test script, which includes at least one instance of each supported instruction type, was loaded into RAM and performed consecutively. To make the test results more clear, the test results are broken into sections and are presented with their corresponding instruction in the sections below. The following instructions are presented in the same order they were executed.

- ADD (R-type): Rs + Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00001	00010	00011	00000	100000

RAM location: 0x000

Instruction in Hex: 0x221820

Result: 4 → R[3]

```
simulator copyright 1991-2014
s proprietary information.
 J-2014.12-SP3; Runtime version J-2014.12-SP3;  Oct 30 10:56 2016

14.12-SP3 Copyright (c) 1991-2014 by Synopsys Inc.

//ADD
   0 CLK = 0 RST = 1 PC_out =          x RAM_Addr =          x RAM_Data =          x RAM_WS = x RAM_OE = x RAM_out = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx IR_out =
   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx REG_WS = x REG_OE = x RegData =          x RegAddr =          x A_Reg =          x B_Reg =          x ALU_OPR1 =          x ALU_OPR2 =          x
   ALU_OUT =          x NF = x ZF = x OF = x BF = x EPC_IN =          x CAUSE_IN =          X STATE =  x
   5 CLK = 0 RST = 0 PC_out =          0 RAM_Addr =          0 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001000100001100000100000 IR_out =
   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          X A_Reg =          x B_Reg =          x ALU_OPR1 =          0 ALU_OPR2 =          1
   ALU_OUT =          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          0 CAUSE_IN =          0 STATE =  0
  10 CLK = 0 RST = 1 PC_out =          0 RAM_Addr =          0 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001000100001100000100000 IR_out =
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          X A_Reg =          x B_Reg =          x ALU_OPR1 =          0 ALU_OPR2 =          1
  ALU_OUT =          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          0 CAUSE_IN =          0 STATE =  0

  25 CLK = 1 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
  00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =      24705 RegAddr =          2 A_Reg =          x B_Reg =          x ALU_OPR1 =          1 ALU_OPR2 =      24704
  ALU_OUT =      24705 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  1

  50 CLK = 0 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
  00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =      24705 RegAddr =          2 A_Reg =          x B_Reg =          x ALU_OPR1 =          1 ALU_OPR2 =      24704
  ALU_OUT =      24705 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  1

  75 CLK = 1 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
  00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =          4 RegAddr =          3 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          3 ALU_OPR2 =          1
  ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  4

 100 CLK = 0 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
 00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =          4 RegAddr =          3 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          3 ALU_OPR2 =          1
 ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  4

 125 CLK = 1 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
 00000000001000100001100000100000 REG_WS = 1 REG_OE = 1 RegData =          4 RegAddr =          3 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          3 ALU_OPR2 =          1
 ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  5

 150 CLK = 0 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out =
 00000000001000100001100000100000 REG_WS = 1 REG_OE = 1 RegData =          4 RegAddr =          3 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          3 ALU_OPR2 =          1
 ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  5
```

Figure 79. Simulation Results for ADD Instruction

Analysis: 1 + 3 → 4 → RegFile[3]. Test results show that RegData = 4 and RegAddr = 3 while the REG_WS signal goes high on the last clock cycle. The next instruction uses R[3] to verify that the value 4 actually got latched.

(NOTE: The operand values in R[1] and R[2] were initially stored into the Register File before the start of simulation using the ***$readmemh*** system Verilog task).

- SUB (R-type): Rs - Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00011	00000	00100	00000	100010

RAM location: 0x001

Instruction in Hex: 0x602022

Result: 2 → R[4]

```
//SUB

175 CLK = 1 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000011000000010000000100010 IR_out =
00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =          2 RegAddr =          2 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 =          1
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  0

200 CLK = 0 RST = 1 PC_out =          1 RAM_Addr =          1 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000011000000010000000100010 IR_out =
00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData =          2 RegAddr =          2 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 =          1
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          1 CAUSE_IN =          0 STATE =  0

225 CLK = 1 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =      32906 RegAddr =          0 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =      32904
ALU_OUT =      32906 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  1

250 CLK = 0 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =      32906 RegAddr =          0 A_Reg =          3 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =      32904
ALU_OUT =      32906 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  1

275 CLK = 1 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =          2 RegAddr =          4 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          4 ALU_OPR2 =          2
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  4

300 CLK = 0 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =          2 RegAddr =          4 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          4 ALU_OPR2 =          2
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  4

325 CLK = 1 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 1 REG_OE = 1 RegData =          2 RegAddr =          4 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          4 ALU_OPR2 =          2
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  5

350 CLK = 0 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011000000010000000100010 IR_out =
00000000011000000010000000100010 REG_WS = 1 REG_OE = 1 RegData =          2 RegAddr =          4 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          4 ALU_OPR2 =          2
ALU_OUT =          2 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  5
```

Figure 80. Simulation Results for SUB Instruction

Analysis: 4 - 2 → 2 → RegFile[4]. Simulation results show that RegData = 2 and RegAddr = 4 while the REG_WS signal goes high on the last clock cycle. The first operand, extracted from location R[3], was generated in the previous instruction, indicating it was latched properly.

(NOTE: The second operand value in R[0] was initially stored into the Register File before the start of simulation using the ***$readmemh*** system Verilog task).

- SW (I-type): M[Rs + (sign extended I[15:0])] ← Rt

Opcode	Rs	Rt	Address/Constant
101011	11110	11011	0000 0000 0100 0110

RAM location: 0x002

Instruction in Hex: 0xAFDB0046

Result: 50 → M[100]

```
//SW

375 CLK = 1 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 10101111110110110000000001000110 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =          3 RegAddr =          0 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          1
ALU_OUT =          3 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  0

400 CLK = 0 RST = 1 PC_out =          2 RAM_Addr =          2 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 10101111110110110000000001000110 IR_out =
00000000011000000010000000100010 REG_WS = 0 REG_OE = 1 RegData =          3 RegAddr =          0 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          1
ALU_OUT =          3 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          2 CAUSE_IN =          0 STATE =  0

425 CLK = 1 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        283 RegAddr =         27 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          3 ALU_OPR2 =        280
ALU_OUT =        283 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  1

450 CLK = 0 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        283 RegAddr =         27 A_Reg =          4 B_Reg =          2 ALU_OPR1 =          3 ALU_OPR2 =        280
ALU_OUT =        283 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  1

475 CLK = 1 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  7

500 CLK = 0 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  7

525 CLK = 1 RST = 1 PC_out =          3 RAM_Addr =        100 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE = 17

550 CLK = 0 RST = 1 PC_out =          3 RAM_Addr =        100 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE = 17

575 CLK = 1 RST = 1 PC_out =          3 RAM_Addr =        100 RAM_Data =         50 RAM_WS = 1 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE = 34

600 CLK = 0 RST = 1 PC_out =          3 RAM_Addr =        100 RAM_Data =         50 RAM_WS = 1 RAM_OE = 0 RAM_out = 10101111110110110000000001000110 IR_out =
10101111110110110000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE = 34
```

Figure 81. Simulation Results for SW Instruction

Analysis: M[100] ← M[R[30] + 70] ← RegFile[27]

R[27] = 50 R[30] = 30

Test results show that RAM_Data = 50 and RAM_Addr = 100 while the RAM_WS signal goes high on the last clock cycle.

(NOTE: R[27] and R[30] were initially stored into the Register File before the start of simulation using the ***$readmemh*** system Verilog task).

- LW (I-type): Rt ← M[Rs + (sign extended I[15:0])] RegFile[5] ← M[30 + 70]

Opcode	Rs	Rt	Address/Constant
100011	11110	00101	0000 0000 0100 0110

RAM location: 0x003

Instruction in Hex: 0x8FC50046

Result: 50 → R[5]

```
//LW

625 CLK = 1 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =         50 RAM_WS = 0 RAM_OE = 1 RAM_out = 10001111111000101000000000001000110 IR_out =
10101111111011011000000000001000110 REG_WS = 0 REG_OE = 1 RegData =          4 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =          3 ALU_OPR2 =          1
ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  0

650 CLK = 0 RST = 1 PC_out =          3 RAM_Addr =          3 RAM_Data =         50 RAM_WS = 0 RAM_OE = 1 RAM_out = 10001111111000101000000000001000110 IR_out =
10101111111011011000000000001000110 REG_WS = 0 REG_OE = 1 RegData =          4 RegAddr =         27 A_Reg =         30 B_Reg =         50 ALU_OPR1 =          3 ALU_OPR2 =          1
ALU_OUT =          4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          3 CAUSE_IN =          0 STATE =  0

675 CLK = 1 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10001111111000101000000000001000110 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =        284 RegAddr =          5 A_Reg =         30 B_Reg =         50 ALU_OPR1 =          4 ALU_OPR2 =        280
ALU_OUT =        284 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  1

700 CLK = 0 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =         50 RAM_WS = 0 RAM_OE = 0 RAM_out = 10001111111000101000000000001000110 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =        284 RegAddr =          5 A_Reg =         30 B_Reg =         50 ALU_OPR1 =          4 ALU_OPR2 =        280
ALU_OUT =        284 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  1

725 CLK = 1 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10001111111000101000000000001000110 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  7

750 CLK = 0 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10001111111000101000000000001000110 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  7

775 CLK = 1 RST = 1 PC_out =          4 RAM_Addr =        100 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000000000000000110010 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =         50 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE = 18

800 CLK = 0 RST = 1 PC_out =          4 RAM_Addr =        100 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000000000000000110010 IR_out =
10001111111000101000000000001000110 REG_WS = 0 REG_OE = 1 RegData =         50 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE = 18

825 CLK = 1 RST = 1 PC_out =          4 RAM_Addr =        100 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000000000000000110010 IR_out =
10001111111000101000000000001000110 REG_WS = 1 REG_OE = 1 RegData =         50 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE = 19

850 CLK = 0 RST = 1 PC_out =          4 RAM_Addr =        100 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000000000000000110010 IR_out =
10001111111000101000000000001000110 REG_WS = 1 REG_OE = 1 RegData =         50 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         70
ALU_OUT =        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE = 19
```

Figure 82. Simulation Results for LW Instruction

Analysis: RegFile[5] ← M[100] ← M[R[30] + 70]

RAM location M[100] contains the value 50 as stored in the previous instruction proving the value 50 actually got latched properly.

(NOTE: This level of verification (i.e. using values in registers that were previously generated in other instructions) is done purposely throughout the test process in order to verify proper latching. This will not be highlight again beyond this point).

Simulation results show that RegData = 50 and RegAddr = 5 while the REG_WS signal goes high on the last clock cycle.

- AND (R-type): Rs AND Rt →Rd

(110010) AND (011110) → RegFile[6]

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00101	11110	00110	00000	100100

RAM location: 0x004

Instruction in Hex: 0xBE3024

Result: 010010 → 18 → R[6]

```
//AND

875 CLK = 1 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000101111100011000000100100 IR_out =
10001111110001010000000001000110 REG_WS = 0 REG_OE = 1 RegData =          5 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =          4 ALU_OPR2 =          1
ALU_OUT =          5 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  0

900 CLK = 0 RST = 1 PC_out =          4 RAM_Addr =          4 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000101111100011000000100100 IR_out =
10001111110001010000000001000110 REG_WS = 0 REG_OE = 1 RegData =          5 RegAddr =          5 A_Reg =         30 B_Reg =          x ALU_OPR1 =          4 ALU_OPR2 =          1
ALU_OUT =          5 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          4 CAUSE_IN =          0 STATE =  0

925 CLK = 1 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =      49301 RegAddr =         30 A_Reg =         30 B_Reg =          x ALU_OPR1 =          5 ALU_OPR2 =      49296
ALU_OUT =      49301 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  1

950 CLK = 0 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =      49301 RegAddr =         30 A_Reg =         30 B_Reg =          x ALU_OPR1 =          5 ALU_OPR2 =      49296
ALU_OUT =      49301 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  1

975 CLK = 1 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =         18 RegAddr =          6 A_Reg =         50 B_Reg =         30 ALU_OPR1 =         50 ALU_OPR2 =         30
ALU_OUT =         18 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  4

1000 CLK = 0 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =         18 RegAddr =          6 A_Reg =         50 B_Reg =         30 ALU_OPR1 =         50 ALU_OPR2 =         30 ALU_OUT
=         18 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  4

1025 CLK = 1 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 1 REG_OE = 1 RegData =         18 RegAddr =          6 A_Reg =         50 B_Reg =         30 ALU_OPR1 =         50 ALU_OPR2 =         30 ALU_OUT
=         18 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  5

1050 CLK = 0 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000101111100011000000100100 IR_out =
00000000101111100011000000100100 REG_WS = 1 REG_OE = 1 RegData =         18 RegAddr =          6 A_Reg =         50 B_Reg =         30 ALU_OPR1 =         50 ALU_OPR2 =         30 ALU_OUT
=         18 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  5
```

Figure 83. Simulation Results for AND Instruction

Analysis: Rs = R[5] Rt = R[30] Rd = R[6]

R[5] = 50 R[30] = 30.

Therefore, (110010) AND (011110) → 010010 → RegFile[6].

Simulation results show that RegData = 18 and RegAddr = 6 while the REG_WS signal goes high on the last clock cycle.

- BEQ (I-type): if Rs = Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000100	00101	00010	0000 0000 0000 0000

RAM location: 0x005

Instruction in Hex: 0x 10A20000

Result: No Branch (PC ← PC + 1)

```
//BEQ (NO BRANCH)

1075 CLK = 1 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010000101000100000000000000000 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =          6 RegAddr =         30 A_Reg =         50 B_Reg =         30 ALU_OPR1 =          5 ALU_OPR2 =          1 ALU_OUT
=          6 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  0

1100 CLK = 0 RST = 1 PC_out =          5 RAM_Addr =          5 RAM_Data =         30 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010000101000100000000000000000 IR_out =
00000000101111100011000000100100 REG_WS = 0 REG_OE = 1 RegData =          6 RegAddr =         30 A_Reg =         50 B_Reg =         30 ALU_OPR1 =          5 ALU_OPR2 =          1 ALU_OUT
=          6 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          5 CAUSE_IN =          0 STATE =  0

1125 CLK = 1 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000101000100000000000000000 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =          6 RegAddr =          2 A_Reg =         50 B_Reg =         30 ALU_OPR1 =          6 ALU_OPR2 =          0 ALU_OUT
=          6 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  1

1150 CLK = 0 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =         30 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000101000100000000000000000 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =          6 RegAddr =          2 A_Reg =         50 B_Reg =         30 ALU_OPR1 =          6 ALU_OPR2 =          0 ALU_OUT
=          6 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  1

1175 CLK = 1 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000101000100000000000000000 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =         49 RegAddr =          2 A_Reg =         50 B_Reg =          1 ALU_OPR1 =         50 ALU_OPR2 =          1 ALU_OUT
=         49 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  9

1200 CLK = 0 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000101000100000000000000000 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =         49 RegAddr =          2 A_Reg =         50 B_Reg =          1 ALU_OPR1 =         50 ALU_OPR2 =          1 ALU_OUT
=         49 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  9
```

Figure 84. Simulation Results for BEQ Instruction Part 1

Analysis: Rs = R[5] = 50 Rt = R[2] = 1

Since (50 ≠ 1) this is a no branch condition and next PC will be (PC + 1). Simulation results show that PC_out for this instruction is 5 so the next instruction should be at PC_out = 6.

- BEQ (I-type): if Rs = Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000100	00000	00100	0000 0000 0000 1010

RAM location: 0x006

Instruction in Hex: 0x 1004000A

Result: Branch (PC ← 47)

```
//BEQ (BRANCH)

1225 CLK = 1 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010000000001000000000000001010 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =          7 RegAddr =          2 A_Reg =         50 B_Reg =          1 ALU_OPR1 =          6 ALU_OPR2 =          1 ALU_OUT
=          7 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  0

1250 CLK = 0 RST = 1 PC_out =          6 RAM_Addr =          6 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010000000001000000000000001010 IR_out =
00010000101000100000000000000000 REG_WS = 0 REG_OE = 1 RegData =          7 RegAddr =          2 A_Reg =         50 B_Reg =          1 ALU_OPR1 =          6 ALU_OPR2 =          1 ALU_OUT
=          7 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          6 CAUSE_IN =          0 STATE =  0

1275 CLK = 1 RST = 1 PC_out =          7 RAM_Addr =          7 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000000001000000000000001010 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          4 A_Reg =         50 B_Reg =          1 ALU_OPR1 =          7 ALU_OPR2 =         40 ALU_OUT
=         47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          7 CAUSE_IN =          0 STATE =  1

1300 CLK = 0 RST = 1 PC_out =          7 RAM_Addr =          7 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000000001000000000000001010 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          4 A_Reg =         50 B_Reg =          1 ALU_OPR1 =          7 ALU_OPR2 =         40 ALU_OUT
=         47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          7 CAUSE_IN =          0 STATE =  1

1325 CLK = 1 RST = 1 PC_out =          7 RAM_Addr =          7 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000000001000000000000001010 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          2 ALU_OUT
=          0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =          7 CAUSE_IN =          0 STATE =  9

1350 CLK = 0 RST = 1 PC_out =          7 RAM_Addr =          7 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010000000001000000000000001010 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          2 ALU_OUT
=          0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =          7 CAUSE_IN =          0 STATE =  9
```

Figure 85. Simulation Results for BEQ Instruction Part 2

Analysis: Rs = R[0] = 2 Rt = R[4] = 2

Since (2 = 2) the branch is executed and the next instruction will be at PC + 1 + ((sign extended I[15:0]) || 00).

PC + 1 = 7 ((sign extended I[15:0]) || 00) = 10 x 4 = 40

The next instruction should be at PC_out = 47

(NOTE: This instruction is at PC_out = 6, which is an indication that the previous instruction executed properly).

- ADDi (I-type): Rt ← Rs + (sign extended I[15:0])

Opcode	Rs	Rt	Address/Constant
001000	00110	00111	0000 0000 0001 0110

RAM location: 0x02F (47 in decimal)

Instruction in Hex: 0x 20C70016

Result: R[7] ← 40

```
//ADDi

1375 CLK = 1 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00100000110001110000000000010110 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =         48 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         47 ALU_OPR2 =          1 ALU_OUT
=          48 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE =  0

1400 CLK = 0 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00100000110001110000000000010110 IR_out =
00010000000001000000000000001010 REG_WS = 0 REG_OE = 1 RegData =         48 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         47 ALU_OPR2 =          1 ALU_OUT
=          48 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE =  0

1425 CLK = 1 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =        136 RegAddr =          7 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         48 ALU_OPR2 =         88 ALU_OUT
=         136 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  1

1450 CLK = 0 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =        136 RegAddr =          7 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         48 ALU_OPR2 =         88 ALU_OUT
=         136 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  1

1475 CLK = 1 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =         40 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         18 ALU_OPR2 =         22 ALU_OUT
=          40 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  7

1500 CLK = 0 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =         40 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         18 ALU_OPR2 =         22 ALU_OUT
=          40 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  7

1525 CLK = 1 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 1 REG_OE = 1 RegData =         40 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         18 ALU_OPR2 =         22 ALU_OUT
=          40 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  8

1550 CLK = 0 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100000110001110000000000010110 IR_out =
00100000110001110000000000010110 REG_WS = 1 REG_OE = 1 RegData =         40 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         18 ALU_OPR2 =         22 ALU_OUT
=          40 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  8
```

Figure 86. Simulation Results for ADDi Instruction

Analysis: Rs = R[6] = 18 Rt = R[7]

R[7] ← 18 + 22

Simulation results show that RegData = 40 and RegAddr = 7 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 47, which is an indication that the previous BEQ instruction executed properly).

- BNE (I-type): if Rs ≠ Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000101	00000	00100	0000 0000 0000 0001

RAM location: 0x030 (48 in decimal)

Instruction in Hex: 0x14040001

Result: No Branch

```
//BNE (NO BRANCH)

1575 CLK = 1 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00010100000001000000000000000001 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =         49 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         48 ALU_OPR2 =          1 ALU_OUT
=          49 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  0

1600 CLK = 0 RST = 1 PC_out =         48 RAM_Addr =         48 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00010100000001000000000000000001 IR_out =
00100000110001110000000000010110 REG_WS = 0 REG_OE = 1 RegData =         49 RegAddr =          7 A_Reg =         18 B_Reg =          x ALU_OPR1 =         48 ALU_OPR2 =          1 ALU_OUT
=          49 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         48 CAUSE_IN =          0 STATE =  0

1625 CLK = 1 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100000001000000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =         53 RegAddr =          4 A_Reg =         18 B_Reg =          x ALU_OPR1 =         49 ALU_OPR2 =          4 ALU_OUT
=          53 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE =  1

1650 CLK = 0 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100000001000000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =         53 RegAddr =          4 A_Reg =         18 B_Reg =          x ALU_OPR1 =         49 ALU_OPR2 =          4 ALU_OUT
=          53 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE =  1

1675 CLK = 1 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100000001000000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          2 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE = 10

1700 CLK = 0 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100000001000000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          2 ALU_OPR2 =          2 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE = 10
```

Figure 87. Simulation Results for BNE Instruction Part 1

Analysis: Rs = R[0] = 2 Rt = R[4] = 2

Since (2 = 2) this is a no branch condition and next PC will be (PC + 1). Simulation results show that PC_out for this instruction is 48, so the next instruction should be at PC_out = 49.

- BNE (I-type): if Rs ≠ Rt then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000101	00111	00110	0000 0000 0000 0001

RAM location: 0x031 (49 in decimal)

Instruction in Hex: 0x 14E60001

Result: Branch (PC ← 54)

```
//BNE (BRANCH)

1725 CLK = 1 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010100111001100000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =         50 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         49 ALU_OPR2 =          1 ALU_OUT
=          50 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE =  0

1750 CLK = 0 RST = 1 PC_out =         49 RAM_Addr =         49 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00010100111001100000000000000001 IR_out =
00010100000001000000000000000001 REG_WS = 0 REG_OE = 1 RegData =         50 RegAddr =          4 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         49 ALU_OPR2 =          1 ALU_OUT
=          50 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         49 CAUSE_IN =          0 STATE =  0

1775 CLK = 1 RST = 1 PC_out =         50 RAM_Addr =         50 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100111001100000000000000001 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         54 RegAddr =          6 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         50 ALU_OPR2 =          4 ALU_OUT
=          54 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         50 CAUSE_IN =          0 STATE =  1

1800 CLK = 0 RST = 1 PC_out =         50 RAM_Addr =         50 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100111001100000000000000001 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         54 RegAddr =          6 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         50 ALU_OPR2 =          4 ALU_OUT
=          54 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         50 CAUSE_IN =          0 STATE =  1

1825 CLK = 1 RST = 1 PC_out =         50 RAM_Addr =         50 RAM_Data =         18 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100111001100000000000000001 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         22 RegAddr =          6 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         40 ALU_OPR2 =         18 ALU_OUT
=          22 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         50 CAUSE_IN =          0 STATE = 10

1850 CLK = 0 RST = 1 PC_out =         50 RAM_Addr =         50 RAM_Data =         18 RAM_WS = 0 RAM_OE = 0 RAM_out = 00010100111001100000000000000001 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         22 RegAddr =          6 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         40 ALU_OPR2 =         18 ALU_OUT
=          22 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         50 CAUSE_IN =          0 STATE = 10
```

Figure 88. Simulation Results for BNE Instruction Part 2

Analysis: Rs = R[7] = 40 Rt = R[6] = 18

Since (40 ≠ 18) the branch is executed and the next instruction will be at PC + 1 + ((sign extended I[15:0]) || 00).

PC + 1 = 50 ((sign extended I[15:0]) || 00) = 1 x 4 = 4

The next instruction should be at PC_out = 54

(NOTE: This instruction is at PC_out = 49, which is an indication that the previous BNE instruction executed properly by not branching).

- Jump (J-type): PC ← PC[31:28] || Inst[25:0] || 00

Opcode	Jump Target
000010	00 0000 0000 0000 0000 0000 0010

RAM location: 0x036 (54 in decimal)

Instruction in Hex: 0x8000002

Result: Branch (PC ← 8)

```
//JUMP

1875 CLK = 1 RST = 1 PC_out =         54 RAM_Addr =         54 RAM_Data =         18 RAM_WS = 0 RAM_OE = 1 RAM_out = 00001000000000000000000000000010 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         55 RegAddr =          6 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         54 ALU_OPR2 =          1 ALU_OUT
=          55 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         54 CAUSE_IN =          0 STATE =  0

1900 CLK = 0 RST = 1 PC_out =         54 RAM_Addr =         54 RAM_Data =         18 RAM_WS = 0 RAM_OE = 1 RAM_out = 00001000000000000000000000000010 IR_out =
00010100111001100000000000000001 REG_WS = 0 REG_OE = 1 RegData =         55 RegAddr =          6 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         54 ALU_OPR2 =          1 ALU_OUT
=          55 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         54 CAUSE_IN =          0 STATE =  0

1925 CLK = 1 RST = 1 PC_out =         55 RAM_Addr =         55 RAM_Data =         18 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001000000000000000000000000010 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         55 ALU_OPR2 =          8 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         55 CAUSE_IN =          0 STATE =  1

1950 CLK = 0 RST = 1 PC_out =         55 RAM_Addr =         55 RAM_Data =         18 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001000000000000000000000000010 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =         40 B_Reg =         18 ALU_OPR1 =         55 ALU_OPR2 =          8 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         55 CAUSE_IN =          0 STATE =  1

1975 CLK = 1 RST = 1 PC_out =         55 RAM_Addr =         55 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001000000000000000000000000010 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         55 ALU_OPR2 =          8 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         55 CAUSE_IN =          0 STATE =  2

2000 CLK = 0 RST = 1 PC_out =         55 RAM_Addr =         55 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001000000000000000000000000010 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =         55 ALU_OPR2 =          8 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         55 CAUSE_IN =          0 STATE =  2
```

Figure 89. Simulation Results for Jump Instruction

Analysis: PC[31:28] || Inst[25:0] || 00 = 0000 || 00000000000000000000000010 || 00
= 1000 = 8

(NOTE: This instruction is at PC_out = 54, which is an indication that the previous BNE instruction executed properly).

- OR (R-type): Rs OR Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	01000	01001	01010	00000	100101

RAM location: 0x008

Instruction in Hex: 0x1095025

Result: FFFFFFFF → 4294967295 → R[10]

```
//OR

2025 CLK = 1 RST = 1 PC_out =          8 RAM_Addr =          8 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001000010010101000000100101 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          9 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          8 ALU_OPR2 =          1 ALU_OUT
=          9 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          8 CAUSE_IN =          0 STATE =  0

2050 CLK = 0 RST = 1 PC_out =          8 RAM_Addr =          8 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001000010010101000000100101 IR_out =
00001000000000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          9 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          8 ALU_OPR2 =          1 ALU_OUT
=          9 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          8 CAUSE_IN =          0 STATE =  0

2075 CLK = 1 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData =      82077 RegAddr =          9 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          9 ALU_OPR2 =      82068 ALU_OUT
=      82077 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  1

2100 CLK = 0 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData =      82077 RegAddr =          9 A_Reg =          2 B_Reg =          2 ALU_OPR1 =          9 ALU_OPR2 =      82068 ALU_OUT
=      82077 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  1

2125 CLK = 1 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData = 4294967295 RegAddr =         10 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 = 2863311530 ALU_OPR2 = 1431655765 ALU_OUT
=         -1 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  4

2150 CLK = 0 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData = 4294967295 RegAddr =         10 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 = 2863311530 ALU_OPR2 = 1431655765 ALU_OUT
=         -1 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  4

2175 CLK = 1 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 1 REG_OE = 1 RegData = 4294967295 RegAddr =         10 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 = 2863311530 ALU_OPR2 = 1431655765 ALU_OUT
=         -1 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  5

2200 CLK = 0 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000010010101000000100101 IR_out =
00000001000010010101000000100101 REG_WS = 1 REG_OE = 1 RegData = 4294967295 RegAddr =         10 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 = 2863311530 ALU_OPR2 = 1431655765 ALU_OUT
=         -1 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  5
```

Figure 90. Simulation Results for OR Instruction

Analysis: Rs = R[8] = 10101010101010101010101010101010 = 0xAAAAAAAA

Rt = R[9] = 01010101010101010101010101010101 = 0x55555555 Rd = R[10]

(0xAAAAAAAA) OR (0x55555555) → 0xFFFFFFFF → 4294967295 → R[10]

Simulation results show that RegData = 4294967295 and RegAddr = 10 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 8, which is a verification that the previous JUMP instruction executed properly).

(NOTE: The operand values in R[8] and R[9] were initially stored into the Register ile before the start of simulation using the ***$readmemh*** system Verilog task).

- SH (I-type): M[Rs + (sign extended I[15:0])] ← Rt [15:0]

Opcode	Rs	Rt	Address/Constant
101001	11110	01000	0000 0000 0100 0111

RAM location: 0x009

Instruction in Hex: 0xA7C80047

Result: 1111 1111 1111 1111 1010 1010 1010 1010 → 4294945450 →M[101]

```
//SH

2225 CLK = 1 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 1 RAM_out = 10100111110010000000000001000111 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData =         10 RegAddr =          9 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 =          9 ALU_OPR2 =          1 ALU_OUT
=          10 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  0

2250 CLK = 0 RST = 1 PC_out =          9 RAM_Addr =          9 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 1 RAM_out = 10100111110010000000000001000111 IR_out =
00000001000010010101000000100101 REG_WS = 0 REG_OE = 1 RegData =         10 RegAddr =          9 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 =          9 ALU_OPR2 =          1 ALU_OUT
=          10 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =          9 CAUSE_IN =          0 STATE =  0

2275 CLK = 1 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        294 RegAddr =          8 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 =         10 ALU_OPR2 =        284 ALU_OUT
=         294 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  1

2300 CLK = 0 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        294 RegAddr =          8 A_Reg = 2863311530 B_Reg = 1431655765 ALU_OPR1 =         10 ALU_OPR2 =        284 ALU_OUT
=         294 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  1

2325 CLK = 1 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  7

2350 CLK = 0 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  7

2375 CLK = 1 RST = 1 PC_out =         10 RAM_Addr =        101 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE = 27

2400 CLK = 0 RST = 1 PC_out =         10 RAM_Addr =        101 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE = 27

2425 CLK = 1 RST = 1 PC_out =         10 RAM_Addr =        101 RAM_Data = 4294945450 RAM_WS = 1 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE = 34

2450 CLK = 0 RST = 1 PC_out =         10 RAM_Addr =        101 RAM_Data = 4294945450 RAM_WS = 1 RAM_OE = 0 RAM_out = 10100111110010000000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE = 34
```

Figure 91. Simulation Results for SH Instruction

Analysis: Rs = R[30] = 30 Rt[15:0] = R[8] [15:0] = 0xFFFFAAAA = 4294945450

M[101] ← M[30 + 71] ← 4294945450

Test results show that RAM_Data = 4294945450 and RAM_Addr = 101 while the RAM_WS signal goes high on the last clock cycle.

(NOTE: 0xFFFFAAAA is the half word of R[8] sign extended with ones).

- LB (I-type): Rt ← M[Rs + (sign extended I[15:0])] (Signed least significant byte only)

Opcode	Rs	Rt	Address/Constant
100000	11110	01011	0000 0000 0100 0111

RAM location: 0x00A

Instruction in Hex: 0x83CB0047

Result: 1111 1111 1111 1111 1111 1111 1010 1010 → 4294967210 → R[11]

```
//LB

2475 CLK = 1 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 1 RAM_out = 10000011110010110000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =         11 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         10 ALU_OPR2 =          1 ALU_OUT
=          11 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  0

2500 CLK = 0 RST = 1 PC_out =         10 RAM_Addr =         10 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 1 RAM_out = 10000011110010110000000001000111 IR_out =
10100111110010000000000001000111 REG_WS = 0 REG_OE = 1 RegData =         11 RegAddr =          8 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         10 ALU_OPR2 =          1 ALU_OUT
=          11 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         10 CAUSE_IN =          0 STATE =  0

2525 CLK = 1 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 10000011110010110000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =        295 RegAddr =         11 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         11 ALU_OPR2 =        284 ALU_OUT
=         295 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  1

2550 CLK = 0 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 10000011110010110000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =        295 RegAddr =         11 A_Reg =         30 B_Reg = 2863311530 ALU_OPR1 =         11 ALU_OPR2 =        284 ALU_OUT
=         295 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  1

2575 CLK = 1 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000011110010110000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  7

2600 CLK = 0 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000011110010110000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  7

2625 CLK = 1 RST = 1 PC_out =         11 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData = 4294967210 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE = 23

2650 CLK = 0 RST = 1 PC_out =         11 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData = 4294967210 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE = 23

2675 CLK = 1 RST = 1 PC_out =         11 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000011110010110000000001000111 REG_WS = 1 REG_OE = 1 RegData = 4294967210 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE = 19

2700 CLK = 0 RST = 1 PC_out =         11 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000011110010110000000001000111 REG_WS = 1 REG_OE = 1 RegData = 4294967210 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE = 19
```

Figure 92. Simulation Results for LB Instruction

Analysis: Rs = R[30] = 30 Rt= R[11]

R[11] ← 4294967210 ← 0xFFFFFFAA ← M[101] ← M[30 + 71]

Test results show that RegData = 4294967210 and RegAddr = 11 while the REG_WS signal goes high on the last clock cycle.

(NOTE: 0xFFFFFFAA is the (signed) sign extended LSB of RAM location M[101]).

- LBu (I-type): Rt ← M[Rs + (sign extended I[15:0])] (Unsigned least significant byte only)
  RegFile[12] ← M[30 + 71] (Signed least significant byte only)

Opcode	Rs	Rt	Address/Constant
100100	11110	01100	0000 0000 0100 0111

RAM location: 0x00B

Instruction in Hex: 0x93CC0047

Result: 0000 0000 0000 0000 0000 0000 1010 1010 → 170 → R[12]

```
//LBu

2725 CLK = 1 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10010011110011000000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =         12 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         11 ALU_OPR2 =          1 ALU_OUT
=          12 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  0

2750 CLK = 0 RST = 1 PC_out =         11 RAM_Addr =         11 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10010011110011000000000001000111 IR_out =
10000011110010110000000001000111 REG_WS = 0 REG_OE = 1 RegData =         12 RegAddr =         11 A_Reg =         30 B_Reg =          x ALU_OPR1 =         11 ALU_OPR2 =          1 ALU_OUT
=          12 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         11 CAUSE_IN =          0 STATE =  0

2775 CLK = 1 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010011110011000000000001000111 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        296 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         12 ALU_OPR2 =        284 ALU_OUT
=         296 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  1

2800 CLK = 0 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010011110011000000000001000111 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        296 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         12 ALU_OPR2 =        284 ALU_OUT
=         296 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  1

2825 CLK = 1 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010011110011000000000001000111 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  7

2850 CLK = 0 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010011110011000000000001000111 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  7

2875 CLK = 1 RST = 1 PC_out =         12 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        170 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE = 22

2900 CLK = 0 RST = 1 PC_out =         12 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =        170 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE = 22

2925 CLK = 1 RST = 1 PC_out =         12 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010011110011000000000001000111 REG_WS = 1 REG_OE = 1 RegData =        170 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE = 19

2950 CLK = 0 RST = 1 PC_out =         12 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010011110011000000000001000111 REG_WS = 1 REG_OE = 1 RegData =        170 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE = 19
```

Figure 93. Simulation Results for LBu Instruction

Analysis: Rs = R[30] = 30 Rt= R[12]

R[12] ← 170 ← 0x000000AA ← M[101] ← M[30 + 71]

Test results show that RegData = 170 and RegAddr = 12 while the REG_WS signal goes high on the last clock cycle.

(NOTE: 0x000000AA is the (unsigned) sign extended LSB of RAM location M[101]).

- XOR (R-type): Rs XOR Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	01011	01100	01101	00000	100110

RAM location: 0x00C

Instruction in Hex: 0x16C6826

Result: 1111 1111 1111 1111 1111 1111 0000 0000 → 4294967040 → R[13]

```
//XOR

2975 CLK = 1 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001011011000110100000100110 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =         13 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         12 ALU_OPR2 =          1 ALU_OUT
=          13 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  0

3000 CLK = 0 RST = 1 PC_out =         12 RAM_Addr =         12 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001011011000110100000100110 IR_out =
10010011110011000000000001000111 REG_WS = 0 REG_OE = 1 RegData =         13 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         12 ALU_OPR2 =          1 ALU_OUT
=          13 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         12 CAUSE_IN =          0 STATE =  0

3025 CLK = 1 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData =     106661 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         13 ALU_OPR2 =     106648 ALU_OUT
=      106661 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  1

3050 CLK = 0 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData =     106661 RegAddr =         12 A_Reg =         30 B_Reg =          x ALU_OPR1 =         13 ALU_OPR2 =     106648 ALU_OUT
=      106661 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  1

3075 CLK = 1 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData = 4294967040 RegAddr =         13 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 = 4294967210 ALU_OPR2 =        170 ALU_OUT
=        -256 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  4

3100 CLK = 0 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData = 4294967040 RegAddr =         13 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 = 4294967210 ALU_OPR2 =        170 ALU_OUT
=        -256 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  4

3125 CLK = 1 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 1 REG_OE = 1 RegData = 4294967040 RegAddr =         13 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 = 4294967210 ALU_OPR2 =        170 ALU_OUT
=        -256 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  5

3150 CLK = 0 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001011011000110100000100110 IR_out =
00000001011011000110100000100110 REG_WS = 1 REG_OE = 1 RegData = 4294967040 RegAddr =         13 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 = 4294967210 ALU_OPR2 =        170 ALU_OUT
=        -256 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  5
```

Figure 94. Simulation Results for XOR Instruction

Analysis: Rs = R[11] = 0xFFFFFFAA

Rt = R[12] = 0xAA Rd = R[13]

(0xFFFFFFAA) XOR (0xAA) → 0xFFFFFF00 → 4294967040 → R[13]

Simulation results show that RegData = 4294967040 and RegAddr = 13 while the REG_WS signal goes high on the last clock cycle.

- SB (I-type): M[Rs + (sign extended I[15:0])] ← Rt [7:0]

Opcode	Rs	Rt	Address/Constant
101000	11110	01001	0000 0000 0100 1000

RAM location: 0x00D

Instruction in Hex: 0xA3C90048

Result: 0000 0000 0000 0000 0000 0000 0101 0101 → 85 → M[102]

```
//SB

3175 CLK = 1 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 1 RAM_out = 10100011110010010000000001001000 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData =         14 RegAddr =         12 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 =         13 ALU_OPR2 =          1 ALU_OUT
=          14 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  0

3200 CLK = 0 RST = 1 PC_out =         13 RAM_Addr =         13 RAM_Data =        170 RAM_WS = 0 RAM_OE = 1 RAM_out = 10100011110010010000000001001000 IR_out =
00000001011011000110100000100110 REG_WS = 0 REG_OE = 1 RegData =         14 RegAddr =         12 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 =         13 ALU_OPR2 =          1 ALU_OUT
=          14 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         13 CAUSE_IN =          0 STATE =  0

3225 CLK = 1 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        302 RegAddr =          9 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 =         14 ALU_OPR2 =        288 ALU_OUT
=         302 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  1

3250 CLK = 0 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        302 RegAddr =          9 A_Reg = 4294967210 B_Reg =        170 ALU_OPR1 =         14 ALU_OPR2 =        288 ALU_OUT
=         302 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  1

3275 CLK = 1 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  7

3300 CLK = 0 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  7

3325 CLK = 1 RST = 1 PC_out =         14 RAM_Addr =        102 RAM_Data =         85 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE = 26

3350 CLK = 0 RST = 1 PC_out =         14 RAM_Addr =        102 RAM_Data =         85 RAM_WS = 0 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE = 26

3375 CLK = 1 RST = 1 PC_out =         14 RAM_Addr =        102 RAM_Data =         85 RAM_WS = 1 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE = 34

3400 CLK = 0 RST = 1 PC_out =         14 RAM_Addr =        102 RAM_Data =         85 RAM_WS = 1 RAM_OE = 0 RAM_out = 10100011110010010000000001001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =        102 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         30 ALU_OPR2 =         72 ALU_OUT
=         102 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE = 34
```

Figure 95. Simulation Results for SB Instruction

Analysis: Rs = R[30] = 30 Rt[7:0] = R[9] [7:0] = 0101 0101 = 85

M[102] ← M[30 + 72] ← 85

Test results show that RAM_Data = 85 and RAM_Addr = 102 while the RAM_WS signal goes high on the last clock cycle.

(NOTE: '0101 0101' is the (signed) sign extended LSB of R[9]).

- JR (R-type): PC ← Rs

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00111	00000	00000	00000	001000

RAM location: 0x00E

Instruction in Hex: 0xE00008

Result: PC ← 40

```
//JR

3425 CLK = 1 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000111000000000000000001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =         15 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         14 ALU_OPR2 =          1 ALU_OUT
=          15 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  0

3450 CLK = 0 RST = 1 PC_out =         14 RAM_Addr =         14 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000111000000000000000001000 IR_out =
10100011110010010000000001001000 REG_WS = 0 REG_OE = 1 RegData =         15 RegAddr =          9 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         14 ALU_OPR2 =          1 ALU_OUT
=          15 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         14 CAUSE_IN =          0 STATE =  0

3475 CLK = 1 RST = 1 PC_out =         15 RAM_Addr =         15 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000111000000000000000001000 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         15 ALU_OPR2 =         32 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         15 CAUSE_IN =          0 STATE =  1

3500 CLK = 0 RST = 1 PC_out =         15 RAM_Addr =         15 RAM_Data = 1431655765 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000111000000000000000001000 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =         30 B_Reg = 1431655765 ALU_OPR1 =         15 ALU_OPR2 =         32 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         15 CAUSE_IN =          0 STATE =  1

3525 CLK = 1 RST = 1 PC_out =         15 RAM_Addr =         15 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000111000000000000000001000 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         15 ALU_OPR2 =         32 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         15 CAUSE_IN =          0 STATE = 20

3550 CLK = 0 RST = 1 PC_out =         15 RAM_Addr =         15 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000111000000000000000001000 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         15 ALU_OPR2 =         32 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         15 CAUSE_IN =          0 STATE = 20
```

Figure 96. Simulation Results for JR Instruction

Analysis: Rs = R[7] = 40 PC ← 40

The next instruction will be from PC_out = 40

- ANDi (I-type): Rt ← Rs AND (sign extended I[15:0])

Opcode	Rs	Rt	Address/Constant
001100	01101	01110	0101 0011 1010 1010

RAM location: 0x028 (40 in decimal)

Instruction in Hex: 0x31AE53AA

Result: 0000 0000 0000 0000 0101 0011 0000 0000 → 21248 → R[14]

```
//ANDi

3575 CLK = 1 RST = 1 PC_out =         40 RAM_Addr =         40 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00110001101011100101001110101010 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         41 RegAddr =          0 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         40 ALU_OPR2 =          1 ALU_OUT
=          41 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         40 CAUSE_IN =          0 STATE =  0

3600 CLK = 0 RST = 1 PC_out =         40 RAM_Addr =         40 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00110001101011100101001110101010 IR_out =
00000000111000000000000000001000 REG_WS = 0 REG_OE = 1 RegData =         41 RegAddr =          0 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         40 ALU_OPR2 =          1 ALU_OUT
=          41 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         40 CAUSE_IN =          0 STATE =  0

3625 CLK = 1 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 0 REG_OE = 1 RegData =      85713 RegAddr =         14 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         41 ALU_OPR2 =      85672 ALU_OUT
=       85713 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  1

3650 CLK = 0 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 0 REG_OE = 1 RegData =      85713 RegAddr =         14 A_Reg =         40 B_Reg =          2 ALU_OPR1 =         41 ALU_OPR2 =      85672 ALU_OUT
=       85713 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  1

3675 CLK = 1 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 0 REG_OE = 1 RegData =      21248 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 = 4294967040 ALU_OPR2 =      21418 ALU_OUT
=       21248 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE = 14

3700 CLK = 0 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 0 REG_OE = 1 RegData =      21248 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 = 4294967040 ALU_OPR2 =      21418 ALU_OUT
=       21248 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE = 14

3725 CLK = 1 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 1 REG_OE = 1 RegData =      21248 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 = 4294967040 ALU_OPR2 =      21418 ALU_OUT
=       21248 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  8

3750 CLK = 0 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110001101011100101001110101010 IR_out =
00110001101011100101001110101010 REG_WS = 1 REG_OE = 1 RegData =      21248 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 = 4294967040 ALU_OPR2 =      21418 ALU_OUT
=       21248 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  8
```

Figure 97. Simulation Results for ANDi Instruction

Analysis: Rs = R[13] = 0xFFFFFF00 Rt = R[14]

R[14] ← '1111 1111 1111 1111 1111 1111 0000 0000' AND '0101 0011 1010 1010'

R[14] ← 21248 ← '0000 0000 0000 0000 0101 0011 0000 0000'

Simulation results show that RegData = 21248 and RegAddr = 14 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 40, which is a verification that the previous JR instruction executed properly).

- ORi (I-type): Rt ← Rs OR (sign extended I[15:0])

Opcode	Rs	Rt	Address/Constant
001101	01110	01111	1000 0101 1000 0101

RAM location: 0x029 (41 in decimal)

Instruction in Hex: 0x35CF8585

Result: 1111 1111 1111 1111 1101 0111 1000 0101 → 4294956933 → R[15]

```
//ORi

3775 CLK = 1 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00110101110011111000010110000101 IR_out =
00110001101011001010011101010101 REG_WS = 0 REG_OE = 1 RegData =         42 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 =         41 ALU_OPR2 =          1 ALU_OUT
=          42 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  0

3800 CLK = 0 RST = 1 PC_out =         41 RAM_Addr =         41 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00110101110011111000010110000101 IR_out =
00110001101011001010011101010101 REG_WS = 0 REG_OE = 1 RegData =         42 RegAddr =         14 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 =         41 ALU_OPR2 =          1 ALU_OUT
=          42 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         41 CAUSE_IN =          0 STATE =  0

3825 CLK = 1 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData = 4294841918 RegAddr =         15 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 =         42 ALU_OPR2 = 4294841876 ALU_OUT
=     -125378 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  1

3850 CLK = 0 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData = 4294841918 RegAddr =         15 A_Reg = 4294967040 B_Reg =          x ALU_OPR1 =         42 ALU_OPR2 = 4294841876 ALU_OUT
=     -125378 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  1

3875 CLK = 1 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData = 4294956933 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =      21248 ALU_OPR2 = 4294935941 ALU_OUT
=      -10363 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE = 15

3900 CLK = 0 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData = 4294956933 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =      21248 ALU_OPR2 = 4294935941 ALU_OUT
=      -10363 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE = 15

3925 CLK = 1 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 1 REG_OE = 1 RegData = 4294956933 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =      21248 ALU_OPR2 = 4294935941 ALU_OUT
=      -10363 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  8

3950 CLK = 0 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00110101110011111000010110000101 IR_out =
00110101110011111000010110000101 REG_WS = 1 REG_OE = 1 RegData = 4294956933 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =      21248 ALU_OPR2 = 4294935941 ALU_OUT
=      -10363 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  8
```

Figure 98. Simulation Results for ORi Instruction

Analysis: Rs = R[14] = '0101 0011 0000 0000' Rt = R[15]

R[15] ← '0101 0011 0000 0000' OR '1111 1111 1111 1111 1000 0101 1000 0101'

R[15] ← 4294956933 ← '1111 1111 1111 1111 1101 0111 1000 0101'

Simulation results show that RegData = 4294956933 and RegAddr = 15 while the REG_WS signal goes high on the last clock cycle.

- XORi (I-type):Rt ← Rs XOR (sign extended I[15:0])

Opcode	Rs	Rt	Address/Constant
001110	01111	10000	1111 1111 0000 0000

RAM location: 0x02A (42 in decimal)

Instruction in Hex: 0x39F0FF00

Result: 0000 0000 0000 0000 0010 1000 1000 0101 → 10373 → R[16]

```
//XORi

3975 CLK = 1 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00111001111100001111111100000000 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData =         43 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =         42 ALU_OPR2 =          1 ALU_OUT
=          43 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  0

4000 CLK = 0 RST = 1 PC_out =         42 RAM_Addr =         42 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00111001111100001111111100000000 IR_out =
00110101110011111000010110000101 REG_WS = 0 REG_OE = 1 RegData =         43 RegAddr =         15 A_Reg =      21248 B_Reg =          x ALU_OPR1 =         42 ALU_OPR2 =          1 ALU_OUT
=          43 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         42 CAUSE_IN =          0 STATE =  0

4025 CLK = 1 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData = 4294966315 RegAddr =         16 A_Reg =      21248 B_Reg =          x ALU_OPR1 =         43 ALU_OPR2 = 4294966272 ALU_OUT
=        -981 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  1

4050 CLK = 0 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData = 4294966315 RegAddr =         16 A_Reg =      21248 B_Reg =          x ALU_OPR1 =         43 ALU_OPR2 = 4294966272 ALU_OUT
=        -981 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  1

4075 CLK = 1 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData =      10373 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 = 4294967040 ALU_OUT
=       10373 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE = 16

4100 CLK = 0 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData =      10373 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 = 4294967040 ALU_OUT
=       10373 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE = 16

4125 CLK = 1 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 1 REG_OE = 1 RegData =      10373 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 = 4294967040 ALU_OUT
=       10373 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  8

4150 CLK = 0 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00111001111100001111111100000000 IR_out =
00111001111100001111111100000000 REG_WS = 1 REG_OE = 1 RegData =      10373 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 = 4294967040 ALU_OUT
=       10373 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  8
```

Figure 99. Simulation Results for XORi Instruction

Analysis: Rs = R[15] = '1111 1111 1111 1111 1101 0111 1000 0101' Rt = R[16]

R[16] ← '0101 0011 0000 0000' OR '1111 1111 1111 1111 1000 0101 1000 0101'

R[16] ← 10373 ← '0000 0000 0000 0000 0010 1000 1000 0101'

Simulation results show that RegData = 10373 and RegAddr = 16 while the REG_WS signal goes high on the last clock cycle.

- LH (I-type): Rt ← M[Rs + (sign extended I[15:0])] (Signed least significant word only)

Opcode	Rs	Rt	Address/Constant
100001	11110	10001	0000 0000 0100 0111

RAM location: 0x02B (43 in decimal)

Instruction in Hex: 0x87D10047

Result: 1111 1111 1111 1111 1010 1010 1010 1010 → 4294945450 → R[17]

```
//LH

4175 CLK = 1 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10000111110100010000000001000111 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData =         44 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         43 ALU_OPR2 =          1 ALU_OUT
=          44 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  0

4200 CLK = 0 RST = 1 PC_out =         43 RAM_Addr =         43 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10000111110100010000000001000111 IR_out =
00111001111100001111111100000000 REG_WS = 0 REG_OE = 1 RegData =         44 RegAddr =         16 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         43 ALU_OPR2 =          1 ALU_OUT
=          44 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         43 CAUSE_IN =          0 STATE =  0

4225 CLK = 1 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000111110100010000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =        328 RegAddr =         17 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         44 ALU_OPR2 =        284 ALU_OUT
=         328 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  1

4250 CLK = 0 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000111110100010000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =        328 RegAddr =         17 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         44 ALU_OPR2 =        284 ALU_OUT
=         328 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  1

4275 CLK = 1 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000111110100010000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  7

4300 CLK = 0 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10000111110100010000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  7

4325 CLK = 1 RST = 1 PC_out =         44 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE = 25

4350 CLK = 0 RST = 1 PC_out =         44 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE = 25

4375 CLK = 1 RST = 1 PC_out =         44 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000111110100010000000001000111 REG_WS = 1 REG_OE = 1 RegData = 4294945450 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE = 19

4400 CLK = 0 RST = 1 PC_out =         44 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10000111110100010000000001000111 REG_WS = 1 REG_OE = 1 RegData = 4294945450 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE = 19
```

Figure 100. Simulation Results for LH Instruction

Analysis: Rs = R[30] = 30 Rt= R[17]

R[17] ← 4294945450 ← 0xFFFFAAAA ← M[101] ← M[30 + 71]

Test results show that RegData = 4294945450 and RegAddr = 17 while the REG_WS signal goes high on the last clock cycle.

(NOTE: 0xFFFFAAAA is the (signed) sign extended least significant half word of RAM location M[101]).

- LHu (I-type): Rt ← M[Rs + (sign extended I[15:0])] (Unsigned least significant word only)

Opcode	Rs	Rt	Address/Constant
100101	11110	10010	0000 0000 0100 0111

RAM location: 0x02C (44 in decimal)

Instruction in Hex: 0x97D20047

Result: 0000 0000 0000 0000 1010 1010 1010 1010 → 43690 → R[18]

```
//LHu

4425 CLK = 1 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10010111110100100000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =         45 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         44 ALU_OPR2 =          1 ALU_OUT
=          45 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  0

4450 CLK = 0 RST = 1 PC_out =         44 RAM_Addr =         44 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 10010111110100100000000001000111 IR_out =
10000111110100010000000001000111 REG_WS = 0 REG_OE = 1 RegData =         45 RegAddr =         17 A_Reg =         30 B_Reg =          x ALU_OPR1 =         44 ALU_OPR2 =          1 ALU_OUT
=          45 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         44 CAUSE_IN =          0 STATE =  0

4475 CLK = 1 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010111110100100000000001000111 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =        329 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         45 ALU_OPR2 =        284 ALU_OUT
=         329 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  1

4500 CLK = 0 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010111110100100000000001000111 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =        329 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         45 ALU_OPR2 =        284 ALU_OUT
=         329 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  1

4525 CLK = 1 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010111110100100000000001000111 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  7

4550 CLK = 0 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 10010111110100100000000001000111 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =        101 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  7

4575 CLK = 1 RST = 1 PC_out =         45 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =      43690 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE = 24

4600 CLK = 0 RST = 1 PC_out =         45 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =      43690 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE = 24

4625 CLK = 1 RST = 1 PC_out =         45 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010111110100100000000001000111 REG_WS = 1 REG_OE = 1 RegData =      43690 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE = 19

4650 CLK = 0 RST = 1 PC_out =         45 RAM_Addr =        101 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 11111111111111111010101010101010 IR_out =
10010111110100100000000001000111 REG_WS = 1 REG_OE = 1 RegData =      43690 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         30 ALU_OPR2 =         71 ALU_OUT
=         101 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE = 19
```

Figure 101. Simulation Results for LHu Instruction

Analysis: Rs = R[30] = 30 Rt= R[18]

R[18] ← 43690 ← 0x0000AAAA ← M[101] ← M[30 + 71]

Test results show that RegData = 43690 and RegAddr = 18 while the REG_WS signal goes high on the last clock cycle.

(NOTE: 0x0000AAAA is the (unsigned) sign extended least significant half word of RAM location M[101]).

- BLTZ (I-type): if Rs < 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000001	00001	00000	0000 0000 0000 0100

RAM location: 0x02D (45 in decimal)

Instruction in Hex: 0x4200004

Result: No Branch (PC ← PC + 1)

```
//BLTZ (NO BRANCH)

4675 CLK = 1 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000100001000000000000000000100 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =         46 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         45 ALU_OPR2 =          1 ALU_OUT
=          46 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  0

4700 CLK = 0 RST = 1 PC_out =         45 RAM_Addr =         45 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000100001000000000000000000100 IR_out =
10010111110100100000000001000111 REG_WS = 0 REG_OE = 1 RegData =         46 RegAddr =         18 A_Reg =         30 B_Reg =          x ALU_OPR1 =         45 ALU_OPR2 =          1 ALU_OUT
=          46 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         45 CAUSE_IN =          0 STATE =  0

4725 CLK = 1 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000100001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         62 RegAddr =          0 A_Reg =         30 B_Reg =          x ALU_OPR1 =         46 ALU_OPR2 =         16 ALU_OUT
=          62 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE =  1

4750 CLK = 0 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000100001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         62 RegAddr =          0 A_Reg =         30 B_Reg =          x ALU_OPR1 =         46 ALU_OPR2 =         16 ALU_OUT
=          62 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE =  1

4775 CLK = 1 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000100001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =          3 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =          3 ALU_OPR2 =          0 ALU_OUT
=           3 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE = 13

4800 CLK = 0 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000100001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =          3 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =          3 ALU_OPR2 =          0 ALU_OUT
=           3 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE = 13
```

Figure 102. Simulation Results for BLTZ Instruction Part 1

Analysis: Rs = R[1] = 3

Since 3 is not less than 0, this is a no branch condition and next PC will be (PC + 1). Simulation results show that PC_out for this instruction is 45 so the next instruction should be at PC_out = 46.

- BLTZ (I-type): if Rs < 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000001	10001	00000	0000 0000 0000 0100

RAM location: 0x02E (46 in decimal)

Instruction in Hex: 0x6200004

Result: 47 + (4 x 4) = 63 Branch (PC ← 63)

```
//BLTZ (BRANCH)

4825 CLK = 1 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000110001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =         46 ALU_OPR2 =          1 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE =  0

4850 CLK = 0 RST = 1 PC_out =         46 RAM_Addr =         46 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000110001000000000000000000100 IR_out =
00000100001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         47 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =         46 ALU_OPR2 =          1 ALU_OUT
=          47 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         46 CAUSE_IN =          0 STATE =  0

4875 CLK = 1 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000110001000000000000000000100 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =         47 ALU_OPR2 =         16 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE =  1

4900 CLK = 0 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000110001000000000000000000100 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         63 RegAddr =          0 A_Reg =          3 B_Reg =          2 ALU_OPR1 =         47 ALU_OPR2 =         16 ALU_OUT
=          63 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE =  1

4925 CLK = 1 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000110001000000000000000000100 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =          0 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 = 4294945450 ALU_OPR2 =          0 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE = 13

4950 CLK = 0 RST = 1 PC_out =         47 RAM_Addr =         47 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000110001000000000000000000100 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =          0 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 = 4294945450 ALU_OPR2 =          0 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         47 CAUSE_IN =          0 STATE = 13
```

Figure 103. Simulation Results for BLTZ Instruction Part 2

Analysis: Rs = R[17] = 0xFFFFAAAA

Since 0xFFFFAAAA is less than 0, the branch is executed and the next instruction will be at PC + 1 + ((sign extended I[15:0]) || 00).

PC + 1 = 47 ((sign extended I[15:0]) || 00) = 4 x 4 = 16

The next instruction should be at PC_out = 47 + 16 = 63

(NOTE: This instruction is at PC_out = 46, which is an indication that the previous BLTZ instruction executed properly by not branching).

- NOR (R-type): Rs NOR Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	10001	10010	10011	00000	100111

RAM location: 0x03F (63 in decimal)

Instruction in Hex: 0x2329827

Result: 0101 0101 0101 0101 → 21845 → R[19]

```
//NOR

4975 CLK = 1 RST = 1 PC_out =         63 RAM_Addr =         63 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000010001100101001100000100111 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         64 RegAddr =          0 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 =         63 ALU_OPR2 =          1 ALU_OUT
=          64 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         63 CAUSE_IN =          0 STATE =  0

5000 CLK = 0 RST = 1 PC_out =         63 RAM_Addr =         63 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000010001100101001100000100111 IR_out =
00000110001000000000000000000100 REG_WS = 0 REG_OE = 1 RegData =         64 RegAddr =          0 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 =         63 ALU_OPR2 =          1 ALU_OUT
=          64 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         63 CAUSE_IN =          0 STATE =  0

5025 CLK = 1 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData = 4294861020 RegAddr =         18 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 =         64 ALU_OPR2 = 4294860956 ALU_OUT
=     -106276 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  1

5050 CLK = 0 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData = 4294861020 RegAddr =         18 A_Reg = 4294945450 B_Reg =          2 ALU_OPR1 =         64 ALU_OPR2 = 4294860956 ALU_OUT
=     -106276 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  1

5075 CLK = 1 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData =      21845 RegAddr =         19 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 = 4294945450 ALU_OPR2 =      43690 ALU_OUT
=       21845 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  4

5100 CLK = 0 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData =      21845 RegAddr =         19 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 = 4294945450 ALU_OPR2 =      43690 ALU_OUT
=       21845 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  4

5125 CLK = 1 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 1 REG_OE = 1 RegData =      21845 RegAddr =         19 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 = 4294945450 ALU_OPR2 =      43690 ALU_OUT
=       21845 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  5

5150 CLK = 0 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000010001100101001100000100111 IR_out =
00000010001100101001100000100111 REG_WS = 1 REG_OE = 1 RegData =      21845 RegAddr =         19 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 = 4294945450 ALU_OPR2 =      43690 ALU_OUT
=       21845 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  5
```

Figure 104. Simulation Results for NOR Instruction

Analysis: Rs = R[17] = 0xFFFFAAAA

Rt = R[18] = 0x0000AAAA Rd = R[19]

(0xFFFFAAAA) NOR (0x0000AAAA) → 0101 0101 0101 0101 → 21845 → R[19]

Simulation results show that RegData = 21845 and RegAddr = 19 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 63, which is an indication that the previous BLTZ instruction executed properly).

- ADDiu (I-type): Rt ← Rs + (sign extended I[15:0]) (Unsigned constant)

Opcode	Rs	Rt	Address/Constant
001001	01111	10100	1111 0000 1111 0000

RAM location: 0x040 (64 in decimal)

Instruction in Hex: 0x25F4F0F0

Result: 1 0000 0000 0000 0000 1100 1000 0111 0101 → 51317 → R[20]

```
//ADDui

5175 CLK = 1 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 1 RAM_out = 00100101111101001111000011110000 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData =          65 RegAddr =         18 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 =         64 ALU_OPR2 =          1 ALU_OUT
=          65 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  0

5200 CLK = 0 RST = 1 PC_out =         64 RAM_Addr =         64 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 1 RAM_out = 00100101111101001111000011110000 IR_out =
00000010001100101001100000100111 REG_WS = 0 REG_OE = 1 RegData =          65 RegAddr =         18 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 =         64 ALU_OPR2 =          1 ALU_OUT
=          65 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         64 CAUSE_IN =          0 STATE =  0

5225 CLK = 1 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData = 4294951937 RegAddr =         20 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 =         65 ALU_OPR2 = 4294951872 ALU_OUT
=      -15359 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  1

5250 CLK = 0 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =      43690 RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData = 4294951937 RegAddr =         20 A_Reg = 4294945450 B_Reg =      43690 ALU_OPR1 =         65 ALU_OPR2 = 4294951872 ALU_OUT
=      -15359 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  1

5275 CLK = 1 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData =      51317 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 =      61680 ALU_OUT
=       51317 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE = 29

5300 CLK = 0 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData =      51317 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 =      61680 ALU_OUT
=       51317 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE = 29

5325 CLK = 1 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 1 REG_OE = 1 RegData =      51317 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 =      61680 ALU_OUT
=       51317 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  8

5350 CLK = 0 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00100101111101001111000011110000 IR_out =
00100101111101001111000011110000 REG_WS = 1 REG_OE = 1 RegData =      51317 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 = 4294956933 ALU_OPR2 =      61680 ALU_OUT
=       51317 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  8
```

Figure 105. Simulation Results for ADDiu Instruction

Analysis: Rs = R[15] = '1111 1111 1111 1111 1101 0111 1000 0101' Rt = R[20]

R[20] ← '1111 1111 1111 1111 1101 0111 1000 0101' + '1111 0000 1111 0000'

R[20] ← 51317 ← '1 0000 0000 0000 0000 1100 1000 0111 0101'

Simulation results show that RegData = 51317 and RegAddr = 20 while the REG_WS signal goes high on the last clock cycle.

(NOTE: Most significant bit is carried out resulting in an ALU output of 51317)

- SLTiu (I-type): if Rs < (sign extended I[15:0]) then Rt ← 1
  else Rt ← 0

Opcode	Rs	Rt	Address/Constant
001011	10100	10101	0000 0000 0000 1000

RAM location: 0x041 (65 in decimal)

Instruction in Hex: 0x2E950008

Result: R[21] ← 0

```
//SLTiu

5375 CLK = 1 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101110100101010000000000001000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData =         66 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         65 ALU_OPR2 =          1 ALU_OUT
=          66 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  0

5400 CLK = 0 RST = 1 PC_out =         65 RAM_Addr =         65 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101110100101010000000000001000 IR_out =
00100101111101001111000011110000 REG_WS = 0 REG_OE = 1 RegData =         66 RegAddr =         20 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         65 ALU_OPR2 =          1 ALU_OUT
=          66 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         65 CAUSE_IN =          0 STATE =  0

5425 CLK = 1 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =         98 RegAddr =         21 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         66 ALU_OPR2 =         32 ALU_OUT
=          98 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  1

5450 CLK = 0 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =         98 RegAddr =         21 A_Reg = 4294956933 B_Reg =          x ALU_OPR1 =         66 ALU_OPR2 =         32 ALU_OUT
=          98 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  1

5475 CLK = 1 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =      51317 ALU_OPR2 =          8 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE = 28

5500 CLK = 0 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =      51317 ALU_OPR2 =          8 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE = 28

5525 CLK = 1 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =      51317 ALU_OPR2 =          8 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  8

5550 CLK = 0 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110100101010000000000001000 IR_out =
00101110100101010000000000001000 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =      51317 ALU_OPR2 =          8 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  8
```

Figure 106. Simulation Results for SLTiu Instruction Part 1

Analysis: Rs = R[20] = 4294895253 Rt = R[21]

Since 4294895253 is not less than 8 (R[21] ← 0)

Simulation results show that RegData = 0 and RegAddr = 21 while the REG_WS signal goes high on the last clock cycle.

- SLTiu (I-type): if Rs < (sign extended I[15:0]) then Rt ← 1
  else Rt ← 0

Opcode	Rs	Rt	Address/Constant
001011	10101	10101	1000 0000 0000 0001

RAM location: 0x042 (66 in decimal)

Instruction in Hex: 0x2EB58001

Result: R[21] ← 1

```
//SLTiu

5575 CLK = 1 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101110101101011000000000000001 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =         67 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =         66 ALU_OPR2 =          1 ALU_OUT
=          67 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  0

5600 CLK = 0 RST = 1 PC_out =         66 RAM_Addr =         66 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101110101101011000000000000001 IR_out =
00101110100101010000000000001000 REG_WS = 0 REG_OE = 1 RegData =         67 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =         66 ALU_OPR2 =          1 ALU_OUT
=          67 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         66 CAUSE_IN =          0 STATE =  0

5625 CLK = 1 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData = 4294836295 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =         67 ALU_OPR2 = 4294836228 ALU_OUT
=     -131001 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  1

5650 CLK = 0 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData = 4294836295 RegAddr =         21 A_Reg =      51317 B_Reg =          x ALU_OPR1 =         67 ALU_OPR2 = 4294836228 ALU_OUT
=     -131001 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  1

5675 CLK = 1 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =          0 ALU_OPR2 =      32769 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE = 28

5700 CLK = 0 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =          0 ALU_OPR2 =      32769 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE = 28

5725 CLK = 1 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =          0 ALU_OPR2 =      32769 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  8

5750 CLK = 0 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101110101101011000000000000001 IR_out =
00101110101101011000000000000001 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =          0 ALU_OPR2 =      32769 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  8
```

Figure 107. Simulation Results for SLTiu Instruction Part 2

Analysis: Rs = R[21] = 0 Rt = R[21]

Since 0 is less than 32769 (R[21] ← 1)

Simulation results show that RegData = 1 and RegAddr = 21 while the REG_WS signal goes high on the last clock cycle.

- SLTi (I-type): If Rs < (sign extended I[15:0]) then Rt ← 1
  else Rt ← 0

Opcode	Rs	Rt	Address/Constant
001010	01000	10110	0000 0000 0000 1000

RAM location: 0x043 (67 in decimal)

Instruction in Hex: 0x29160008

Result: R[22] ← 1

```
//SLTi

5775 CLK = 1 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 1 RAM_out = 00101001000101100000000000001000 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData =         68 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =         67 ALU_OPR2 =          1 ALU_OUT
=         68 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  0

5800 CLK = 0 RST = 1 PC_out =         67 RAM_Addr =         67 RAM_Data =          0 RAM_WS = 0 RAM_OE = 1 RAM_out = 00101001000101100000000000001000 IR_out =
00101110101101011000000000000001 REG_WS = 0 REG_OE = 1 RegData =         68 RegAddr =         21 A_Reg =          0 B_Reg =          0 ALU_OPR1 =         67 ALU_OPR2 =          1 ALU_OUT
=         68 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         67 CAUSE_IN =          0 STATE =  0

5825 CLK = 1 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         22 A_Reg =          0 B_Reg =          0 ALU_OPR1 =         68 ALU_OPR2 =         32 ALU_OUT
=        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  1

5850 CLK = 0 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          0 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         22 A_Reg =          0 B_Reg =          0 ALU_OPR1 =         68 ALU_OPR2 =         32 ALU_OUT
=        100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  1

5875 CLK = 1 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 = 2863311530 ALU_OPR2 =          8 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  6

5900 CLK = 0 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 = 2863311530 ALU_OPR2 =          8 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  6

5925 CLK = 1 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 = 2863311530 ALU_OPR2 =          8 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  8

5950 CLK = 0 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101001000101100000000000001000 IR_out =
00101001000101100000000000001000 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 = 2863311530 ALU_OPR2 =          8 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  8
```

Figure 108. Simulation Results for SLTi Instruction Part 1

Analysis: Rs = R[8] = 0xAAAAAAAA Rt = R[22]

Since AAAAAAAA is less than 8 (R[22] ← 1)

Simulation results show that RegData = 1 and RegAddr = 22 while the REG_WS signal goes high on the last clock cycle.

- SLTi (I-type): if Rs < (sign extended I[15:0]) then Rt ← 1
  else Rt ← 0

Opcode	Rs	Rt	Address/Constant
001010	10110	10110	1000 0001 1000 0001

RAM location: 0x044 (68 in decimal)

Instruction in Hex: 0x2AD68181

Result: R[22] ← 0

```
//SLTi

5975 CLK = 1 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101010110101101000000110000001 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =         69 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 =         68 ALU_OPR2 =          1 ALU_OUT
=          69 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  0

6000 CLK = 0 RST = 1 PC_out =         68 RAM_Addr =         68 RAM_Data =          x RAM_WS = 0 RAM_OE = 1 RAM_out = 00101010110101101000000110000001 IR_out =
00101001000101100000000000001000 REG_WS = 0 REG_OE = 1 RegData =         69 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 =         68 ALU_OPR2 =          1 ALU_OUT
=          69 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         68 CAUSE_IN =          0 STATE =  0

6025 CLK = 1 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData = 4294837833 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 =         69 ALU_OPR2 = 4294837764 ALU_OUT
=     -129463 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  1

6050 CLK = 0 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          x RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData = 4294837833 RegAddr =         22 A_Reg = 2863311530 B_Reg =          x ALU_OPR1 =         69 ALU_OPR2 = 4294837764 ALU_OUT
=     -129463 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  1

6075 CLK = 1 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 = 4294934913 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  6

6100 CLK = 0 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 = 4294934913 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  6

6125 CLK = 1 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 = 4294934913 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  8

6150 CLK = 0 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00101010110101101000000110000001 IR_out =
00101010110101101000000110000001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =          1 ALU_OPR2 = 4294934913 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  8
```

Figure 109. Simulation Results for SLTi Instruction Part 2

Analysis: Rs = R[22] = 1 Rt = R[22]

Since 1 is not less than '1000 0001 1000 0001' (R[22] ← 0)

Simulation results show that RegData = 0 and RegAddr = 22 while the REG_WS signal goes high on the last clock cycle.

- BLEZ (I-type): if Rs ≤ 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000110	10101	00000	0000 0000 0000 0010

RAM location: 0x045 (69 in decimal)
Instruction in Hex: 0x1AA00002
Result: No Branch (PC ← PC + 1)

```
//BLEZ (NO BRANCH)

6175 CLK = 1 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011010101000000000000000000010 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData =         70 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =         69 ALU_OPR2 =          1 ALU_OUT
=          70 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  0

6200 CLK = 0 RST = 1 PC_out =         69 RAM_Addr =         69 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011010101000000000000000000010 IR_out =
00101010110101101000000110000001 REG_WS = 0 REG_OE = 1 RegData =         70 RegAddr =         22 A_Reg =          1 B_Reg =          1 ALU_OPR1 =         69 ALU_OPR2 =          1 ALU_OUT
=          70 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         69 CAUSE_IN =          0 STATE =  0

6225 CLK = 1 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010101000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         78 RegAddr =          0 A_Reg =          1 B_Reg =          1 ALU_OPR1 =         70 ALU_OPR2 =          8 ALU_OUT
=          78 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE =  1

6250 CLK = 0 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010101000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         78 RegAddr =          0 A_Reg =          1 B_Reg =          1 ALU_OPR1 =         70 ALU_OPR2 =          8 ALU_OUT
=          78 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE =  1

6275 CLK = 1 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010101000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE = 11

6300 CLK = 0 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010101000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE = 11
```

Figure 110. Simulation Results for BLEZ Instruction Part 1

Analysis: Rs = R[21] = 1
Since 1 is not less than or equal to 0, this is a no branch condition and next PC will be (PC + 1). Simulation results show that PC_out for this instruction is 69 so the next instruction should be at PC_out = 70.

- BLEZ (I-type): if Rs ≤ 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000110	10110	00000	0000 0000 0000 0010

RAM location: 0x046 (70 in decimal)

Instruction in Hex: 0x1AC00002

Result: 71 + (2 x 4) = 79 (PC ← 79)

```
//BLEZ (BRANCH)

6325 CLK = 1 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011010110000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         71 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         70 ALU_OPR2 =          1 ALU_OUT
=          71 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE =  0

6350 CLK = 0 RST = 1 PC_out =         70 RAM_Addr =         70 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011010110000000000000000000010 IR_out =
00011010101000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         71 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         70 ALU_OPR2 =          1 ALU_OUT
=          71 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         70 CAUSE_IN =          0 STATE =  0

6375 CLK = 1 RST = 1 PC_out =         71 RAM_Addr =         71 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010110000000000000000000010 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         79 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         71 ALU_OPR2 =          8 ALU_OUT
=          79 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         71 CAUSE_IN =          0 STATE =  1

6400 CLK = 0 RST = 1 PC_out =         71 RAM_Addr =         71 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010110000000000000000000010 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         79 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         71 ALU_OPR2 =          8 ALU_OUT
=          79 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         71 CAUSE_IN =          0 STATE =  1

6425 CLK = 1 RST = 1 PC_out =         71 RAM_Addr =         71 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010110000000000000000000010 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =          0 ALU_OPR2 =          0 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         71 CAUSE_IN =          0 STATE = 11

6450 CLK = 0 RST = 1 PC_out =         71 RAM_Addr =         71 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011010110000000000000000000010 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =          0 ALU_OPR2 =          0 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         71 CAUSE_IN =          0 STATE = 11
```

Figure 111. Simulation Results for BLEZ Instruction Part 2

Analysis: Rs = R[22] = 0

Since 0 is equal to 0, the branch is executed and the next instruction will be at PC + 1 + ((sign extended I[15:0]) || 00).

PC + 1 = 71 ((sign extended I[15:0]) || 00) = 2 x 4 = 8

The next instruction should be at PC_out = 71 + 8 = 79

(NOTE: This instruction is at PC_out = 70, which is an indication that the previous BLEZ instruction executed properly by not branching).

- SLL (R-type): Rd ← Rt shifted left by Shamt Rt = R[12] = 0xAA Rd = R[23] Shamt = 8

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00000	01100	10111	01000	000000

RAM location: 0x04F (79 in decimal)

Instruction in Hex: 0xCBA00

Result: 1010 1010 0000 0000 → 43520 → R[23]

```
//SLL

6475 CLK = 1 RST = 1 PC_out =         79 RAM_Addr =         79 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000011001011101000000000 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         80 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         79 ALU_OPR2 =          1 ALU_OUT
=          80 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         79 CAUSE_IN =          0 STATE =  0

6500 CLK = 0 RST = 1 PC_out =         79 RAM_Addr =         79 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000011001011101000000000 IR_out =
00011010110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         80 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         79 ALU_OPR2 =          1 ALU_OUT
=          80 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         79 CAUSE_IN =          0 STATE =  0

6525 CLK = 1 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData = 4294895696 RegAddr =         12 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         80 ALU_OPR2 = 4294895616 ALU_OUT
=      -71600 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  1

6550 CLK = 0 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData = 4294895696 RegAddr =         12 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         80 ALU_OPR2 = 4294895616 ALU_OUT
=      -71600 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  1

6575 CLK = 1 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData =      43520 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =          2 ALU_OPR2 =        170 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  4

6600 CLK = 0 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData =      43520 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =          2 ALU_OPR2 =        170 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  4

6625 CLK = 1 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 1 REG_OE = 1 RegData =      43520 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =          2 ALU_OPR2 =        170 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  5

6650 CLK = 0 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000011001011101000000000 IR_out =
00000000000011001011101000000000 REG_WS = 1 REG_OE = 1 RegData =      43520 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =          2 ALU_OPR2 =        170 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  5
```

Figure 112. Simulation Results for SLL Instruction

Analysis: Rt = R[12] = 0xAA Rd = R[23] Shamt = 8

1010 1010 0000 0000 → 43520 → R[23]

Simulation results show that RegData = 43520 and RegAddr = 23 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 79, which is an indication that the previous BLEZ instruction executed properly).

- SLLV (R-type): Rd ← Rt shifted left by Rs

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00011	10111	11000	00000	000100

RAM location: 0x050 (80 in decimal)
Instruction in Hex: 0x77C004
Result: 1010 1010 0000 0000 0000 → 696320 → R[24]

```
//SLLV

6675 CLK = 1 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000011101111100000000000100 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData =         81 RegAddr =         12 A_Reg =          2 B_Reg =        170 ALU_OPR1 =         80 ALU_OPR2 =          1 ALU_OUT
=         81 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  0

6700 CLK = 0 RST = 1 PC_out =         80 RAM_Addr =         80 RAM_Data =        170 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000011101111100000000000100 IR_out =
00000000000011001011101000000000 REG_WS = 0 REG_OE = 1 RegData =         81 RegAddr =         12 A_Reg =          2 B_Reg =        170 ALU_OPR1 =         80 ALU_OPR2 =          1 ALU_OUT
=         81 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         80 CAUSE_IN =          0 STATE =  0

6725 CLK = 1 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData = 4294901857 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =         81 ALU_OPR2 = 4294901776 ALU_OUT
=      -65439 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  1

6750 CLK = 0 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =        170 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData = 4294901857 RegAddr =         23 A_Reg =          2 B_Reg =        170 ALU_OPR1 =         81 ALU_OPR2 = 4294901776 ALU_OUT
=      -65439 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  1

6775 CLK = 1 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData =     696320 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =          4 ALU_OPR2 =      43520 ALU_OUT
=      696320 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  4

6800 CLK = 0 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData =     696320 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =          4 ALU_OPR2 =      43520 ALU_OUT
=      696320 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  4

6825 CLK = 1 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 1 REG_OE = 1 RegData =     696320 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =          4 ALU_OPR2 =      43520 ALU_OUT
=      696320 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  5

6850 CLK = 0 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000011101111100000000000100 IR_out =
00000000011101111100000000000100 REG_WS = 1 REG_OE = 1 RegData =     696320 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =          4 ALU_OPR2 =      43520 ALU_OUT
=      696320 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  5
```

Figure 113. Simulation Results for SLLV Instruction

Analysis: Rt = R[23] = 0xAA00 Rs = R[3] = 4 Rd = R[24]
1010 1010 0000 0000 0000 → 696320 → R[24]
Simulation results show that RegData = 696320 and RegAddr = 24 while the REG_WS signal goes high on the last clock cycle.

- SRL (R-type): Rd ← Rt shifted right by Shamt

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00000	11000	11001	00100	000010

RAM location: 0x051 (81 in decimal)

Instruction in Hex: 0x18C902

Result: 1010 1010 0000 0000 → 43520 → R[25]

```
//SRL

6875 CLK = 1 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000110001100100100000010 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData =         82 RegAddr =         23 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =         81 ALU_OPR2 =          1 ALU_OUT
=          82 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  0

6900 CLK = 0 RST = 1 PC_out =         81 RAM_Addr =         81 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000110001100100100000010 IR_out =
00000000011101111100000000000100 REG_WS = 0 REG_OE = 1 RegData =         82 RegAddr =         23 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =         81 ALU_OPR2 =          1 ALU_OUT
=          82 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         81 CAUSE_IN =          0 STATE =  0

6925 CLK = 1 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData = 4294911066 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =         82 ALU_OPR2 = 4294910984 ALU_OUT
=      -56230 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  1

6950 CLK = 0 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData = 4294911066 RegAddr =         24 A_Reg =          4 B_Reg =      43520 ALU_OPR1 =         82 ALU_OPR2 = 4294910984 ALU_OUT
=      -56230 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  1

6975 CLK = 1 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData =      43520 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =          2 ALU_OPR2 =     696320 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  4

7000 CLK = 0 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData =      43520 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =          2 ALU_OPR2 =     696320 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  4

7025 CLK = 1 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 1 REG_OE = 1 RegData =      43520 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =          2 ALU_OPR2 =     696320 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  5

7050 CLK = 0 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000110001100100100000010 IR_out =
00000000000110001100100100000010 REG_WS = 1 REG_OE = 1 RegData =      43520 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =          2 ALU_OPR2 =     696320 ALU_OUT
=       43520 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  5
```

Figure 114. Simulation Results for SRL Instruction

Analysis: Rt = R[24] = 0xAA000 Rd = R[25] Shamt = 4

1010 1010 0000 0000 → 43520 → R[25]

Simulation results show that RegData = 43520 and RegAddr = 25 while the REG_WS signal goes high on the last clock cycle.

- SRLV (R-type): Rd ← Rt shifted right by Rs

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00001	11001	11010	00000	000110

RAM location: 0x052 (82 in decimal)

Instruction in Hex: 0x39D006

Result: 0001 0101 0100 0000 → 5440 → R[26]

```
//SRLV

7075 CLK = 1 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001110011101000000000110 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData =         83 RegAddr =         24 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =         82 ALU_OPR2 =          1 ALU_OUT
=          83 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  0

7100 CLK = 0 RST = 1 PC_out =         82 RAM_Addr =         82 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001110011101000000000110 IR_out =
00000000000110001100100100000010 REG_WS = 0 REG_OE = 1 RegData =         83 RegAddr =         24 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =         82 ALU_OPR2 =          1 ALU_OUT
=          83 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         82 CAUSE_IN =          0 STATE =  0

7125 CLK = 1 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData = 4294918251 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =         83 ALU_OPR2 = 4294918168 ALU_OUT
=      -49045 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  1

7150 CLK = 0 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =     696320 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData = 4294918251 RegAddr =         25 A_Reg =          2 B_Reg =     696320 ALU_OPR1 =         83 ALU_OPR2 = 4294918168 ALU_OUT
=      -49045 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  1

7175 CLK = 1 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData =       5440 RegAddr =         26 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =          3 ALU_OPR2 =      43520 ALU_OUT
=        5440 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  4

7200 CLK = 0 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData =       5440 RegAddr =         26 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =          3 ALU_OPR2 =      43520 ALU_OUT
=        5440 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  4

7225 CLK = 1 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 1 REG_OE = 1 RegData =       5440 RegAddr =         26 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =          3 ALU_OPR2 =      43520 ALU_OUT
=        5440 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  5

7250 CLK = 0 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001110011101000000000110 IR_out =
00000000001110011101000000000110 REG_WS = 1 REG_OE = 1 RegData =       5440 RegAddr =         26 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =          3 ALU_OPR2 =      43520 ALU_OUT
=        5440 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  5
```

Figure 115. Simulation Results for SRLV Instruction

Analysis: Rt = R[25] = 0xAA00 Rd = R[26] Rs = R[1] = 3

0001 0101 0100 0000 → 5440 → R[26]

Simulation results show that RegData = 5440 and RegAddr = 26 while the REG_WS signal goes high on the last clock cycle.

- SRA (R-type): Rd ← Rt shifted right (Arithmetic) by Shamt

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00000	01000	11100	10000	000011

RAM location: 0x053 (83 in decimal)

Instruction in Hex: 0x8E403

Result: 1111 1111 1111 1111 1010 1010 1010 1010 → 4294945450 → R[28]

```
//SRA

7275 CLK = 1 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000010001110010000000011 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData =         84 RegAddr =         25 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =         83 ALU_OPR2 =          1 ALU_OUT
=          84 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  0

7300 CLK = 0 RST = 1 PC_out =         83 RAM_Addr =         83 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000010001110010000000011 IR_out =
00000000001110011101000000000110 REG_WS = 0 REG_OE = 1 RegData =         84 RegAddr =         25 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =         83 ALU_OPR2 =          1 ALU_OUT
=          84 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         83 CAUSE_IN =          0 STATE =  0

7325 CLK = 1 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData = 4294938720 RegAddr =          8 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =         84 ALU_OPR2 = 4294938636 ALU_OUT
=      -28576 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  1

7350 CLK = 0 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data =      43520 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData = 4294938720 RegAddr =          8 A_Reg =          3 B_Reg =      43520 ALU_OPR1 =         84 ALU_OPR2 = 4294938636 ALU_OUT
=      -28576 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  1

7375 CLK = 1 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =          2 ALU_OPR2 = 2863311530 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  4

7400 CLK = 0 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData = 4294945450 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =          2 ALU_OPR2 = 2863311530 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  4

7425 CLK = 1 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 1 REG_OE = 1 RegData = 4294945450 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =          2 ALU_OPR2 = 2863311530 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  5

7450 CLK = 0 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000010001110010000000011 IR_out =
00000000000010001110010000000011 REG_WS = 1 REG_OE = 1 RegData = 4294945450 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =          2 ALU_OPR2 = 2863311530 ALU_OUT
=      -21846 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  5
```

Figure 116. Simulation Results for SRA Instruction

Analysis: Rt = Rt = R[8] = 0xAAAAAAAA Rd = R[28] Shamt = 16

1111 1111 1111 1111 1010 1010 1010 1010 → 4294945450 → R[28]

Simulation results show that RegData = 4294945450 and RegAddr = 28 while the REG_WS signal goes high on the last clock cycle.

- SRAV (R-type): Rd ← Rt shifted right (Arithmetic) by Rs

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00000	11100	11101	00000	000111

RAM location: 0x054 (84 in decimal)

Instruction in Hex: 0x1CE807

Result: 1111 1111 1111 1111 1110 1010 1010 1010 → 4294961834 → R[29]

```
//SRAV

7475 CLK = 1 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData =         85 RegAddr =          8 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =         84 ALU_OPR2 =          1 ALU_OUT
=          85 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  0

7500 CLK = 0 RST = 1 PC_out =         84 RAM_Addr =         84 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000010001110010000000011 REG_WS = 0 REG_OE = 1 RegData =         85 RegAddr =          8 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =         84 ALU_OPR2 =          1 ALU_OUT
=          85 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         84 CAUSE_IN =          0 STATE =  0

7525 CLK = 1 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData = 4294942833 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =         85 ALU_OPR2 = 4294942748 ALU_OUT
=      -24463 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  1

7550 CLK = 0 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 2863311530 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData = 4294942833 RegAddr =         28 A_Reg =          2 B_Reg = 2863311530 ALU_OPR1 =         85 ALU_OPR2 = 4294942748 ALU_OUT
=      -24463 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  1

7575 CLK = 1 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData = 4294961834 RegAddr =         29 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =          2 ALU_OPR2 = 4294945450 ALU_OUT
=       -5462 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  4

7600 CLK = 0 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData = 4294961834 RegAddr =         29 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =          2 ALU_OPR2 = 4294945450 ALU_OUT
=       -5462 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  4

7625 CLK = 1 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 1 REG_OE = 1 RegData = 4294961834 RegAddr =         29 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =          2 ALU_OPR2 = 4294945450 ALU_OUT
=       -5462 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  5

7650 CLK = 0 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000111001110100000000111 IR_out =
00000000000111001110100000000111 REG_WS = 1 REG_OE = 1 RegData = 4294961834 RegAddr =         29 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =          2 ALU_OPR2 = 4294945450 ALU_OUT
=       -5462 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  5
```

Figure 117. Simulation Results for SRAV Instruction

Analysis: Rt = Rt = R[28] = 0xFFFFAAAA Rd = R[29] Rs = R[0] = 2

1111 1111 1111 1111 1110 1010 1010 1010 → 4294961834 → R[29]

Simulation results show that RegData = 4294961834 and RegAddr = 29 while the REG_WS signal goes high on the last clock cycle.

- BGTZ (I-type): if Rs > 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000111	10110	00000	0000 0000 0000 0010

RAM location: 0x055 (85 in decimal)

Instruction in Hex: 0x1EC00002

Result: No Branch (PC ← PC + 1)

```
//BGTZ (NO BRANCH)

7675 CLK = 1 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011110110000000000000000000010 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData =         86 RegAddr =         28 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =         85 ALU_OPR2 =          1 ALU_OUT
=          86 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  0

7700 CLK = 0 RST = 1 PC_out =         85 RAM_Addr =         85 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011110110000000000000000000010 IR_out =
00000000000111001110100000000111 REG_WS = 0 REG_OE = 1 RegData =         86 RegAddr =         28 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =         85 ALU_OPR2 =          1 ALU_OUT
=          86 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         85 CAUSE_IN =          0 STATE =  0

7725 CLK = 1 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011110110000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         94 RegAddr =          0 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =         86 ALU_OPR2 =          8 ALU_OUT
=          94 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  1

7750 CLK = 0 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data = 4294945450 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011110110000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         94 RegAddr =          0 A_Reg =          2 B_Reg = 4294945450 ALU_OPR1 =         86 ALU_OPR2 =          8 ALU_OUT
=          94 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  1

7775 CLK = 1 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011110110000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =          0 ALU_OPR2 =          0 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE = 12

7800 CLK = 0 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011110110000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =          0 ALU_OPR2 =          0 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE = 12
```

Figure 118. Simulation Results for BGTZ Instruction Part 1

Analysis: Rs = R[22] = 0

Since 0 is not greater than 0, this is a no branch condition and next PC will be (PC + 1). Simulation results show that PC_out for this instruction is 85 so the next instruction should be at PC_out = 86.

- BGTZ (I-type): if Rs > 0 then PC ← PC + 1 + ((sign extended I[15:0]) || 00)

Opcode	Rs	Rt	Address/Constant
000111	00010	00000	0000 0000 0000 0010

RAM location: 0x056 (86 in decimal)

Instruction in Hex: 0x1C400002

Result: 87 + (2 x 4) = 95 (PC ← 95)

```
//BGTZ (BRANCH)

7825 CLK = 1 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011100010000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         87 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         86 ALU_OPR2 =          1 ALU_OUT
=          87 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  0

7850 CLK = 0 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011100010000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         87 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         86 ALU_OPR2 =          1 ALU_OUT
=          87 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  0

7875 CLK = 1 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         95 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         87 ALU_OPR2 =          8 ALU_OUT
=          95 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE =  1

7900 CLK = 0 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         95 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         87 ALU_OPR2 =          8 ALU_OUT
=          95 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE =  1

7925 CLK = 1 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE = 12

7950 CLK = 0 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE = 12
```

Figure 119. Simulation Results for BGTZ Instruction Part 2

Analysis: Rs = R[2] = 1

Since 1 is greater than 0, the branch is executed and the next instruction will be at PC + 1 + ((sign extended I[15:0]) || 00).

PC + 1 = 87 ((sign extended I[15:0]) || 00) = 2 x 4 = 8

The next instruction should be at PC_out = 87 + 8 = 95

(NOTE: This instruction is at PC_out = 86, which is an indication that the previous BGTZ instruction executed properly by not branching).

- SLTu (R-type): if Rs < Rt then Rd ← 1
  else Rd ← 0

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00000	00010	00100	00000	101001

RAM location: 0x05F (95 in decimal)

Instruction in Hex: 0x22029

Result: 0 → R[4]

```
//SLTu

7975 CLK = 1 RST = 1 PC_out =         95 RAM_Addr =         95 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000100010000000101001 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         96 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         95 ALU_OPR2 =          1 ALU_OUT
=          96 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         95 CAUSE_IN =          0 STATE =  0

8000 CLK = 0 RST = 1 PC_out =         95 RAM_Addr =         95 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000100010000000101001 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         96 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         95 ALU_OPR2 =          1 ALU_OUT
=          96 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         95 CAUSE_IN =          0 STATE =  0

8025 CLK = 1 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =      33028 RegAddr =          2 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         96 ALU_OPR2 =      32932 ALU_OUT
=       33028 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  1

8050 CLK = 0 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =      33028 RegAddr =          2 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         96 ALU_OPR2 =      32932 ALU_OUT
=       33028 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  1

8075 CLK = 1 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  4

8100 CLK = 0 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  4

8125 CLK = 1 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  5

8150 CLK = 0 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          4 A_Reg =          2 B_Reg =          1 ALU_OPR1 =          2 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  5
```

Figure 120. Simulation Results for SLTu Instruction Part 1

Analysis: Rs = R[0] = 2 Rt = R[2] = 1 Rd = R[4]

Since 2 is not less than 1, 0 → R[4].

Simulation results show that RegData = 0 and RegAddr = 4 while the REG_WS signal goes high on the last clock cycle.

(NOTE: This instruction is at PC_out = 95, which is an indication that the previous BGTZ instruction executed properly).

- SLTu (R-type): if Rs < Rt then Rd ← 1
  else Rd ← 0

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00100	00010	00100	00000	101001

RAM location: 0x060 (96 in decimal)

Instruction in Hex: 0x822029

Result: 1 → R[4]

```
//SLTu

8175 CLK = 1 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000100000100010000000101001 IR_out =
00000000000000010001000000101001 REG_WS = 0 REG_OE = 1 RegData =         97 RegAddr =          2 A_Reg =          2 B_Reg =          1 ALU_OPR1 =         96 ALU_OPR2 =          1 ALU_OUT
=          97 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  0

8200 CLK = 0 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000100000100010000000101001 IR_out =
00000000000000010001000000101001 REG_WS = 0 REG_OE = 1 RegData =         97 RegAddr =          2 A_Reg =          2 B_Reg =          1 ALU_OPR1 =         96 ALU_OPR2 =          1 ALU_OUT
=          97 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  0

8225 CLK = 1 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =      33029 RegAddr =          2 A_Reg =          2 B_Reg =          1 ALU_OPR1 =         97 ALU_OPR2 =      32932 ALU_OUT
=       33029 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  1

8250 CLK = 0 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =      33029 RegAddr =          2 A_Reg =          2 B_Reg =          1 ALU_OPR1 =         97 ALU_OPR2 =      32932 ALU_OUT
=       33029 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  1

8275 CLK = 1 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          4 A_Reg =          0 B_Reg =          1 ALU_OPR1 =          0 ALU_OPR2 =          1 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  4

8300 CLK = 0 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          4 A_Reg =          0 B_Reg =          1 ALU_OPR1 =          0 ALU_OPR2 =          1 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  4

8325 CLK = 1 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =          4 A_Reg =          0 B_Reg =          1 ALU_OPR1 =          0 ALU_OPR2 =          1 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  5

8350 CLK = 0 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100000100010000000101001 IR_out =
00000000100000100010000000101001 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =          4 A_Reg =          0 B_Reg =          1 ALU_OPR1 =          0 ALU_OPR2 =          1 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  5
```

Figure 121. Simulation Results for SLTu Instruction Part 2

Analysis: Rs = R[4] = 0 Rt = R[2] = 1 Rd = R[4]

Since 0 is less than 1, 1 → R[4].

Simulation results show that RegData = 1 and RegAddr = 4 while the REG_WS signal goes high on the last clock cycle.

- SLT (R-type): if Rs < Rt then Rd ← 1
  else Rd ← 0

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00100	01011	00101	00000	101010

RAM location: 0x061 (97 in decimal)

Instruction in Hex: 0x8B282A

Result: 0 → R[5]

```
//SLT

8375 CLK = 1 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =         98 RegAddr =          2 A_Reg =          0 B_Reg =          1 ALU_OPR1 =         97 ALU_OPR2 =          1 ALU_OUT
=          98 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  0

8400 CLK = 0 RST = 1 PC_out =         97 RAM_Addr =         97 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =         98 RegAddr =          2 A_Reg =          0 B_Reg =          1 ALU_OPR1 =         97 ALU_OPR2 =          1 ALU_OUT
=          98 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         97 CAUSE_IN =          0 STATE =  0

8425 CLK = 1 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =      41226 RegAddr =         11 A_Reg =          0 B_Reg =          1 ALU_OPR1 =         98 ALU_OPR2 =      41128 ALU_OUT
=       41226 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  1

8450 CLK = 0 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =      41226 RegAddr =         11 A_Reg =          0 B_Reg =          1 ALU_OPR1 =         98 ALU_OPR2 =      41128 ALU_OUT
=       41226 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  1

8475 CLK = 1 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          5 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =          1 ALU_OPR2 = 4294967210 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  4

8500 CLK = 0 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          5 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =          1 ALU_OPR2 = 4294967210 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  4

8525 CLK = 1 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          5 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =          1 ALU_OPR2 = 4294967210 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  5

8550 CLK = 0 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000100010110010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          5 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =          1 ALU_OPR2 = 4294967210 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  5
```

Figure 122. Simulation Results for SLT Instruction Part 1

Analysis: Rs = R[4] = 1 Rt = R[11] = 0xFFFFFFAA Rd = R[5]

Since 1 is not less than FFFFFFAA (signed), 0 → R[5].

Simulation results show that RegData = 0 and RegAddr = 5 while the REG_WS signal goes high on the last clock cycle.

- SLT (R-type): if Rs < Rt then Rd ← 1
  else Rd ← 0

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	01000	00010	00101	00000	101010

RAM location: 0x062 (98 in decimal)

Instruction in Hex: 0x102282A

Result: 1 → R[5]

```
//SLT

8575 CLK = 1 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001000000100010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =         99 RegAddr =         11 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =         98 ALU_OPR2 =          1 ALU_OUT
=         99 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  0

8600 CLK = 0 RST = 1 PC_out =         98 RAM_Addr =         98 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001000000100010100000101010 IR_out =
00000000100010110010100000101010 REG_WS = 0 REG_OE = 1 RegData =         99 RegAddr =         11 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =         98 ALU_OPR2 =          1 ALU_OUT
=         99 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         98 CAUSE_IN =          0 STATE =  0

8625 CLK = 1 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =      41227 RegAddr =          2 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =         99 ALU_OPR2 =      41128 ALU_OUT
=      41227 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  1

8650 CLK = 0 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data = 4294967210 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =      41227 RegAddr =          2 A_Reg =          1 B_Reg = 4294967210 ALU_OPR1 =         99 ALU_OPR2 =      41128 ALU_OUT
=      41227 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  1

8675 CLK = 1 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          5 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 = 2863311530 ALU_OPR2 =          1 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  4

8700 CLK = 0 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          5 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 = 2863311530 ALU_OPR2 =          1 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  4

8725 CLK = 1 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =          5 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 = 2863311530 ALU_OPR2 =          1 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  5

8750 CLK = 0 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001000000100010100000101010 IR_out =
00000001000000100010100000101010 REG_WS = 1 REG_OE = 1 RegData =          1 RegAddr =          5 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 = 2863311530 ALU_OPR2 =          1 ALU_OUT
=          1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  5
```

Figure 123. Simulation Results for SLT Instruction Part 2

Analysis: Rs = R[8] = 0xAAAAAAAA Rt = R[2] = 1 Rd = R[5]

Since 0xAAAAAAAA (signed) is less than 1, 1 → R[5].

Simulation results show that RegData = 1 and RegAddr = 5 while the REG_WS signal goes high on the last clock cycle.

- JAL (J-type): PC ← PC[31:28] || Inst[25:0] || 00

Opcode	Jump Target
000011	00 0000 0000 0000 0000 0001 1110

RAM location: 0x063 (99 in decimal)

Instruction in Hex: 0xC00001E

Result: 30 x 4 = 120 (PC ← 120) (R[31] ← 100)

```
//JAL

8775 CLK = 1 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00001100000000000000000000011110 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =          2 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 =         99 ALU_OPR2 =          1 ALU_OUT
=         100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  0

8800 CLK = 0 RST = 1 PC_out =         99 RAM_Addr =         99 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00001100000000000000000000011110 IR_out =
00000001000000100010100000101010 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =          2 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 =         99 ALU_OPR2 =          1 ALU_OUT
=         100 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         99 CAUSE_IN =          0 STATE =  0

8825 CLK = 1 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        220 RegAddr =          0 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE =  1

8850 CLK = 0 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        220 RegAddr =          0 A_Reg = 2863311530 B_Reg =          1 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE =  1

8875 CLK = 1 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         31 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE =  3

8900 CLK = 0 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        100 RegAddr =         31 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE =  3

8925 CLK = 1 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 1 REG_OE = 1 RegData =        100 RegAddr =         31 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE = 46

8950 CLK = 0 RST = 1 PC_out =        100 RAM_Addr =        100 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00001100000000000000000000011110 IR_out =
00001100000000000000000000011110 REG_WS = 1 REG_OE = 1 RegData =        100 RegAddr =         31 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        100 ALU_OPR2 =        120 ALU_OUT
=         220 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        100 CAUSE_IN =          0 STATE = 46
```

Figure 124. Simulation Results for JAL Instruction

Analysis: 0000 || 00000000000000000000011110 || 00 = 1111000 = 120

The next PC will be at PC_out = 120

$ra ← PC + 1 R[31] ← 100

Simulation results show that RegData = 100 and RegAddr = 31 while the REG_WS signal goes high on the last clock cycle. Additionally, the last instruction (JALR) will restore the PC to the $ra value of 100 for further verification.

- ADDu (R-type): Rs + Rt → Rd

Rs = R[10] = FFFFFFFF Rt = R[2] = 1 Rd = R[6]

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	01010	00010	00110	00000	100001

RAM location: 0x078 (120 in decimal)

Instruction in Hex: 0x1423021

Result: 0 → R[6]

```
//ADDu

8975 CLK = 1 RST = 1 PC_out =        120 RAM_Addr =        120 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001010000100011000000100001 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        121 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        120 ALU_OPR2 =          1 ALU_OUT
=        121 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        120 CAUSE_IN =          0 STATE =  0

9000 CLK = 0 RST = 1 PC_out =        120 RAM_Addr =        120 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001010000100011000000100001 IR_out =
00001100000000000000000000011110 REG_WS = 0 REG_OE = 1 RegData =        121 RegAddr =          0 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        120 ALU_OPR2 =          1 ALU_OUT
=        121 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        120 CAUSE_IN =          0 STATE =  0

9025 CLK = 1 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 0 REG_OE = 1 RegData =      49405 RegAddr =          2 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        121 ALU_OPR2 =      49284 ALU_OUT
=       49405 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  1

9050 CLK = 0 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 0 REG_OE = 1 RegData =      49405 RegAddr =          2 A_Reg =          2 B_Reg =          2 ALU_OPR1 =        121 ALU_OPR2 =      49284 ALU_OUT
=       49405 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  1

9075 CLK = 1 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          6 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  4

9100 CLK = 0 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 0 REG_OE = 1 RegData =          0 RegAddr =          6 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  4

9125 CLK = 1 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          6 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  5

9150 CLK = 0 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011000000100001 IR_out =
00000001010000100011000000100001 REG_WS = 1 REG_OE = 1 RegData =          0 RegAddr =          6 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=           0 NF = 0 ZF = 1 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  5
```

Figure 125. Simulation Results for ADDu Instruction

Analysis: Rs = R[10] = 0xFFFFFFFFRt = R[2] = 1 Rd = R[6]

(0xFFFFFFFF) + 1 → 0 → R[6]

Simulation results show that RegData = 0 and RegAddr = 6 while the REG_WS signal goes high on the last clock cycle.

(NOTE: The results are 0 because the 1 gets carried out.)

- SUBu (R-type): Rs - Rt → Rd

Rs = R[10] = FFFFFFFF Rt = R[2] = 1 Rd = R[7]

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	01010	00010	00111	00000	100011

RAM location: 0x079 (121 in decimal)

Instruction in Hex: 0x1423823

Result: 0x FFFFFFFE → 4294967294 → R[7]

```
//SUBu

9175 CLK = 1 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011000000100001 REG_WS = 0 REG_OE = 1 RegData =        122 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        121 ALU_OPR2 =          1 ALU_OUT
=         122 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  0

9200 CLK = 0 RST = 1 PC_out =        121 RAM_Addr =        121 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100001 REG_WS = 0 REG_OE = 1 RegData =        122 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        121 ALU_OPR2 =          1 ALU_OUT
=         122 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        121 CAUSE_IN =          0 STATE =  0

9225 CLK = 1 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData =      57606 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        122 ALU_OPR2 =      57484 ALU_OUT
=       57606 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  1

9250 CLK = 0 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData =      57606 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        122 ALU_OPR2 =      57484 ALU_OUT
=       57606 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  1

9275 CLK = 1 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData = 4294967294 RegAddr =          7 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=          -2 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  4

9300 CLK = 0 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData = 4294967294 RegAddr =          7 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=          -2 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  4

9325 CLK = 1 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 1 REG_OE = 1 RegData = 4294967294 RegAddr =          7 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=          -2 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  5

9350 CLK = 0 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000001010000100011100000100011 IR_out =
00000001010000100011100000100011 REG_WS = 1 REG_OE = 1 RegData = 4294967294 RegAddr =          7 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 = 4294967295 ALU_OPR2 =          1 ALU_OUT
=          -2 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  5
```

Figure 126. Simulation Results for SUBu Instruction

Analysis: Rs = R[10] = FFFFFFFF Rt = R[2] = 1 Rd = R[7]

(0xFFFFFFFF) - 1 → 0xFFFFFFFE → 4294967294 → R[7]

Simulation results show that RegData = 4294967294 and RegAddr = 7 while the REG_WS signal goes high on the last clock cycle.

- JALR (R-type): Rs → PC PC+1 → Rd

Rs = $ra = R[31] = 100 Rd = R[21]

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	11111	00000	10101	00000	001001

RAM location: 0x07A (122 in decimal)

Instruction in Hex: 0x3E0A809

Result: 100 → PC 123 → R[21]

```
//JALR

9375 CLK = 1 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000011111000001010100000001001 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData =        123 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        122 ALU_OPR2 =          1 ALU_OUT
=        123 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  0

9400 CLK = 0 RST = 1 PC_out =        122 RAM_Addr =        122 RAM_Data =          1 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000011111000001010100000001001 IR_out =
00000001010000100011100000100011 REG_WS = 0 REG_OE = 1 RegData =        123 RegAddr =          2 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        122 ALU_OPR2 =          1 ALU_OUT
=        123 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =        122 CAUSE_IN =          0 STATE =  0

9425 CLK = 1 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 0 REG_OE = 1 RegData = 4294877343 RegAddr =          0 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE =  1

9450 CLK = 0 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 0 REG_OE = 1 RegData = 4294877343 RegAddr =          0 A_Reg = 4294967295 B_Reg =          1 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE =  1

9475 CLK = 1 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 0 REG_OE = 1 RegData =        123 RegAddr =         21 A_Reg =        100 B_Reg =          2 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE = 21

9500 CLK = 0 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 0 REG_OE = 1 RegData =        123 RegAddr =         21 A_Reg =        100 B_Reg =          2 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE = 21

9525 CLK = 1 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 1 REG_OE = 1 RegData =        123 RegAddr =         21 A_Reg =        100 B_Reg =          2 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE = 45

9550 CLK = 0 RST = 1 PC_out =        123 RAM_Addr =        123 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000011111000001010100000001001 IR_out =
00000011111000001010100000001001 REG_WS = 1 REG_OE = 1 RegData =        123 RegAddr =         21 A_Reg =        100 B_Reg =          2 ALU_OPR1 =        123 ALU_OPR2 = 4294877220 ALU_OUT
=      -89953 NF = 1 ZF = 0 OF = 0 BF = 0 EPC_IN =        123 CAUSE_IN =          0 STATE = 45
```

Figure 127. Simulation Results for JALR Instruction Part 1

Analysis: Rs = $ra = R[31] = 100 100 → PC

Rd = R[21] PC + 1 → 123 → R[21]

Simulation results show that RegData = 123 and RegAddr = 21 while the REG_WS signal goes high on the last clock cycle.

The Simulation results below show the next instruction is at PC_out = 100, which is indication that the JALR operation executed correctly.

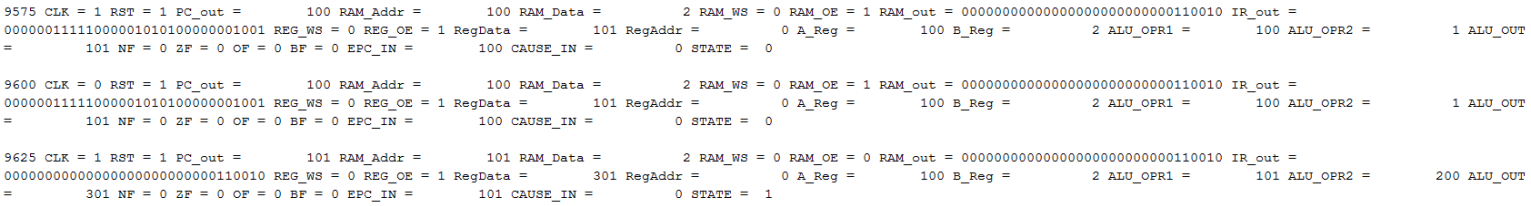


Figure 128. Simulation Results for JALR Instruction Part 2

8 PROCESSOR TIMING

Multi-cycle processors divided the data path into 5 cycles, namely FETCH, DECODE, EXECUTE, MEMORY, and WRITEBACK. Each processor cycle is executed in a single clock cycle. Not every MIPS instruction executed needs to go through each of the 5 cycles; in fact, in most multi-cycle processors, only the load operation utilize all 5 cycles. Some multi-cycle processors use the worst case amount of cycles and go through all 5 cycles for each instruction executed. This is inefficient because most instructions can be executed in 3 or 4 cycles, leaving the additional cycle time to be wasted, resulting in a decrease in throughput.

This multi-cycle processor uses functions in the Sequence Controller and State Register to execute each instruction in only the required number of cycles. As described in section 4.4, the Sequence Controller is comprised of a large state machine. Each state of the Sequence Controller state machine outputs the next state to be executed. This output (i.e. Next_State) is an actual output of the Sequence Controller and is wired as an input to the State Register. The State Register then latches this value in at the rising edge of system clock and outputs the value back into the asynchronous Sequence Controller to jump to that particular state. This process not only assures that the processor cycles are synced with the system clock, it also allows the Sequence Controller to jump to any processor cycle for increased throughput. Figure 129 shows a Branch-If-Greater-Than-Zero (BGTZ) instruction, executed in three clock cycles, followed by a Set-If-Less-Than-Unsigned (SLTu) instruction. Note the STATE output during the execution of the BGTZ instruction only going to states 0, 1, and 12. These are the actual Sequence Controller states for which the BGTZ instruction only requires three. Also note that total time it took to execute the instruction is 150ns (i.e. 7975ns – 7825ns), which is consistent with three clock periods at 50ns per period.

```
//BGTZ (BRANCH)

7825 CLK = 1 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011100010000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         87 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         86 ALU_OPR2 =          1 ALU_OUT
=          87 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  0

7850 CLK = 0 RST = 1 PC_out =         86 RAM_Addr =         86 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00011100010000000000000000000010 IR_out =
00011110110000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         87 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         86 ALU_OPR2 =          1 ALU_OUT
=          87 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         86 CAUSE_IN =          0 STATE =  0

7875 CLK = 1 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         95 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         87 ALU_OPR2 =          8 ALU_OUT
=          95 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE =  1

7900 CLK = 0 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         95 RegAddr =          0 A_Reg =          0 B_Reg =          2 ALU_OPR1 =         87 ALU_OPR2 =          8 ALU_OUT
=          95 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE =  1

7925 CLK = 1 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE = 12

7950 CLK = 0 RST = 1 PC_out =         87 RAM_Addr =         87 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00011100010000000000000000000010 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =          1 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =          1 ALU_OPR2 =          0 ALU_OUT
=           1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         87 CAUSE_IN =          0 STATE = 12


//SLTu

7975 CLK = 1 RST = 1 PC_out =         95 RAM_Addr =         95 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000100010000000101001 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         96 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         95 ALU_OPR2 =          1 ALU_OUT
=          96 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         95 CAUSE_IN =          0 STATE =  0

8000 CLK = 0 RST = 1 PC_out =         95 RAM_Addr =         95 RAM_Data =          2 RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000000000100010000000101001 IR_out =
00011100010000000000000000000010 REG_WS = 0 REG_OE = 1 RegData =         96 RegAddr =          0 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         95 ALU_OPR2 =          1 ALU_OUT
=          96 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         95 CAUSE_IN =          0 STATE =  0

8025 CLK = 1 RST = 1 PC_out =         96 RAM_Addr =         96 RAM_Data =          2 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000000000100010000000101001 IR_out =
00000000000000100010000000101001 REG_WS = 0 REG_OE = 1 RegData =      33028 RegAddr =          2 A_Reg =          1 B_Reg =          2 ALU_OPR1 =         96 ALU_OPR2 =      32932 ALU_OUT
=       33028 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_IN =         96 CAUSE_IN =          0 STATE =  1
```

Figure 129. Processor Timing Example

The multi-cycle processor is only as fast as its slowest cycle. This design was simulated using a 50ns system clock period as a reference, but in real applications, there are gate delays that need to be accounted for. In order to determine the fastest clock period we can use and still have the design function properly, we need to examine each cycle and set the clock period to cover the cycle with the most gate delay. In real life application, the actual delays are determined at the Place and Route phase of the design. For the purpose of this project, the delay estimates shown in Table 9 were developed. In order to perform the estimate, the longest gate path of the particular component was looked at, and then a 1ns delay was assigned for each gate.

Component	Longest Gate Path	Delay (ns)	Comment
Register with load enable	2 inverters, 1 AND, 1 OR, 2 NAND	6	Includes Program Counter, MIR, EPC, and CAUSE registers
Register with no load enable	2 NAND	2	Made up of D flip flops and includes all Registers associated with the ALU
MUX	1 inverter, 1 AND, 1 OR	3	This is the delay estimate for all Mux components
RAM	3 inverters, 2 AND, 2 OR, 2 NAND	9	RAM is a register with a mux on the front end for write capability and another mux on the back end for read capability
Register File	3 inverters, 2 AND, 2 OR, 2 NAND	9	Similar to RAM as both data I/O's are calculated in parallel
ALU Controller	1 AND, 1 OR	2	This is based on the ALU Controller logic shown in Table 3
ALU	3 inverter, 3 AND, 3 OR, 1 XOR	10	Calculated assuming subtraction as the worst case operation using a full adder/subtractor also adding three gate delays for the select circuitry
Combinational Block	2 OR, 1 Inverter, 1 AND	4	

NOTE:

1. Delays for Sequence Generator and State Register do not matter as the control signals are generated in parallel to the processors main data path and are never the slowest process.
2. There is no delay associated with the Sign Extend, Shift, and Concatenate components as they are composed out of wire and do not contain any gates.

Table 9. Component Gate Delays

The longest delay path that occurs during the FETCH is when the next Program Counter is calculated. That data path includes latching the new PC value into the Program Counter, MUX7, which is used to select the first ALU operand, the ALU, used to calculate the next program counter value, and MUX10, which is used to select the value of the next program counter. In accordance with Table 9, the maximum delay for the FETCH cycle is 22ns.

The longest delay path in the DECODE cycle occurs when the immediate branch address is calculated. The path includes MUX6, which is used to select the sign extended immediate value, MUX8, used to select the second operand of the ALU, and the ALU, which is used to calculate the branch address. In accordance with Table 9, the maximum delay for the DECODE cycle is 16ns.

The longest delay path in the EXECUTE cycle occurs during a Branch-If-Greater-Than-Zero operation (BGTZ) when the branch is taken. The path includes latching a new value into the Register File register (i.e. Reg1), MUX7, which is used to select the first ALU operand, the ALU, used to calculate the branch condition, and the Combinational Block, used to determine if the branch should be taken. In accordance with Table 9, the maximum delay for the EXECUTE cycle is 19ns. There is actually another path in the EXECUTE cycle which also consists of a 19ns delay. That path occurs when an immediate value is calculated and consists of MUX6, used to select between the sign extended immediate value, MUX8, used to select the second operand of the ALU, The ALU, which performs the required operation on the first operand and immediate value, and MUX5, used to select the ALU output to be stored in the Register File. As can be seen, both the BGTZ and immediate operation data paths consist of a delay of 19ns.

The longest delay path in the MEMORY cycle occurs during the Load Word operation where a word is read from RAM and propagates to the Register File to be stored on the next clock cycle. The path includes MUX1, which is used to select the RAM address, the RAM, which contains the word to be loaded into the Register File, MUX3, used to select the appropriate value coming out of RAM (sign extended if need be), and MUX5, used to select the data to be written into the Register File. In accordance with Table 9, the maximum delay for the MEMORY cycle is 18ns.

The longest delay path in the WRITEBACK cycle occurs during either the Load operation, where data is being written to the Register File resulting in a delay of 9ns, or a Store operation where data is written to RAM, also results in a 9ns delay.

Based on the analysis described above, the longest cycle is the FETCH cycle with a delay of 22ns. This means that in order to operate at maximum throughput, the clock period must be kept at 22ns. Anything faster would cause the processor to malfunction while a slower clock would negatively affect throughput. Note that this is only an estimate and is not representative of any current technology.

9 CONCLUSION

This project presents an example of a successful design, modeling, simulation, and timing analysis of a MIPS Multi-Cycle Processor using Verilog HDL. Before beginning the hardware design, several MIPS Instruction Set Formats were studied, and a subset of instructions were selected to be include in this project's design. From there, the hardware was built by implementing one instruction at a time, adding additional hardware as the project moved along until there was full conceptual drawing of the processor, including control logic state diagrams. After this, each component was implemented in Verilog HDL and was thoroughly tested, and the proper functionality was verified as each module was designed. Once all components were designed, they were utilized in creating the top-level processor design.

One of the main objectives for the processor design was to keep the execution of each instruction under five clock cycles. When the design architecture was first thought of, it was apparent there would be challenges when it came to actually implementing the hardware. While implementing the hardware, several adjustments had to be instituted, mainly at the top level of the design, in order to achieve the desired timing requirement.

Most problems within the processor's functionality where resolved at the component level. Each component was thoroughly tested with a module level bench test before being instantiated in the processor architecture. This was important in reducing the time it took to troubleshoot at the top level. The Sequence Controller is an exception as it was largely tested at the processor level. This is by design as testing the Sequence Controller at the component level can be very time consuming and, even then, it is not guaranteed that it will be fully functional as the processor design relies so heavily on its operation, particularly in regards to timing.

This project's processor design is fully functional as each supported instruction was tested in a robust manner. Each instruction set was tested using the processor to execute a MIPS code script while paying attention to processor performance. Future improvements, such as pipelining, can be added for increased throughput. This project should include all the information required for a full understanding of the processor functionality and performance.

Works Cited

Bistriceabu, Virgil. *Lab 7 Exceptions in MIPS*. Illinois Institute of Technology, College of Science. 1996. http://www.cs.iit.edu/~virgil/cs470/Labs/Lab7.pdf. Accessed last 10 November 2016.

Patterson, David A., and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 2012. Print.