UNIVERSITY OF CALIFORNIA IRVINE COURSE: ORGANIZATION OF DIGITAL COMPUTERS
LABORATORY (112L)
WINTER 2016

Lab 2 Report - Single-Cycle MIPS Datapath and Control

Prepared by: Team The Powerful Processors
Jon Raphael Apostol - 32302252
Binh Nguyen - 34707912
Yixiang Yan - 16392389
James Yi - 17492099

January 25, 2016

1 INTRODUCTION

In this lab, we implemented the single-cycle MIPS processor in VHDL. Since it is a single-cycle processor, there are no branch instructions, and the ALU only does addition, subtraction, the AND operation, the OR operation, the XOR operation, and the NOR operation. The branch instruction will be implemented in the later labs.

2 DESIGN

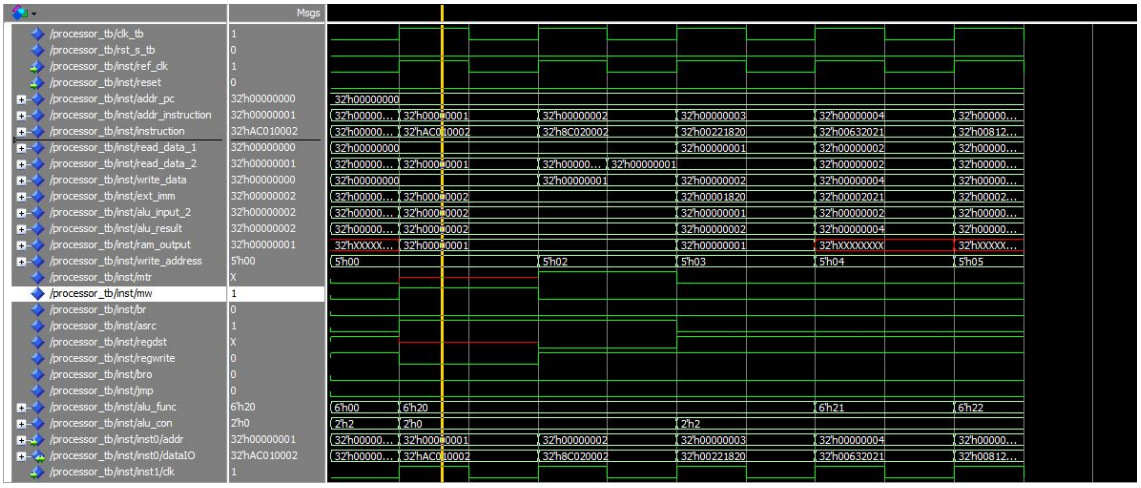
For our processor, we designed it in the following different blocks:

- The Program Counter (PC)
    1. The program counter contains the address of the instruction that is being executed.
    2. For our program counter we add 1 to the address everytime the clock hits a rising edge.
- The Instruction Memory (ROM)
    1. The instruction memory is implemented as a ROM. This is where the processor will receive the instructions from.
    2. The imem.h file is where we store instructions in hexadecimal form.
- The Register File
    1. The register file contains all the registers that each may store 32-bits of data.
- The Data Memory (RAM)
    1. The data memory will be a 512 long array of words with 32-bits each word.
- The ALU
    1. The ALU is where the mathematics of the processor is carried out and then stored in either the register file or the data memory.
- The Sign Extender
    1. The sign extender lengthens the 15 downto 0 bits of the instruction to 32-bits as a way to carry out I-type instructions.
- The Multiplexer
    1. The multiplexer selects one of two choices. This is used throughout the processor to select the type of instruction.
- The Controller
    1. The controller chooses the ALU control and is also a factor in controlling the type of instruction.
- The ALU Control
    1. Currently this is not being implemented, but it a way to increase the speed of the processor.
    2. This selects the operation the ALU will be doing.

3 TESTS

These are the tests we did on multiple commands that were given. We initialized the $1 to be equal to 1 in our code to test these instructions. Each instruction below was done in order.

Store Word (SW):



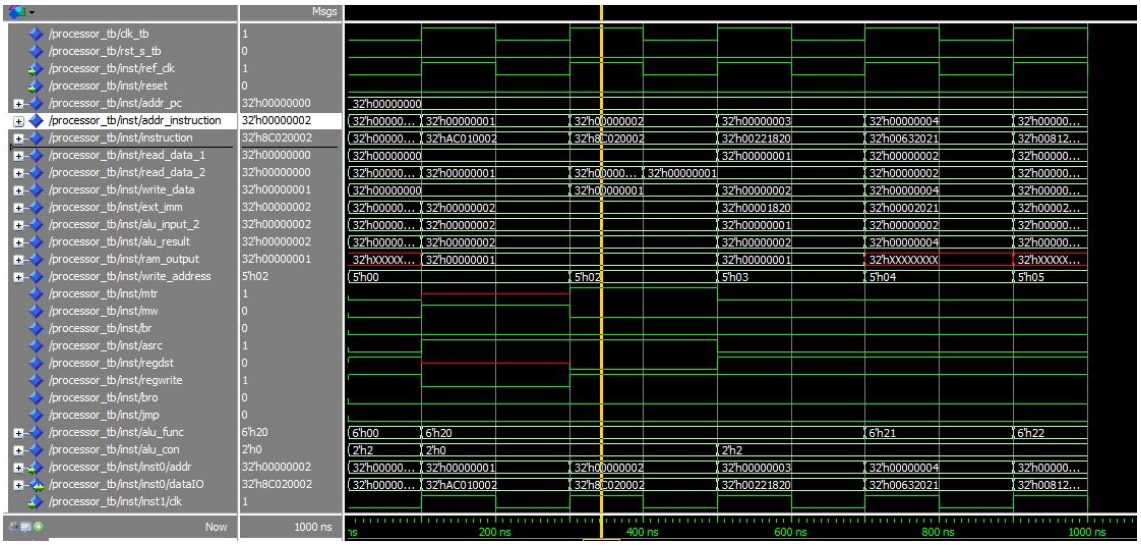
Hexadecimal: ac010002

Binary:
101011 00000 00001 0000000000000010

MIPS Code:
sw $0, $0, $1

This instruction stores the word from register 1 to register 0, thus making register 0 equal to ’1’;

Load Word (LW):



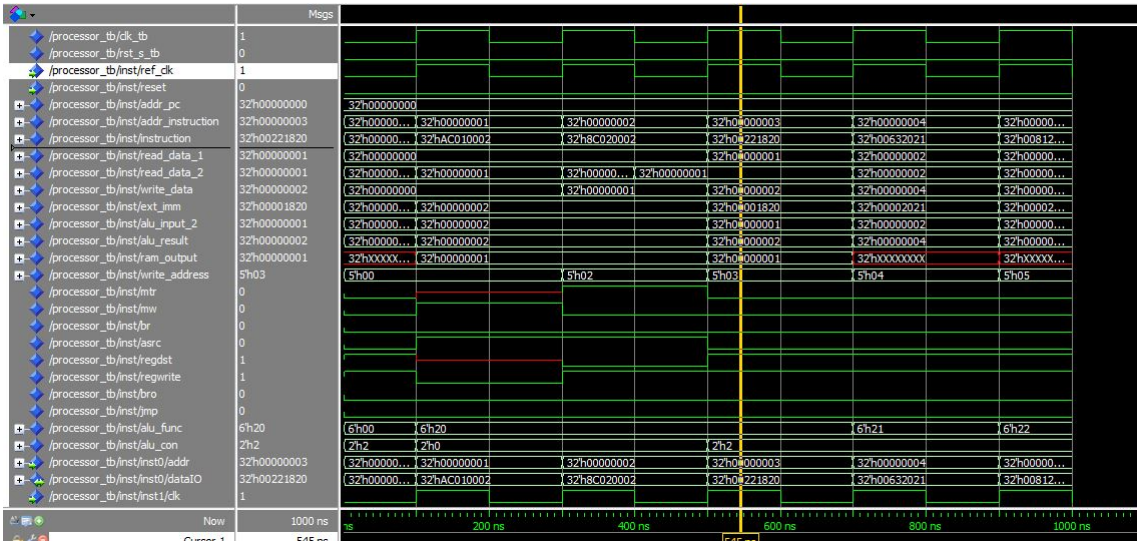
Hexadecimal:
8c020002

Binary:
100011 00000 00010 0000000000000010

MIPS Code:
lw $2, $0, $2

This instruction loads the word from register 0 into register 2, thus making register 2 equal to ’1’.

ADD:



Hexadecimal:
00221820

Binary:
000000 00001 00010 00011 00000 100000

MIPS Code:
add $3, $1, $2

This instruction adds register 1 with register 2 and then stores the value in register 3, thus making register 3 equal to ’2’.

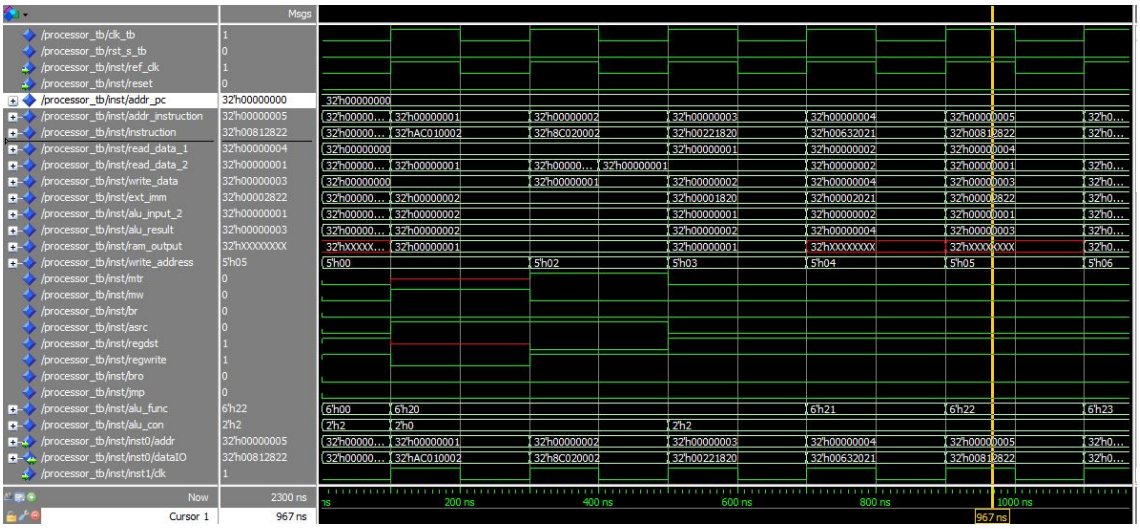
Hexadecimal:
00632021

Binary:
000000 00011 00011 00100 00000 100001

MIPS Code:
add $4, $3, $3

This instruction adds register 3 with register 3 and then stores the value in register 4, thus making register 4 equal to ’4’.

SUB:



Hexadecimal:
00812822

Binary:
000000 00100 00001 00101 00000 100010

MIPS Code:
sub $5, $4, $1

This instruction subtracts the value of register 1 from the value of register 4 then stores the value into register 5, thus making register 5 equal to ’3’.

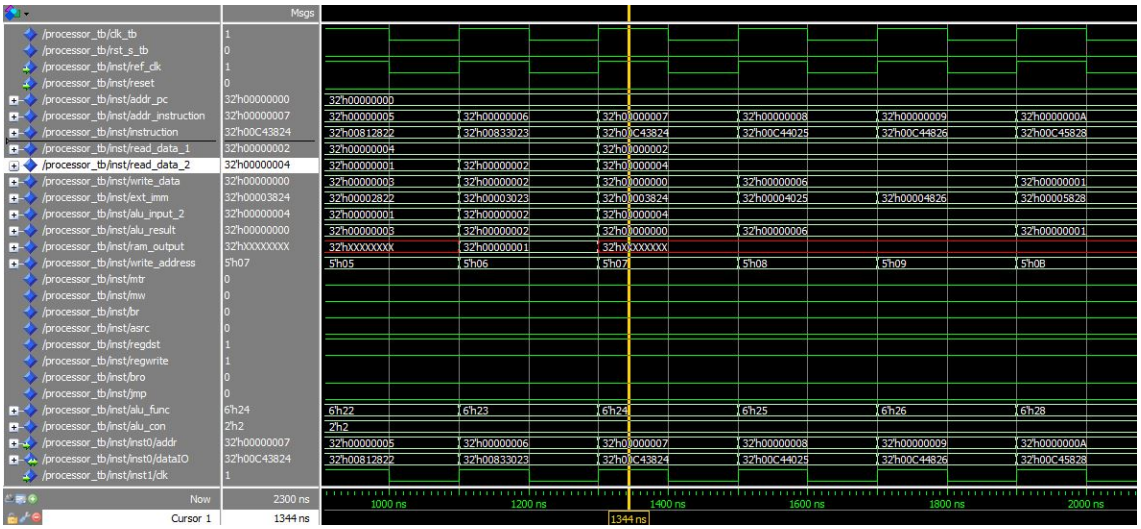
Hexadecimal:
00833023

Binary:
000000 00100 00011 00110 00000 100011

MIPS Code:
sub $6, $4, $3

This instruction subtracts the value of register 3 from the value of register 4 then stores the value into register 6, thus making register 6 equal to ’2’.

AND:



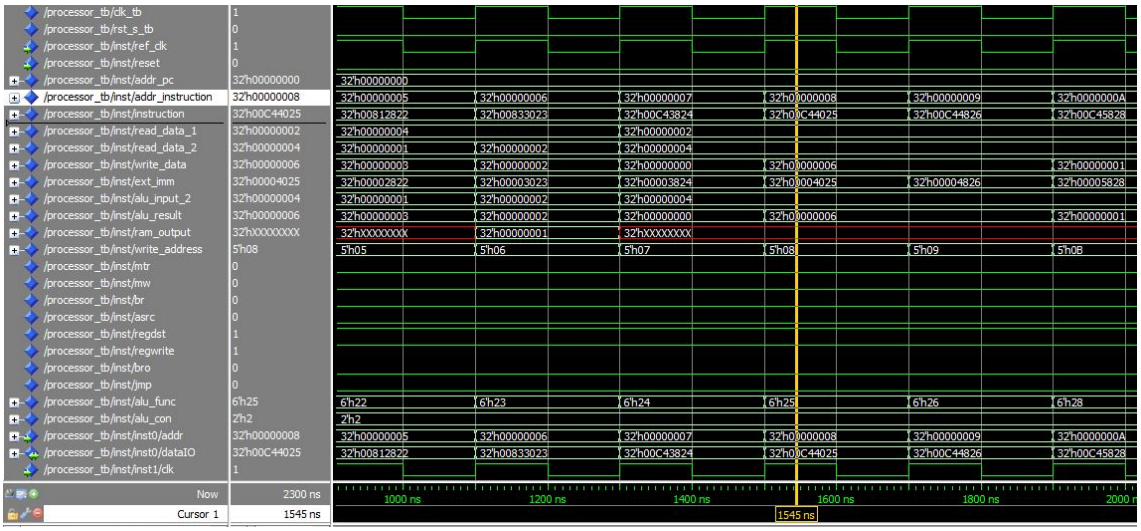
Hexadecimal:
00c43824

Binary:
000000 00110 00100 00111 00000 100100

MIPS Code:
and $7, $6, $4

This instruction takes the bitwise AND of register 6 and register 4 then stores the value into register 7, thus making register 7 equal to ’0’.

OR:



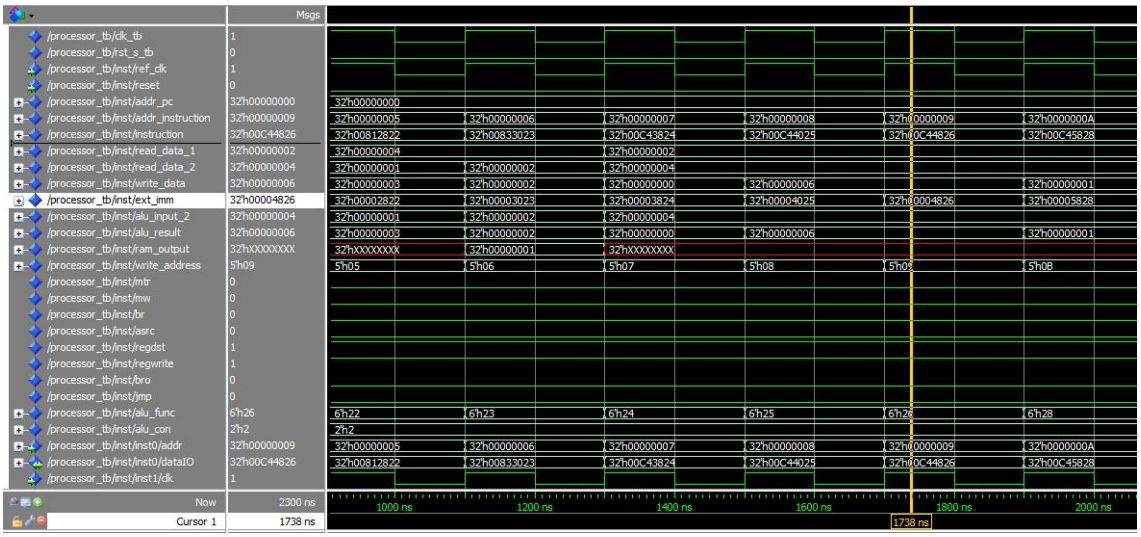
Hexadecimal:
00c44025

Binary:
000000 00110 00100 01000 00000 100101

MIPS Code:
or $8, $6, $4

This instruction takes the bitwise OR of register 6 and register 4 then stores the value into register 8, thus making register 8 equal to ’6’.

XOR:



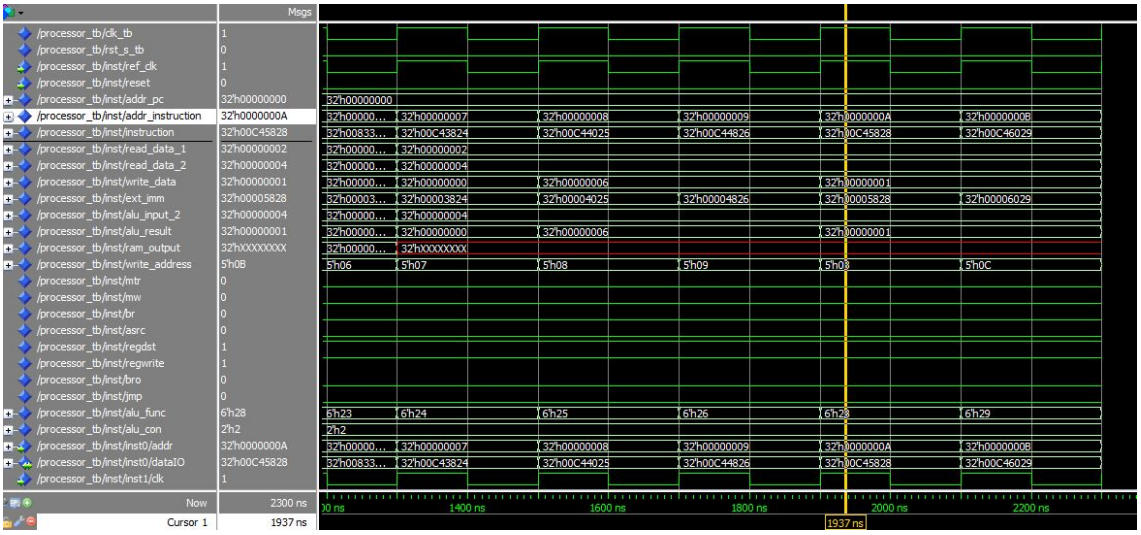
Hexadecimal:
00c44826

Binary:
000000 00110 00100 01001 00000 100110

MIPS Code:
xor $9, $6, $4

This instruction takes the bitwise XOR of register 6 and register 4 then stores the value into register 9, thus making register 9 equal to ’6’.

Set-Less-Than Signed:



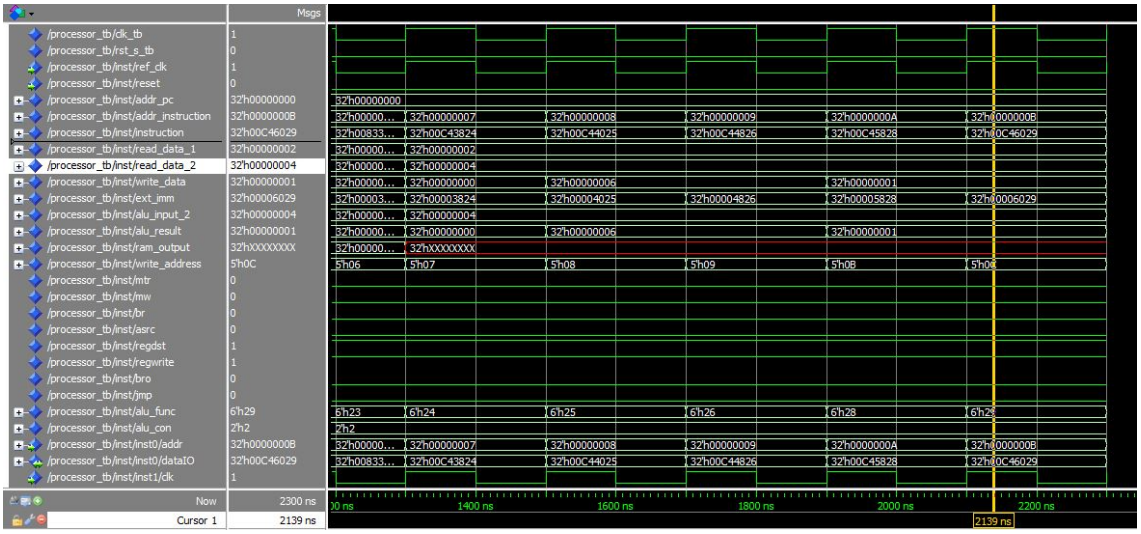
Hexadecimal:
00c45828

Binary:
000000 00110 00100 01011 00000 101000

MIPS Code:
slt $6, $4, $11 (2<4 $11=1)

This instruction checks whether or not the signed value register 6 is less than the signed value register 4 then stores ’0’ (false) or ’1’ (true) into register 11, thus making register 11 equal to ’1’.

Set-Less-Than Unsigned:



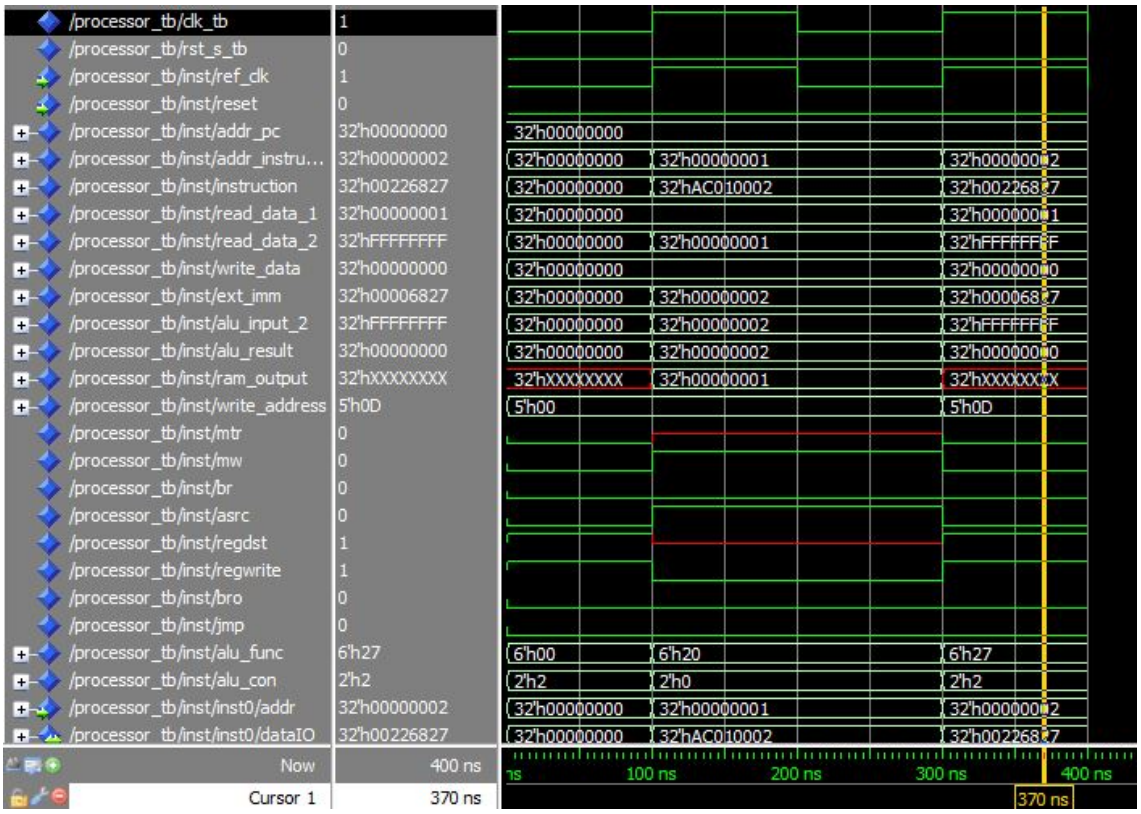
Hexadecimal:
00c46029

Binary: 000000 00110 00100 01100 00000 101001

MIPS Code:
slt(unsigned) $12, $6, $4

This instruction checks whether or not the unsigned value of register 6 is less than unsigned value of register 4 then stores ’0’ (false) or ’1’ (true) into register 12, thus making register 12 equal to ’1’.

NOR:



For the NOR instruction we set register 1 to ’1’ and register 2 to ’11111111’.

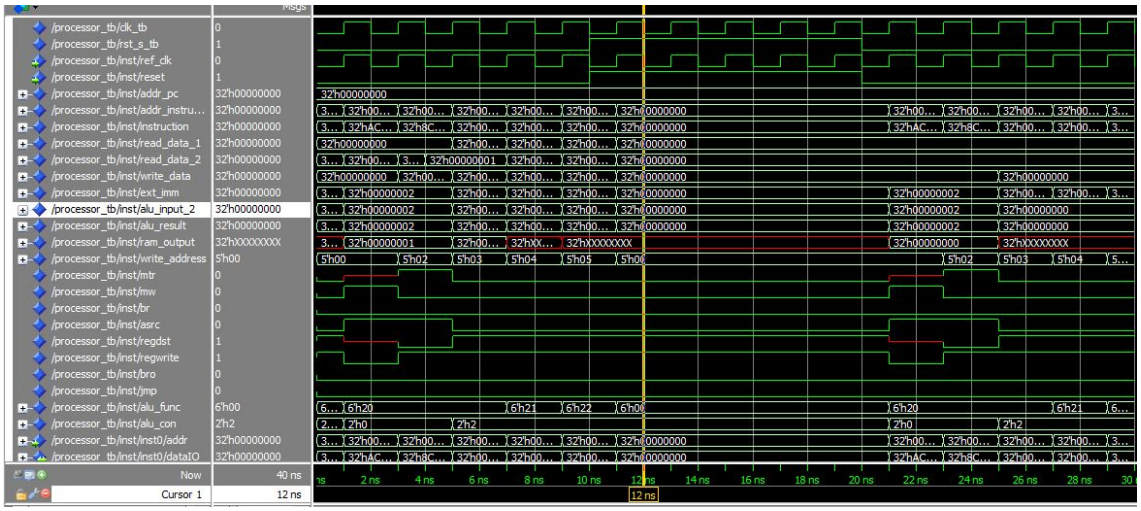
Hexadecimal:
00226827

Binary:
000000 00001 00010 01101 00000 100111

MIPS Code:
nor $13, $1, $2

This instruction takes the bitwise NOR of register 1 and register 2 then stores the value into register 13, thus making register 13 equal to ’0’.

RESET:



The reset after 10 ns

4 Conclusion

In the end, our program compiled fully with some warnings here and there. We also ended up having a run time of less than 1 ns per instruction.