CprE 381 – Computer Organization and
Assembly-Level Programming

Project B: Single-Cycle CPU

Lab Partners:
Ethan McGill
Ehren Fox

Section / Lab Time:
Section 6, Thursday 4:10-6:00PM

Table of Contents

Prelab Questions:

- ***(0A) How are instruction and data memory initialized in the simulation? How does MARS interface with ModelSim?***

  Although we went over this in class at the beginning of the project, we are unsure how to answer this upon completion of the report.

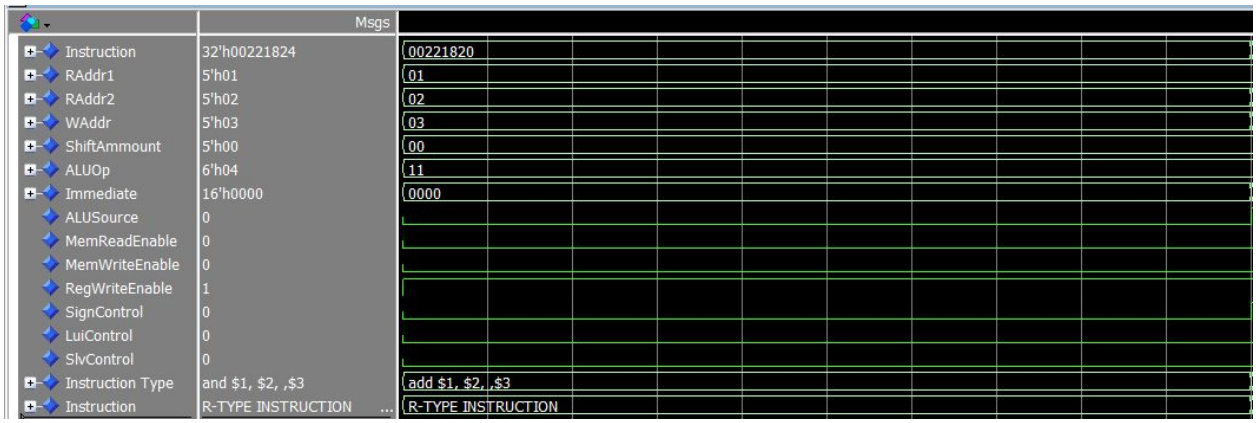
- ***(0B) In the MIPS skeleton VHDL, how is a halting / termination condition detected? What MIPS assembly instruction does this correspond to?***

  We are checking if register 2 contains the value 10 while making a system call. This corresponds to an add immediate 10 to register 2, followed by a system call.

Phase I: Data-Handling Instructions:

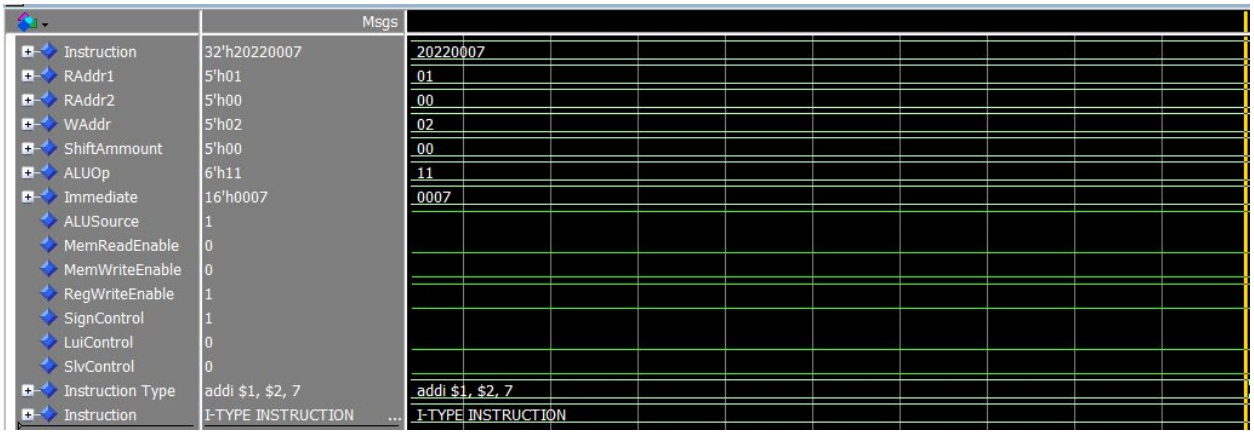
(1B) - Control Unit Waveforms and Discussion:

Add



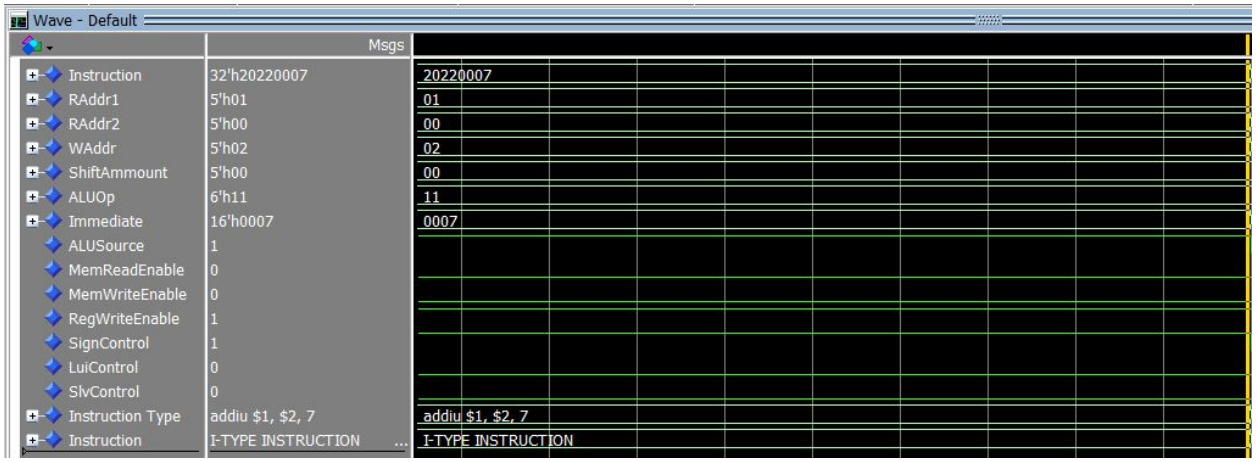
If we look in our control spreadsheet we can see that the add instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be an add, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Addi



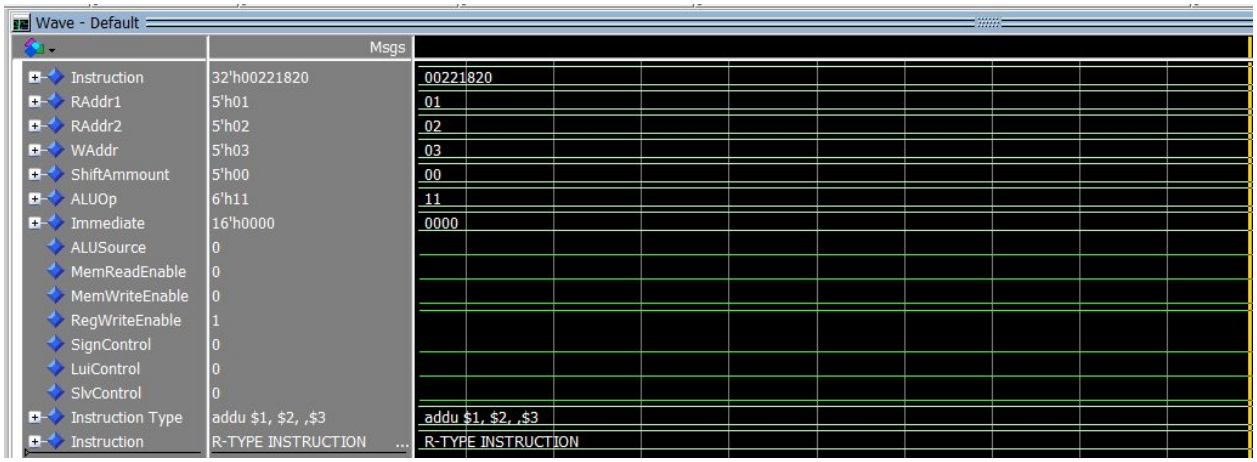
If we look in our control spreadsheet we can see that the addi instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an add, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Addiu



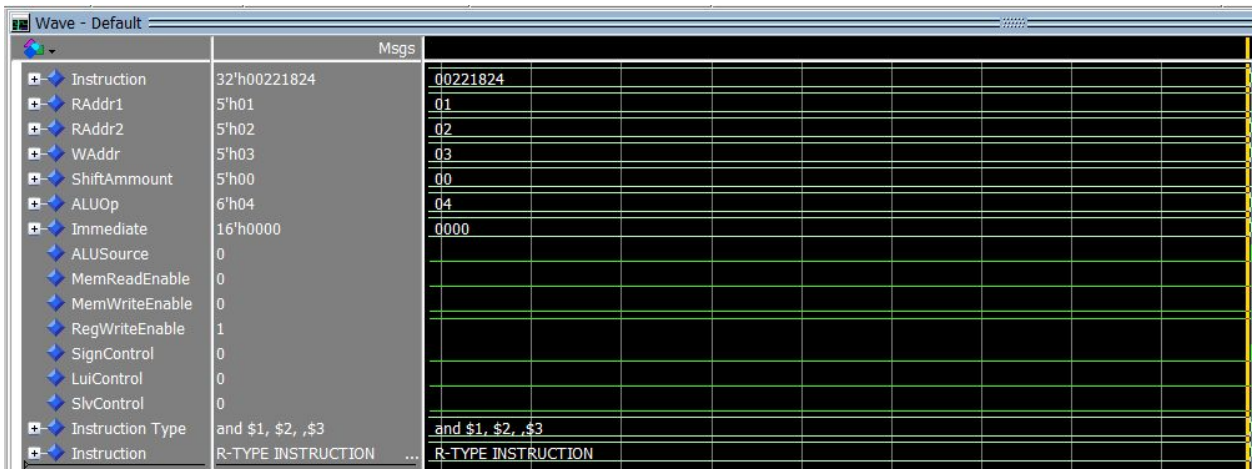
If we look in our control spreadsheet we can see that the addiu instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an add, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Addu



If we look in our control spreadsheet we can see that the addu instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be an add, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

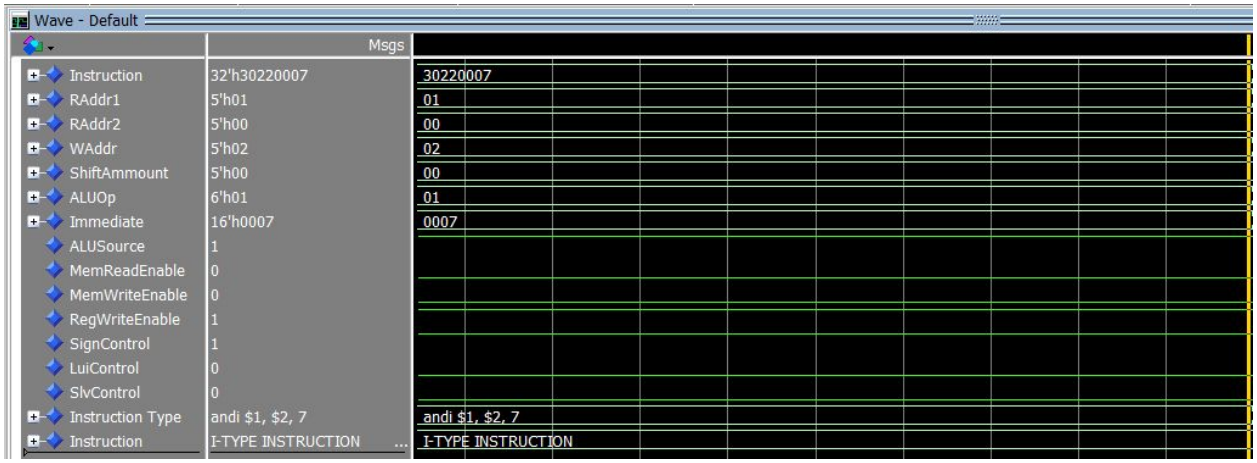
And



If we look in our control spreadsheet we can see that the and instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be an and, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the

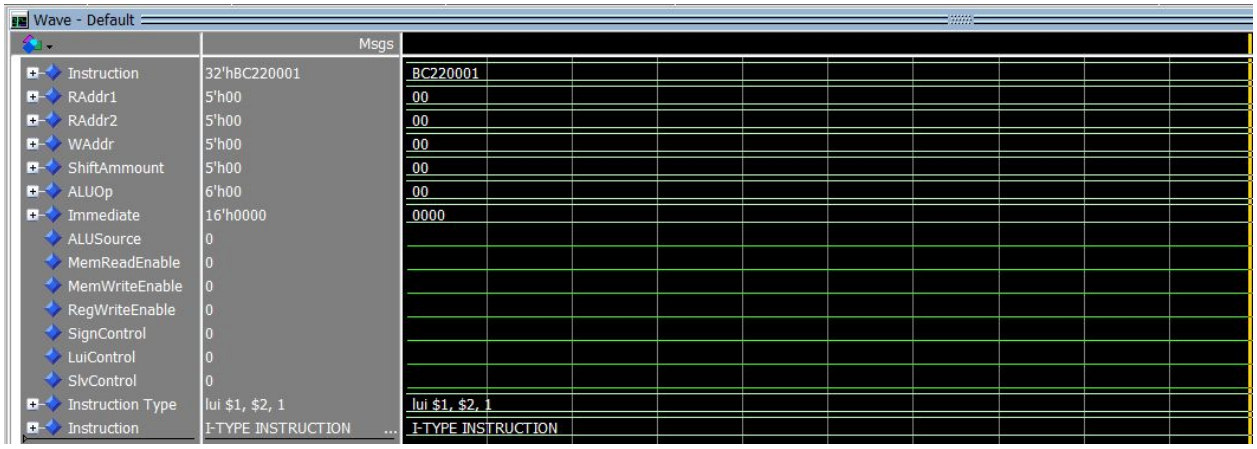
screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Andi



If we look in our control spreadsheet we can see that the andi instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an and, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

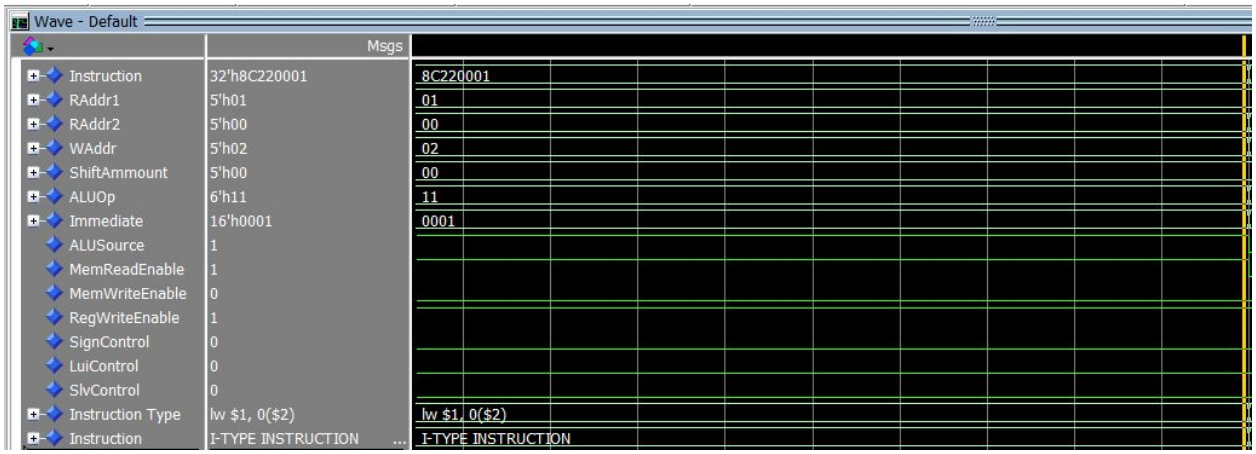
Lui



If we look in our control spreadsheet we can see that the lui instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an add, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 1, and SlvCont should be 0. As seen in the

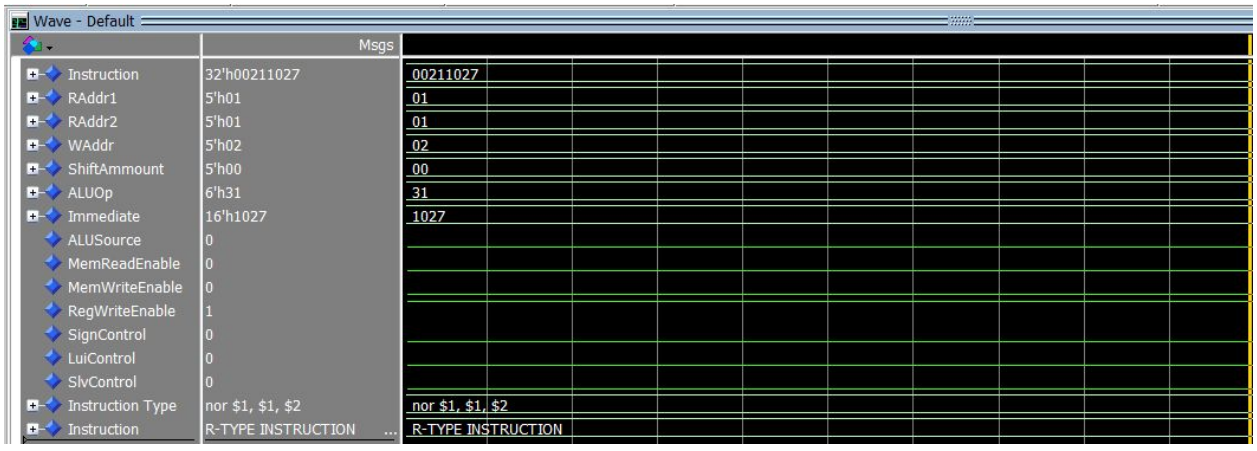
screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Lw



If we look in our control spreadsheet we can see that the lw instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 0, ALUOp should perform an add, MemToReg should be 1, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 1, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

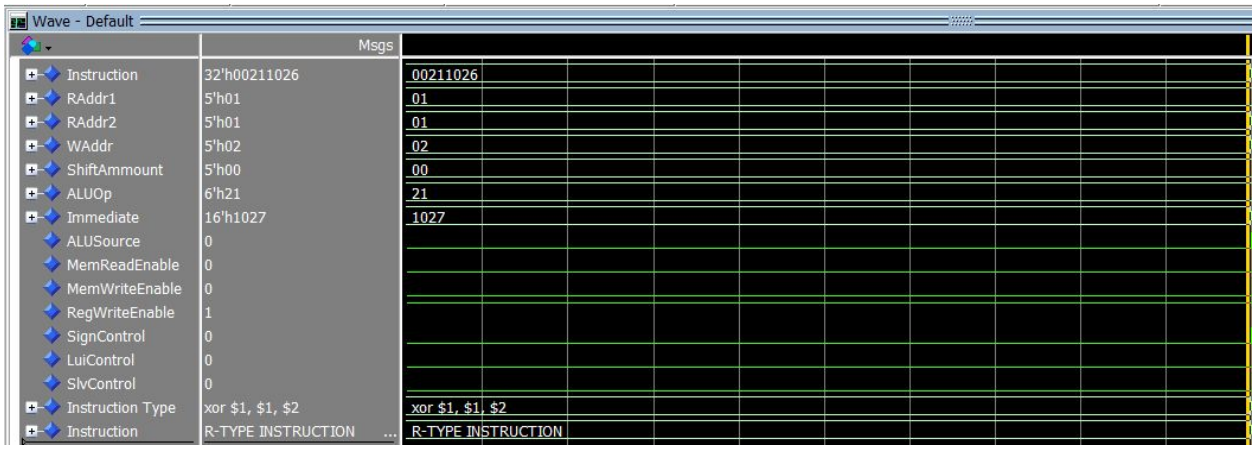
Nor



If we look in our control spreadsheet we can see that the nor instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a nor, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the

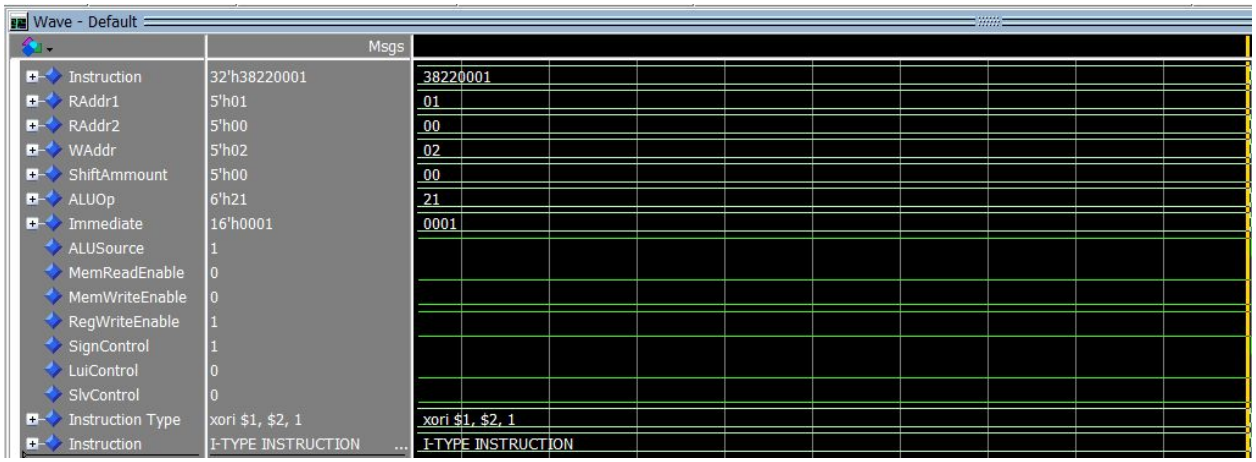
screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Xor



If we look in our control spreadsheet we can see that the xor instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a xor, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

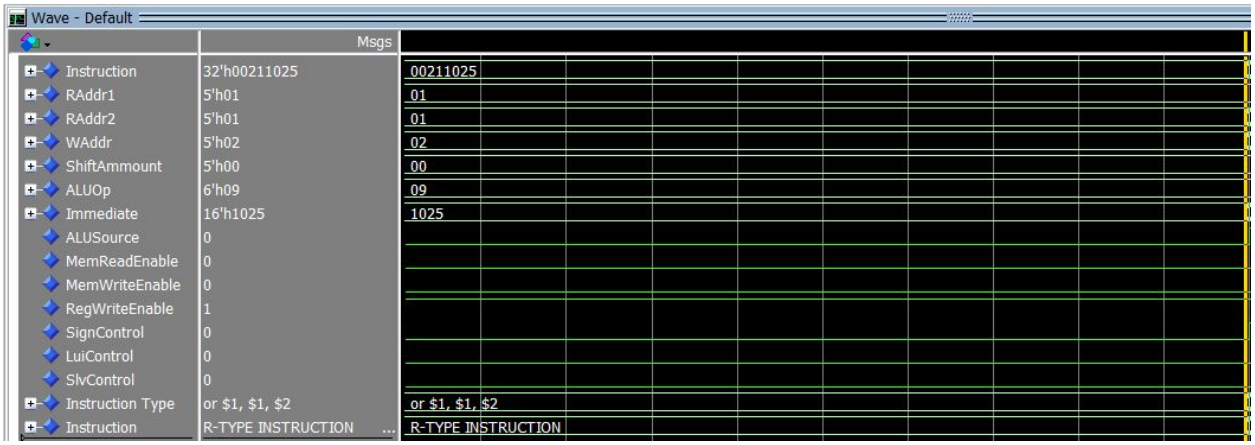
Xori



If we look in our control spreadsheet we can see that the xori instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an xor, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the

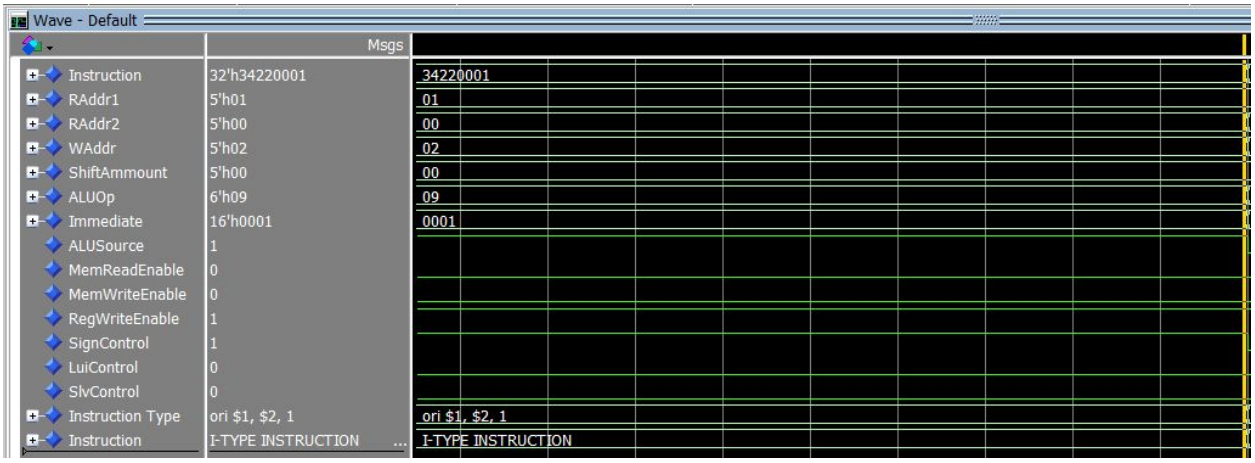
screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Or



If we look in our control spreadsheet we can see that the or instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a or, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

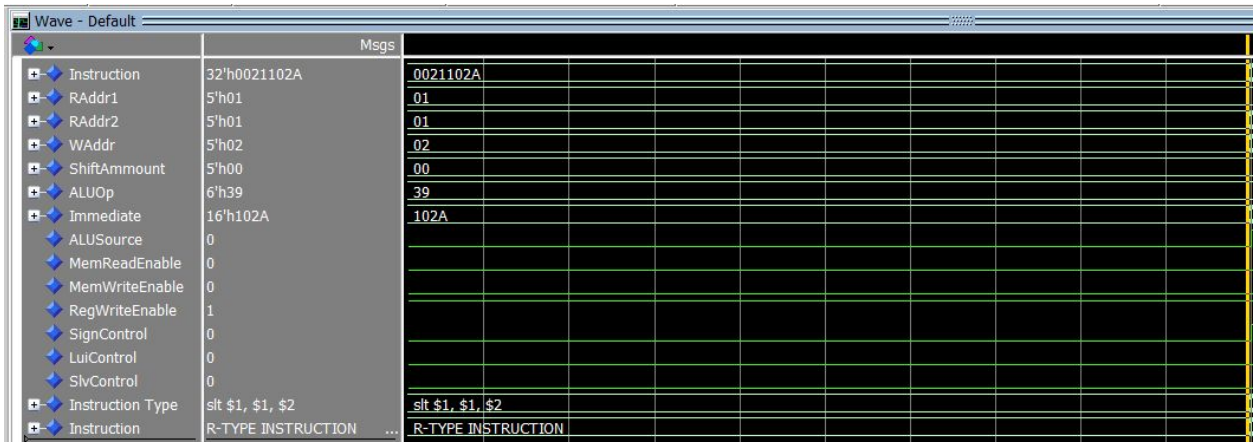
Ori



If we look in our control spreadsheet we can see that the ori instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an or, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the

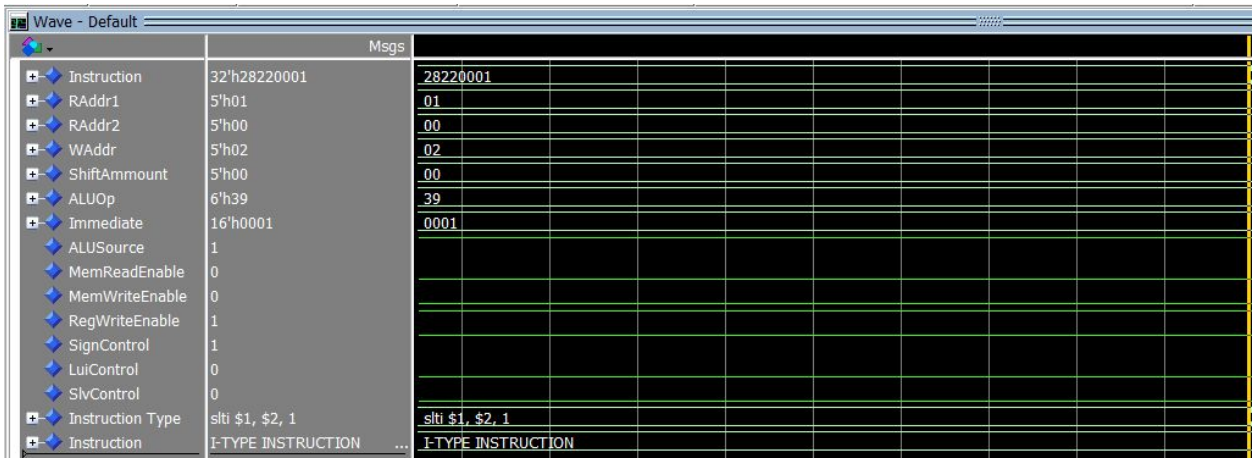
screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Slt



If we look in our control spreadsheet we can see that the slt instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a slt, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

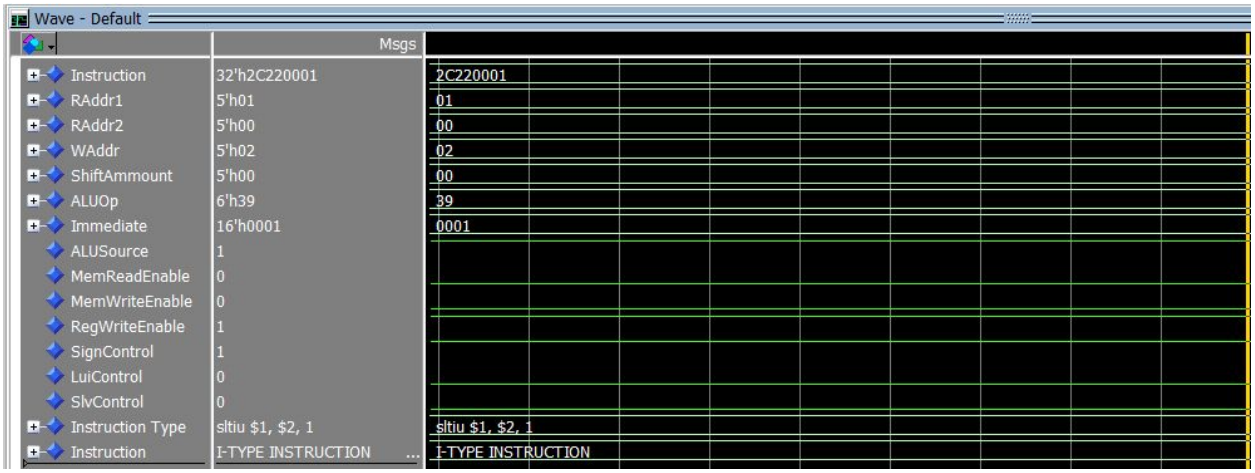
Slti



If we look in our control spreadsheet we can see that the slti instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an slt, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the

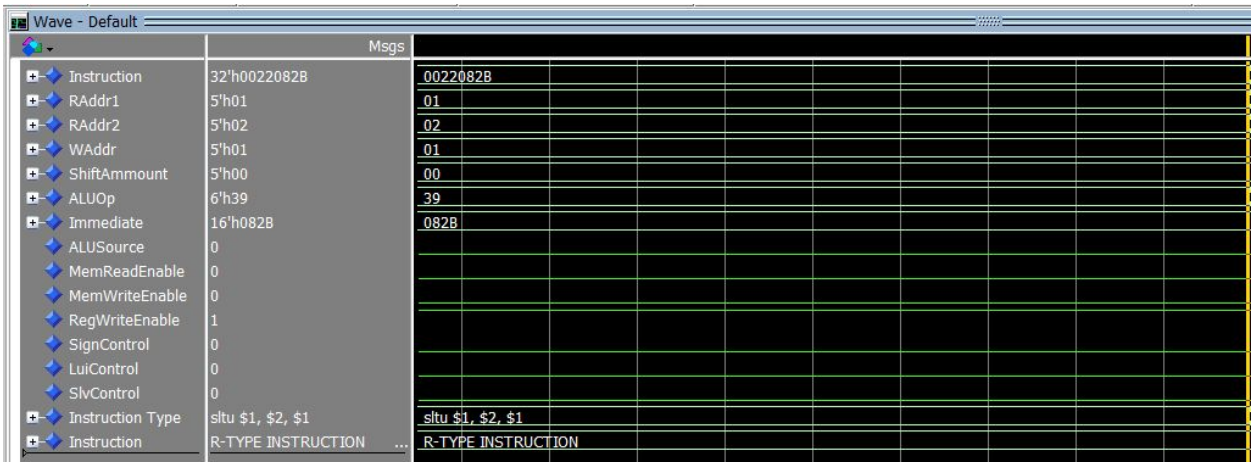
screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

Sltiu



If we look in our control spreadsheet we can see that the sltiu instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an slt, MemToReg should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

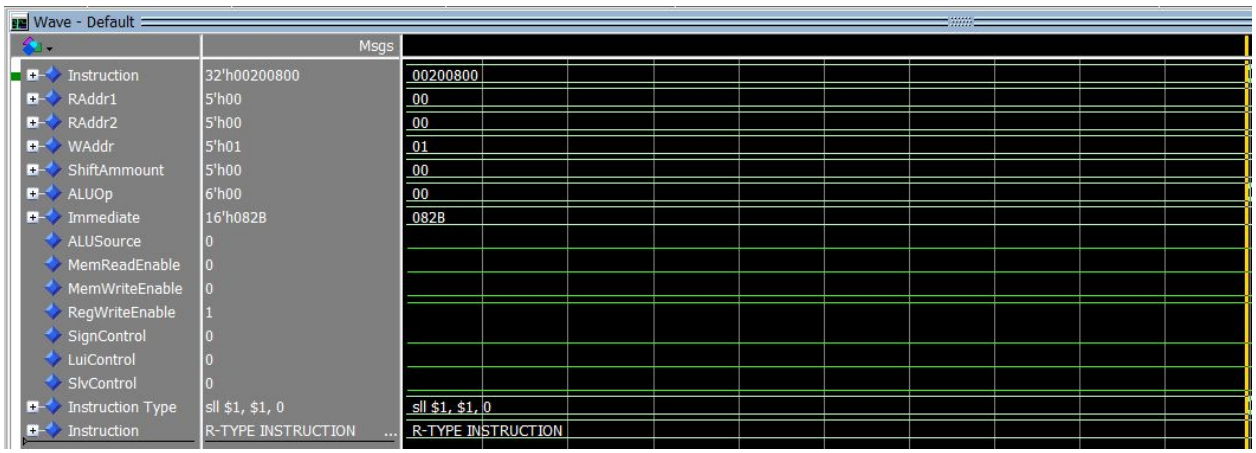
Sltu



If we look in our control spreadsheet we can see that the sltu instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a slt, MemRead should be 0, MemWrite should be 0, RegWrite should be

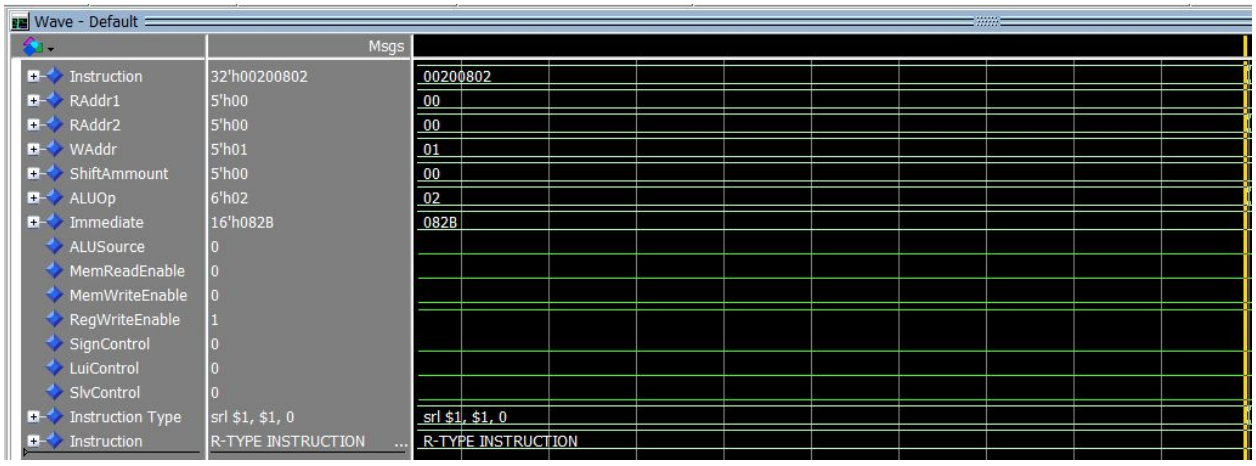
1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Sll



If we look in our control spreadsheet we can see that the sll instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a sll, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

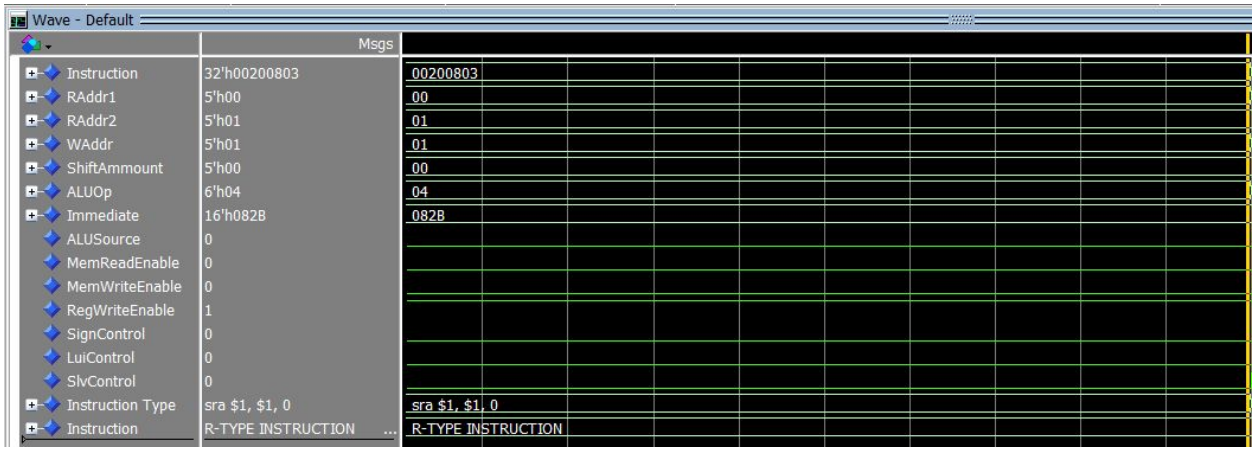
Srl



If we look in our control spreadsheet we can see that the srl instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a srl, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot

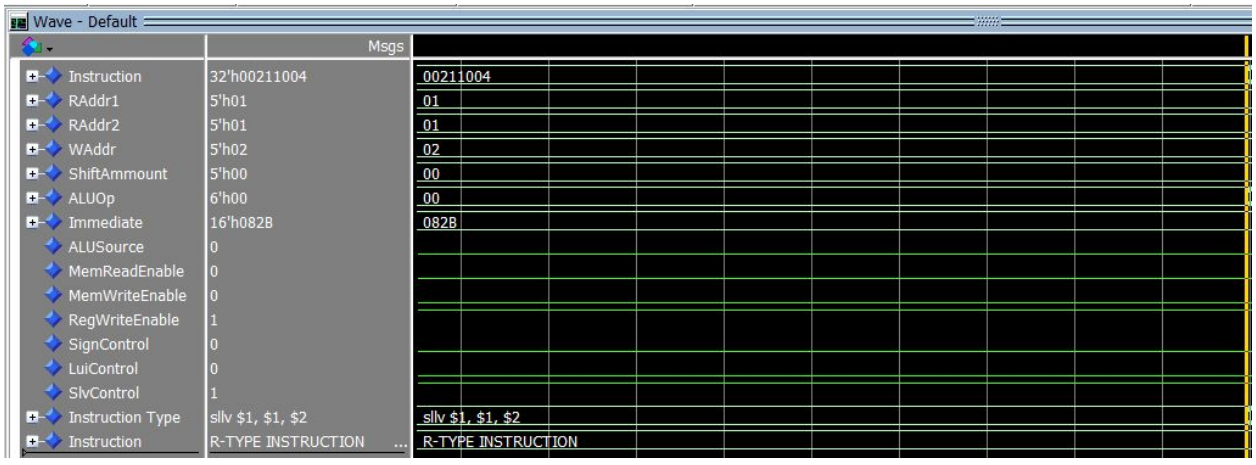
all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Sra



If we look in our control spreadsheet we can see that the sra instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a sra, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

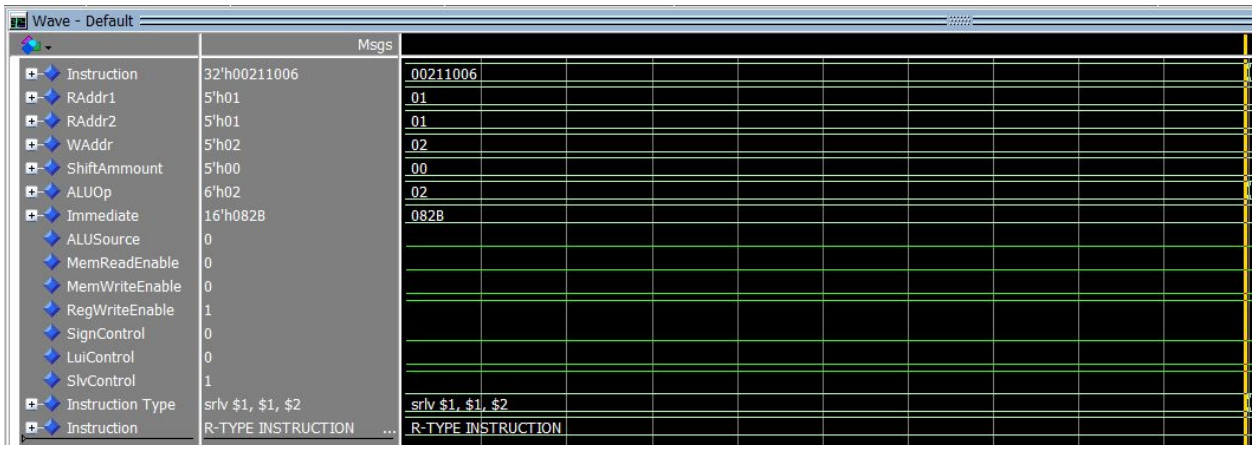
Sllv



If we look in our control spreadsheet we can see that the sllv instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a sllv, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 1. As seen in the

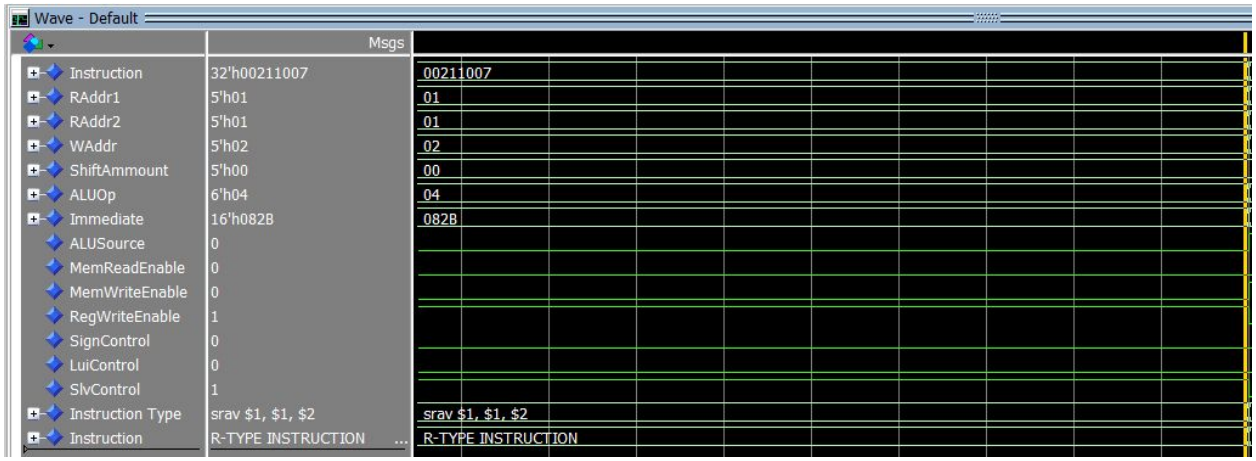
screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Srlv



If we look in our control spreadsheet we can see that the srlv instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a srlv, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 1. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

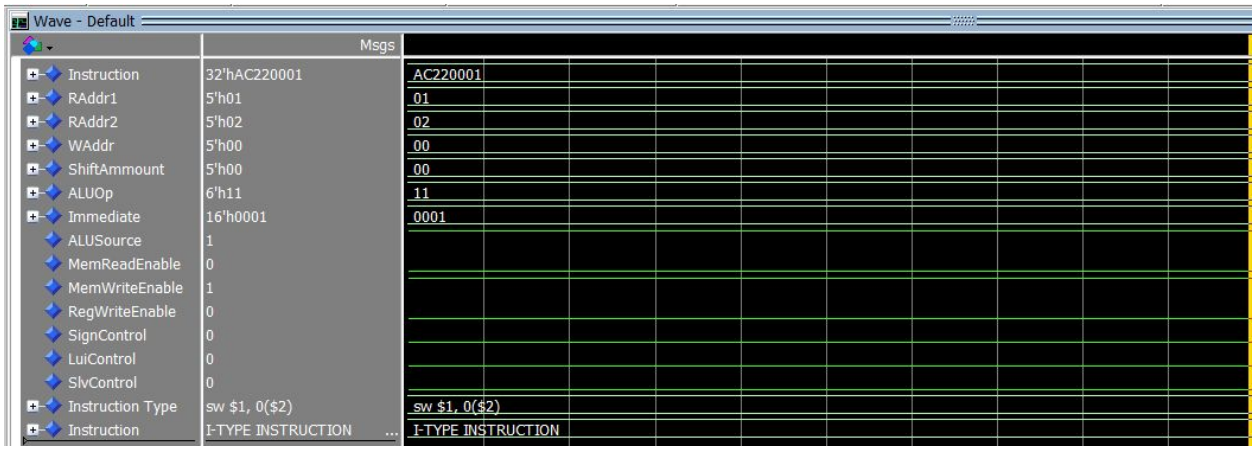
Srav



If we look in our control spreadsheet we can see that the srav instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0,

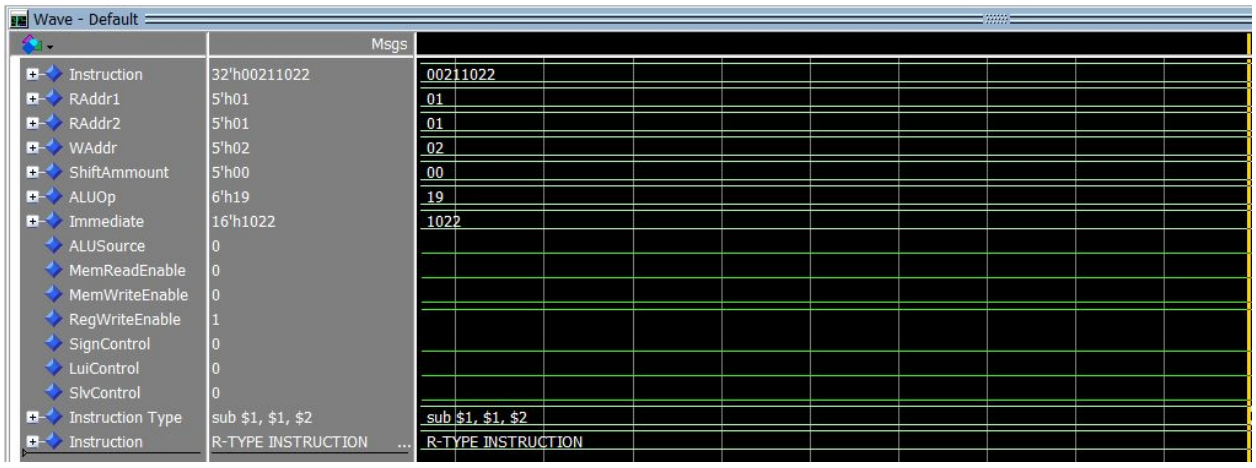
ALUControl should be a srlv, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 1. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Sw



If we look in our control spreadsheet we can see that the sw instruction is an I-Type, so we write to the rt field, and according to our signals ALUSrc should be 1, SignControl should be 1, ALUOp should perform an add, MemToReg should be 0, MemWrite should be 1, RegWrite should be 0, RegDest should be 0, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these control bits are correct, and we deal with the write address inside of our control unit so there is no output signal for RegDst.

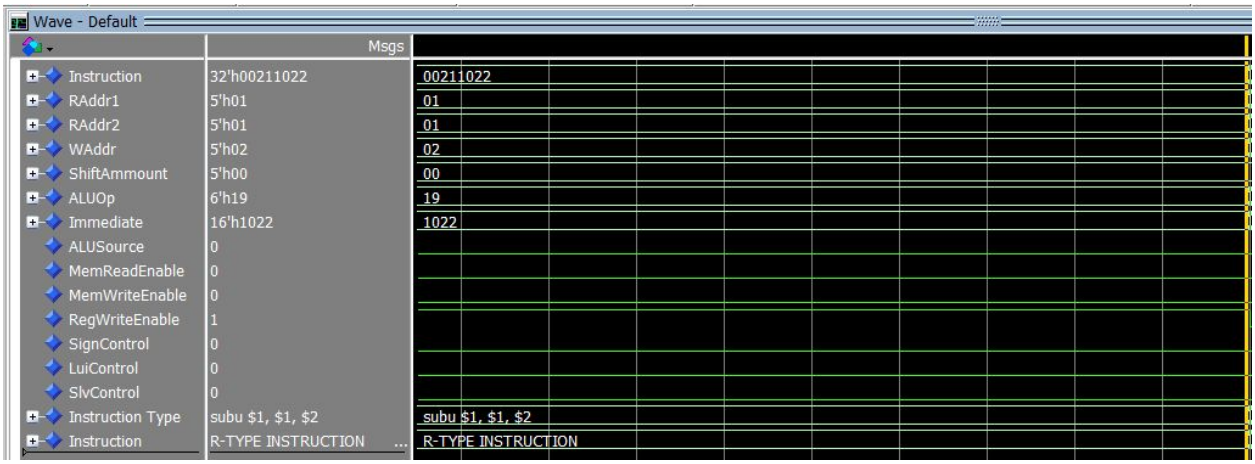
Sub



If we look in our control spreadsheet we can see that the sub instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a sub, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the

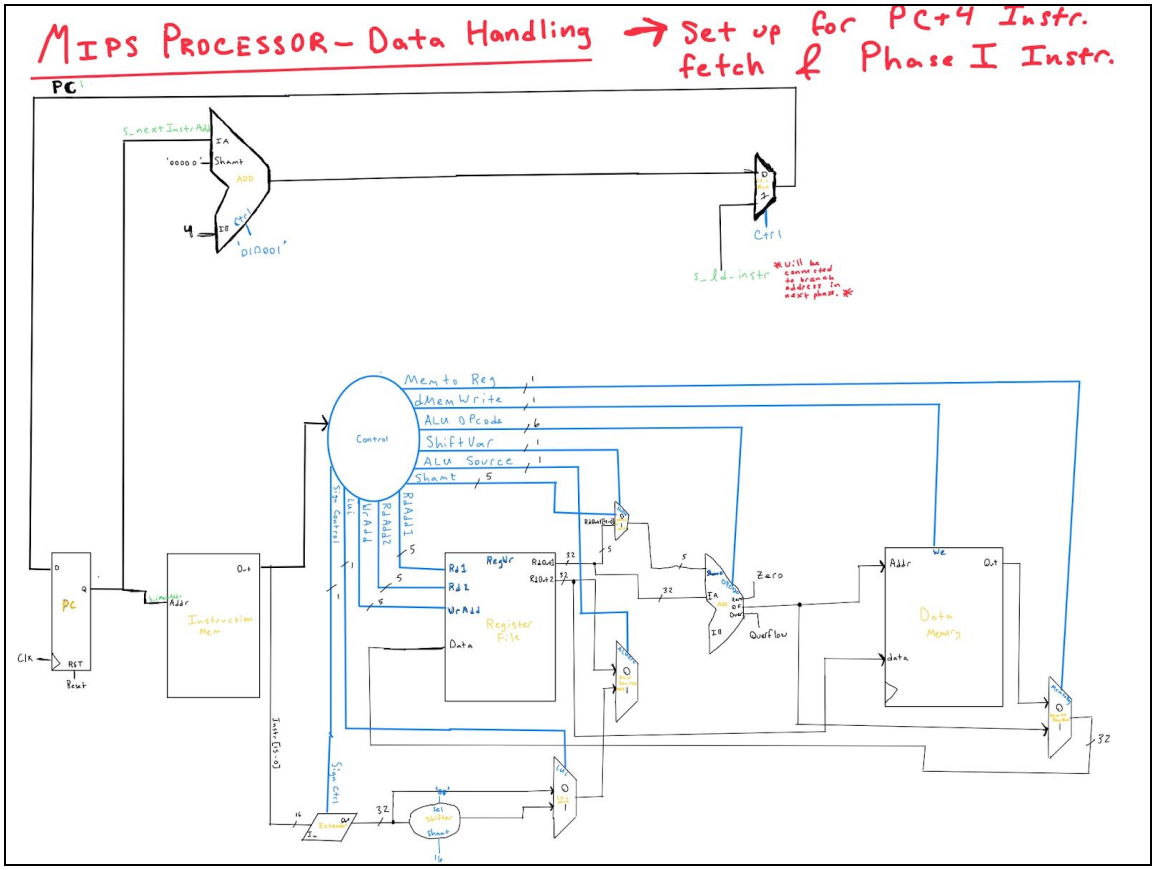
screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

Subu



If we look in our control spreadsheet we can see that the subu instruction is an R-Type, so we write to the rd field, and according to our signals ALUSrc should be 0, SignControl should be 0, ALUControl should be a sub, MemRead should be 0, MemWrite should be 0, RegWrite should be 1, RegDest should be 1, LuiCont should be 0, and SlvCont should be 0. As seen in the screenshot all of these are correct, and we deal with the write address inside of the control unit so there is no RegDest output signal.

(2) Data-Handling Schematic

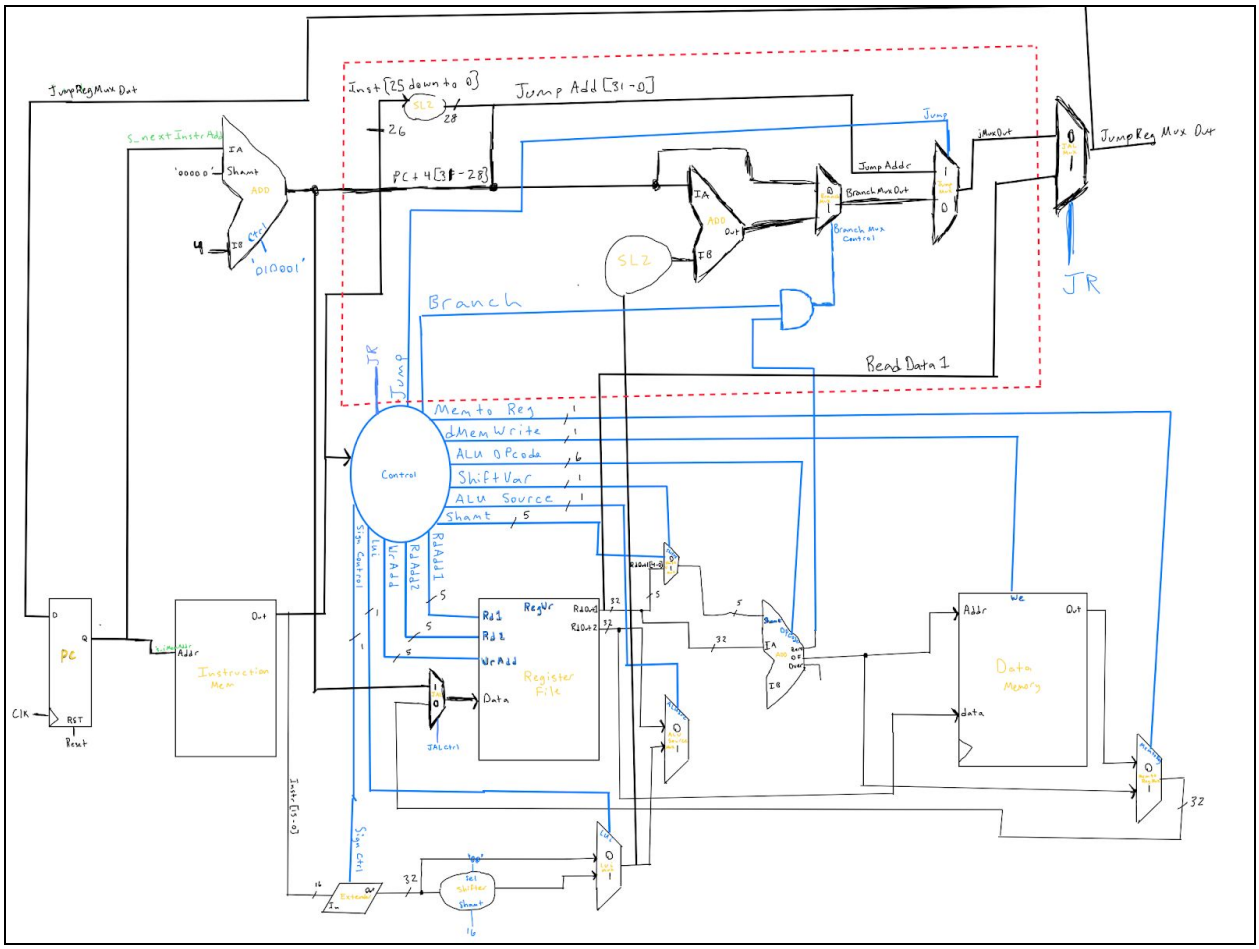


Phase II: Control Flow Instructions and High Level Testing:

- ***Describe the control flow possibilities that the instruction fetch logic must support as a function of the different control flow-related instructions:***

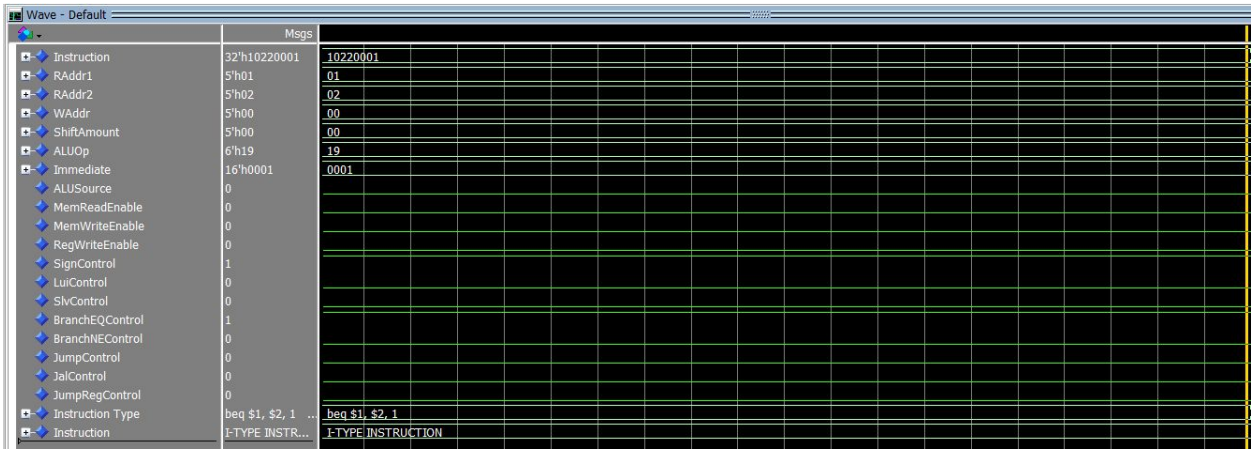
We are adding 5 new control-flow instructions including beq, bne, j, jal, and jr. The added functionality of beq and bne are the same logically, you check if something is equal, or not equal and branch to the given address. To implement this we need to use the zero from our alu, as well as the beq and bne control signals. If the statement is equal, or not equal to, then we simply add the offset from the instruction to the original pc+4 value and that becomes our next instruction address. For jump we want to take the jump address given in the instruction and shift it left 2 bits, then append the three most significant bits from the pc+4. This is fed into a mux that is controlled by the jump control signal that picks between the normal/branch address, and the jump address. The chosen address gets put into the pc as the next instruction address. Jump and link is the same as jump, except while the jump address is being calculated, we are also storing pc+4 into the register 31 in the register file. Finally jump register grabs the value straight from register 31, and feeds that into a mux that chooses between the calculated pc and the value from the register file, the address is chosen by the jump register control bit.

(4C) - MIPS Single-Cycle Processor Schematic (Control Flow + Data-Handling):



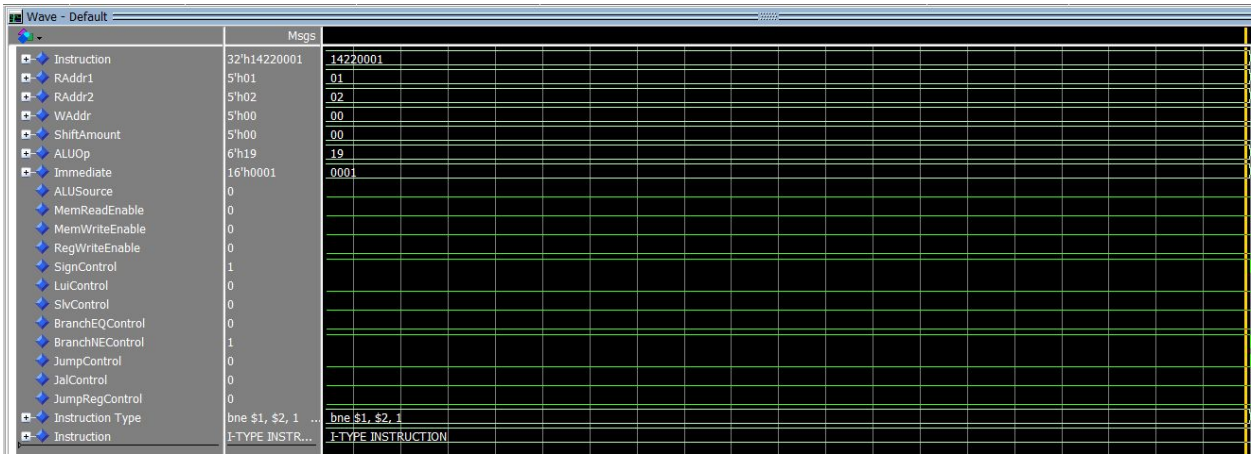
(4D) - MIPS Single-Cycle Processor Waveforms and Discussion:

Beq



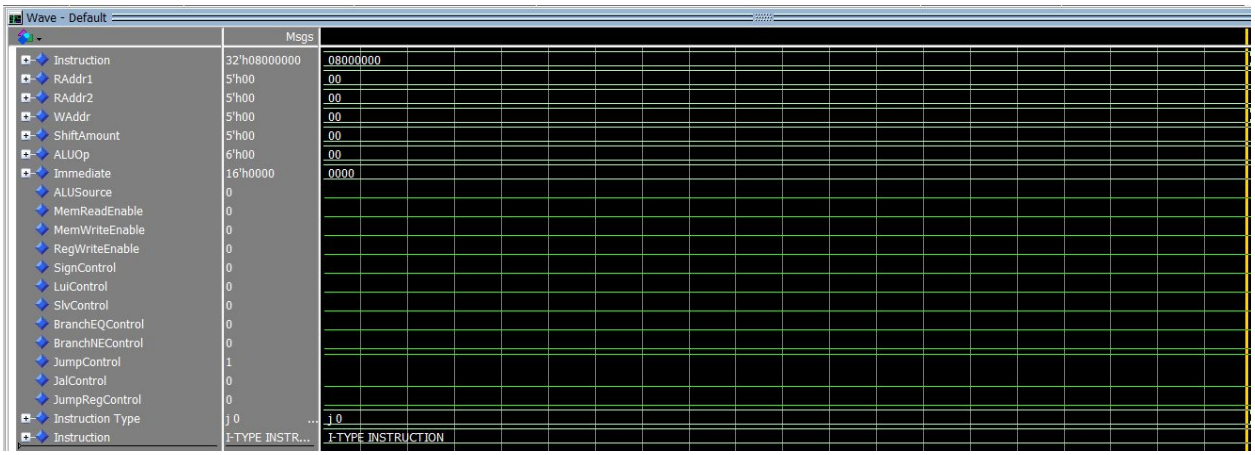
If we look at our control signal spreadsheet we can see that for beq, we expect the ALUSrc to be 0, SignControl to be 1, ALUOp to be a sub, MemToReg to be 0, MemWrite to be 0, RegWrite to be 0, RegDest to be 0, LuiCont to be 0, SlvCont to be 0, Jump to be 0, BranchEQ to be 1, BranchNE to be 0, jalCont to be 0, and jrCont to be 0. As you can see these all match up, we are dealing with the write and read addresses in the control unit so there is no output signal for RegDest.

Bne



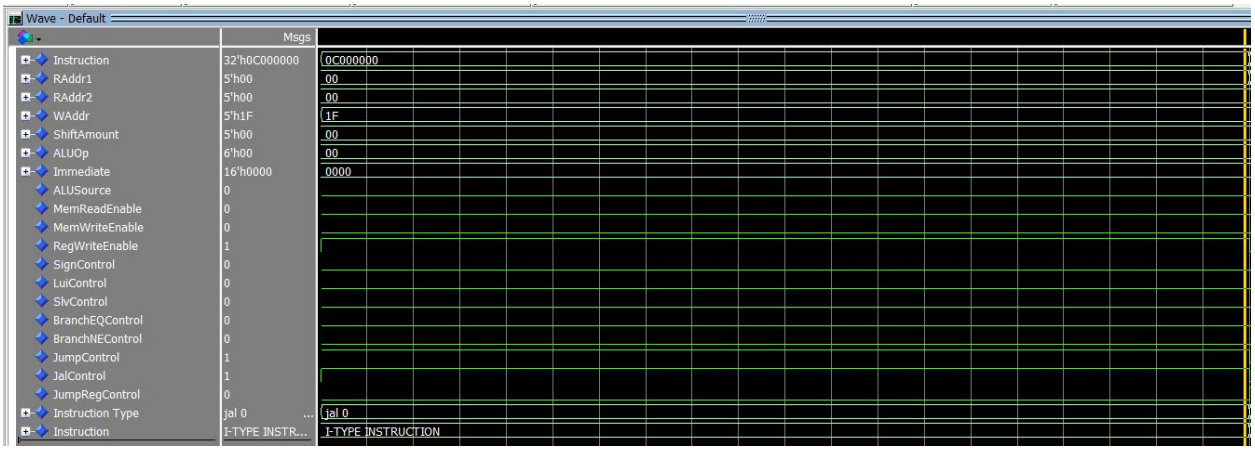
If we look at our control signal spreadsheet we can see that for beq, we expect the ALUSrc to be 0, SignControl to be 1, ALUOp to be a sub, MemToReg to be 0, MemWrite to be 0, RegWrite to be 0, RegDest to be 0, LuiCont to be 0, SlvCont to be 0, Jump to be 0, BranchEQ to be 0, BranchNE to be 1, jalCont to be 0, and jrCont to be 0. As you can see these all match up, we are dealing with the write and read addresses in the control unit so there is no output signal for RegDest.

J



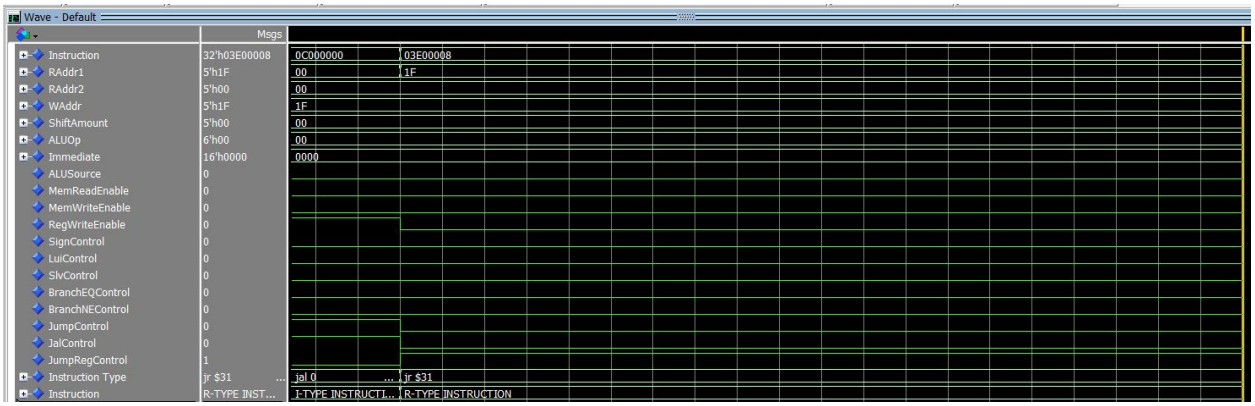
If we look at our control signal spreadsheet we can see that for j, we expect the ALUSrc to be 0, SignControl to be 0, ALUOp doesn't matter, MemToReg to be 0, MemWrite to be 0, RegWrite to be 0, RegDest to be 0, LuiCont to be 0, SlvCont to be 0, Jump to be 1, BranchEQ to be 0, BranchNE to be 0, jalCont to be 0, and jrCont to be 0. As you can see these all match up, we are dealing with the write and read addresses in the control unit so there is no output signal for RegDest.

Jal



If we look at our control signal spreadsheet we can see that for jal, we expect the ALUSrc to be 0, SignControl to be 0, ALUOp doesnt matter, MemToReg to be 0, MemWrite to be 0, RegWrite to be 1, RegDest to be 0, LuiCont to be 0, SlvCont to be 0, Jump to be 1, BranchEQ to be 0, BranchNE to be 0, jalCont to be 1, and jrCont to be 0. As you can see these all match up, we are dealing with the write and read addresses in the control unit so there is no output signal for RegDest.

Jr

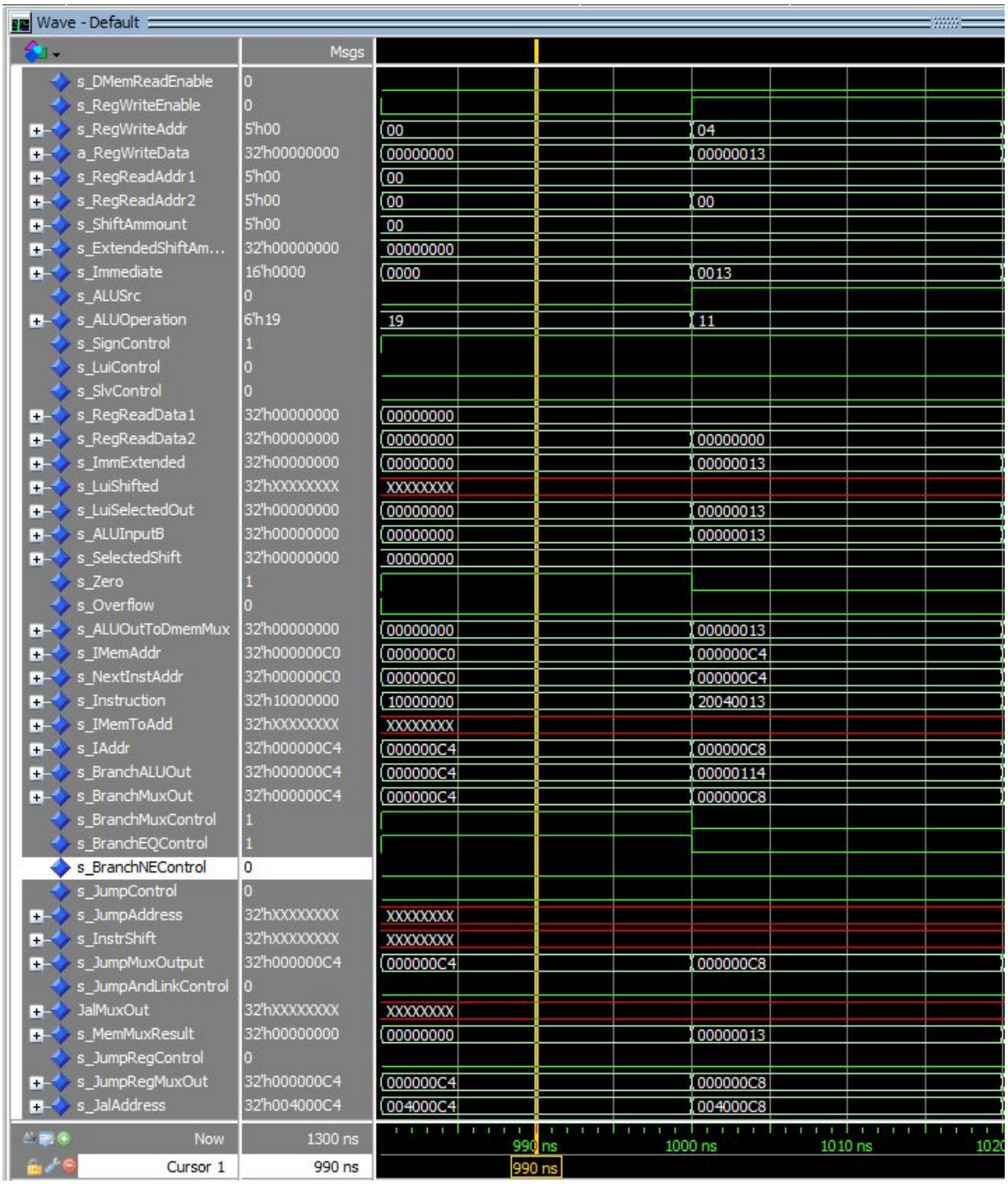


If we look at our control signal spreadsheet we can see that for jal, we expect the ALUSrc to be 0, SignControl to be 0, ALUOp doesn't matter, MemToReg to be 0, MemWrite to be 0, RegWrite to be 0, RegDest to be 0, LuiCont to be 0, SlvCont to be 0, Jump to be 0, BranchEQ to be 0, BranchNE to be 0, jalCont to be 0, and jrCont to be 1. As you can see these all match up, we are dealing with the write and read addresses in the control unit so there is no output signal for RegDest.

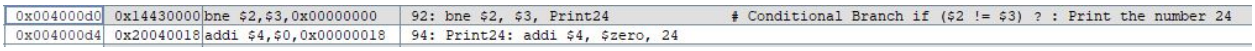
Phase III: Fully Debugging MIPS Single-Cycle Processor

(5A) - Project B Test2 Waveforms and Discussion:

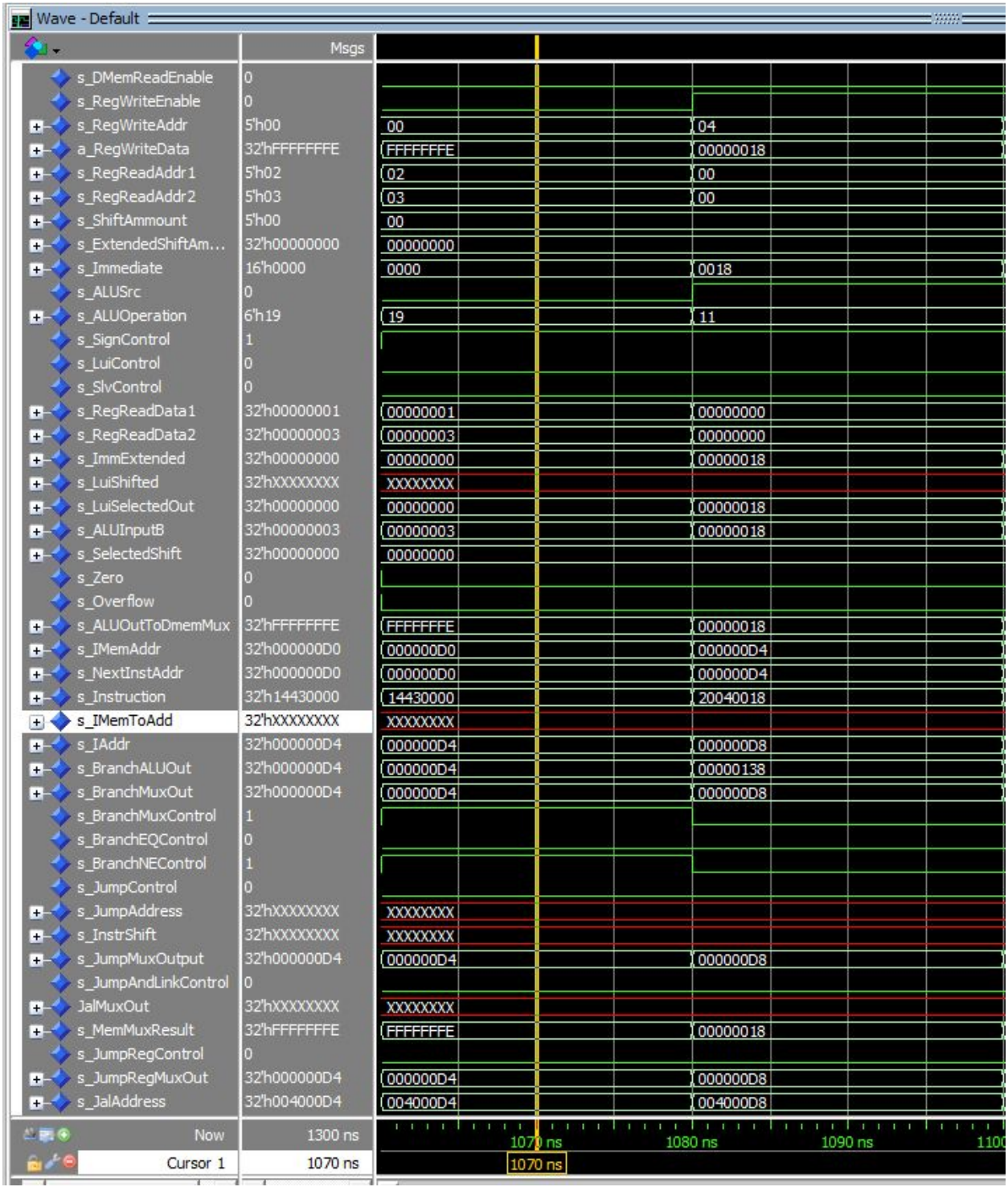
```
0x004000c0  0x10000000  beq $0,$0,0x00000000    86: beq $0, $0, Print19          # Unconditional Branch on Equal Print the number 19
0x004000c4  0x20040013  addi $4,$0,0x00000013   88: Print19: addi $4, $zero, 19
```



As you can see from the Mars screenshot of our test2 code that the address that we are branching to is C4 and the BranchMux out as well as the JumpMuxOut is C4, so that means that the next instruction address will be C4. And as you can see in the next wave, the correct instruction is executed.



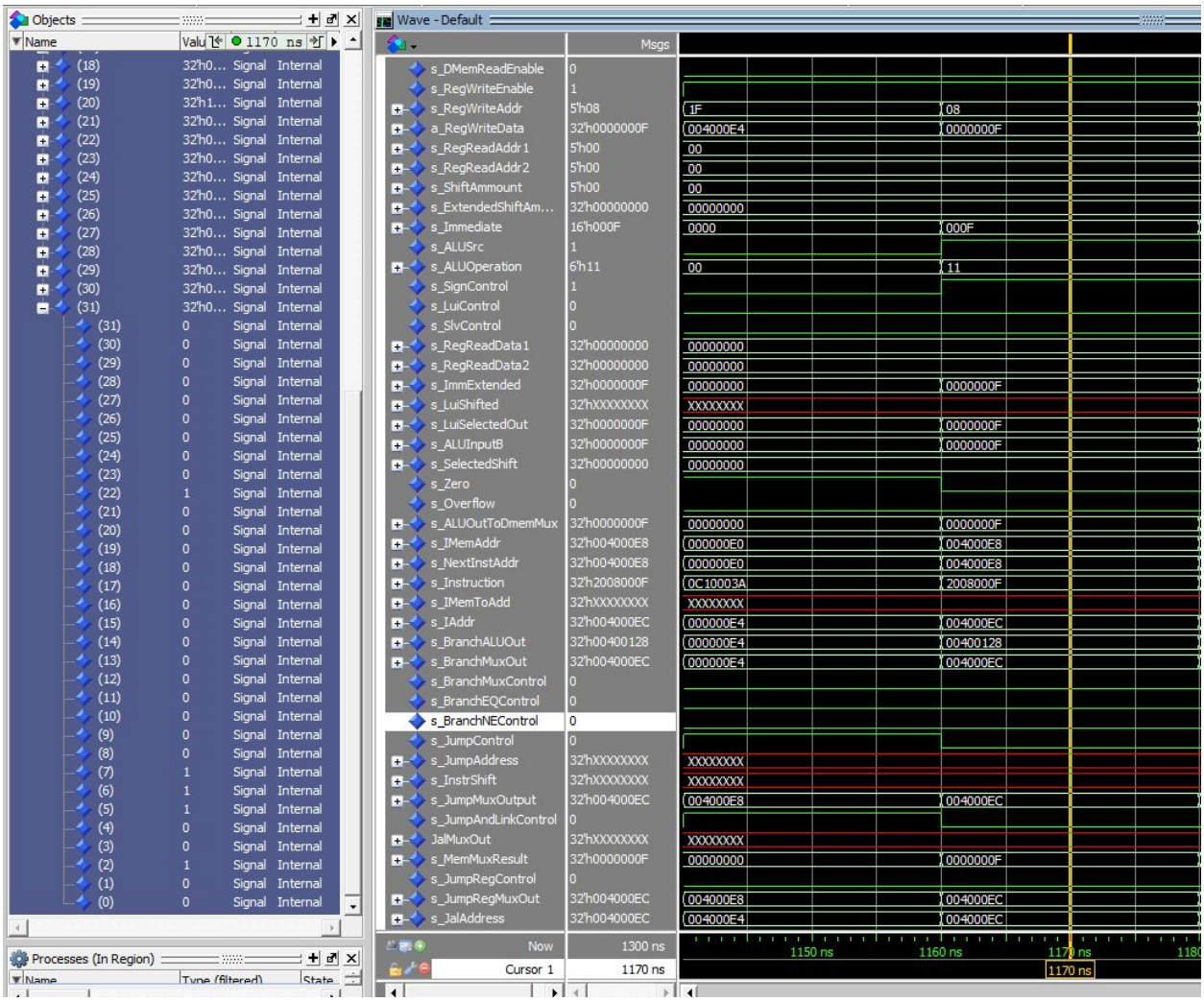
0x004000d0	0x14430000	bne $2,$3,0x00000000	92: bne $2, $3, Print24	# Conditional Branch if ($2 != $3) ? : Print the number 24
0x004000d4	0x20040018	addi $4,$0,0x00000018	94: Print24: addi $4, $zero, 24	



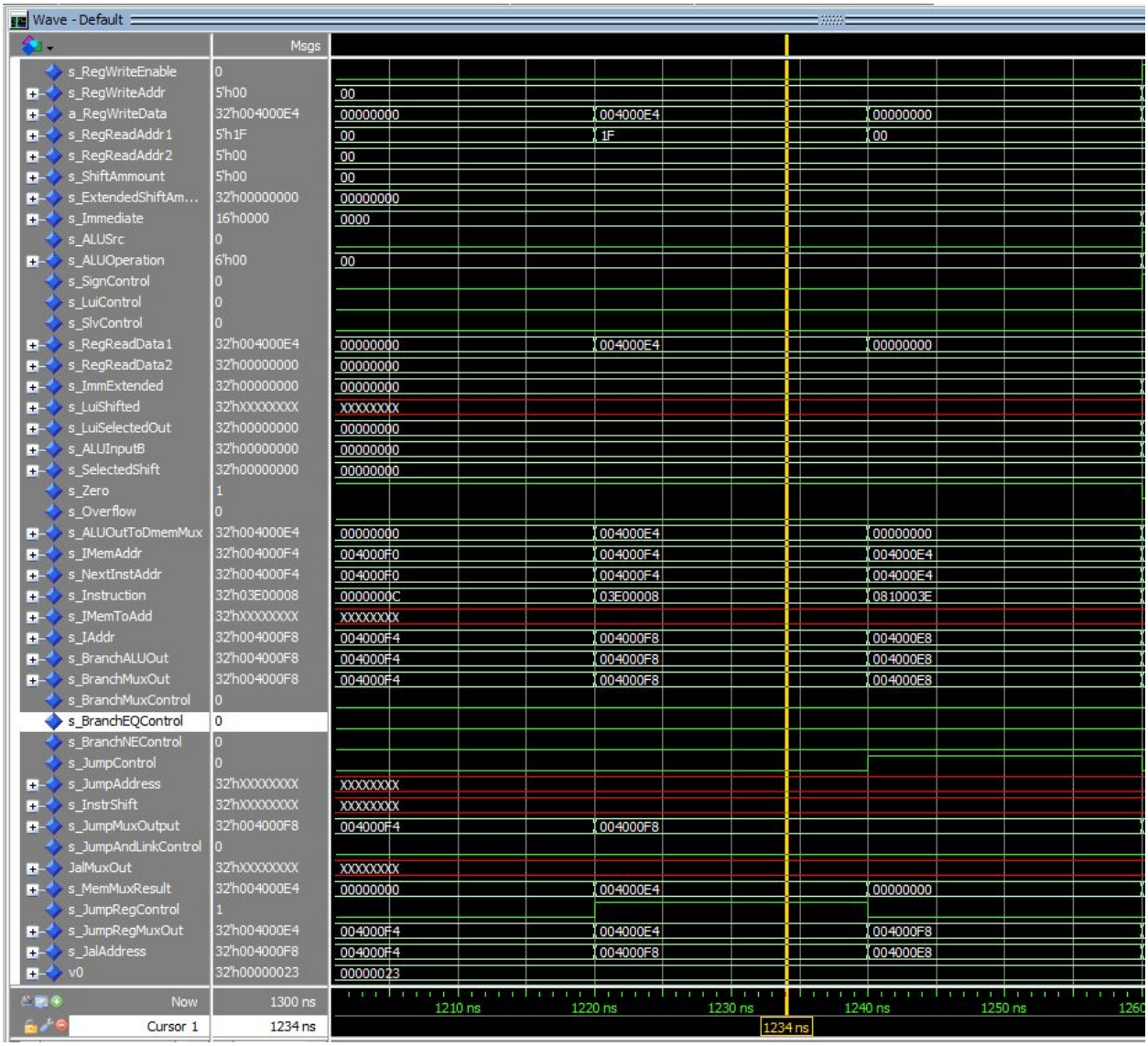
As you can see from the Mars screenshot of our test2 code that the address that we are branching to is C8 and the BranchMux out as well as the JumpMuxOut is C8, so that means that the next instruction address will be C8. And as you can see in the next wave, the correct instruction is executed.

0x004000e0	0x0c10003a	jal 0x004000e8	98: jal printBin # jump to Print the number 15 in 32 bit binary

0x004000e8	0x2008000f	addi $8,$0,0x0000000f	103: printBin: addi $8, $0, 15
0x004000ec	0x20020023	addi $2,$0,0x00000023	104: addi $v0, $0, 35
0x004000f0	0x0000000c	syscall	105: syscall



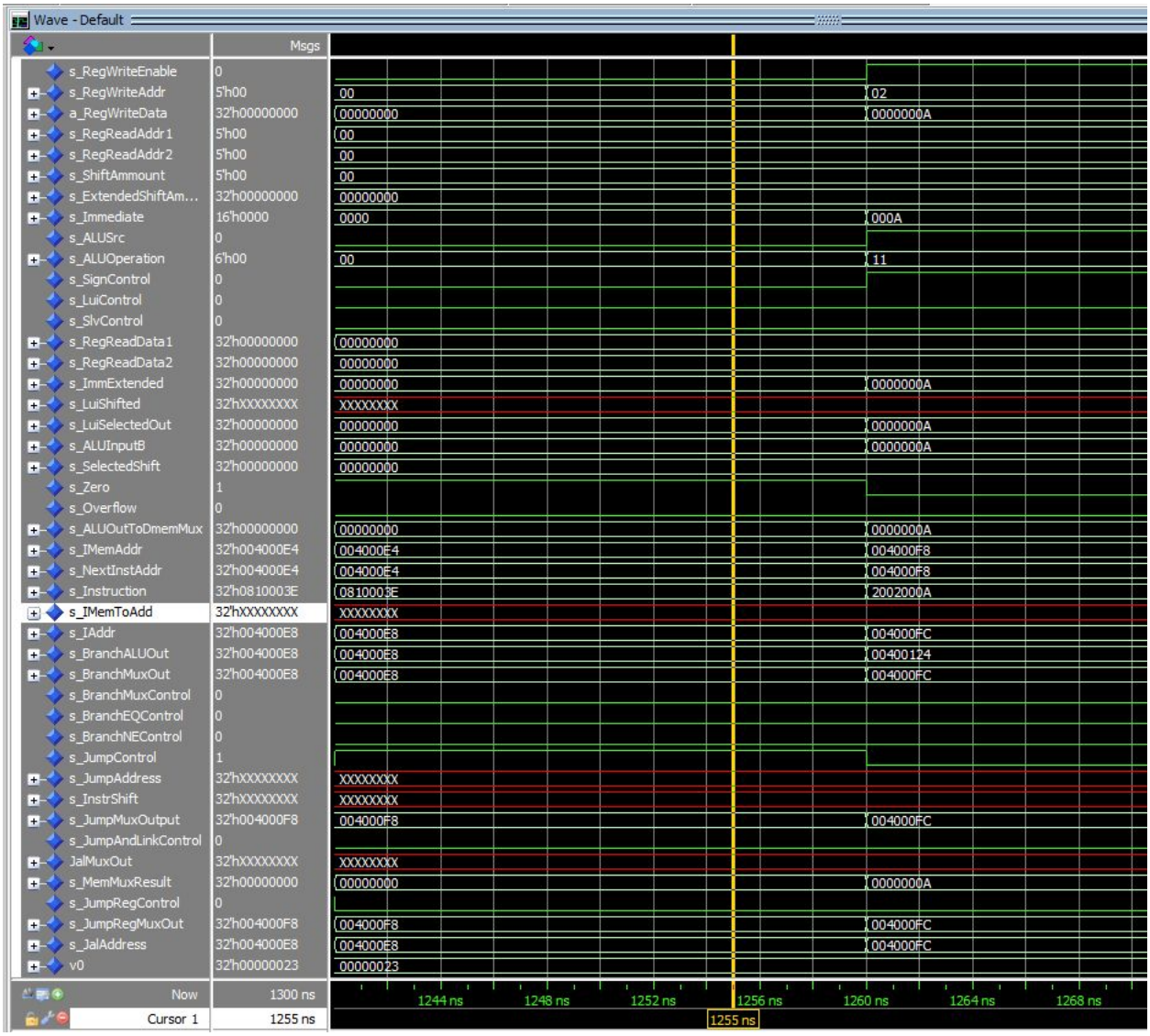
As you can see in the MARs screenshot, we are jumping to address e8 and storing e4 in register 31 to jump to later. In the modelsim screen shot you can see that we go from address e0 to e8 and store 4000e4 in register 31.



If you look at the previous screen shot you can see that we stored 400e4 into register 31, and if you look at this screenshot you can see that we jumped to adress 400e4

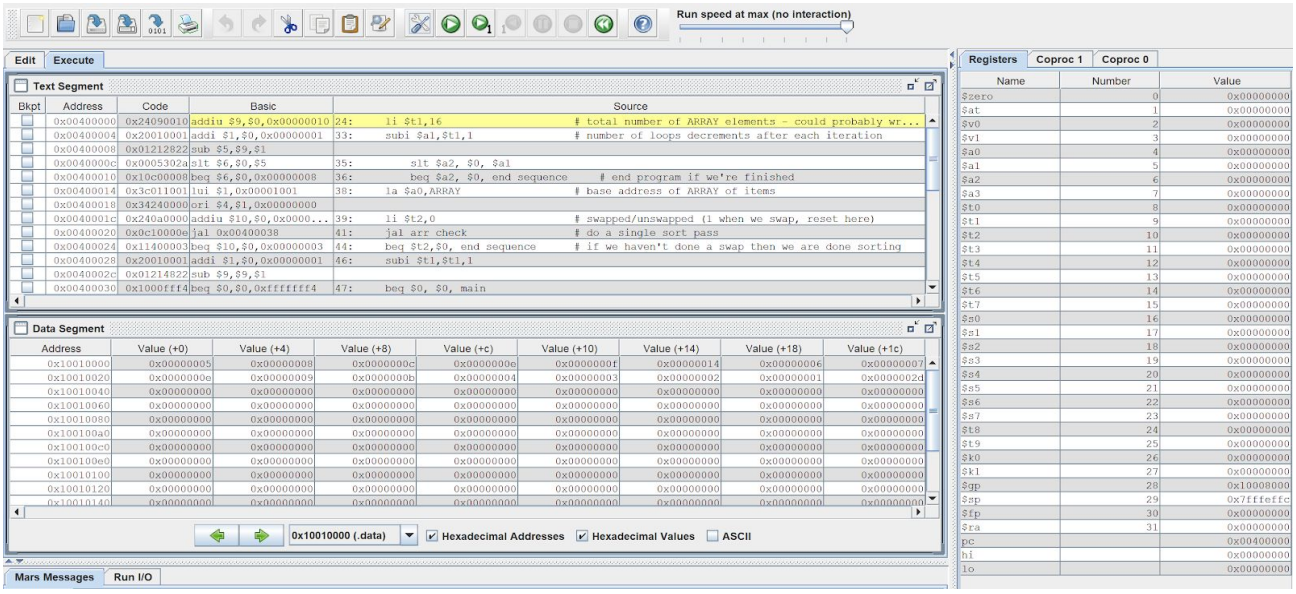
```
0x004000e4  0x0810003e  j 0x004000f8          100: j die
```

```
0x004000f8  0x2002000a  addi $2,$0,0x0000000a  110: die: addi  $2,  $0,  10
```

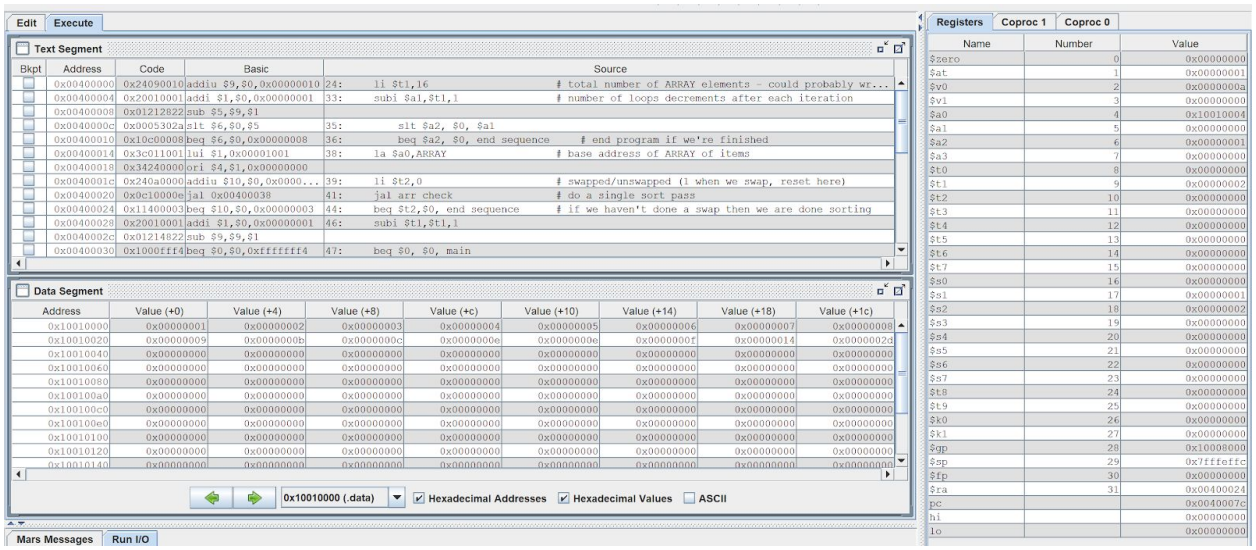


If you look at the Mars screenshot you can see that our jump, jumps to address 400f8, and if you look at the instruction address of the next instruction it matches 400f8 which shows that it jumped to the correct address.

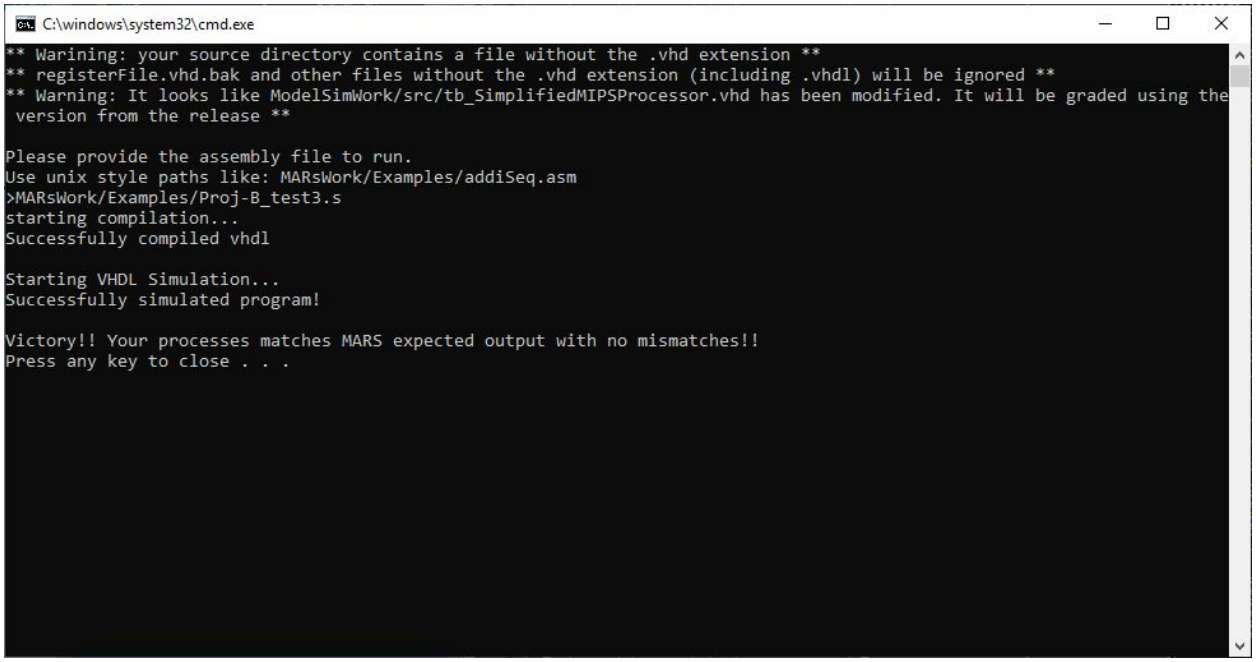
(5B) - Project B Test3 (Bubble Sort) Waveforms and Discussion:



The screenshot above shows the execution window of the bubble sort algorithm in Mars that was implemented during this project. The array that the program was to sort had a base address in the data memory of 0x10010000 (incrementing by 4 from one element to the next), and was made up of the following numbers: (5,8,12,14,15,20,6,7,14,9,11,4,3,2,1,45).



The screenshot above shows that upon running the program, the algorithm successfully places the elements in the array in order from lowest value to the highest value: (1,2,3,4,5,6,7,8,9,11,12,13,13,15,20,45).

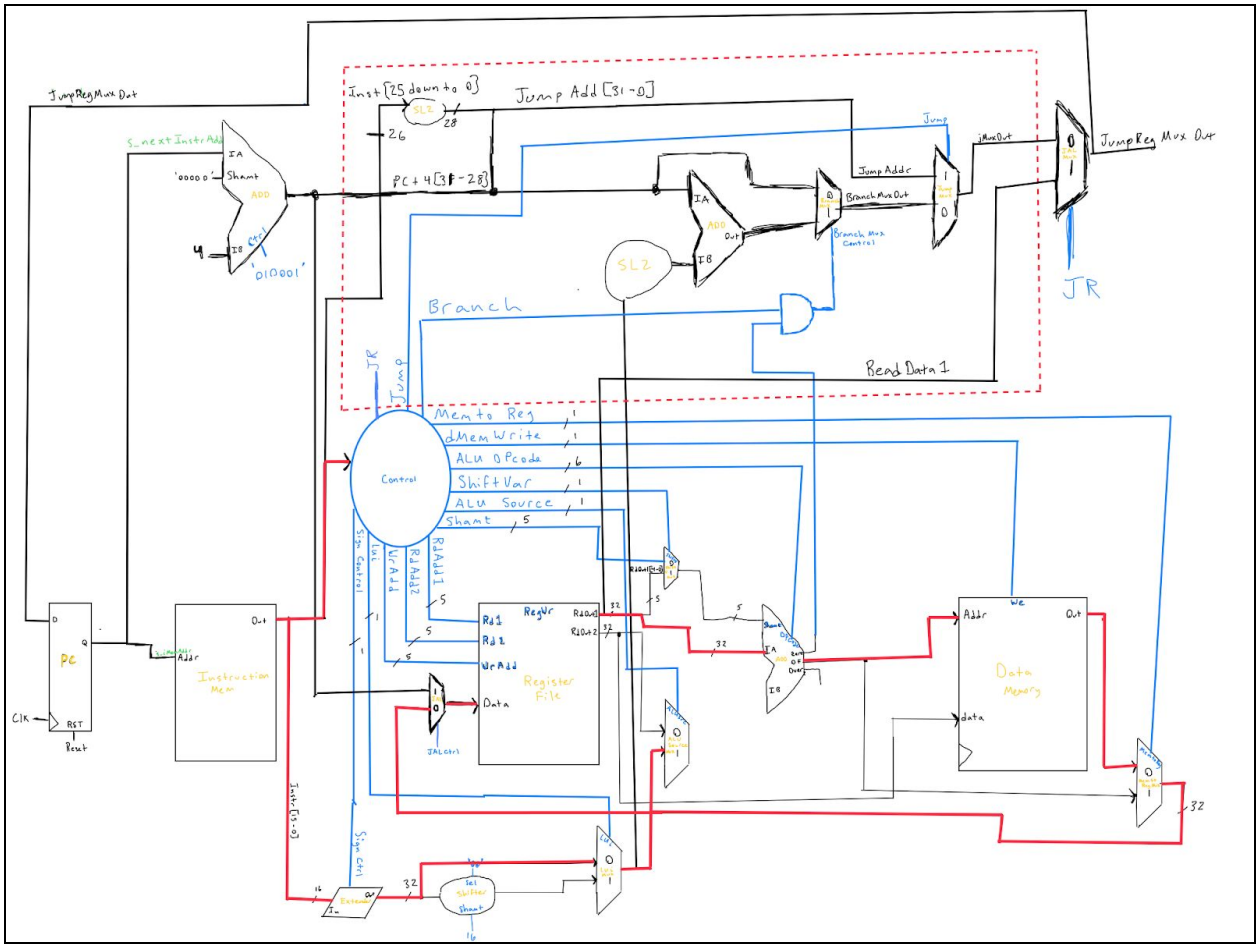


The above screenshot shows that our processor successfully runs our bubble sort algorithm.

Phase IV: Synthesis

- ***What is the maximum frequency this processor can run at?***
  - This processor can run at a frequency of 28.5 MHz.
- ***What is the critical path of this processor?***
  - The critical path of this processor is the path that a load word instruction takes.
- ***Discuss the results of your determined critical path:***
  - When observing the text file generated from synthesis, we can see that the critical path goes through instruction memory, into the barrel shifter at the bottom of the diagram, through the pair of muxes to get to the ALU, where it then goes into the data memory component, proceeds through a pair of muxes to get to the register file, where data is finally written to the corresponding write address.
- ***What components would you focus on to improve frequency?***
  - The synthesis seemed to spend a majority of its time in the alu, so I would try to improve on that

Critical Path in Top-Level Design:



Post-Lab Feedback

a. [Feedback] You must complete this section for your lab to be graded. Please complete each column **separately** for each team member; I expect it to take roughly 10 minutes (do not take more than 20 minutes).

i. How many hours did you spend on this lab?

Task	During lab time			Outside of lab time		
Team Initials	EF	EM		EF	EM	
Reading lab	1	1		1	.5	
Pencil/paper design	1	1		4	1	
VHDL design	3	3		6	12	
Assembly coding				3	.5	
Simulation						
Debugging	3	3		6	8	
Report writing				1.5	2	
Other:						
Total	8	8		21.5	24	

ii. If you could change one thing about the lab experience, what would it be? Why?

EF: I think at the beginning of the lab, there should be a greater focus on diagrams. In the lab manual it says to make a diagram, which everyone undoubtedly did, however, for us as we kept drawing more and more diagrams the issues that we were having became way easier to track down and squish out. I believe there were only a couple of bugs that took us longer than an hour to figure out, and one of those was with our PC counter logic which, after redrawing our diagram we were able to fix swiftly.

EM: Outside of two or three bugs that were relatively easy to fix after we figured out what was wrong, the lab wasn't that bad. It was only very time consuming. Maybe instead of this lab being one big lab, it might be better if phase one and phase 2 were split into two separate labs.

iii. What was the most interesting part of the lab?

EF: The most interesting part of this lab was being able to witness things "going green". Our control unit was relatively straightforward, but when it came to the processor and we would encounter errors, it was immensely satisfying when Ethan and I could turn and look at each other, and high five when we realized that our processor did what we wanted it to do.

EM: The most interesting part was seeing the progress we were making as we were squashing bugs. It feels very satisfying when your processor works better and better each time you run a test program. It is also very cool to see your processor executing a real program at the end of lab.