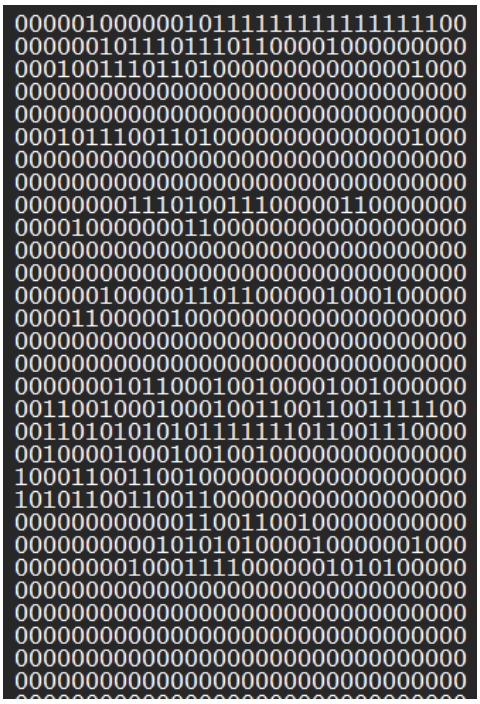
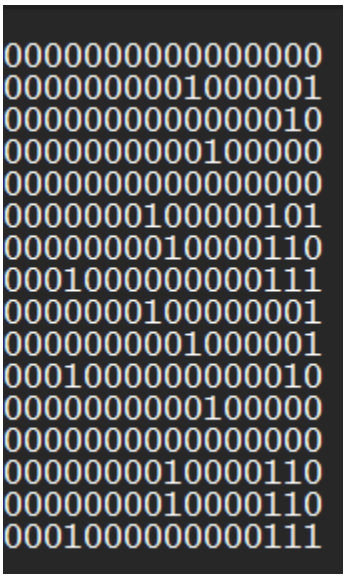
There is a testbench that includes all instructions. I will show and prove them one by one. Let me start with first show you register contents and instructions.

Instructions, (there is 1 instruction per line):



Registers:



Some infos about design:

I put together ALU control unit and main control unit. Function field and opcode field is connected there. My control unit can do both tasks. It generate signals such as MemWr, MemRd, PCSelect. Moreover, it generates ALU select bits.

IMPORTANT!

Load Word is slow instruction. It needs 2 clocks. And I display the current register status in every clock. So, in outputs there are 2 initial register data status output. I print register status twice. You must consider the first print for finding initial status of registers since they may change in instructions that need only 1 clock.

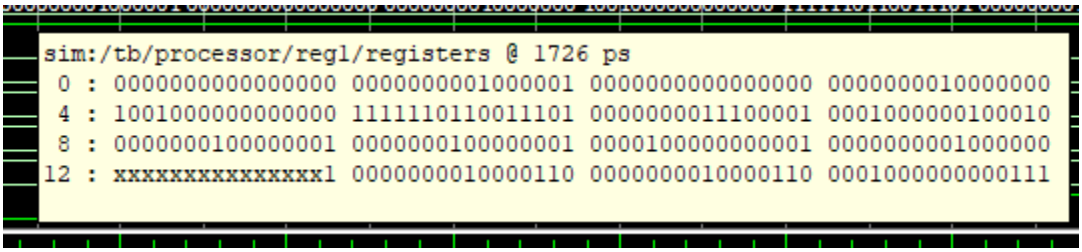
Some infos about tests:

In every instruction, I show the initial target register content and target memory content with $display. After that, I show the result of target register and target memory after instruction is done with $display. If instruction does not touch the memory contents (reading or writing), it may show garbage results. Please, consider outputs with purpose of instruction.

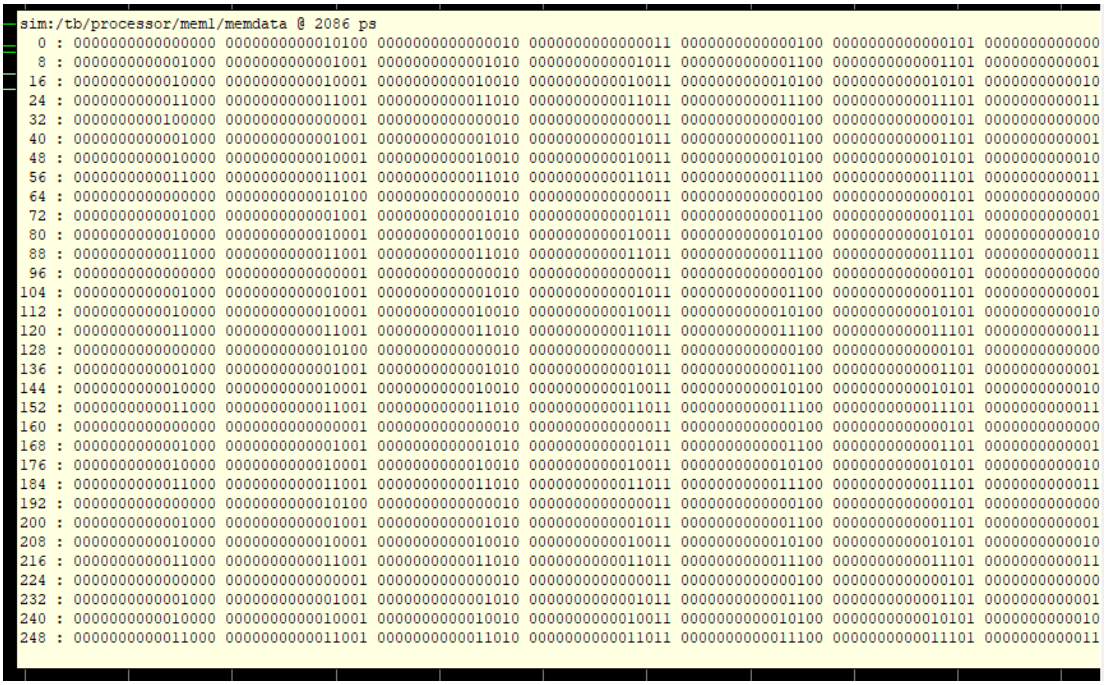
You can check initial values of register and data memory from the texts. They are in simulation/modelsim folder. I tried not to write results same registers so that you can follow up easily. I only overwrite to register 1 and 2.

I show the results with $display, but still if you want, at the end of the instructions, you can check the final results of registers and memory content from wave form.

The final output of registers from wave form:



Memory outputs does not fit here, but I can show some:

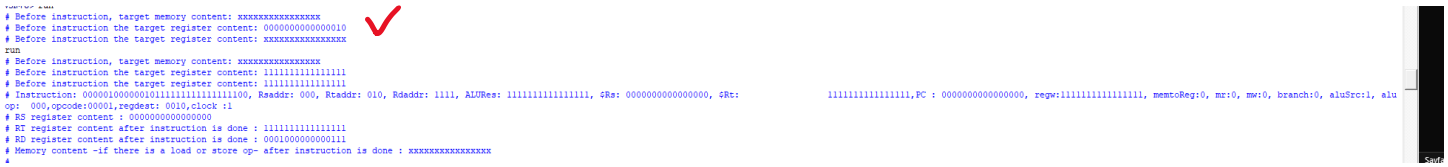


You can check the results both from $display outputs and wave forms. Now, I will show you $display outputs to prove my processor. It is more clear.

Now, let's start with first instruction.

1. Loadi

It takes as 1111111111111111 as immediate input and puts it to register 1011. Let's see the result:

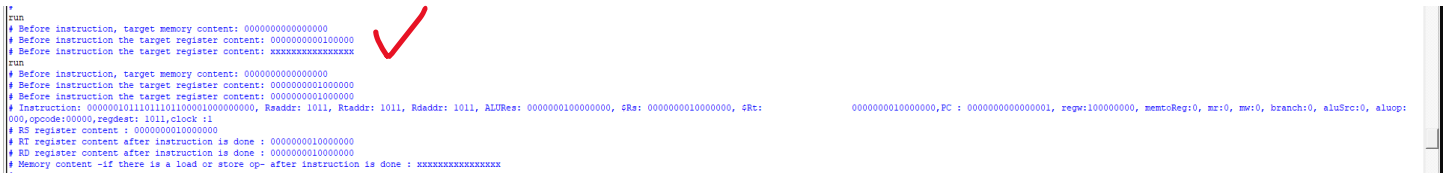


As you see, before instruction occurs the RT content is 0000000000000010 as you can see from the text above. After the instruction, it becomes 1111111111111111. Li instruction runs correctly.

You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

2. Add

It takes register 1011 as RS and register 1011 as RT, adds them and write it to the register 1011. Let's see the result:

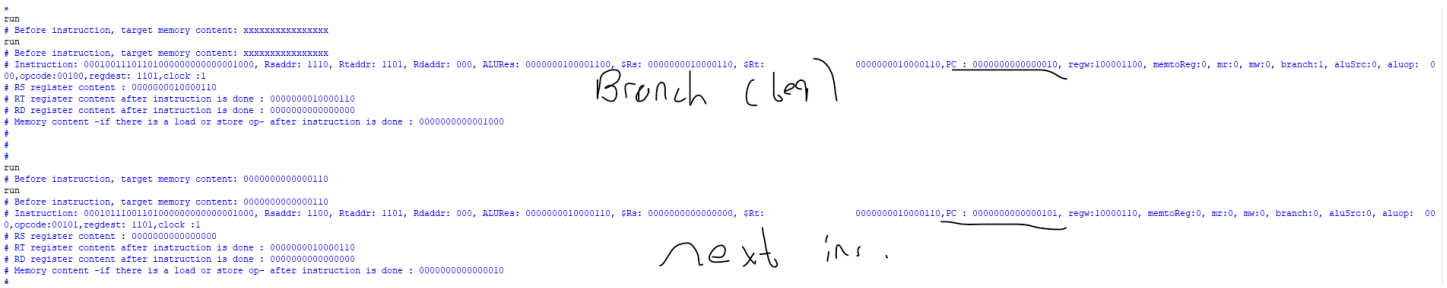


As you see, before instruction occurs RD content is 0000000000100000. After the instruction, it becomes 0000000001000000. It is correct. 32+32=64. Add instruction runs correctly.

You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

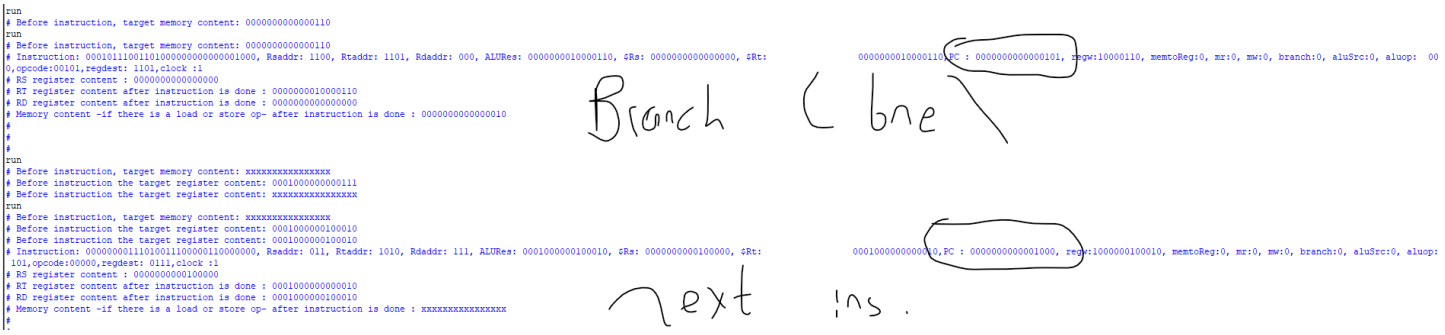
3. Branch Equal (beq)

It takes register 1110 as RS and register 1101 as RT, compares them, and if they are equal, it goes to PC + 1 + branch address(10). In this case, 0000000010000110 and 0000000010000110 is equal (You can check it from below). If you check the instruction list, it must skip 3 instructions 000…0 and 000…0 since branch address is 2 as decimal. As you can see output below, it skips nop instructions successfully. Also you can check the PC value from output. It was 2 before beq instruction, after that it became 5. Branch Equal runs correctly.



4. Branch Not Equal (bne)

I takes 1100 as RS and 1101 as RT, compares them, and if they are NOT equal, it goes to PC + 1 + branch address(10). In this case, 0000000000100000 and 0001000000000010 is not equal (You can check it from below). If you check the instruction list, it must skip 3 instructions 000…0 and 000…0 since branch address is 2 as decimal. As you can see output below, it skips nop instructions successfully. Also you can check the PC value from output. It was 5 before beq instruction, after that it became 8. Branch Equal runs correctly.



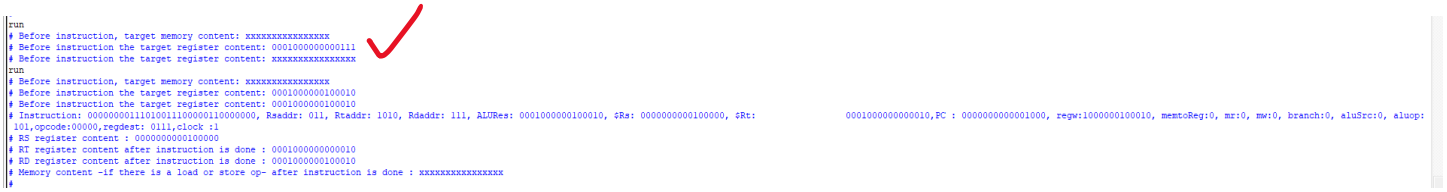
5. Or

It takes register 011 as RS, register 1010 as RT. And makes OR operation on them. It writes the result to RD register 111. In this case it will make an OR operation on data below:

0000000000100000

0001000000000010

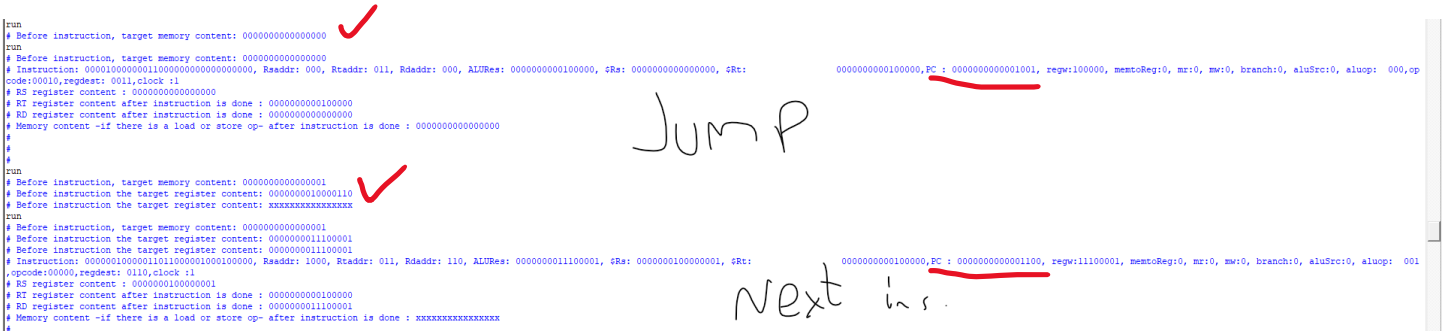
Result must be: 0001000000100010. As you can see output below, it makes the calculation correctly and writes the result to correct address. Or instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

6. Jump (j)

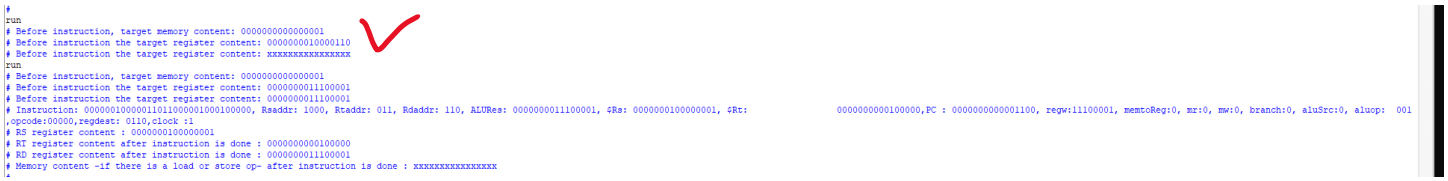
This instruction must go the address 00000000000000000000000000001100=12. As you can see, it skips the 12. Instruction. Please check the next instruction's PC, jump instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

7. Subtraction (sub)

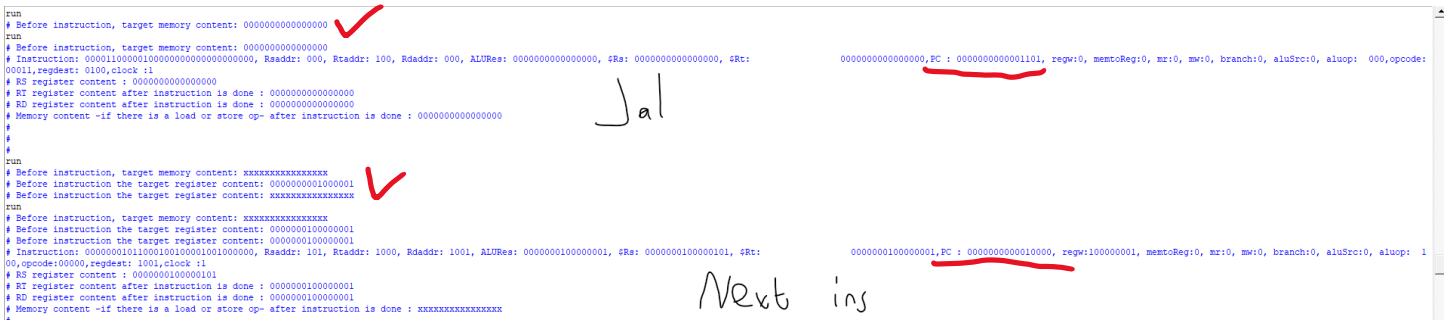
It takes register 1000 as RS, register 011 as RT. Then, it subtract reg 011 content from reg 1000 content and writes the result register 110 as RD. In this case, RS content is 0000000100000001, and RT content is 0000000000100000. So, the result must be 255 – 32 = 223. 223 is 0000000011100001. So, the result is true. Sub instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

8. Jump and Link (jal)

This instruction must go the address 00000000000000000000000000010000=16. As you can see, it skips the 16. Instruction. Please check the next instruction's PC, jump instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

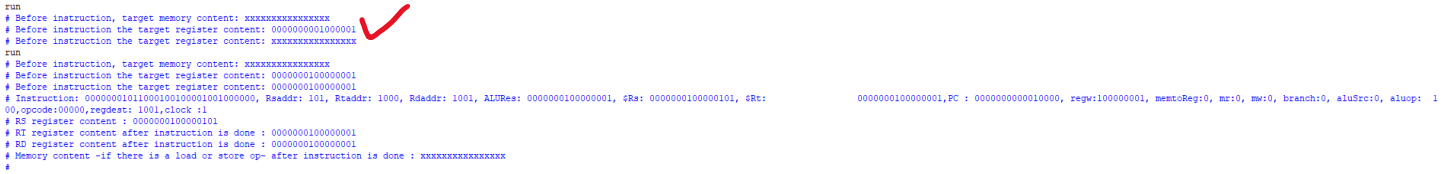
9. And

It takes register 101 as RS, and register 1000 as RT. It makes AND operation on them and writes the result register 1001 as RD. In this case, RS register content is 0000000100000101, RT register content is 0000000100000001.

0000000100000101

0000000100000001

When we apply and operation on data above, result must be 0000000100000001. As you can see, RD content after the instruction occurs, is true. And instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

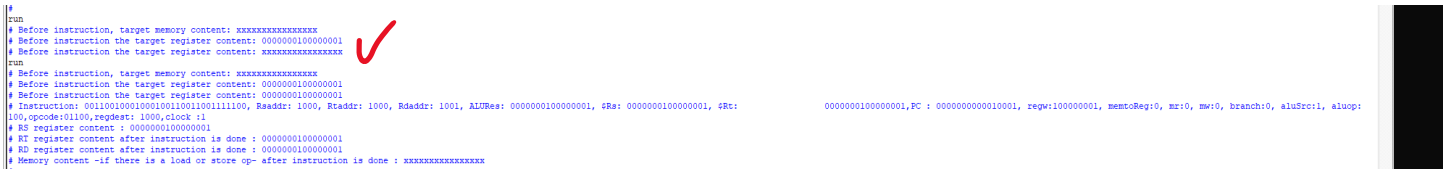
10. And Immediate

It takes register 1000 as RS. Its content is 0000000100000001. Instruction is 00110010001000100110011001111100. Its Immediate part is 1001100110011111. So, when we apply an AND operation on them:

0000000100000001

1001100110011111

The result must be 0000000100000001 and it must be written to the RT register. RT register content after instruction is done is: 0000000100000001. You can check it below. And Immediate instruction runs correctly.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

12 Or Immediate

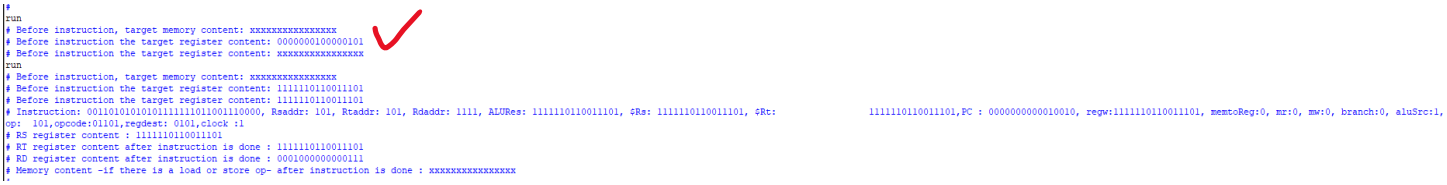
It takes register 101 as RS. Instruction is 00110101010101111111011001110000. So, immediate field is 1111110110011100. RS content is 0000000100000101 before instruction. Now, we will apply an OR operation on them.

0000000100000101

1111110110011100

The or result is: 11111110110011101

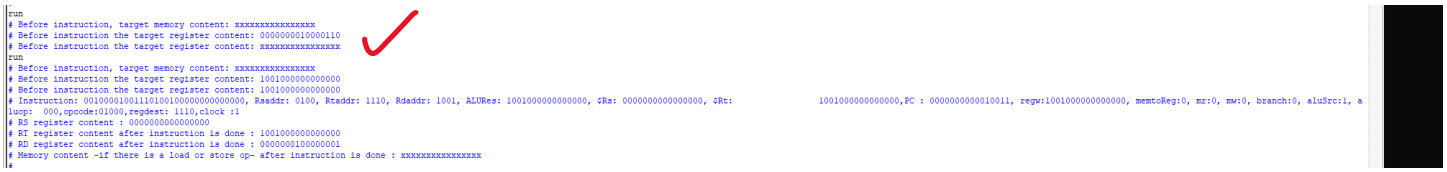
You see the result as 1111110110011101. So, Or Immediate runs correctly. (In the screenshot below RS seems to be changed but if you check the instruction RS and RT address are same. So, since RT is changed RS seems to be changed.)



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

13. Add Immediate

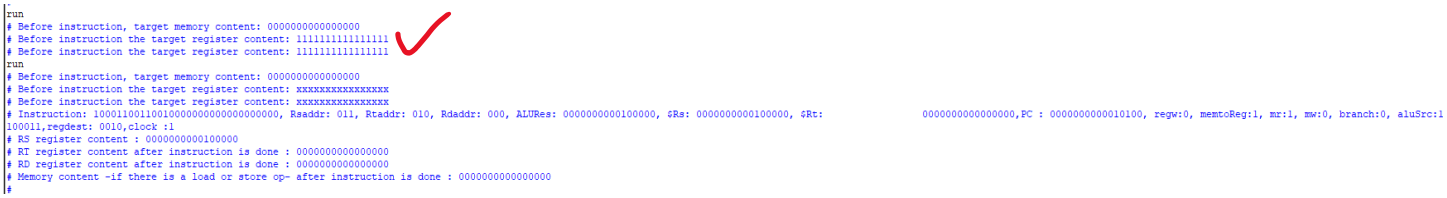
It takes register 100 as RS. Instruction is 00100001000100100100000000000000. So, immediate field is 1001000000. RS content is 0000000000000000 before instruction. RS Content + 1001000000000000 is 1001000000000000. So, Add Immediate runs correctly. (In the screenshot below RS seems to be changed but if you check the instruction RS and RT address are same. So, since RT is changed RS seems to be changed.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

14. Load Word

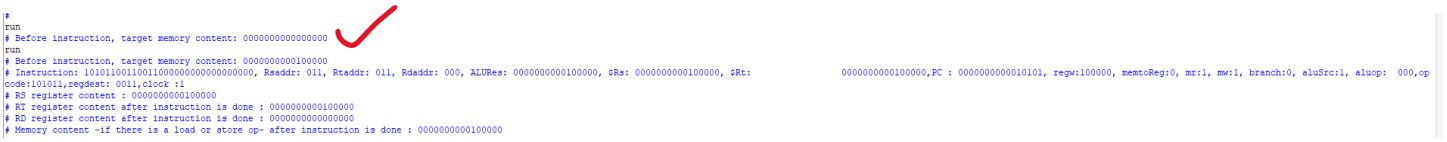
It takes register 011 as RS. The instruction is 10001100110010000000000000000000. So, immediate field is 0. Hence, it goes to address (content of RS=0000000000100000 + 0). It takes that value and put it to register 010 as RT. As you see in data text file and the output below, 32. Data of main memory is 16'd0. It takes it and put the RT. You can check the screenshow below. register is updated. RT address 010 is become 16'd0. So, Load Word runs correctly.



You must consider the first display output for initial register status print. I put a tick on it.

15. Store Word

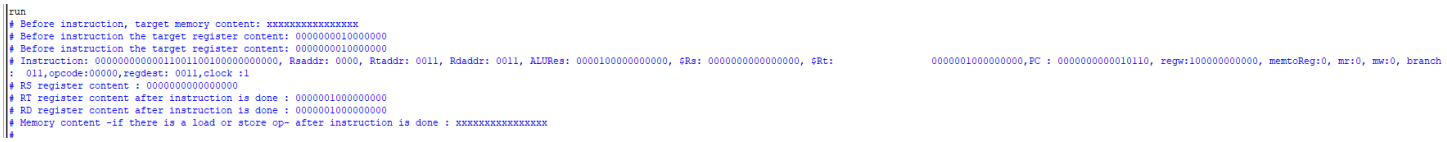
It takes register 011 RS as 0000000000100000. And it takes register 011 RT as 0000000000100000. After that, it will add immediate field to the RS content, then it writes the RT content to the RS content + immediate address. Instruction is 10101100110011000000000000000000. Immediate field is 0. It must take RT content 0000000000100000 and write it to the memory content. As you can see the result, memory content is updated as 0000000000100000. So, store word runs correctly. As I told you initially, you can check it from the final waves of memory content.



You must consider the first display output for initial register status print since It needs 1 clock. I put a tick on it.

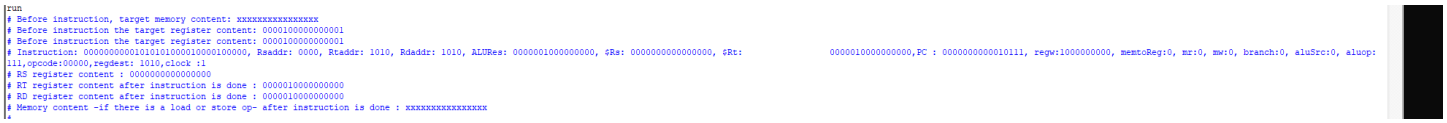
15. Shift Left Logical

It takes register 011 as RT. Its content is 0000000000100000. Instruction is 00000000000011001100100000000000. So, shamt field is 0010. It must be shifted 2 bit towards left. And we check the RD register content after instruction is done, we see 0000000010000000. Result is true. Shift Left Logical runs correctly.



16. Shift Right Logical

It takes register 1010 as RT. Its content is 0010000000000100. Instruction is 00000000001010101000010000001000. So, shamt field is 0001. It must be shifted 1 bit towards right. And we check the RD register content after instruction is done, we see 0000100000000001. Result is true. Shift Right Logical runs correctly.



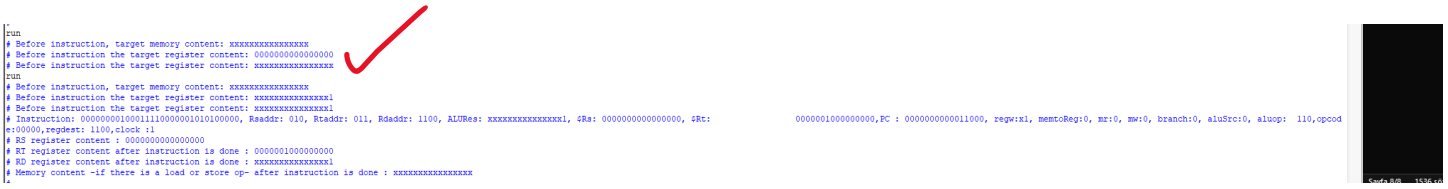
17. Set Less Than

It takes the register 010 as RS. And register 011 as RT. It compares them, if RS content < RT content result is 1, otherwise result is 0. Result will be written to register 1100 as RD.

RS content = 0000000000000000

RT content = 0000000010000000

As you see, RS<RT. So, output must be 1, and Register RD is updated as xx..x1. So, Set Less Than runs correctly



You must consider the first display output for initial register status print. I put a tick on it.