Stage 8

Divyanshu Pabia - RA2111003011373

The gtkwaves shown are the result of the testbench that I have submitted along with my code in this zip file. I have tested my design on a few custom-made examples. They are present in a folder named test_cases. I have tested all the instructions as given in the assignment pdf.

Design Description:

The total number of states in my final FSM are 22. I have added a few more states in this final design. They were for storing the pc value in a register during bl and swi instructions.

The tasks that I have completed in this assignment are

1. Bl performs as expected.
2. Swi instructions perform as expected. If the status bit is '0', it will wait for it to become '1'.
3. Ret and rte will return to the address saved in lr, by essentially doing mov pc,lr.
4. In supervisor mode, we will not be able to access the initial restricted space in the memory.
5. The condition checking is done in register write, memory write and flags are always set based on the s bit of the DP group and multiply group of instructions.

How to test my test cases:

Please ignore the bash script file, it was for my testing purpose. I have attached the test cases as .vhd files. To test them in the code, we must replace the complete code in mem.vhd with the code in the test case file, and then run by typing the command make in the terminal.

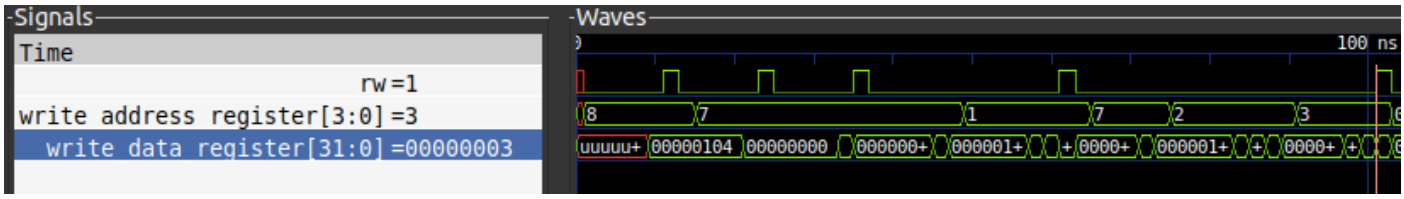
Note: Input port data cannot be accessed in user mode because I have included it in the condition:

Mem_input <= "000000000000000000000000"&input_data(7 downto 0) when memory_address = "001110000" **and mode = '1'** else read_memory_data;
Mode = '1' implies supervisor mode here.

All these programs work in my design. Nothing has been hardcoded, or taken from anybody else. I have included all essentials signals in the gtkwaves that I have submitted. If needed, I can submit more as well. I have extensively tested all commands.

Test Cases and Output:
>Test file 1 – testcase1.s

(Full predication and s-bit)
mov r8, #260
movs r7,#0
add r7,r7,#3
strne r7,[r8]
ldreq r1,[r8]
streq r7,[r8]
ldrne r2,[r8]
ldreq r3,[r8]



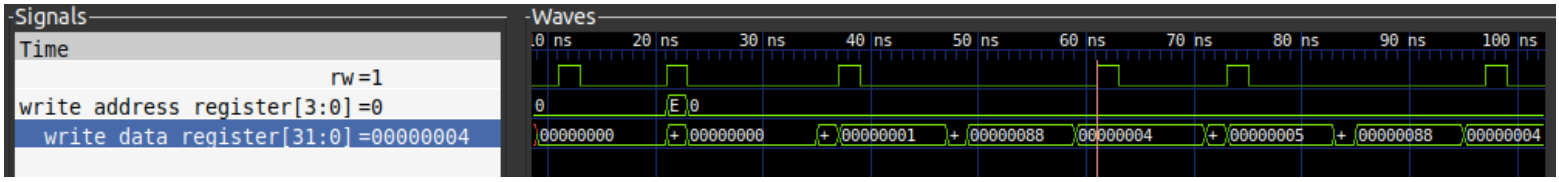
Explanation: Here movs r7,#0 sets the Z flag as visible in the gtkwave. When the flag is set, memory write = “1111”. Then I use strne and as expected, the value of r7 is not stored(we can see in the next ldreq line that the value of r1 is 0 and not 3). The next instruction is streq, which means that r7 is now stored at the location given by r8. The next instruction ldrne implies that no register will be written(predicate = false). The final instruction writes the value at location r8 into the register r3, just as expected.

The same logic for predication has been used for branch instructions therefore I have not shown the output. (branch output has been shown in previous assignments).

>Test file 2 – testcase2.s

(testing bl and ret/rte)

```
mov r0, #0
bl inc
mov r0,#4
inc: add r0,r0,#1
ret
```



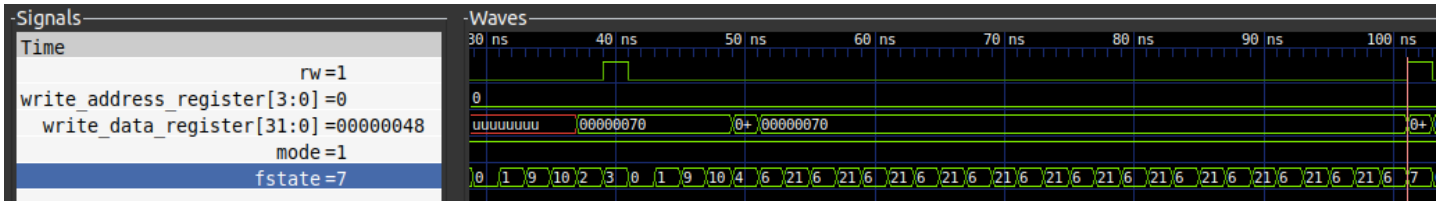
Explanation:

In the output, r0 first takes the value 1, then 4 then 5 then it goes in an infinite loop between 4 and 5. This the expected output because, after branching to inc, pc has to come back to the next instruction as well. Therefore, after adding 1 it comes back to mov r0,#4 and then proceeds normally. Now, when it encounters ret again, it goes to the value still stored in r14, which is the address of mov r0,#4.

>Test file 3 – testcase3.s

(testing swi and status bit)

Instruction: swi



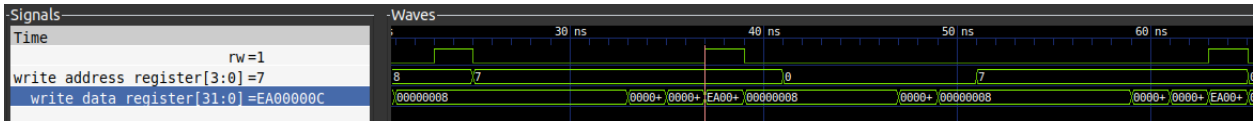
Explanation:

In my testbench, the status bit is initially = 0 and changed to 1 at t = 100ns. Which is why the program is stuck between the control states 6 and 21 and at 100ns it exits it. The program will continue to alternate between these two states until the status bit is "ready". The input 8 bits were "48" which is then stored in r0. The mode is also set to 1 = supervisor mode at the beginning.

>Test file 4 – testcase4.s

(restricted usage of user mode and supervisor mode)

```
mov r0, #4
mov r8,#8
ldr r7,[r8]
str r0,[r8]
ldr r7,[r8]
```



Explanation:

I have printed out the value of r7 before and after the str statement. Here, the address in str points to a location in the restricted space of the supervisor mode, therefore the data is NOT written. We can see that the same garbage value is present in r7 before and after the str statement.