COL216 Assignment 2 Stage 7 – 25/03/2022
Aryan Dua – 2020CS50475

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 methods of using all the features of multiply group instructions as given in the assignment pdf. **Smlal** and **smull** work as expected.

Design Description:
I have added another module named multiplier. I have used the **ALU to add** 2 64-bit numbers as I felt I could do it after I already implemented the simple 64-bit design as mentioned in the pdf, if you want, I can submit that later as well.

I have also added 6 more control states to my FSM. The first is for reading an extra register from the register file, the second for the multiplier. The next 4 alternate between **ALU adder** and Register file **write**. (We need 2 each because of long multiply instructions). I feel that this is not the most efficient design, and I will optimize on the number of states in the next stage once there is nothing more to be added. The **S bit instructions** have also been taken care of while integrating with this complex ALU + multiplier design.

Some difficulties I faced are: I had to change a lot of connections to the 2 operands of the ALU. I also had to take care of the **C flag** while doing the addition of 64 bit numbers using a 32 bit ALU.

I have attached the gtkwave outputs of as many signals as I could/are relevant. I have tested: all 6 multiply/accumulate instructions as given in the table.

How to test my test cases:

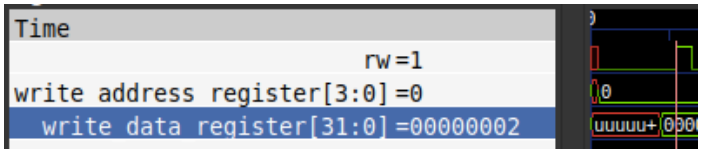
Please ignore the bash script file and the makefile, they were for my testing purposes. I have attached the test cases as .vhd files. To test them in the code, we must replace the complete code in mem.vhd with the code in the test case file, and then run using ghdl. To check if the required output is being stored correctly, read below:

Guide to my gtkwave output:
1. Check write_data_register [31:0] and write_address_register when RW = 1.
2. write_data_register [31:0] is the value going into the register file.
3. write_address_register is the address at which the value is being entered

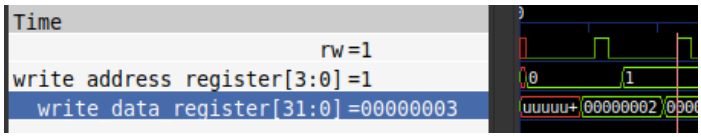
All these programs work in my design. Nothing has been hardcoded, or taken from anybody else. I have included all essentials signals in the gtkwaves that I have submitted. If needed, I can submit more as well. I have extensively tested all commands.

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

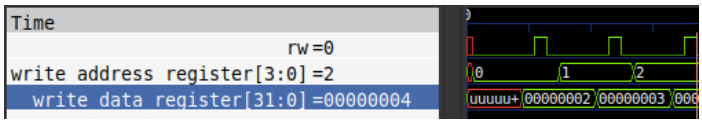
1. mov r0,#2



2. mov r1,#3

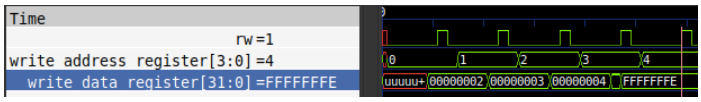


3. mov r2,#4

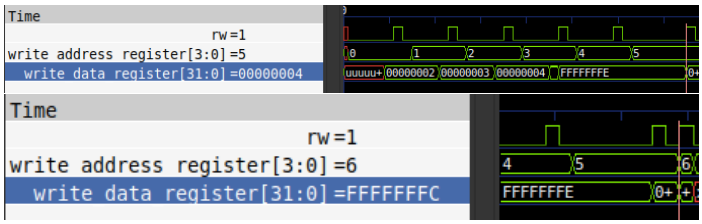


4. sub r3,r0,r2

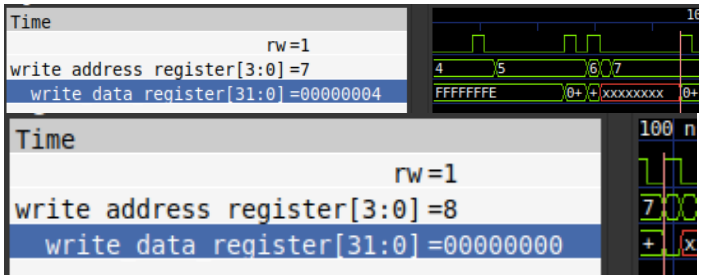
5. mov r4, r3



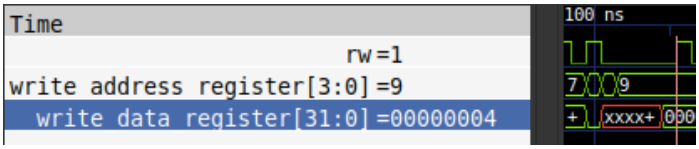
6. umull r5,r6,r3, r4



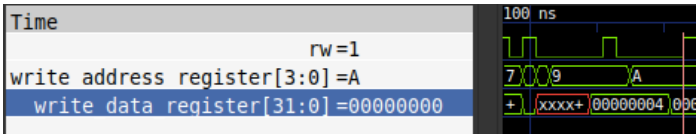
7. smull r7,r8,r3, r4



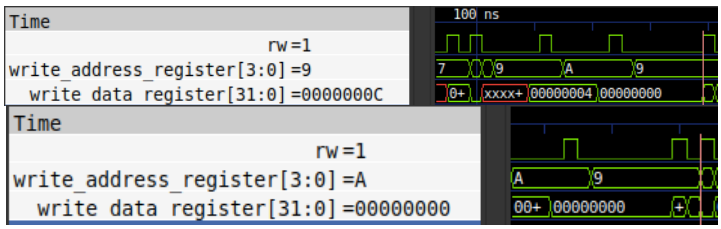
8. mov r9,r7



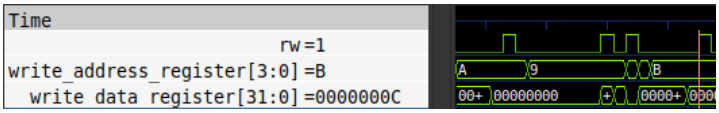
9. mov r10,r8



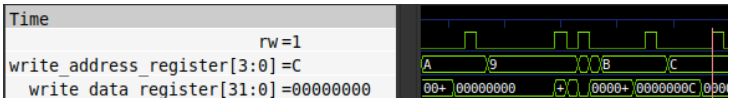
10. umlal r9,r10,r0, r2



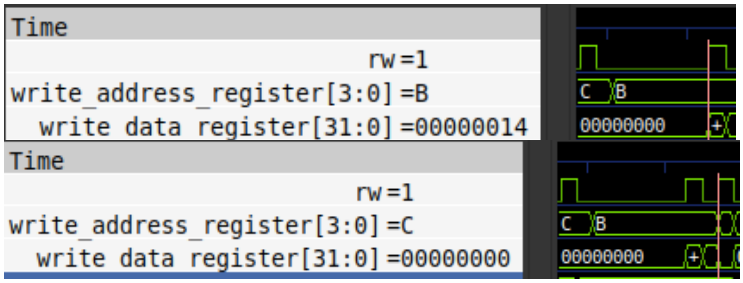
11. mov r11,r9



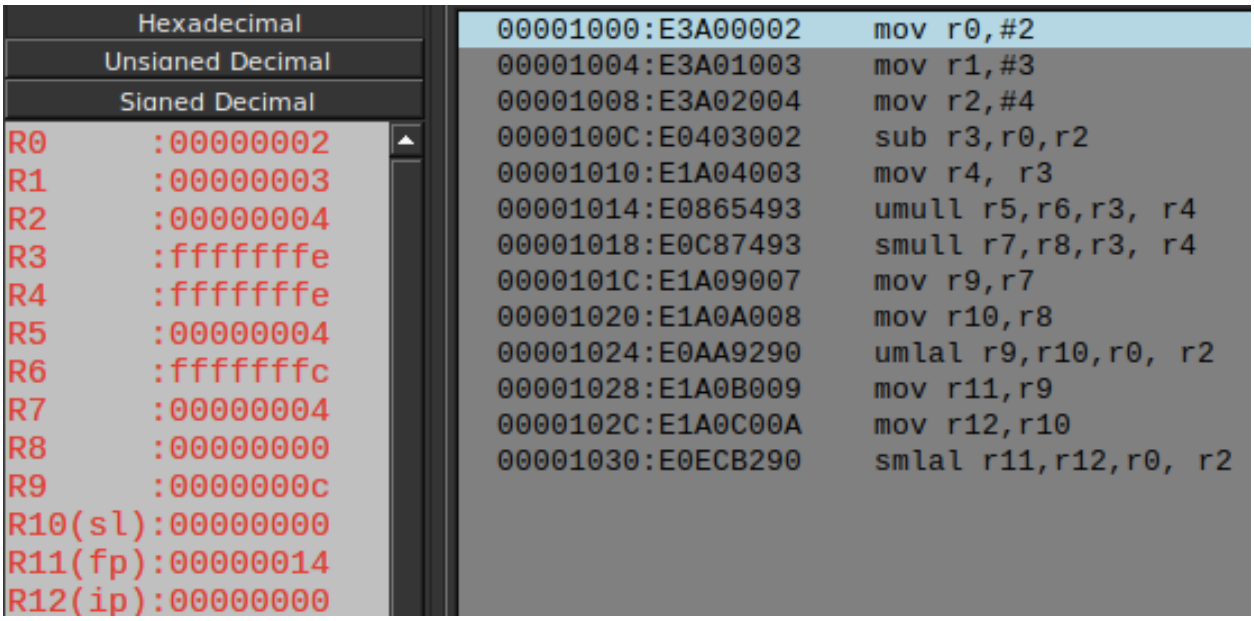
12. mov r12,r10



13. smlal r11,r12,r0, r2

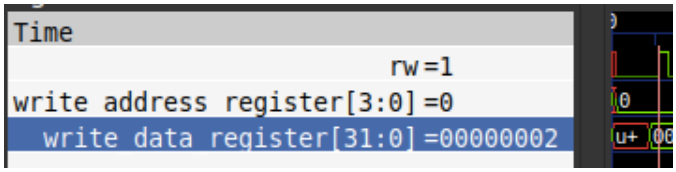


ARMSim output:-

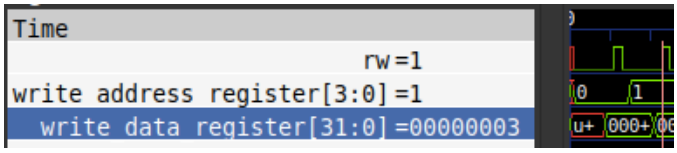


>Test file 2 – testcase2.s

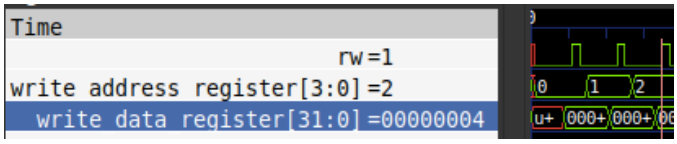
1. mov r0,#2



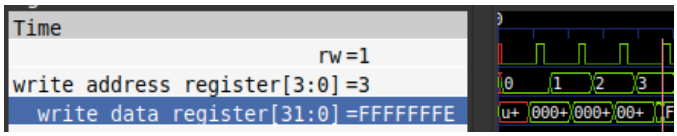
2. mov r1,#3



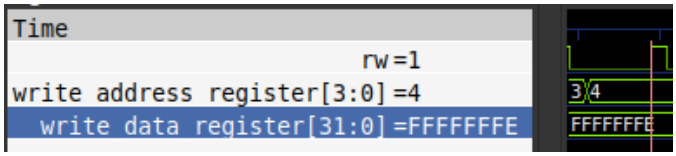
3. mov r2,#4



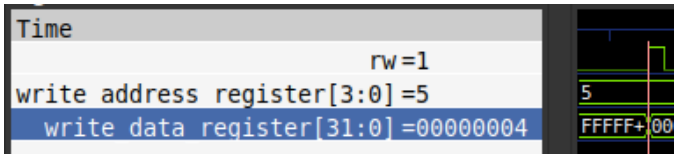
4. sub r3,r0,r2



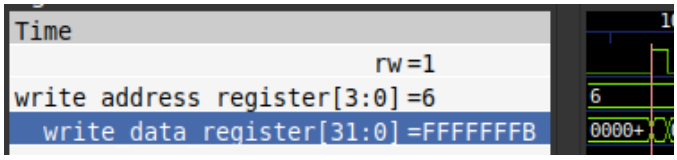
5. mov r4, r3



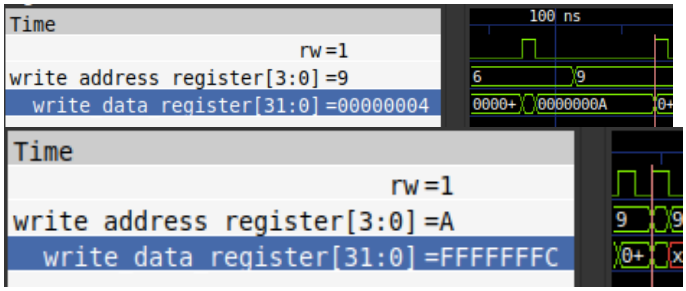
6. mul r5,r3, r4



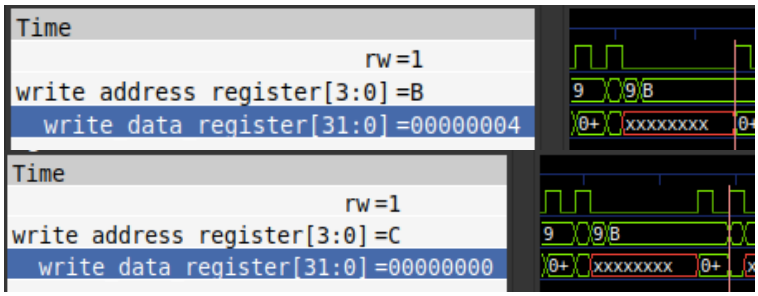
7. mla r6,r5, r4, r1



8. umull r9,r10,r3, r4



9. smull r11,r12,r3, r4



ARMSim output: