Lab4 – Rayan Hassan

Explanation

I changed my multiplier to have only 2 inputs (A and B) and one output (R). A and B are going to be fed directly from the instruction file (getting rs and rt). The output will be divided into HI and LO going into register HI and register LO respectively. The output of each of these registers is fed to the multiplexer that has the MemtoReg enable signal, which chooses which part is going to be written in the register file and where.

InstrF and InstrD decode and the same for all instructions but for MULTU, selHi and selLo are going to be '1' to load the registers HI and LO directly. And that's it for MULTU (only has 2 stages fetch and decode). For MFLO and MFHI, they go to MultCompl stage after InstrD, where RegWrite='1' and MemtoReg = "100" for MFLO and "010" for MFHI, and RegDst='1' because we want to write at address rd. So in total, I have 3 stages for these instructions, InstrF-InstrD-MultCompl.

Testbench

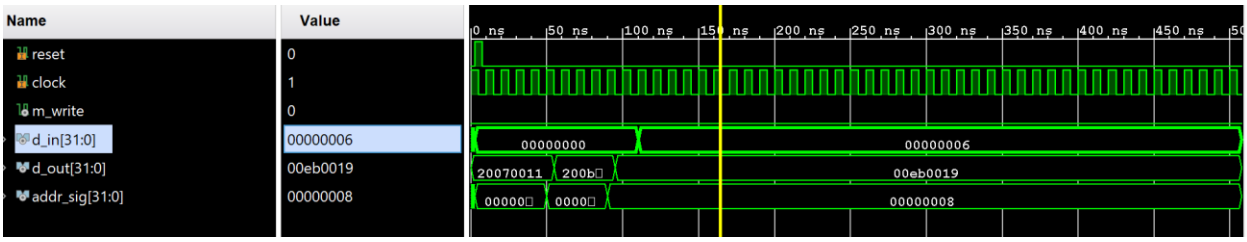
Here is my testbench to test these instructions (I also included it in the zip file "Mult_tb.tcl")

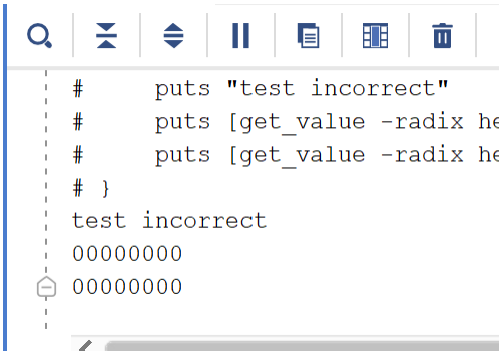
```
# restart the simulation
restart

#top-level CPU testbench is named cpu_tb
#this instruction will add the internal signals and ports of a component name U_1, which in this case is the memory block.
#this should be replaced by the name of the componenet in your top-level testbench
#add_wave {{/cpu_tb/U_1}}

#addi $7, $0, 17
#addi $11, $0, 6
#multu &7, &11
#mfhi $1
#mflo $2
#sw $1 15($8)
#sw $2 19($8)

# you can use any of the following commands as an example on how to initilaize a memory location with a value
# the first 4 memory locations are initialized with the instruction codes correpsonding to the 4 instructions above.
add_force {/cpu_tb/U_1/mw_U_0ram_table[0]} -radix hex {20070011}
add_force {/cpu_tb/U_1/mw_U_0ram_table[1]} -radix hex {200B0006}
add_force {/cpu_tb/U_1/mw_U_0ram_table[2]} -radix hex {00EB0019}
add_force {/cpu_tb/U_1/mw_U_0ram_table[3]} -radix hex {00000810}
add_force {/cpu_tb/U_1/mw_U_0ram_table[4]} -radix hex {00001012}
add_force {/cpu_tb/U_1/mw_U_0ram_table[5]} -radix hex {ACE1000F}
add_force {/cpu_tb/U_1/mw_U_0ram_table[6]} -radix hex {AD020013}
add_force {/cpu_tb/U_1/mw_U_0ram_table[7]} -radix hex {00000000}

#forcing a clock with 10 ns period
add_force clock 1 {0 5ns} -repeat_every 10ns

#give a reset signal
add_force reset 0
run 2500ps
add_force reset 1
run 5 ns
add_force reset 0

run 500 ns

if { [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[8]} ] == {00000000} && [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[9]} ] == {
    puts "test correct"
    puts [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[8]}  ]
    puts [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[9]}  ]
} else {
    puts "test incorrect"
    puts [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[8]} ]
    puts [get_value -radix hex {/cpu_tb/U_1/mw_U_0ram_table[9]}  ]
}
```

I should have “00000000” at the 8th location of memory and “00000066” at the 9th one because we are multiplying 17 and 6 and putting the result (102) in the register file (HI at $1 and LO at $2), and then we are storing these values in memory.

Wrong output and how to fix it

Here is the output waveform. It did not store the expected values in memory.





Solution

To fix that I think I have to put all these instructions in the same stages and not have a different one for MULTU, because we want to grab the values stores in the register file directly after. So I would have the same 3 stages for MFHI and MFLO but with MULTU included and with if statements I can give the right signals accordingly.

Another way I can fix it is by implementing the multiplier unit as it was given to us (which is what I did first, but it didn’t work as well so I had to try something else). If I try it again using that implementation (with done signal, clock and reset), MULTU would go from InstrD to MultCompl only if done = 1, else it goes back to InstrD. In MultCompl, the values coming out of the multiplier and loaded in registers HI and LO will be written back (through the same multiplexer, the one that has MemtoReg signal) to the register file. As for MFLO and MFHI, the registers HI and LO will get loaded during InstrD.