ECE 212 – Digital Circuits II

Lab11 – Pipelined Processor (Part1)

Otieno Maurice (Scribe)

Alex Villalba

https://github.com/otienomaurice1/ece212_alex_maurice.git

Introduction:

The goal of this lab was to confer the mips pipelined processor the ability to compute jal and jump register instructions. These instructions are needed for function calls and returns.

a) The jal instruction

During jump and link, we store the address of the next instruction to be computed i.e. pc+4 in the register file. This address Will later be read out of the register file during the jump register instruction. We also fetch a given instruction in memory. The address of this instruction is given by concatenating the upper four bits of the pc, with the address in the immediate field shifted to the left by four. The latter process is similar to that of the jump instruction. Since the data path is already capable of handling the jump instruction, we can use the same path for the jump and link instruction. We now design the path to store the pc+4 address into the register file. The datapath already computes pc+4 in the decode stage. We take this value in the decode stage and pass it all the way to the write back stage through the pipeline registers. In the writeback stage, we add a multiplexer to choose between pc+4 and a value computed by the alu. The selector signal for this multiplexer is a signal named jal_w. This is a one-bit signal that makes the multiplexer choose pc+4_w when it is high. The jal_w signal is a signal generated in the control unit during the decode stage and passed along the pipelined registers all the way to the writeback stage. We created this signal since it was not part of the original outputs of the decoder. The main decoder is responsible for generating this signal. It checks the opcode and compares it to that of jal and if they match, it drives jal high. Recall that we use the same data path for the jal as that of the jump instruction. This means that the jump signal must also be high during the decode stage of the jal instruction.

Another important modification to the datapath for the jump and link instruction is to specify the register into which pc+4 will be written. This register is specified to be register 31 by the mips architecture. We need to specify this register in the execute stage. To do this, we extend the register destination multiplexer from a mux2 into a mux 3 with the third input being the five-bit constant value 31. Extending the mux to a three-input mux will mean extending the selector signal for this multiplexer to a two bit signal in the main decoder. The selector signal regdest now select register rd when its valu is 0, register rt when its value is 1 and register 31 when its value is 2.

b) The jr instruction

During the jr instruction, we jump to the address specified in the register file. This value is read out of port rd1. We need to update the pc to this value after the next clock edge. To do this, we add a multiplexer in the datapath to choose between the jump and register addresses. The selector signal

for this new multiplexer is the jumpr_d signal. When it is high, the multiplexer should select the value from port rd1 in the decode stage. This signal should be driven high when the instruction opcode is similar to that of the jr instruction.

In general we, created jal, jumpr and regdst[1:0] signals , added two mux2's and modified one mux2 into a mux3.

The top-level diagram showing the changes is given below.

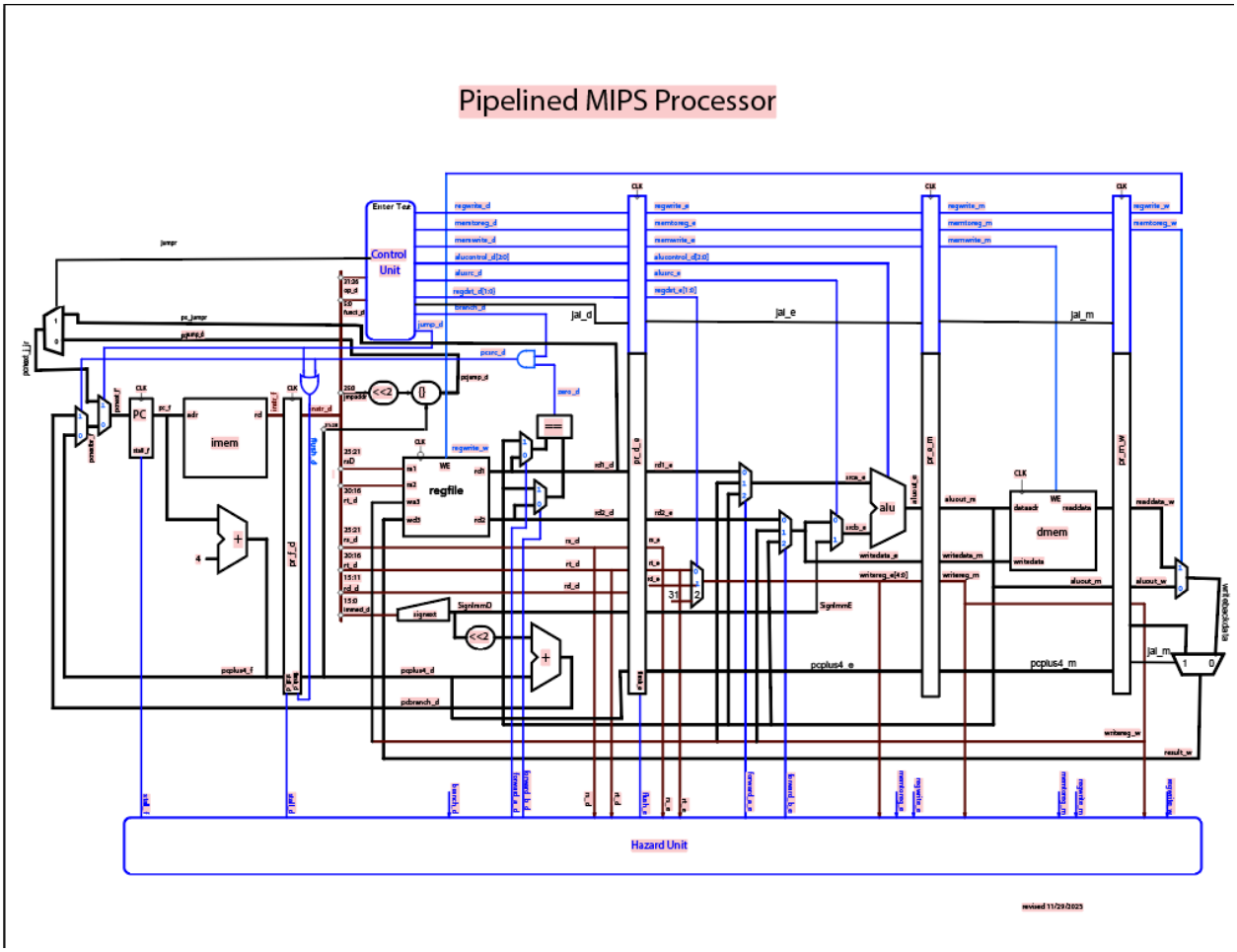

Figure 1 top level diagram showing modifications to the pipelined processor to execute jal and jr instructions

Results

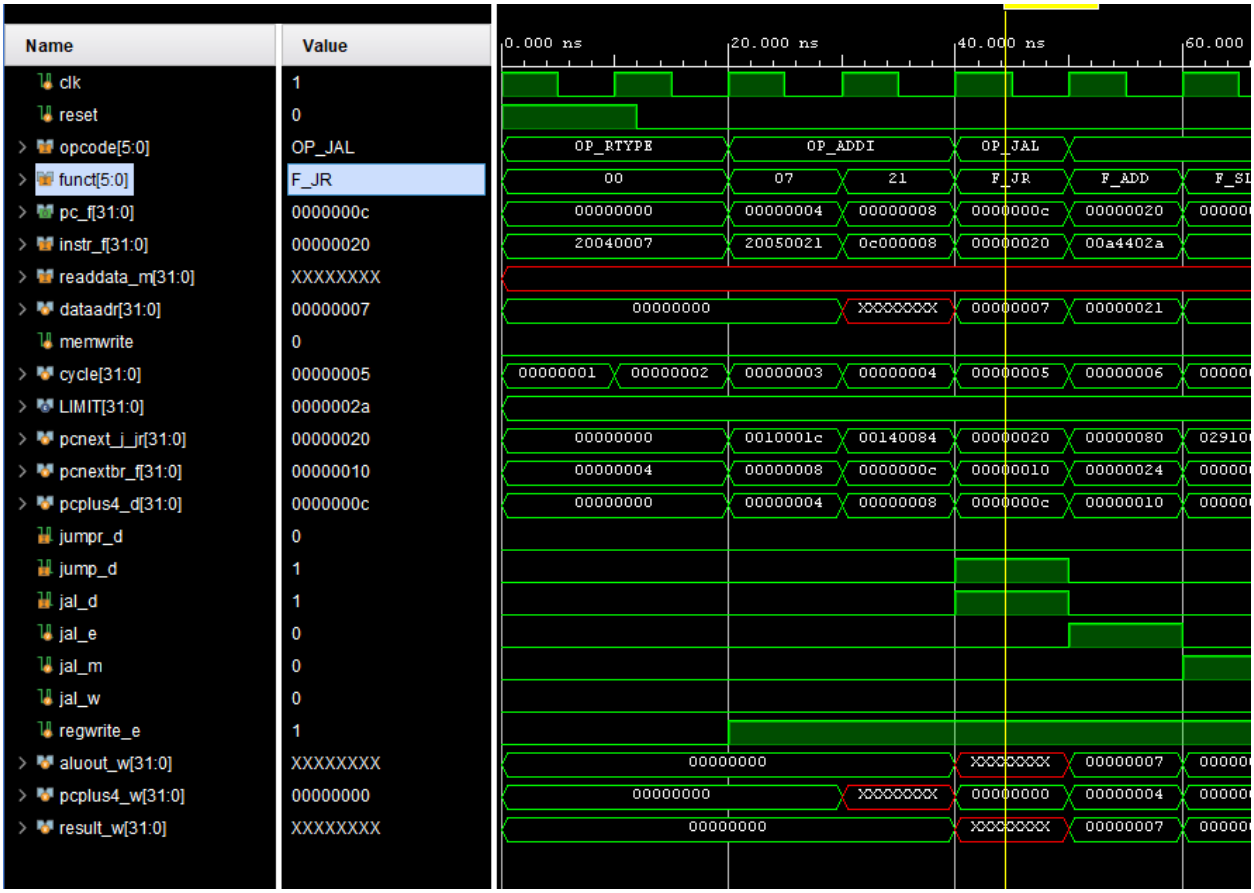

Figure 2simulation scope showing the execution the jal instr

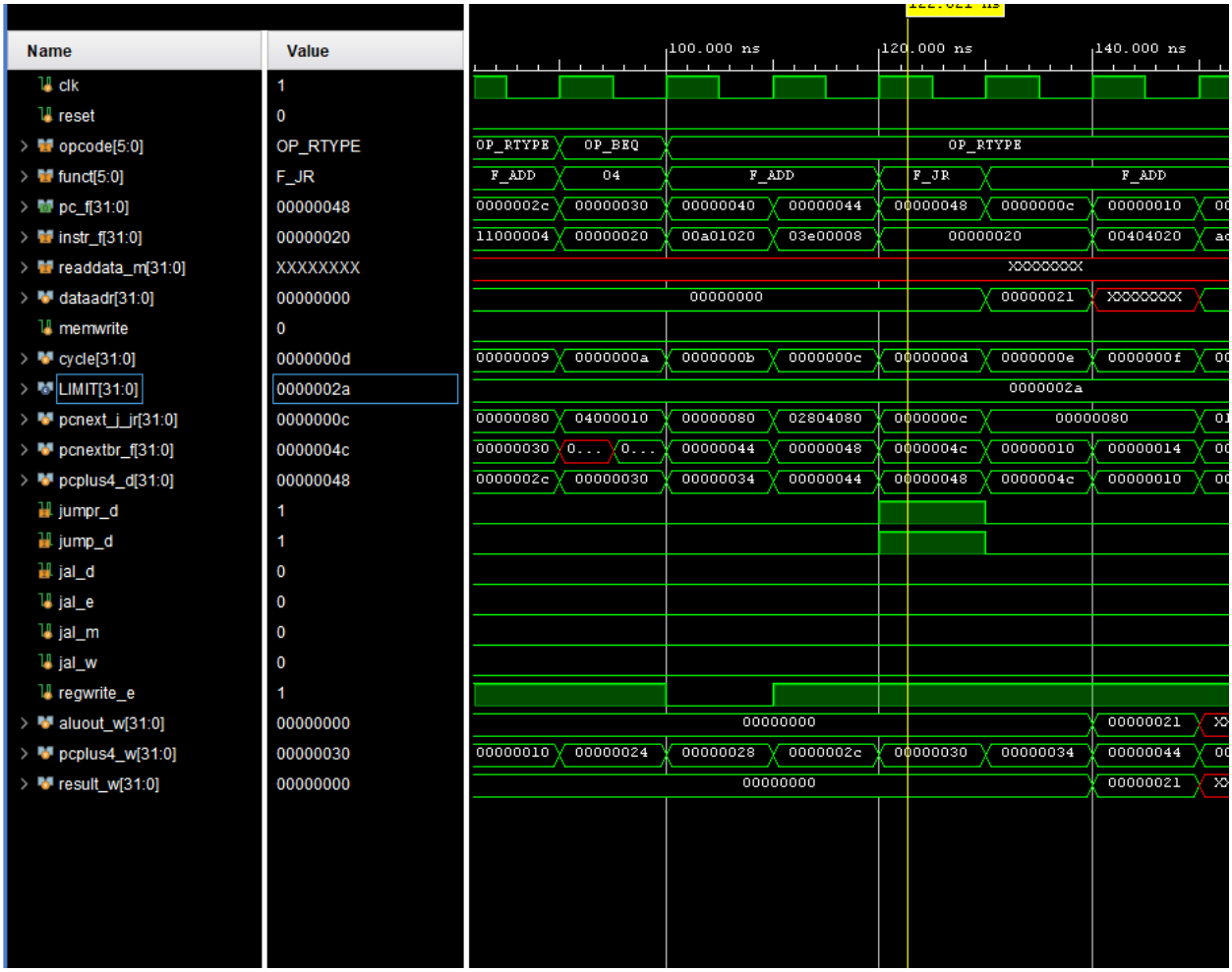

Figure 3 simulation scope showing the jr instruction

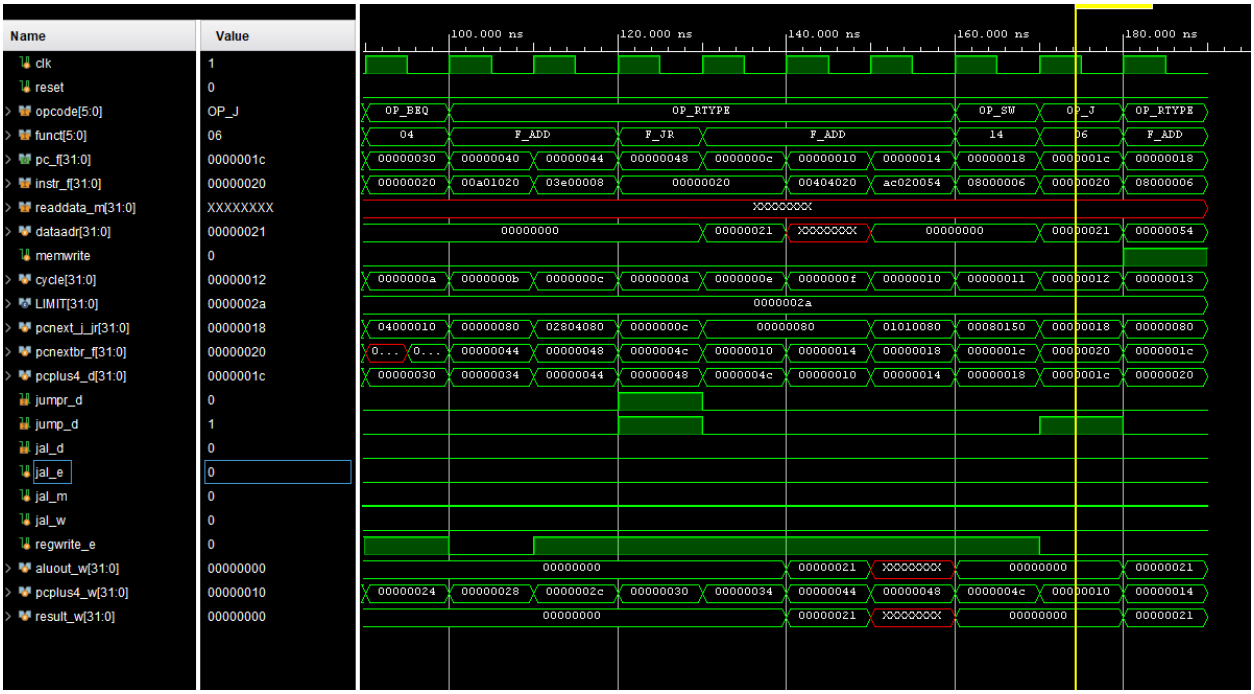

Figure 4 simulation scope showing the sw and jump instruction

The value 33 is written into memory. The test program was supposed to compute the largest value between 7 and 33 , return the largest value and store it in memory location 84.

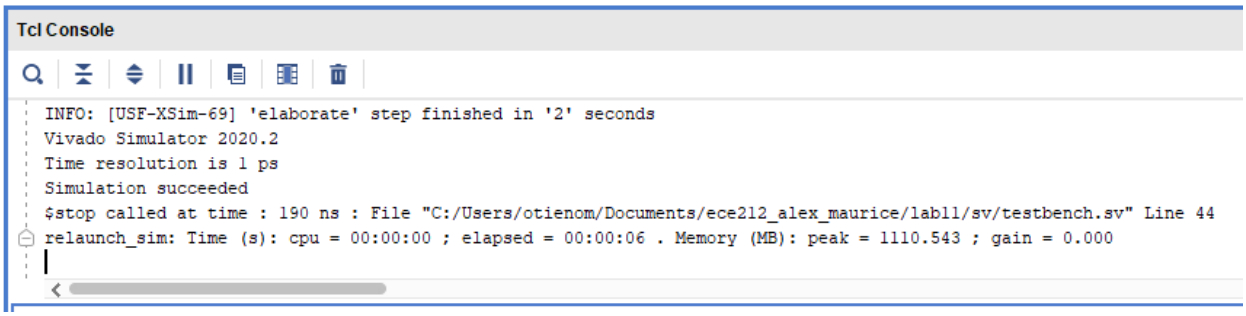

Figure 5 TCL console screenshot showing success of the jal_jr_test.dat

Regression test results

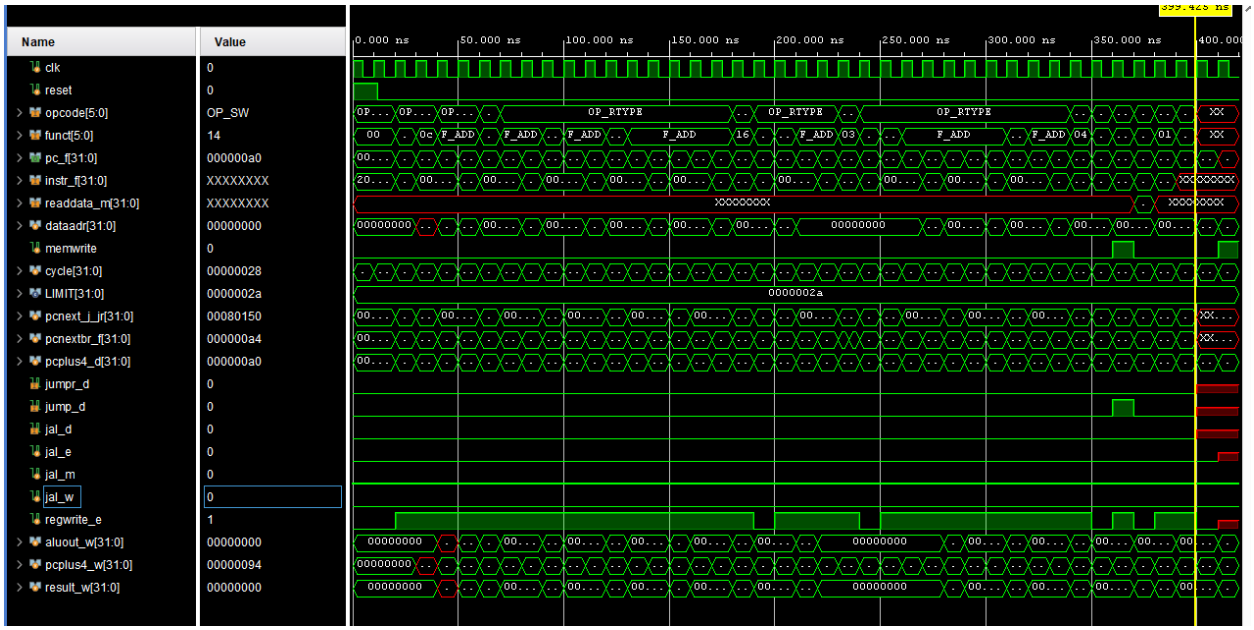

Figure 6 simulation scope for the regression test (test program lab11memfile.dat)

The regression test program runs successfully. The value 7 is written into memory location 84. This means that our changes did not break the processor.

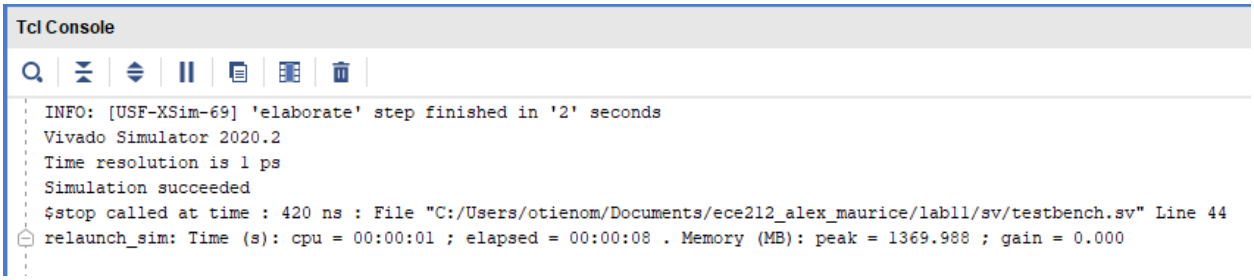

Figure 7 tcl console showing regression test success

Conclusion:

This was an exciting lab as the process of designing the changes to the Datapath required careful thinking. The biggest mistake we made was not specifying the register into which the value of pc+4 is written. Consequently, the program broke at the jr instruction. The knowledge acquired in this lab would be essential in designing the pipelined processor to handle even more instruction in the future.

Time spent on lab:

8 hrs