Comp-Arch: Lab 3
10/10/2018
Vicky McDermott
Minju Kang
Allison Basore

CompArch Lab 4 - Uploading single cycle CPU to FPGA

For Lab 4, our group decided to upload the single cycle CPU to FPGA to simulate our implemented CPU from Lab 3 to analyze on the memory usage and performance. Below is the block diagram for the implemented single cycle CPU. Our verilog implementation contains both behavioral and structural verilog but our tests are written purely with behavioral verilog.

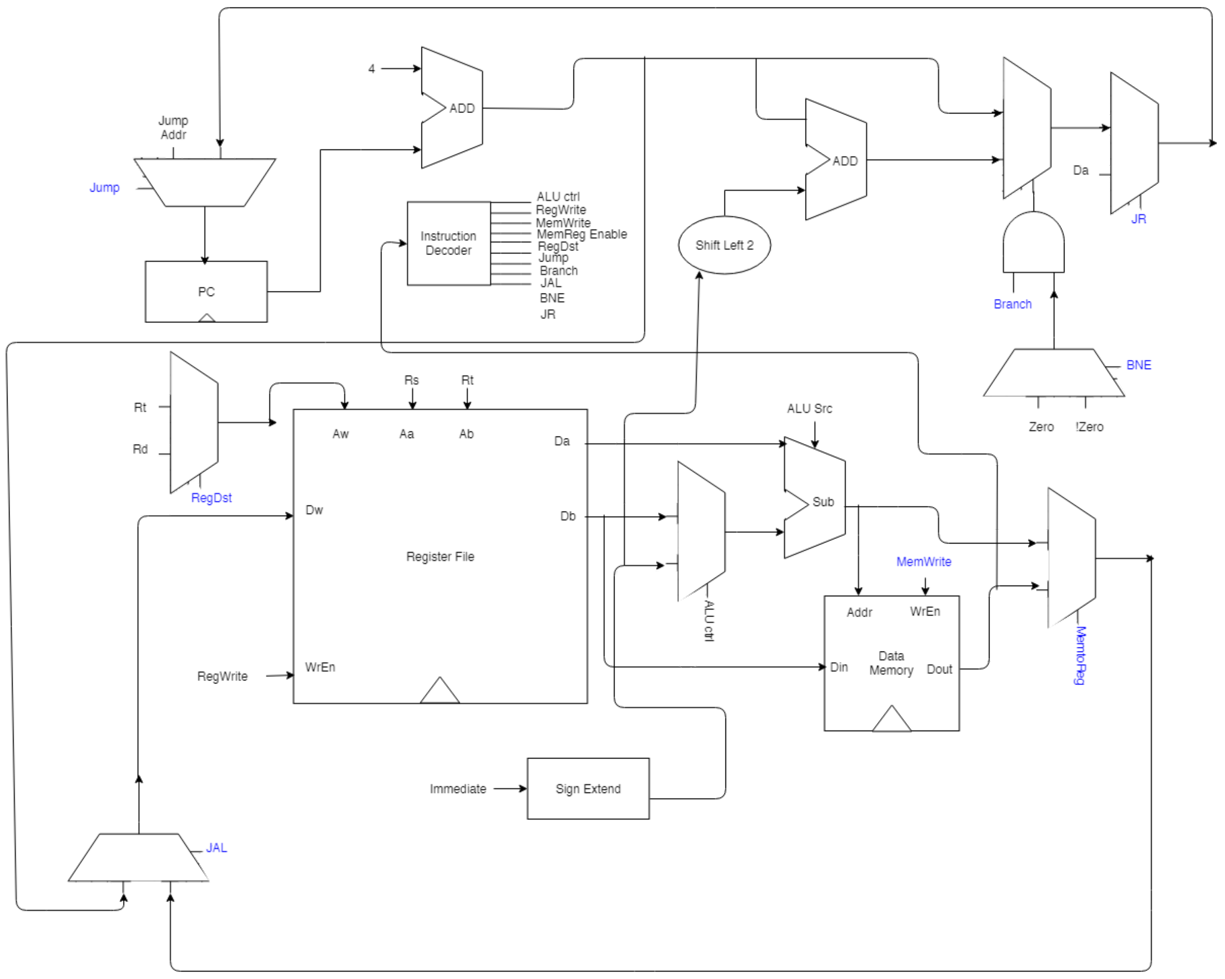

Figure 1: Block Diagram of implemented CPU

Interaction with FPGA

We wanted to test the FPGA's performance with manual inputs. When button 0 is pressed, the current values of the switches are read as the "operation mode". When button 1 is pressed, the current values of the switches are read to determine the least 4 significant digits of the immediate value to give to the CPU. Pressing button 2 displays the value of the immediate that was set with the switches and pressing button 3 displays the least 4 significant digits of the output coming from the CPU on the LEDs.

Operation Mode	Switch Input	6 Bits Binary Control Code	Function values
ADD	0000	000000	h20
ADDI	0001	001000	Not R type (I)
SUB	0010	000000	h22
SLT	0100	000000	h2a
XORI	0000	001110	Not R type (I)
BNE	0011	000101	Not R type (I)
BEQ	0111	000100	Not R type (I)
JAL	1111	000011	Not R type (J)
JR	1000	000000	b001000
J	1100	000010	Not R type (J)
SW	1110	101011	Not R type (I)
LW	0101	100011	Not R type (I)

Table 1: Operation Mode based on the switch input

For example, if you want to test ADDI, you have to preload corresponding assembly test onto FPGA memory first. In that way, the instruction decoder doesn't uses the memory block. Then, you set the immediate values you want to add with pre-set integer (set in assembly code) and op function (reference the table above. In this case, switch input will be 0001). Once you threw all values in, pressing button 3 will display the result.

In our modified CPU, we set the immediate value and op funct manually, instead of having the whole instruction include both.

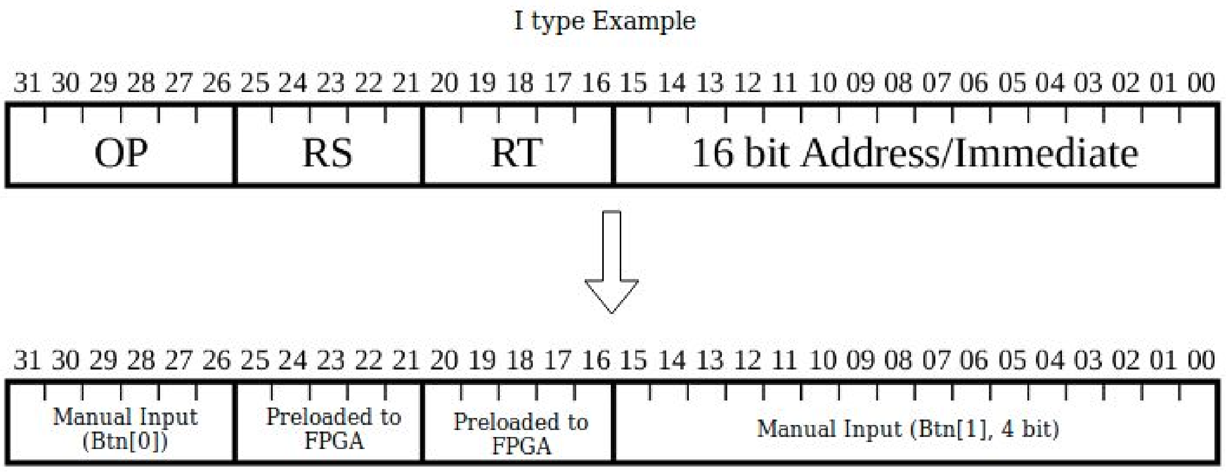

Figure 2: representation of I type instruction into the structure of how FPGA recognize

Instead of having 32 bits long single instruction, we have manual inputs incorporated. For first 6 bits, we have operation codes for each function (Table 1) read from switch inputs of FPGA. Then, Rs and Rt are preset by the uploaded assembly test. The 16 bit Address/Immediate are manual input set by button 1. This way of implementation has some degree of limitation. We are not fully flexible in setting Rs and Rt. They can only be accessed through assembly codes that we upload to FPGA. In other words, if we want to change the value of what we operate (e.g. change the value of integer for ADDI), we have to re-upload the assembly codes to FPGA).

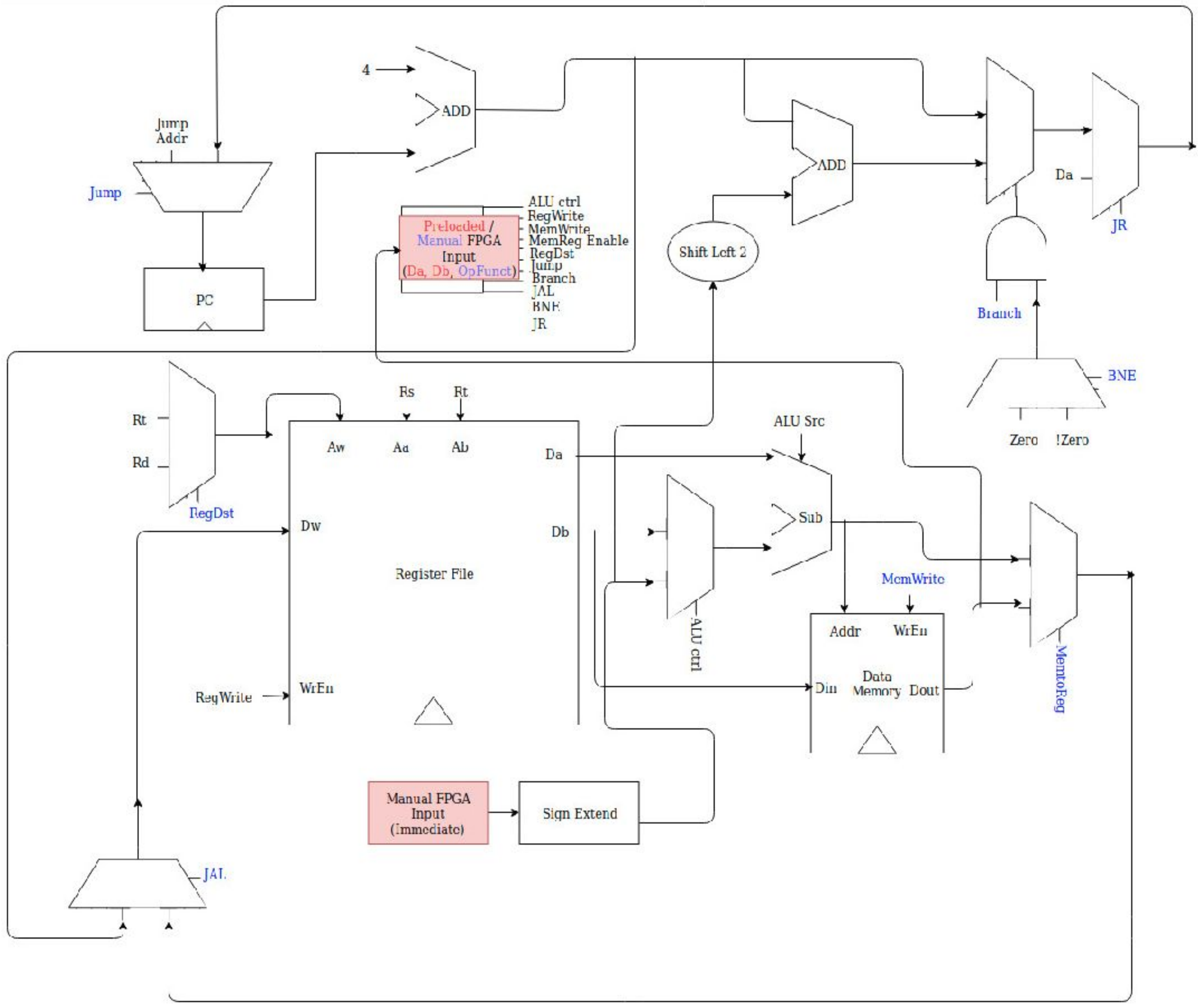

Figure 3: Proposed architecture of modified CPU to be uploaded on FPGA

Work summary

First, since we did not have fully functional CPU verilog implementation for lab 3, we had to make sure that our CPU passed all of our assembly test. We found several errors including that we were calling a non-existing assembly register and we had switched the inputs and the outputs. After fixing all these errors, our CPU passed all of our assembly tests. Our next step was to test all pieces of our CPU individually on the FPGA (e.g. data memory - see video linked here). Based on written tests for each functionality, we planned and proposed the master test that tests all functionality of CPU on FPGA. However, we encountered several challenges. The proposed CPU design had to be modified to be uploaded onto FPGA so that CPU.v takes manual inputs from FPGA. CPU verilog module originally took clock and reset signal as its input but we modified it to take *opmode, functval, immediate,* and *leddisplay*.

Therefore, CPU verilog module no longer takes inputs from pre-written assembly test, but the manual inputs of FPGA.

Challenges

During this lab, we encountered several challenges including learning how to load assembly tests to the FPGA, determining how to refactor our code to ensure that our CPU module was completely functional, and dealing with two conflicting inputs to CPU. Having a completely functional CPU module could be achieved with the instructor's help. We had to go over trial and error for loading assembly test to FPGA and started with writing small tests and then build up from there. In the process, we realized that we have conflicting inputs for verilog modules of CPU and FPGA's CPU. We could overcome such challenge by modifying our original verilog CPU.

Conclusion

Therefore, to overcome our limitations, we will try to explore many options that integrate an enhanced CPU with FPGA with attachable accessories for the next lab. We're looking forward to incorporating Pmods components so that we can have more switch inputs and LED outputs to test our CPU more thoroughly and overcome our limitation. For this lab, we did not have plans set up other than test plan due to the lab's flexibility. We were a bit disorganized about executing our plans (Test Plan) without having any written assignment and for the next lab, we will come up with a written lab plan and then organize ourselves better.