Pipelined CPU - Build Instructions

Shivam Desai, Kris Groth, Sarah Strohkorb, Eric Westman

Work Plan Reflection:

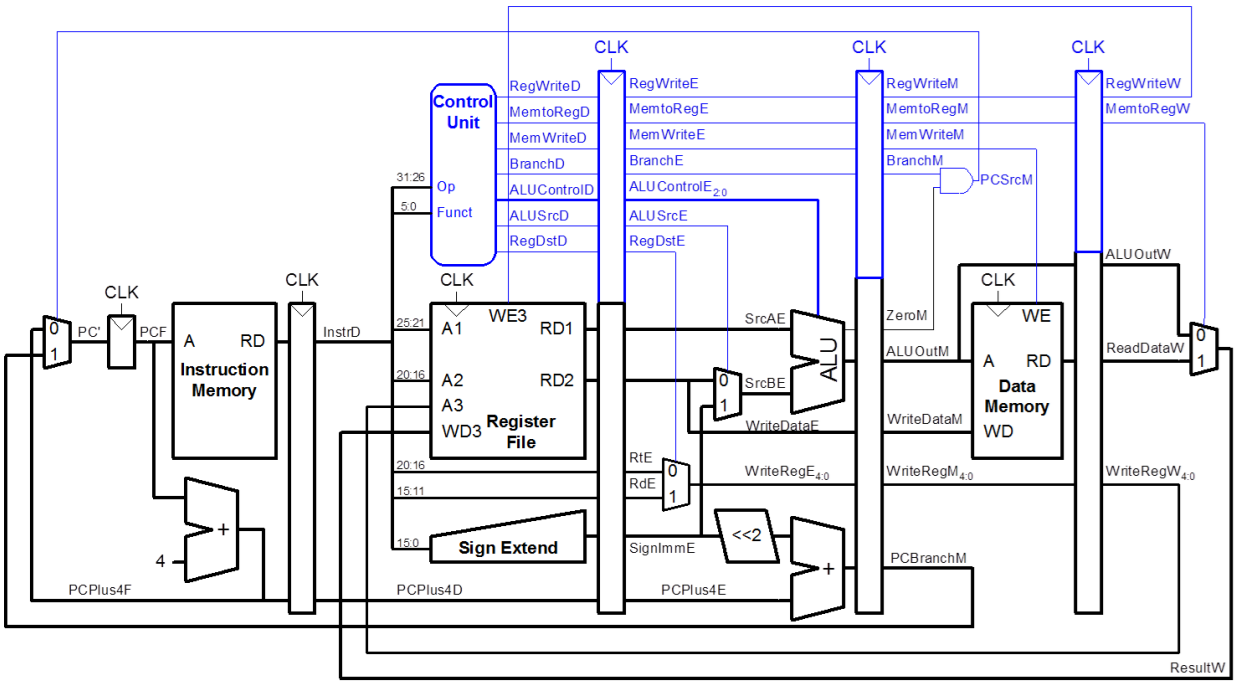
Our goal for the project was to build a pipelined CPU that handled at least one hazard and process more instructions than we have been doing for the labs. We were not able to get the pipelined CPU to be fully operational but most of the instructions work and the other instructions work in isolation.

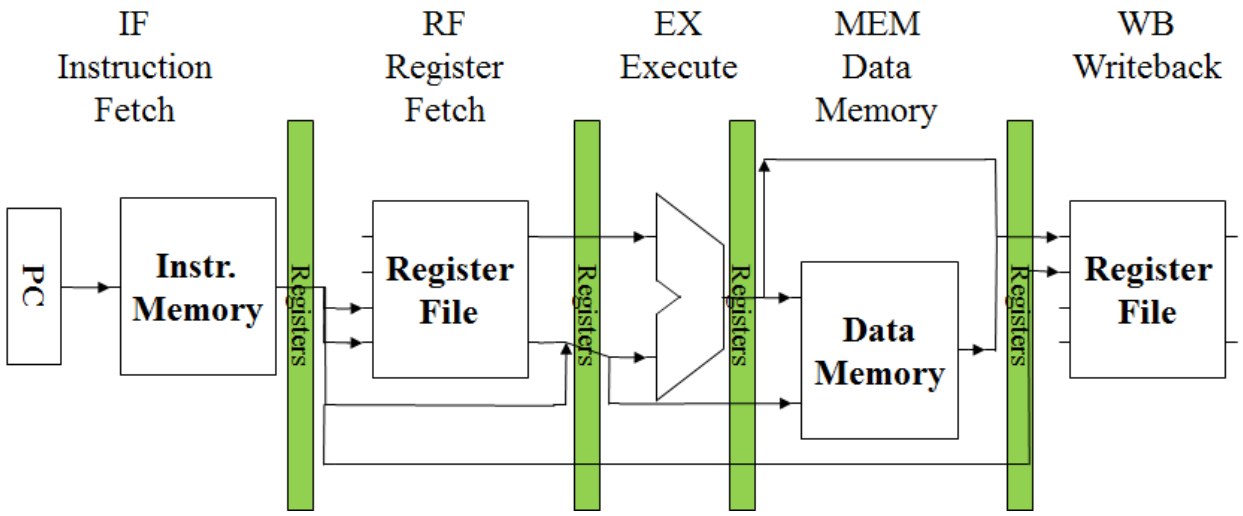
We made slight modifications to our schedule as we came across many problems implementing the pipeline. We decided to implement a pipelined format then add instructions as we progressed instead of trying to complete the entire thing in one pass.

To Do:

- Check the JAL and JR functionality
- Check the Forwarding logic to make sure it is implemented correctly, especially with the storing word instructions
- Redesign and implement a system to handle jumping and branching

Schematic:





Build Instructions:

How to compile Assembly program using Mars4_4 to machine code:

1. Compile/assemble the Assembly program (F3)
2. Dump the file to memory (File -> Dump Memory or Control + D)
3. Change dump format to Binary Text
4. Save file

How to run a program in machine code on our pipelined CPU using ModelSim:

1. Make sure that the line
   $readmemb(FILENAME, Memory)
   uses the correct filename, such as “example.dat” and that the path is reference correctly if the machine code file is not in the same directory as the verilog file.
2. Open ModelSim. If you don’t have a project for this CPU, then make one. Add the files “pipelinedCPU.v” and “pipelinedCPU.do” to the project, and make sure the .dat file is in the same directory.
3. Open the project. Type “do pipelinedCPU.do” into the command line and press enter.
4. This will run the program in the machine code file using the pipelined CPU. You may need to manually set the run time to a convenient time for the program.

Gotcha’s, watchouts:

- Debugging will take longer than expected. Always.
- Get comfortable with the ins and outs of Verilog
- Addressing memory may be different in your implementation of the CPU as compared to MIPS Assembly if you use 32-bit words instead of 8-bit words
- Design your hazard avoidance systems before you look at the code and try to implement

it, draw it out first and discuss it before you go to implement in the code