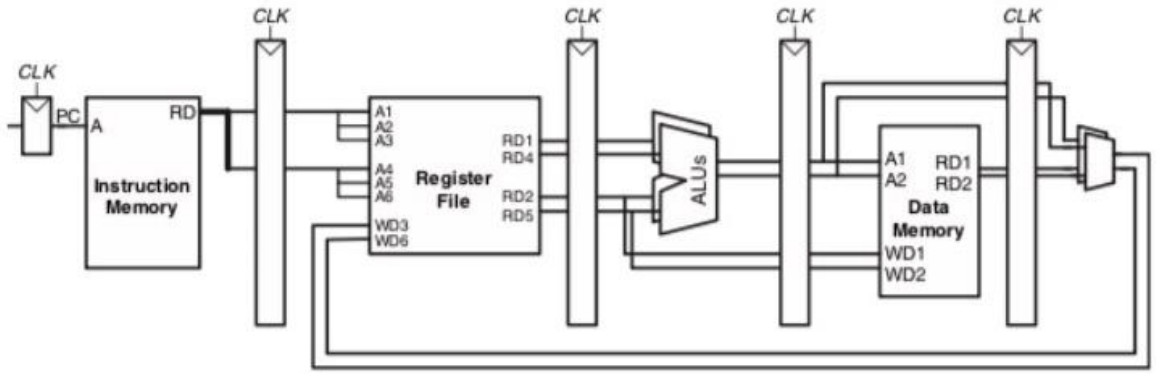
Tyler Gardenhire
Darryl Derico
CPE 404

Superscalar Project Proposal

Goal:

For our final project, we plan to modify our MIPS pipelined processor with the superscalar advanced technique implemented. The picture below is an example of a two-way superscalar datapath. This will allow two instructions to be fetched and executed on each clock cycle, meaning that adding duplicate hardware and functionality to existing hardware should, in theory, half the execution time. Both the register file and data memory will require double the ports. In addition, there will be two ALU's instead of one. Essentially, an n-way superscalar processor will require nearly n times the parts to make a single pipelined processor. For each additional way, extra hardware will be used, and as a result, the power required will be substantially increased.



Implementation:

The same five registers that were used in the pipelined processor should stay the same, but each wire that feeds into the register should be duplicated to allow for the second set of instructions to be ran. Additionally, the wires from the control unit will need to be duplicated as well. There will be a PC wire that feeds into instruction memory and will fetch two sets of instructions (PC and PC+4) to be used in the next register. In the decode stage, one register file will still be used, but every input and output will be duplicated to allow for the second instruction to be decoded at the same time. The execute stage will now need two ALU's (which is quite a bit of added hardware and power) to compute the two instructions, and the data memory file in the memory stage will need added functionality like the register file. Finally, in the write back stage a multiplexer will be added to see if each instruction will end up writing to memory. The schematic shows two 2:1 multiplexers, but we could use a single 4:1 multiplexer if the hardware is more efficient.

Challenges and Possible Solutions:

The biggest problem with building a superscalar processor is the hazards that come with executing multiple instructions at the same time. Resembling when we had to add a hazard unit to our pipelined processor, we must account for branches and data hazards, such as data dependencies. To do this, we will use the existing hazard unit, duplicate each wire to account for

both instructions being executed, and add more code that will stall the registers if one instruction depends on the other one being executed first. This will slow down our processor and the execution time will increase but having the right output is more important than speed.

Simulation and Emulation:

Building off our pipelined processor code in Verilog, a new naming convention for almost every wire will be needed. The use of letter differentiation such as A for the first instruction loaded and B for the second could be used. For example, we could use AaluoutE or writedataMB. Alternatively, if there is confusion from this, using numbers in place of or in conjunction with A and B would be acceptable. There is probably a way to instantiate two versions of the pipelined processor code which would save time instead of writing hundreds of lines of extra code, but for now, the main goal is having an idea of where to start, and how to complete it. Optimization is a secondary priority.

To complete a large project such as this one, breaking up the project up into smaller portions and debugging each would be the most efficient route. As with the pipelined processor, starting with the hazard unit and using test vectors to see if the superscalar portion of the unit is working properly. Ideally, duplicating all control unit wires one at a time and seeing if they are feeding and processing into each stage (fetch, decode, execute, memory, writeback) correctly would be the next goal. Once finished debugging every piece and ensuring they work correctly on their own, we will combine them all to create the final superscalar processor. This will be tested using the memfile.dat file that has been used for the last few projects. Finally, the goal will be to calculate the overall execution time, then compare it with both the stand-alone pipelined processor execution time as well as the execution time of a standard single cycle processor. The execution times should resemble a similar pattern to the chart provided below.

Once simulation is completed, the focus of the final project will be on emulating it on the DE2-115 FPGA board and setting up the necessary pins and buttons required including a button to control the reset signal. How the LEDs, screens, and other assets available on the DE2 will be utilized is a topic that still needs to be discussed and is of low priority until simulation is completed. For starters, the LEDs might be used to display the output of the memfile.dat in binary. If time allows, one of the buttons may be used to act as a clock cycle per press, the LCD screen could show current instruction performed, and the 7-segment displays could be used to show current clock cycle in either decimal or hexadecimal.