Armando Moncada - moncadaa@mit.edu - Github: moncadaa
Yeabsira Hawaz - yhawaz@mit.edu - Github: Yhawaz
Orion Li - haoxuanli@mit.edu - Github: OrionLi545

Our first goal is to make the pipelined processor superscalar. We plan to approach this stage by stage. First in Fetch, we'll attempt to process 2 instructions simultaneously, unless the cache boundary makes it too difficult. We plan to just use a Superscalar FIFO so we can enqueue and dequeue two instructions at a time. Next is Decode, first we deq both instructions. We'll also need to double our scoreboard and register files, since we need to duplicate the decode logic. Then if the 2 instructions end up having a dependency, then we'd partially stall(only dequeue one of the two instructions in Decode. In Execute we will take a similar approach to fetch, where if two instructions are difficult to handle, such as 2 branches, or 2 memory instructions we will handle it each instruction at a time. If it's a straightforward pair of instructions, such as arithmetic, we will execute them both at the same time. Then for write back, we just increase the ports on all the epochs we use to make it more doable to write 2 instructions in at a time.

On top of superscalar, another control flow enhancement we would like to explore is vector machine. Specifically, we are interested in implementing a modern vector machine that exceeds the basic functionalities of an MMX vector processing unit. We plan to build a vector machine that is capable to carry out vector ALU operations, shuffles, as well as explore the possibilities of doing strided loads and stores. Considering the effect of loading large vectors from the caches, we are simultaneously interested in combining our vector machine implementation with different cache architectures (such as multiple ways) and examine how much performance improvement this can bring upon.

Another 2 goals we will try to tackle are Branch Prediction, along with some basic FPGA analysis. The main goal will be to reduce annulments through an improved prediction model other than pc + 4. Implementations will include a Branch Prediction Buffer, Branch History Table, and Return Address Stack. We will look into different ways to train these data structures to add/remove branches in an optimal manner. As for the FPGA analysis, we already have access to an RealDigital Urbana board. This board has a Xilinx Spartan-7 FPGA and many useful breakout interfaces that can be used for interacting with the FPGA. Throughout our development of the processor, we can document changes in timing, area, and resource usage.

Branch Prediction Weekly Goals:
- Week of April 19: Get a current processor implementation working on an FPGA, test with some basic programs and possibly MMIO. Try to integrate some of the provided BTB and BHT code.
- Week of April 26: Choose to use BTB or BHT based on FPGA metrics. Start implementing Return Address Stack.
- Week of May 3: Finish implementing the full Branch Prediction system, and have a full suite of tests for the FPGA to demonstrate Processor capability.
- Week of May 10: Finalize design and Presentation.

VM Weekly Goals:
Week of April 19: Implement basic SIMD instructions and a separate vector processing unit, adapting to
RISC-V architecture
Week of April 26: Implement shuffle as well as multiple caches
Week of May 3: Try to implement strided loads and stores. Combine the caches to test out the performances
Week of May 10: Finalize design and presentation

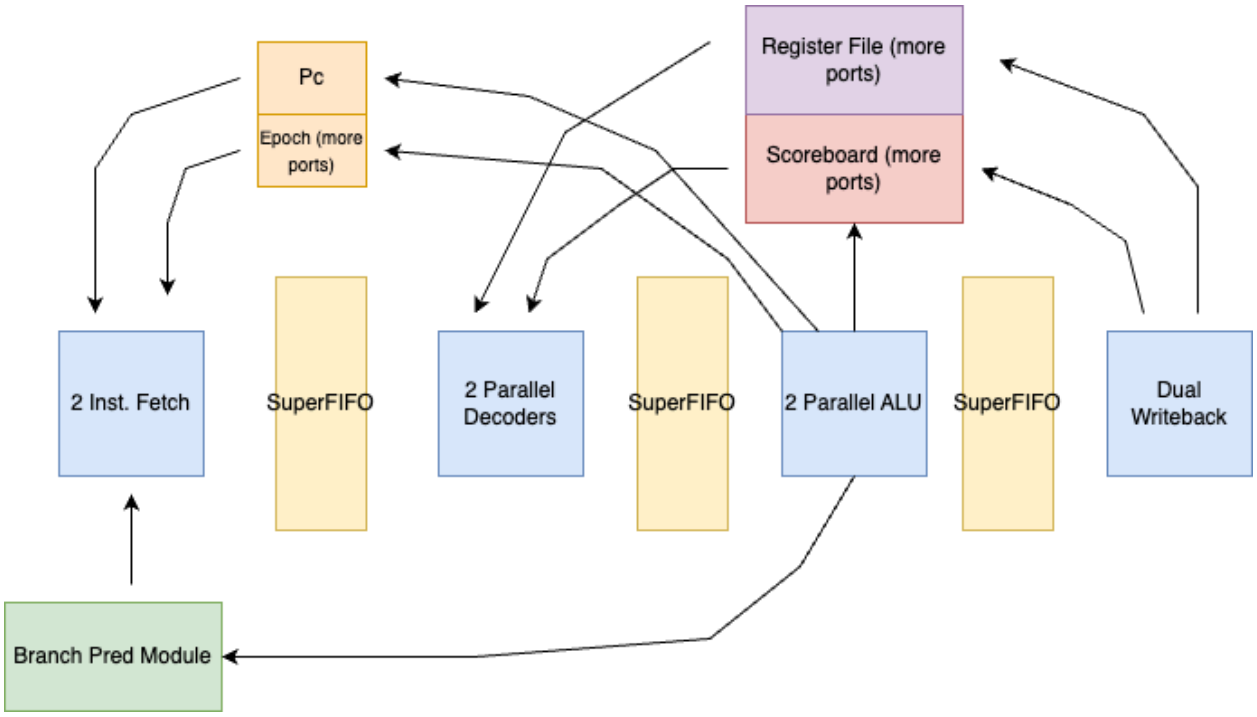

Figure 1: A rough diagram of our Processor with superscalar and branch prediction