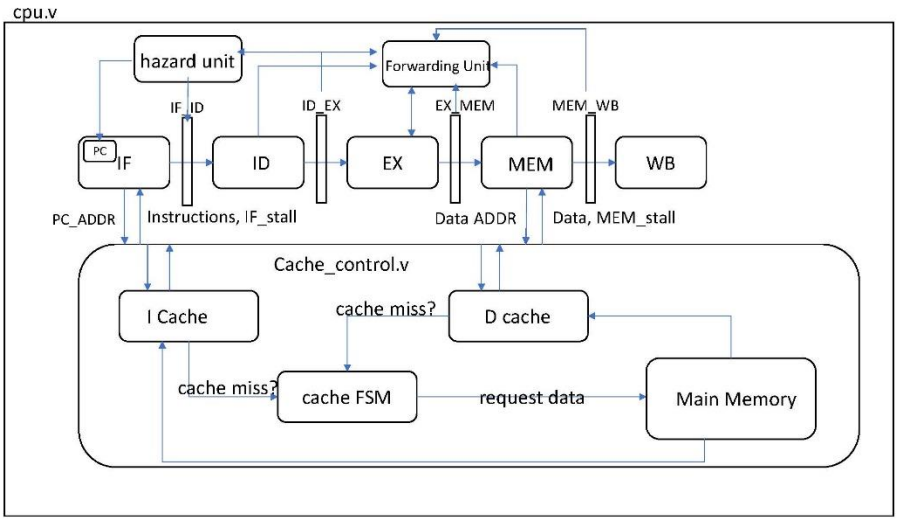
ECE/CS 552 Fall 2018: Final Project Report

1. Overview

a) Brief description and introduction to project design

In the first phase, we implemented the WISC-F18 ISA and designed a single cycle processor. On each clock cycle, the processor first read from the instruction memory, executed the instructions and store the results. Then the processor decides which PC address to use and fetch the next instruction. The compute instructions, control instructions and memory instructions are mainly implemented in this phase. Also, reset sequence and flags registers are implemented to help the processor work correctly. In the second phase, we designed a 5-stage pipelined processor to implement the WISC-F18 ISA based on our work in phase one. We first divided the single cycle processor into 5 stages by adding essential registers, then implemented the control blocks for control and data hazard detection which handles all dependencies. We modified control and data path for full data forwarding, including ex-to-ex, mem-to-ex and mem-to-mem forwarding and register file bypassing. The stalls, flushes and HLT signals are set during this phase. In the final phase, we replaced instructions memory and data memory with I cache, D cache and main memory with 5-stage pipeline in phase 2. We used cache control module as the interface with our pipeline and instantiated one I cache, one D cache, one cache FSM and one main memory inside it. Cache reads and writes are implemented following the cache read and cache write policies. Cache miss handler FSM was implemented to handle instruction or data misses in order and output the stall signal to stall the whole pipeline. Memory contention on cache miss is not well handled in our phase 3.



2. Responsibility/Task Breakdown

	Phase 1	Phase 2	Phase 3	Percentage
Shubham	**Memory instructions and control instructions implementation;**	**Schematic design, pipeline latches and Stall/Flush implementation;**	**Cache control, cache fill FSM and pipeline merging implementation;**	**34%**
Arushi	**Compute instructions, ALU implementations and Flags implementation;**	**Hazard and Forwarding Units implementation;**	**Cache control, I cache and D cache implementation;**	**33%**
Yucheng	**Top level CPU module design;**	**Hazard and Forwarding Units implementation;**	**Cache control and LRU implementation;**	**33%**

3. Special features

Tried to avoid stalling the entire pipeline while cache misses were being handled, but due to added complexity from forwarding logic, this approach was abandoned.

4. Completeness

Phase 1: passed testcase 1 without testing testcase 2 and testcase 3;

Phase 2: passed all 3 testcases; full control and data forwarding units are implemented and tested; stall and flush are implemented and flush works fine, however, load to use stalls are buggy; HLT and reset sequences are implemented and tested;

Phase 3: passed testcase 1 and testcase 3, but failed testcase 2 and testcase 4 with infinite loops; cache writes and reads are implemented and tested following cache writes and reads policies; data cache miss and instruction cache miss are handled correctly when separated; memory contention on cache misses are not handled which led to testcases failed;

After demo: during demo testcase 3 was not completing correctly due to stall logic, hence we changed it to stall the entire pipeline to ensure forwarding logic functions correctly before write-back.

5. Testing

The methodology we used to test the correctness of our design is to test all the testcases provided and derived the correct outcome of each testcase file. Then we compared the difference between our implementation and the expected outcome. We compared the signals which determined the errors to find out the units and modules which not work correctly.

Initial testing of individual components was performed using testbenches, however,

several changes were made when putting everything together in the top-level module. Due to time constraints we did not update our testbenches but rather tested the top-level module on a cycle by cycle basis. Necessary intermediate instructions, for instance, metadata array signals passed through to the top-level module were added to aid inspection of signals and later removed from the final implementations.

6. Results

	Success	Cycle Time	Notes
Testcase 1	Y	42	
Testcase 2	N	Infinite loop	Memory contention causes the instruction cache to be incorrectly loaded with data that should be loaded in data cache, resulting in unknown values which then propagate through the pipeline resulting in an infinite loop.
Testcase 3	Y	52	
Testcase 4	N	Infinite loop	Same issue as testcase 2