CS/ECE 552 Spring 2020

Extra Credit Write-up

Team-12

Qinjun Jiang, Ruokai Yin

We have implemented the following extra credit options:

- Branching Decisions in Decode
- Additional Forwarding Paths (EX-D, MEM-MEM)
- Exception Handling
- LRU Cache Replacement
- Synthesis (partial)

The directory structure of our submission is as follow:

The root directory is “extra-credit”, inside it there are

1. verilog-ec: all verilog files required and their vcheck files
2. verification-ec:
    a. Branch-Decode
    b. Exceptions
    c. Forwarding-paths
    d. LRU
    e. Synth
    f. baseline-summary

Each containing files such as tests, summary log, and other files related. All test results are in the pdf document as screenshots. Baseline-summary is the folder that contains all summary log files indicating that our extra-credit design is still able to pass all demo3 tests.

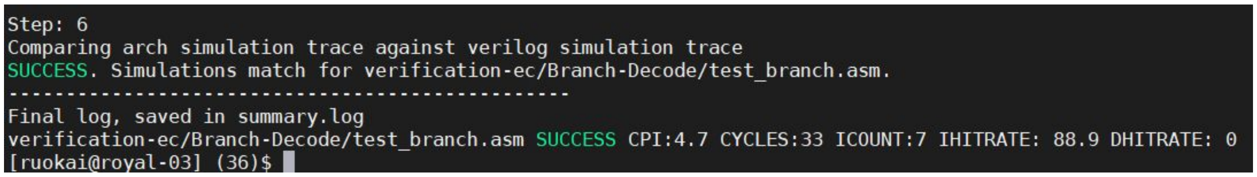
3. ExtraCreditWriteUp.pdf

1. Branching Decisions in Decode

This requires adding address calculation and condition evaluation logic in Decode. It also requires one more forwarding path (EX-D) and two more stalling situations (data dependency of Branch/JR/JALR on previous instruction, data dependency of Branch/JR/JALR on the load before previous instruction). Meanwhile, it reduces the number of cycles to stall because of incorrect branch prediction down to 1.

Since we already implemented this in phase2, we can not do a comparison to test the performance. But the overall CPI in final tests should show the benefits.

Our own test result:



```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for verification-ec/Branch-Decode/test_branch.asm.
------------------------------------------------
Final log, saved in summary.log
verification-ec/Branch-Decode/test_branch.asm SUCCESS CPI:4.7 CYCLES:33 ICOUNT:7 IHITRATE: 88.9 DHITRATE: 0
[ruokai@royal-03] (36)$
```

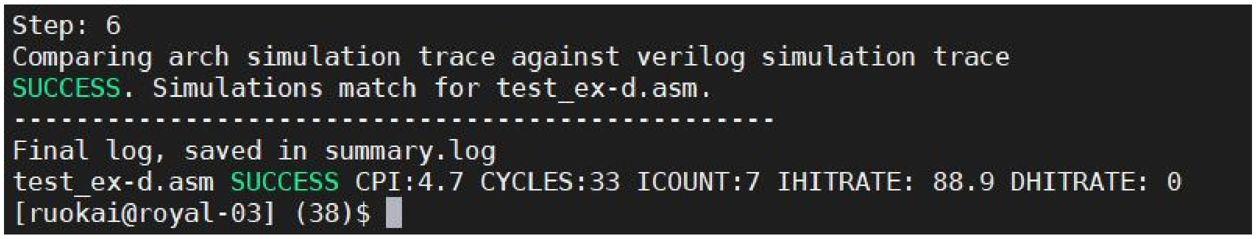
2. Additional Forwarding Paths

a) EX-D

Since we have made branching decisions in Decode as described above, EX-D forwarding will save one cycle of stall if Branch/JR/JALR instructions are dependent on the previous instruction. The overhead is to add logic in the Decode stage to choose data from either register file or EX/MEM register.

Since we already implemented this in phase3, we can not do a comparison to test the performance. But the overall CPI in final tests should show the benefits.

Our own test result:



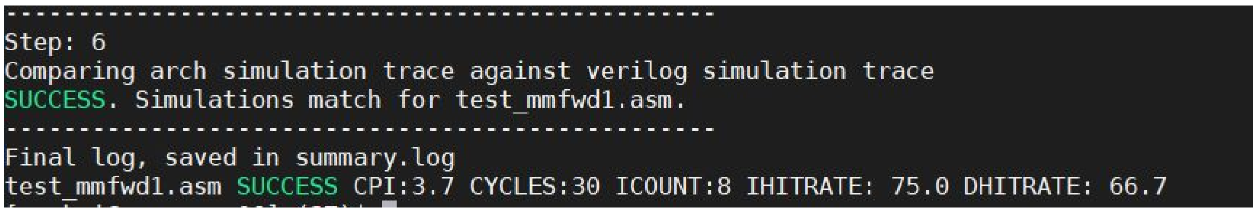
```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for test_ex-d.asm.
------------------------------------------------
Final log, saved in summary.log
test_ex-d.asm SUCCESS CPI:4.7 CYCLES:33 ICOUNT:7 IHITRATE: 88.9 DHITRATE: 0
[ruokai@royal-03] (38)$
```

b) MEM-MEM

This forwarding path is specifically for store after load on the same register situation. It helps save 2 stalling cycles because in our previous version the store instruction needs to stall at Decode for 2 cycles to get the value to store in memory. The overhead is to add logic in the Memory stage to choose data from either EX/MEM register or MEM/WB register.

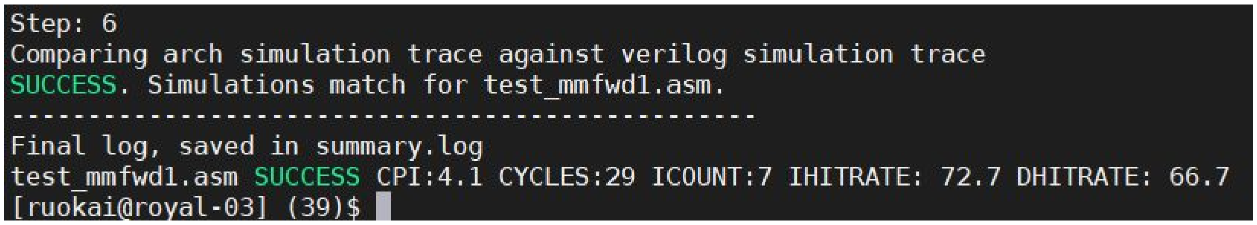
The benefit can be proved by the decreased cycle count when running a test that contains the store after load. However, in the test bench, the instruction count is based on the condition of Halt/RegWrt/MemWrt. Also, as the WB stage of the load (has RegWrt signaled) and Memory stage of the store (has MemWrt signaled) would happen simultaneously because of the MEM-MEM forwarding, the instruction count will decrease by one and thus results in increased CPI. We have tried to change the connection of MemWrt and RegWrt signals in the test bench, but it turned out that there is no way to solve the problem based on the current testbench. You may consider changing the way of instruction counting for next semester. The order of printing test log information could also mistakenly report a correct implementation as an error.

Before:



```
------------------------------------------------
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for test_mmfwd1.asm.
------------------------------------------------
Final log, saved in summary.log
test_mmfwd1.asm SUCCESS CPI:3.7 CYCLES:30 ICOUNT:8 IHITRATE: 75.0 DHITRATE: 66.7
```

After:

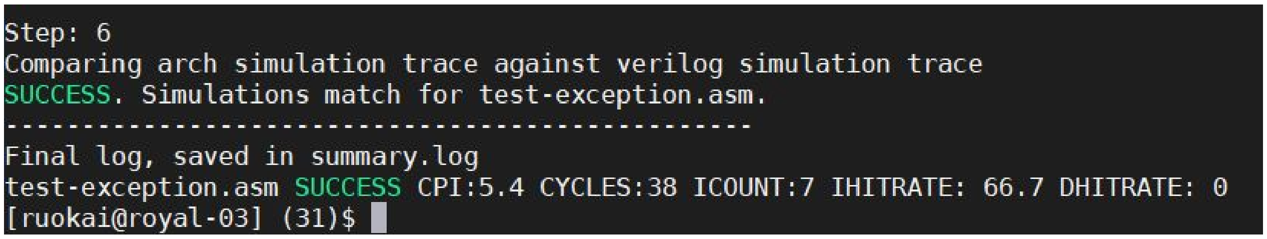


```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for test_mmfwd1.asm.
------------------------------------------------
Final log, saved in summary.log
test_mmfwd1.asm SUCCESS CPI:4.1 CYCLES:29 ICOUNT:7 IHITRATE: 72.7 DHITRATE: 66.7
[ruokai@royal-03] (39)$
```

3. Exception Handling

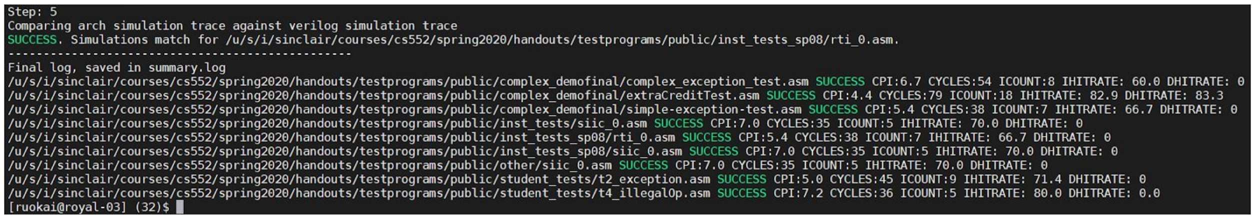
We implemented exception handling as described in the project ISA. The design can now pass all exception related tests. The overhead is to recognize the SIIC and RTI instructions in Decode, make an ETC register, and when the two instructions appear, stalling and flushing for one cycle to either save EPC and load 0x02 to PC or load EPC to PC.

Pass our own test:



```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for test-exception.asm.
--------------------------------------------------
Final log, saved in summary.log
test-exception.asm SUCCESS CPI:5.4 CYCLES:38 ICOUNT:7 IHITRATE: 66.7 DHITRATE: 0
[ruokai@royal-03] (31)$
```

Pass all tests in
/u/s/i/sinclair/public/html/courses/cs552/spring2020/handouts/testprograms/public/exceptions.list
:



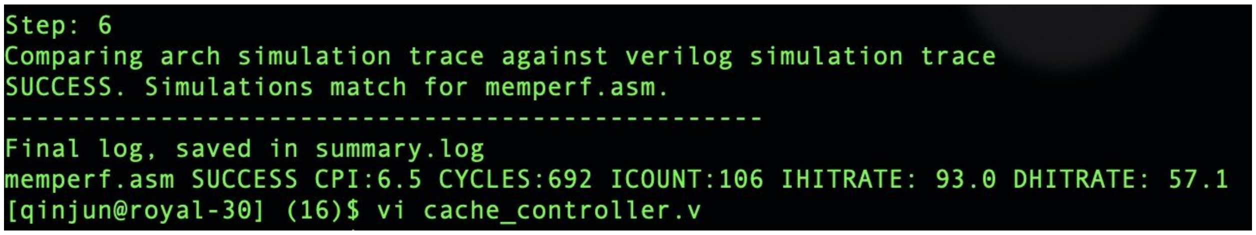
```
Step: 5
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for /u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/inst_tests_sp08/rti_0.asm.
--------------------------------------------------
Final log, saved in summary.log
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/complex_demofinal/complex_exception_test.asm SUCCESS CPI:6.7 CYCLES:54 ICOUNT:8 IHITRATE: 60.0 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/complex_demofinal/extraCreditTest.asm SUCCESS CPI:4.4 CYCLES:79 ICOUNT:18 IHITRATE: 82.9 DHITRATE: 83.3
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/complex_demofinal/simple-exception-test.asm SUCCESS CPI:5.4 CYCLES:38 ICOUNT:7 IHITRATE: 66.7 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/inst_tests/siic_0.asm SUCCESS CPI:7.0 CYCLES:35 ICOUNT:5 IHITRATE: 70.0 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/inst_tests_sp08/rti_0.asm SUCCESS CPI:5.4 CYCLES:38 ICOUNT:7 IHITRATE: 66.7 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/inst_tests_sp08/siic_0.asm SUCCESS CPI:7.0 CYCLES:35 ICOUNT:5 IHITRATE: 70.0 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/other/siic_0.asm SUCCESS CPI:7.0 CYCLES:35 ICOUNT:5 IHITRATE: 70.0 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/student_tests/t2_exception.asm SUCCESS CPI:5.0 CYCLES:45 ICOUNT:9 IHITRATE: 71.4 DHITRATE: 0
/u/s/i/sinclair/courses/cs552/spring2020/handouts/testprograms/public/student_tests/t4_illegalOp.asm SUCCESS CPI:7.2 CYCLES:36 ICOUNT:5 IHITRATE: 80.0 DHITRATE: 0.0
[ruokai@royal-03] (32)$
```

4. Cache Replacement - Least Recently Used

We implemented LRU cache replacement policy by making a 256-bit register each bit for storing the most recently used cache line. Every time we need to evict a cache line, we choose the one that is not used most recently. Every time we access a cache line, we update the bit stored in the register. One thing worth of attention here is that the updates needed to be done in both hit and miss situations. This gives a large decrease in cycle numbers and improves CPI.

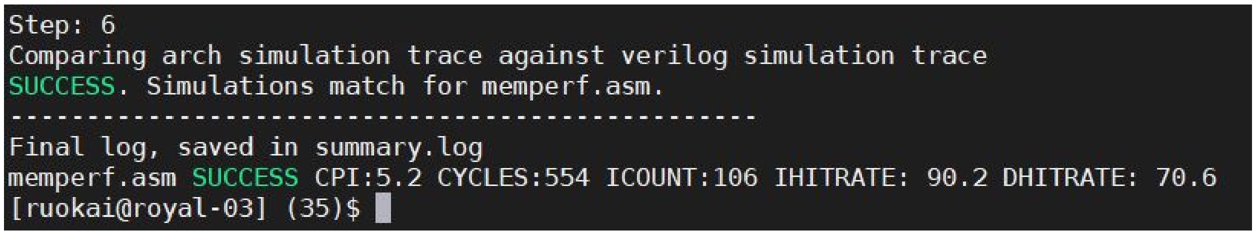
When comparing our design's performances on the designed test for LRU we can see that there is a significant improvement in CPI, Cycles and D-hitrate. About 140 cycles are reduced, while data memory hit rate increases for 13% percent.

Before:



```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for memperf.asm.
--------------------------------------------------
Final log, saved in summary.log
memperf.asm SUCCESS CPI:6.5 CYCLES:692 ICOUNT:106 IHITRATE: 93.0 DHITRATE: 57.1
[qinjun@royal-30] (16)$ vi cache_controller.v
```

After:



```
Step: 6
Comparing arch simulation trace against verilog simulation trace
SUCCESS. Simulations match for memperf.asm.
------------------------------------------------
Final log, saved in summary.log
memperf.asm SUCCESS CPI:5.2 CYCLES:554 ICOUNT:106 IHITRATE: 90.2 DHITRATE: 70.6
[ruokai@royal-03] (35)$
```

5. Synthesizing Your Design (partial)

We first follow the guidance on the website and successfully synthesis our design. One thing that is important is that the list.txt file should contain all the final_memory.syn.v and cmem.syn.v instead of the normal .v version.

All the results are in the /synth folder. Roughly speaking, our result on cell-area is around 1300000 and we violate the timing by -2 Slack.

For testing on post_synthesis, we have met some difficulties. At first, we did not choose the right files for running tests. By discussing them with Professor on piazza, we solve the problem by copying gscl45nm library and .v file to our repo. Then includes only the essential file for running the test, they are: proc_hier_pbench proc.syn.v proc_hier_pbench.v clkrst.v proc_hier.v gscl45nm.v. We can now run the tests. But we encountered some new errors. We solved one by adding time-scale `timescale1ns/10ps to pbench and clkrst file. Finally we failed on the “not a downward path” error. The error is induced by testbench cannot recognize the signal in our .syn.v file. We try to add the required signals as the output port in our proc.v file, as suggested by the professor. But this method does not solve the problem at last.

So for this part, we finally succeeded in finishing synthesis but failed in running the tests.