EECS 470 Term Project Report
Group 6: Branch Oracles

Yufan Yue, Haoxuan Zhu, Junyan Zhai, Andrea Bejarano, Jiayi Chen

I. Introduction

We successfully implemented a MIPS R10K-style 2-way superscalar out-of-order processor with a **clock period of 10.5 ns** and an average **CPI of 1.76**. Our processor includes several advanced features that enhance the system performance, including load-store processing and memory enhancements, as well as a more sophisticated dynamic branch predictor.

II. Processor Design

Our out-of-order pipeline is composed of 6 instruction stages: fetch, dispatch, issue, execute, complete and retire. A full top-level diagram of our processor and the data structure blocks and advanced features that were implemented is included in the appendix.

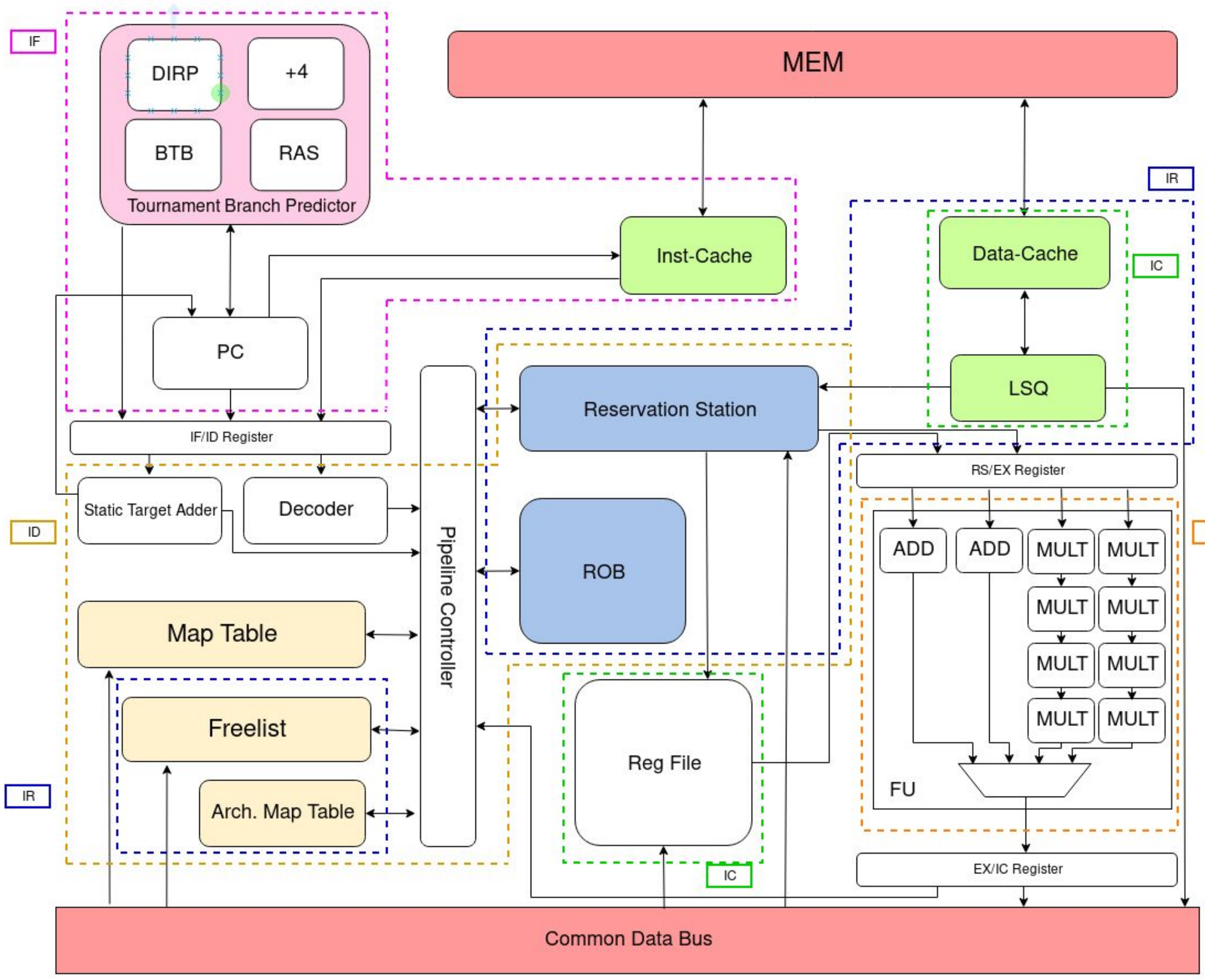

Figure 1. Top-level Diagram

Figure 1 is the top-level diagram of our 2-way OoO processor. It shows critical data flows among key modules. Not all signals and ports are shown in the diagram. Details will be discussed in the following sections.

Summary of Key Modules		
Module	**Entry Num.**	**Comments**
Reorder buffer	32	
Reservation station*	16	RS uses a mix of issue after tag broadcast and issue before tag broadcast to eliminate serial CAM operations.
Physical register file	64	Accessed by instructions at the complete stage.
Map table	32	Restored by the architectural map table during recovery.
Architectural map table	32	Updated when the ROB retires the head entry.
Freelist	32	Has a tag entry and a valid entry, which allows the processor to recover in one cycle. When a tag is being used by the ROB, the valid bit is set to 0. In the case of an exception or misprediction, it can avoid serial roll-back by setting all valid bits to 1 to recover.
Load/store queue*	16	Separate 8-entry LQ and 8-entry SQ.
Return address stack*	4	Predicts return address of JALR instructions.
Branch direction predictor*		Tournament predictor including a gshare predictor, a per-PC correlation predictor and a speculative chooser.
Branch target buffer	32	Partial tag comparison to reduce complexity.
Functional units*		Two integer adders, two 4-stage multipliers.
Data cache	32	4-way, 8 sets, with pLRU policy.
Instruction cache	32	Direct mapped cache.
* fully parameterized modules		

Data Structures:

The processor contains the characteristic elements of a MIPS R10K processor that enable out-of-order instruction execution, including reorder buffer (ROB), reservation station (RS), map table, architectural map table, freelist and a physical register file (PRF). Our design also includes separate load queue and store queues.

Functional Units:

The functional units consist of two integer ALUs and two pipelined multipliers. Integer ALUs are responsible for all integer addition and subtraction, while multipliers are for Integer multiplication. The multipliers are parametrized so that we could see how the number of pipeline stages affected our clock period and CPI. After this analysis, 4-stage multipliers are used in our final design.

Memory Structure:

A 4-way set associative cache with a block size of 8B was implemented for the data cache. This increases the complexity of the memory hardware, but decreases the miss-rate and the average memory access time (AMAT), which decreases the CPI. The instruction cache implemented is direct-mapped.

III. Advanced Features

Feature Summary

Features	**Value**	**Comments**
Automated Regression Testing	*	Automated testing using bash script.
2-way Superscalar	***	
Store-to-load forwarding in LSQ	*	Load value forwarded from store queue if same address detected by age logic.
Load issue out-of-order past pending stores (speculatively)	**	Roll back if memory order violation detected.
Write-back data cache	*	Write-back and write-allocate.
Associativity = 4 > 1	*	Associativity is parametrized. 4 is the best result after testing.
Tournament direction predictor w/BTB	*	Implemented gshare, per-PC correlated predictor and speculative chooser.
Return address stack	*	Size is parametrized.

Superscalar Design:

The superscalar processor can handle two parallel instructions at every stage of the pipeline. Data can be forwarded from the first to the second way if needed.

Load-store Processing Enhancements:

The LSQ supports store-to-load forwarding and speculative load issuing, meaning that they can be issued out-of-order past pending stores and roll back after memory order violation detected.

Memory Enhancements:

To improve the performance of the data cache, other than making the cache associative (discussed in Section II), it was also implemented with a write-back cache policy along with a write-through policy.

Branch Prediction:

To implement branch prediction, we used a branch target buffer (BTB), a return address stack (RAS) and a tournament predictor. The target PC is predicted either by the BTB, the RAS or by PC + 4. If there is a target misprediction, the target PC can be changed by the correct results from dispatch or retire.

The tournament predictor is composed of a global Gshare predictor and a PAs correlation predictor. The tournament predictor uses 6 bits of the PC to index the PAs branch history table, as well as to XOR the 6-bit global branch history register to index the Gshare branch history table. The entry of the corresponding BHT is used alongside 4 bits of the PC to index the PAs pattern history table. The direction predictions from the Gshare and tournament predictors are selected according to the chooser FSM.

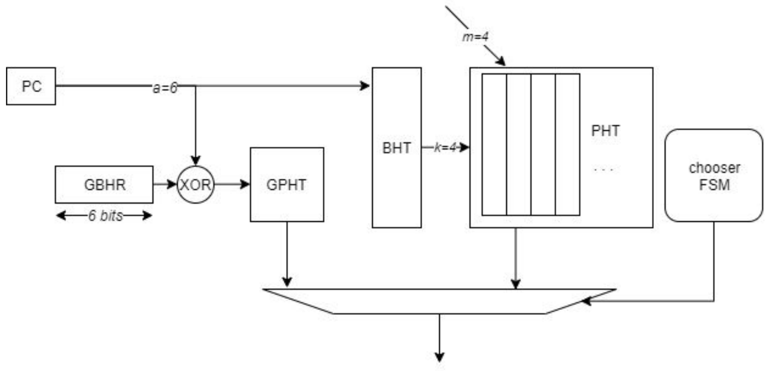

Figure 2: Tournament Predictor Architecture

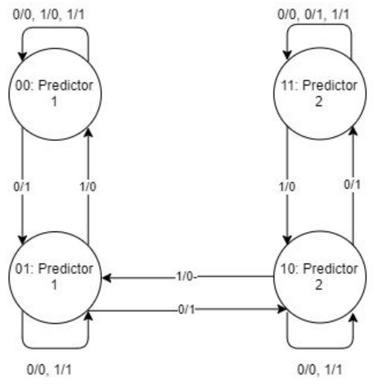

Figure 3: Chooser FSM

IV. Testing Methodology

We took a hierarchical approach at the initial stages of testing our processor. Each module was debugged independently and made it easier to be integrated. We then tested the entire pipeline by appending subsequent stages once previous stages were debugged. This made it easier to find errors than having to debug an entire 6 stage pipeline at once.

We then recursively debugged the pipeline using the provided test programs as well as our C programs. Our design successfully passed all public test programs before and after synthesis for all possible optimization flags.

V. Design Analysis and Results

Pipeline Evaluation (Figure 4):

The 2-way superscalar pipeline implemented all of the advanced features discussed above. The performance of our design on different public test cases is shown in Figure 4.

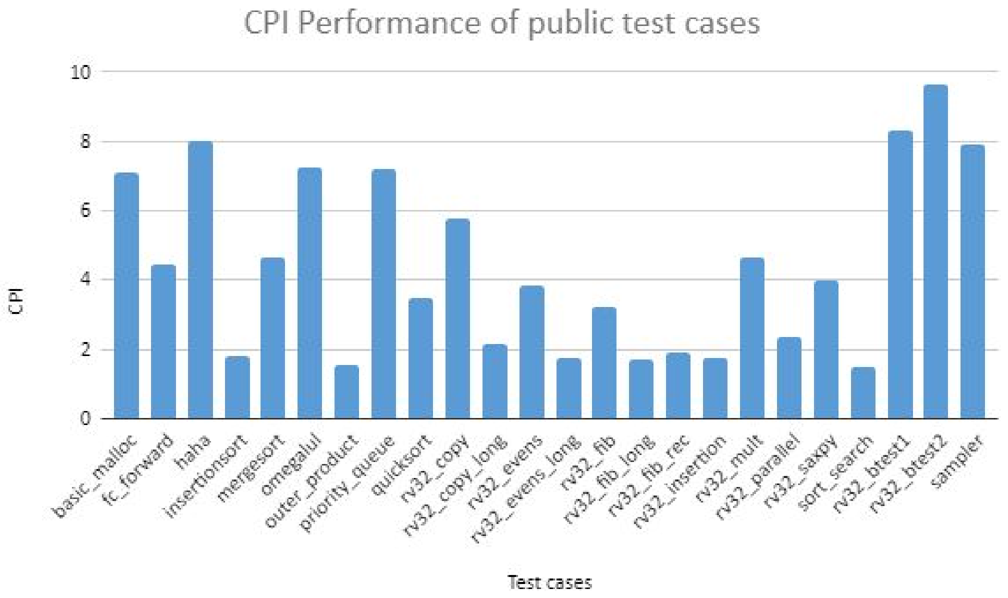

Figure 4. Pipeline performance on different test cases

Load-store Processing Enhancements (Figure 5):

The CPI of different LSQ configurations varies according to the test programs. Compared with 16 lines, LSQ with 8 lines reduces the CPI by 1%. A smaller LSQ is more advantageous for workloads that have less memory accesses, which is clearly demonstrated in the case *rv32_btest2*. Furthermore, the clock period can be smaller due to the smaller size of the CAMs required by LSQ logic. But when we tried to decrease further, there is no significant performance improvements. After inspecting the utility rate, we realize that the average utilization of LSQ is around 4 entries. However, when we use LSQ with 8 lines, it no longer is the critical path. Therefore, the line number of LSQ is eventually chosen to be 8.

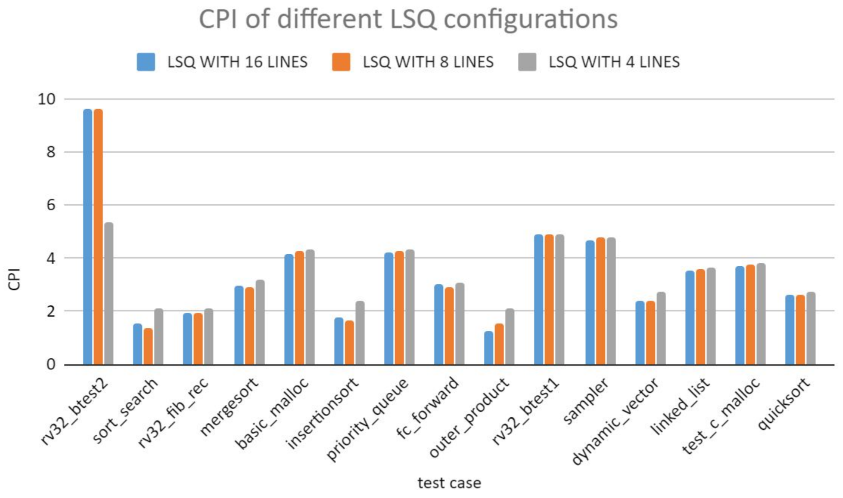

Figure 5: Average CPI of different LSQ configurations for various test cases

<u>Branch Prediction (Figure 6)</u>:

The tournament predictor achieves a 84.7% prediction accuracy using the test cases shown in Figure 6. It greatly outperforms the other two predictors for most test-cases, as the 1-bit correlation predictor has an average prediction accuracy of 28% and predicting always not-taken has an even lower accuracy of 26%. As a result, the CPI is reduced by an average of 6.6% when compared to using a 1-bit correlating predictor and by 7% when compared to predicting always not-taken.

The return address stack reduced the CPI by 1.8% on average. Because the system is restored quite frequently, the RAS is often cleared and the benefits were reduced. This explains why increasing the RAS size from 4 to 8 had no benefit for any of the test cases. Therefore, we chose to keep the size of the RAS to 4 to reduce complexity.

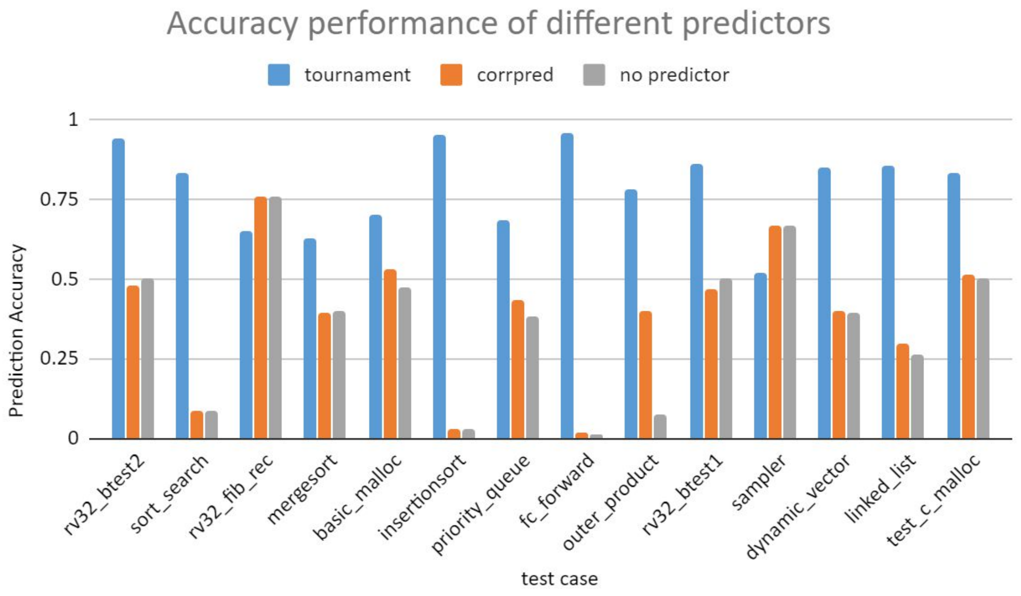

Figure 6: Performance Comparison for Various Direction Predictors

ALU Optimization (Figure 7):

The number of multiplier stages was changed from 8 to 4 as it didn't become the critical path but reduced the CPI by 3.7% on average, particularly for programs with a large number of multiply instructions such as *rv32_mult_no_lsq* and *rv32_mult*, where the CPI decreased by 33.5%.

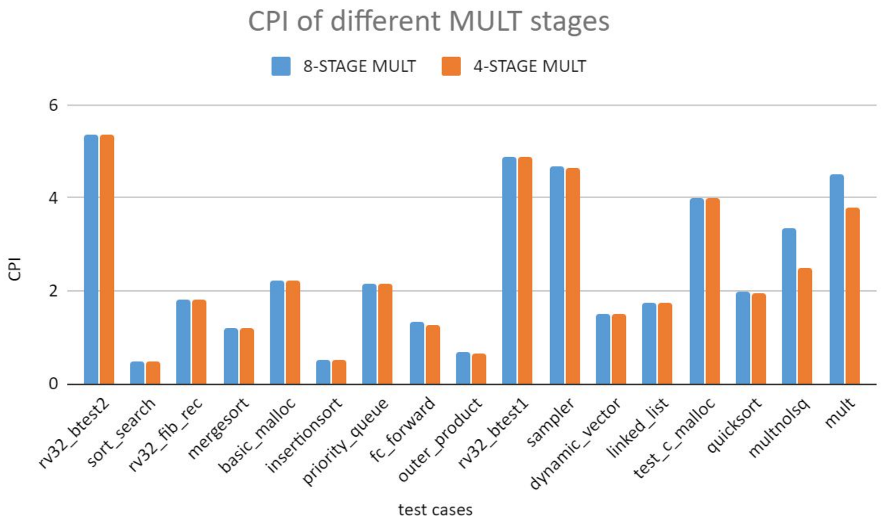

Figure 7: Performance Comparison for multipliers with different stages

VI. Critical Path Optimization

After adding all features, our processor has a critical path of ~30 ns. We moved on to optimize clock cycle by eliminating critical paths. We first analyzed RS and LSQ utilization and changed the size from 32 to 16 and from 16 to 8, respectively. This reduced the clock period to 18.3 ns. Furthermore, instead of issuing after tag broadcast, the processor now process issue and tag broadcast in parallel and reached a clock cycle. With slight increase of CPI, the processor reached a clock period of 10.5 ns.

VII. Conclusion

We have successfully implemented an R10K-style 2-way superscalar out-of-order processor based on RISC-V ISA. The processor exploits fundamental computer architecture concepts, from techniques such as register renaming, to advanced features including LSQ and tournament predictor.

From this experience, we've gained hands-on experience designing modern processors with SystemVerilog, and developing rounded testing infrastructure to validate our design. Designing processors requires solid understanding of the modules from top-level to each detail. During integration, we need to evaluate each module, and analyze whether the benefits exceed the drawbacks. Our reasoning and evaluations are listed in **V Analysis** Section.

By the end of our final submission, our processor is fully functionable, and can be synthesized with a clock period of **10.5 ns**, yielding the average CPI of **1.76**.

VIII. Member Contributions

Junyan	(20%):	LSQ, ROB, ir_stage, integration, debugging, critical path optimization, synthesis.
Haoxuan	(20%):	if-stage and integration, caches, test cases & tenchbenches, automated testing infrastructure, debugging.
Yufan	(20%):	caches, if_stage, id_stage, ex_stage, correlated predictor, BTB, integration, debugging, critical path optimization, synthesis.
Jiayi	(20%):	functional units, return address stack, ex_stage.
Andrea	(20%):	tournament predictor, RAS, ic_stage, is_stage, RS, RAT/RRAT, result analysis.