Project Part 3

James Gaul

Andy Eslick

Introduction

During this semester, we created three processors that implement the RISC-V ISA using three different implementation strategies: Single Cycle, Software Scheduled Pipeline, and Hardware Scheduled Pipeline. For each processor, we sketched a high level design, implemented the processor, wrote tests, debugged, and did performance analysis after the processor was confirmed to be working.

Benchmarking

Single Cycle	**Instruction Count**	**Total Cycles to execute**	**CPI**	**Maximum Cycle Time**	**Total Execution Time**
Testbench	115	115	1.0	36.2713094 ns	4 171.05 ns
Grendel	2 129	2129	1.0	36.2713094 ns	77 218.83 ns
Mergesort	549	549	1.0	36.2713094 ns	19 912.23 ns

Software Scheduled Pipeline	**Instruction Count**	**Total Cycles to execute**	**CPI**	**Maximum Cycle Time**	**Total Execution Time**
Testbench				17.7242113 ns	
Grendel	6 013	6275	1.04	17.7242113 ns	110 812.37 ns
Mergesort	86	97	1.13	17.7242113 ns	1 484.95 ns

Hardware Scheduled Pipeline	**Instruction Count**	**Total Cycles to execute**	**CPI**	**Maximum Cycle Time**	**Total Execution Time**
Testbench	115	214	1.86	19.4212468 ns	4 153.93 ns
Grendel	2 129	7073	3.32	19.4212468 ns	137 265.99 ns
Mergesort	549	1551	2.83	19.4212468 ns	30 172.27 ns

Performance Analysis

Instruction Count×CPI×Maximum Cycle Time gives us our total execution time in nanoseconds. For each processor, we synthesized it to get our Maximum Cycle Time, and ran each test program (Testbench, Grendel, and Mergesort) using the provided toolflow to get the instruction count from RARS, the cycle count, and CPI.

Each processor has its own pros and cons.

	Single Cycle	**Software Scheduled**	**Hardware Scheduled**
Pros	No hazards, small instruction count	No hazard detection overhead	Same instruction count as Single Cycle
Cons	Large cycle time	Large instruction count	Hazard Detection overhead

Our Hazard Detection unit contributed to about 4 ns each cycle, about 20% of the overall cycle time to *each cycle*. The time is split between decoding the instruction in IF, and actually sending the signals for avoiding hazards. 2 ns was spent actually decoding each instruction stage. If we were smarter about how we decoded instructions, we could improve our cycle time. Completing our forwarding unit would save cycles, improving our performance further.

Software Optimization

Hardware Optimization

It Depends

Challenges

Demo