Micro-architectural Processor Simulator

Andrew Hilton adh39
Tami Lehman ts179
Patrick Huang zq28

For our project, we will develop a micro-architectural processor simulator capable of simulating the x86_64 (user space) ISA. This processor will load and execute user-level binaries. We will emulate system calls functionally in the simulator—we do not intend to implement full system simulation.

1 Functional Requirements

Memory We will implement a functional model of physical memory, as well as giving each process its own virtual address space and translation. As we will not simulate an operating system, the virtual to physical translations for a program will be fixed once they are created.

ELF binary loading We will implement an ELF loader capable of reading a statically linked ELF binary into our simulated memory. We reserve loading dynamically linked binaries for a stretch goal.

Instruction Decoding We will implement an x86_64 instruction decoder, capable of decoding the subset of x86_64 instructions which appear in the SPEC2006 benchmark suite. Note that we do not intend to implement *all* x86_64 instructions, as there are quite a large number of them.

Micro-op cracking We will crack the x86_64 micro-ops into a simple RISC-style micro-ISA.

Micro-op execution We will provide a functional model capable of executing our micro-ops (and thus the instructions we can decode).

2 Stretch Goals

Simple timing model Our firs stretch goal will be to provide a simple 5 stage in-order scalar timing model, with a 2 level cache hierarchy. The timing model will report IPC, but no other stats.

Branch Prediction Our second stretch goal will be to implement a branch predictor for our timing model.

Super Scalar Our third stretch goal will be to add an option to increase the super scalar width of the processor.

Task	Expected Person-Hours			Start	End	Person
	Implement	Test	Debug			
Memory Model	2	3	3	Nov 3	Nov 9	Tami
ELF Loader	4	6	4	Nov 3	Nov 12	Patrick
Decode/Crack	4	4	4	Nov 3	Nov 15	Drew
Execution (75%)	6	6	3	Nov 9	Nov 21	Tami
Execution (25%)	2	2	1	Nov 9	Nov 21	Patrick
Stretch Goals						
Simple Timing Model	3	3	4	Nov 15	Nov 21	Drew
Branch Prediction	2	2	3	Nov 21	Nov 27	Tami
Super Scalar	3	3	3	Nov 21	Nov 27	Drew
Dynamicly linked ELF	4	3	3	Nov 21	Nov 27	Patrick

Table 1: Summary of work breakdown and time expectations.

Dynamically linked ELF loading Our fourth stretch goal will be to support loading of dynamically linked ELF binaries. simulations.

3 Software Design Plan

Our first step will be to get the binary image loaded into memory. We see two discrete tasks in this step: creating the memory model, and writing the ELF loader. We plan to split these tasks between two group members. As we build this, we will create functionality in our memory model to dump its contents, and use that to test the ELF loader functionality.

At the same time that is being developed, our third group member will begin making the instruction decoder. We plan to build a tool to let us write a convenient description of the instruction encoding, and generate an FSM for the decoder. This tool will also allow us to specify the micro-op crackings of each instruction. As this piece is developed, we will write down the encodings of many instructions and use them as test cases. We expect to test this by examining the FSM (C code) output by the tool.

Once the memory model and/or ELF loader are complete, those teammates will transition to working on the execution capabilities. At this point, we will be able to test by loading and actual binaries (starting simple, and working to more complex ones).

Once we are to this phase, it will mostly be a matter of running programs until we find an instruction we cannot handle, adding that functionality, and running again.

Table 1 sumarizes our expected time breakdown by task, as well as who is planning to do each task, and what our exected schedule of starting/completing the tasks is.

4 Specific Expertise

This project requires significant expertise in processor micro-architecture, as well as the x86_64 ISA. Our group members have much expertise in this area, so we do not expect many problems.

We do not remember the details of the ELF format (especially for dynamically linked ELF binaries), but expect to be able to find that in online documentation.

We have an ISA manual for x86_64, so we can find whatever details we need there as we implement the instructions.