Frankie

TEAM 3V

Maura Coriale
Ben Gothard
Matthew Lyons
Joy Stockwell

Executive Summary

Over the course of seven weeks, the team worked to create a “Frankenstein’s Monster” architecture consisting of a blend of accumulator, stack, and load-store architectures. Its nickname is Frankie. The processor accomplishes necessary computations using two accumulators, dubbed mary and shelley, along with a stack and a few other registers.All team members contributed to all steps in the process. They both participated in discussions and wrote code, as well as contributing to the documentation and testing.

Frankie started as a purely accumulator-based architecture, featuring one accumulator (mary). It quickly evolved to include a stack, and later another, secondary accumulator (shelley) that could swap values with the main. After team members established a rough idea of how the processor would work overall, the first step in the development process was the formulation of the instruction set. It includes twenty-five basic mnemonics, many of which containing variants determined by a flagbit accompanying the OP code. The team then worked to create a register transfer language and finite state machine for the processor. From this, a “shopping list” of necessary components was assembled. The team proceeded to implemented each component: registers, a memory block, an ALU, and a control unit. Finally, all the components were integrated step-by-step, with accompanying tests for each new iteration of the datapath. Final tests consisted of entire instructions and short programs. The team used these to debug the datapath and instruction set. They also conducted tests using their own implementation of the Euclidean algorithm to make final improvements and collect performance data.

Introduction

Students from Rose-Hulman Institute of Technology's Computer Architecture course created unique computer processing units of their own design. This team consisted of four sophomore Computer Science majors: Maura Coriale, Ben Gothard, Matthew Lyons, and Joy Stockwell. They developed their own instruction sets, register transfer languages, control states, and data paths. Furthermore, they wrote and ran code using the instructions they developed to demonstrate their understanding of low-level programming and its interaction with hardware. They constructed the processor top-down and implemented it in Verilog.

Instruction Set Design

The team began the process of creating the instruction set architecture with a mostly accumulator-based CPU that had some facets of stack. They then began brainstorming together about commands and instruction types, as well as deciding conventions for procedure calls, instruction layouts, etc. (The design document contains more detail about these.) They also began writing a first draft of the Euclid's method program using their instructions and current syntax. Matthew began writing code snippets as well. However, there did not seem to be a way to keep track of more than three variables. This exposed the need for two accumulators, an idea proposed by Ben. Maura proposed that a user specify a flagbit that determined whether the instruction would affect the backup accumulator or not. This was later changed to the @ symbol, which the assembler (written by Ben) converted to a bit in the machine code.

Team members then divided the list of instructions into quarters and each wrote RTL for one quarter of the instructions. They decided as a group to make a multicycle datapath. They tested the RTL by recording the state of the machine before and after each step in the RTL and making sure that the beginning and ending states were appropriate. This included establishing control bits. After reviewing these together, team members worked together to assemble a list of parts that would be necessary to support the instructions.

After more progress had been made into implementation, Joy diagramed the datapath. She then checked the RTL again by tracing the part of the datapath that each instruction used and writing the control bits in the appropriate places. By changing when the control bits updated, bugs were eliminated, but no major changes were made to the instructions.

Implementation

Implementation began with Ben, Maura, and Matthew each implementing a component of Frankie individually. Joy researched I/O and reported findings back to the group. It was decided that input would be handled with interrupts, while output would be automatic. Ben, Matthew, and Joy then began integrating the individual components, while Maura worked on the control unit. At first, Ben, Matthew, and Joy tried to integrate in an “onion” way: implementing the memory, then testing it in conjunction with the registers that gave it input and output, then testing those with the components that connected to them, etc. However, they quickly discovered that it was faster to each integrate and test a “block” of the datapath themselves and then integrate them so that they could work individually. Ben took charge of memory and the instruction register, Matthew of the ALU, mary, shelley, ra, ALUOUT, and comp, and Joy of pc and sp. Matthew and Joy then worked on integrating these blocks with each other and control. Ben and Matthew worked together to implement memory-mapped I/O, then added interrupts.

Xilinx Model

Frankie was implemented targeting the Spartan3E family, device xc3s500e, package fg320, speed grade -4. The version of Xilinx used was the ISE design suite 14.7 for Windows.

Testing

Unit Testing

Each individual component was unit tested.

Integration Testing

Joy tested the pc and sp blocks by manually setting the control bit for their muxes and checking that the values output by the pc and sp registers were correct on the waveform. (The test file's name is pc_block_tb2.v due to a merge conflict; the other test file has since been deleted.) She tested the integration of the pc, sp, and memory blocks by manually loading values into memory and checking that the proper values were output. She checked both right after writing them and later, going backward through memory this time.

Full Processor Testing

Joy, Ben, and Matthew tested Frankie as a whole by testing each type of instruction. They converted short programs into machine code and loaded them into memory. They ran the programs, then checked that the waveform displayed both the correct output and the expected values at each step in the RTL of instructions of interest. If they encountered a problem, they manually inspected the machine code as well as their Verilog.
They tested Euclid's algorithm by loading it into Frankie's memory. (Not all numbers tested are present in the testbench; the one in the testbench was backspaced and retyped for brevity's sake.) The testbench was run with 5, 10, 3030, and 5040, among other numbers, checking the output of each against a known value.

Performance

Conclusion

Team members take away a great deal of valuable experience from the creation of Frankie. Unlike many CS students, they have an appreciation that the limitations that hardware imposes on programs are not arbitrary. Instead, they are the result of carefully calculated tradeoffs, which the team now has experience making for themselves. They have experience implementing and testing modules in Verilog, as well as working directly with machine code. They created diagrams, tables, and documentation, which can be viewed in the design document. These both helped them communicate with each other and allowed them to practice preparing to communicate with future users and/or editors of their code. Finally, team members gained valuable insight into the importance of communication and how to make decisions as a group.