A Computer Engineering Capstone Design Project: A Harvard Architecture Assembly Simulator

Frank Di Natale, Joe Di Natale John Mercer, and Donald Ray
University of South Florida

For our capstone design project we participated in the 2010 IEEE Computer Society System Competition. The purpose of the competition is, "to promote excellence in the design of a system by a team of students." This year's problem was to design and develop a computer processor (CPU) simulator where originality of the processor architecture, functionality, quality, versatility, and the use of good software engineering techniques are key criteria. Our inspiration was the National Cyber Leap Year Summit of 2009 calling for a change in hardware to increase security in computer transactions. One possible improvement is to be found in the Harvard architecture. By separating the data memory from the instruction memory, buffer overflow attacks cannot be used to inject and run code. In order to familiarize students with the Harvard architecture and its potential tradeoffs, we designed and developed an assembly simulator and an associated instruction set architecture. Key innovations included implementation of a Harvard architecture instruction set, compliance with Section 508 of the Rehabilitation Act, and high-level debugging capabilities. By introducing the idea of secure computing to students as they are learning the foundations of assembly programming, we can help guide the move to a more secure Internet marketplace.

Corresponding Author: Frank Di Natale, fdinatal@mail.usf.edu

Introduction

The authors of this paper are all senior students in the Department of Computer Science and Engineering at the University of South Florida (USF). A requirement for the BS in Computer Engineering degree is CIS 4910 Senior Project.[1] The course takes students through the standard design process, requiring that documentation be submitted at each stage. This allows for experience in a team environment to be acquired, as well as gives valuable experience into how the design process works.[1]

The Senior Design course at USF uses the design process shown in Figure 1. The goal of the course is to allow students to become familiar with the development process and complete a software or hardware project. Each student selects an industry-based project, forming a team of four to six with other students. The formal development process is followed with requirements, specification, and test plan documents being produced along its progression. At the end of the semester students will write a final report, make a poster, write a press release, and give a final oral presentation about the results of their project.

One of the key aspects of the course is industry involvement.[1] The course gives companies in central Florida the opportunity to provide a non-critical project to students to be completed over the course of a semester, but also gives them a means of interacting with the graduating class. One such interaction is through guest lecturing. Speakers are asked to give a lecture about their particular company, giving them the opportunity to become familiar within the class as a whole. Involvement by companies makes the experience feel much more authentic by making the projects feel important.

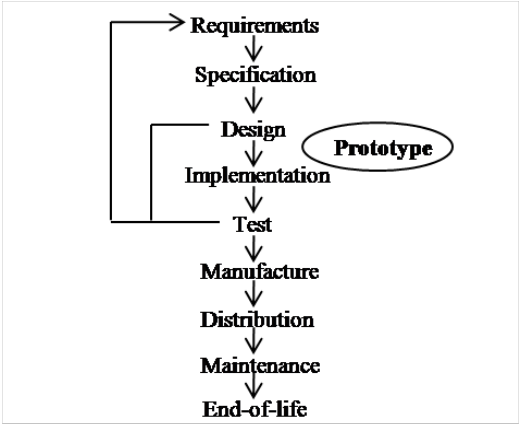

Figure 1: The Design Process

IEEE Computer Society Competition

In the Spring 2010 semester of Senior Design, there was the opportunity to participate in the IEEE-CS Competition. The competition has been run yearly since the year 2000, seeking to give Computer Science and Engineering students internationally a challenge similar

to those faced in the real world.[2] The goal behind providing such a problem is to promote teamwork, centering around the idea that it's not about individual competition. The IEEE-CS Competition seeks to accomplish that by providing a task which requires good teamwork to win.[2]

The 2010 IEEE Computer Society System Competition focuses on the use of sound software engineering principles.[3] This year's competition goal is to design an instruction set architecture and implement a simulator for it. The project as a whole will be judged by the following criteria:

- "The originality of the architecture designed
- Functionality, quality, and versatility of the simulator.
- The use of software engineering in the design of the simulator."

For the winning team, a $7,000 prize is awarded for the project that meets the listed criteria, as well as three individual $1,000 prizes for groups who are the best in a specific category.

Required contest deliverables included a report of no more than 40 pages, the simulator itself, and a ten minute video demonstration. In order to produce these deliverables our group was broken down into assigned roles, each person handling a specialty within the project. The general flow of the team was led by ISA development, followed by implementation within the simulator, followed by additions to the report. We found that this flow worked well because it allowed for tasks to be pipelined quite efficiently.

This paper will highlight our team's instruction set architecture and simulator. It will also take a brief look at related work, future work, and analyze our team's experience in our design class.

The Instruction Set Architecture

The first step in the design process was to design the instruction set architecture (ISA). The South Florida Instruction Set Architecture (SFI) is based the Harvard Memory Architecture and RISC.

The Harvard Architecture allows for much needed introduction to the concept of security. Harvard architecture separates instruction and data memory allowing for buffer overflow attacks to be deterred. By introducing this concept of attacks at the architecture level it becomes easier to introduce the concept of security to students. Using RISC aligns well with the statistical data presented in Patterson and Hennessy's *Computer Architecture: a Quantitative Approach*, which determined what instructions and addressing modes are most common, and found that a minority of instructions are used a majority of the time.[4] Based on the competition guidelines and our research, the following requirements were drafted:

1. The data and instruction memory shall be separated as seen in the Harvard Memory Architecture.
2. The instruction set shall use load-store architecture.
3. The instruction set shall support conditional and unconditional branching.
    a. Jump instruction.
    b. Branch equal, unequal, and to Subroutine.
4. All signed values shall be represented in 2's complement form.
5. The instruction set shall support the following arithmetic instructions:
    a. Signed and unsigned addition, subtraction, multiplication, division, and modulus instructions.
    b. Arithmetic right shifting operation.
6. The instruction set shall support the following logical instructions:
    a. AND, OR, NOT, and XOR instructions.
    b. Left and right shift operations.

Instruction Classes

SFI has a total of 46 hardware level instructions spanning over three instruction classes. Our ISA takes two operands as inputs, using the first operand as both a source and the destination, making these types suitable to handle arithmetic instructions as well as some variations of the load instruction. The Flow Control type takes only one argument, a 16 bit immediate. This class of instruction is used solely for forms of the jump, branch, load, and store commands.

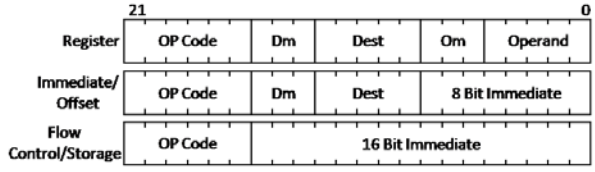

Figure 2: The three instruction classes

Registers

The SFI architecture contains 32 general purpose registers (GPRs), each 32 bits in length. Each register is addressed by using the name R# in the assembly code, where # is a number from 0 to 31. Each GPR is also split into a number of slices and can be seen in Figure 3 on the nest page. Instructions that contain operands that are registers are accompanied by three bits which represent the register slice mode. The slice break down of a register is shown below. Register slices are accessed by appending the name of the slice to the register's name (Ex. R1.W0 or R3.B2).

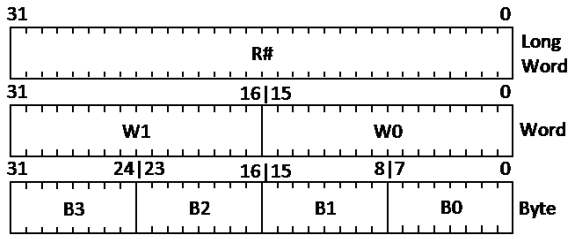

Figure 3: Layout of the register slices

Address Modes

Our architecture, SFI, supports five address modes: immediate, relative, register-direct, register-indirect, and register-indirect with offset. Each addressing mode can be used for both instruction and data memory as each instruction format provides. Instruction memory is addressable by instruction (22 bit blocks) and data memory is addressable by long word (32 bit blocks).

The immediate addressing mode allows for constants to be passed in through the instruction to be operated on. This mode allows for either a small constant to be operated on or an address location to be passed within the instruction.

The relative addressing mode uses an offset to address locations near the PC. This ability makes flow control much simpler.

Register-direct is the addressing mode used to directly address a register in the register file of the processor. There are 32 registers in the register file, therefore the register-direct mode uses 5 bits to address.

Register-indirect with offset uses a register to store a base memory address and an 8 bit immediate offset or another register to provide offsets greater than 8 bits. The offset is added to the base address and then accessed or written to. This mode is also used to provide register-indirect (without an offset), simply by keeping the offset zero. When a register is addressed in the assembly, it is automatically assumed that the offset is zero if no offset is provided. Register-indirect addressing is denoted by the "@" symbol with the register and offset encapsulated in parentheses.

The Simulator

As SFI has no hardware developed for it, a simulator is a requirement to test the instruction set and to provide users with an introduction to its features. The SFI Simulator implements the SFI Architecture, provides debugging abilities, visual and audio alerts, configurable syntax coloring, and an overview of the machine's state during execution.

Our simulator was designed in two parts: the GUI and the simulated processor. Because of the modular nature of the design, our team decided that C# was the best choice for implementation. Using the .NET Framework, the simulator contains a fully functional simulated SFI processor and functional GUI which includes full compliance to United States Section 508 standards.[5] Complying with the standard allows for the SFI simulator to be more readily accessible to those who are visually and hearing impaired.

The user interface provides the basics that a simulator needs to have in order to display the machine's state. Like our architecture, the SFI Simulator embraces the concept of simplicity and power. An uncluttered interface allows easy access to the main features of the simulator.

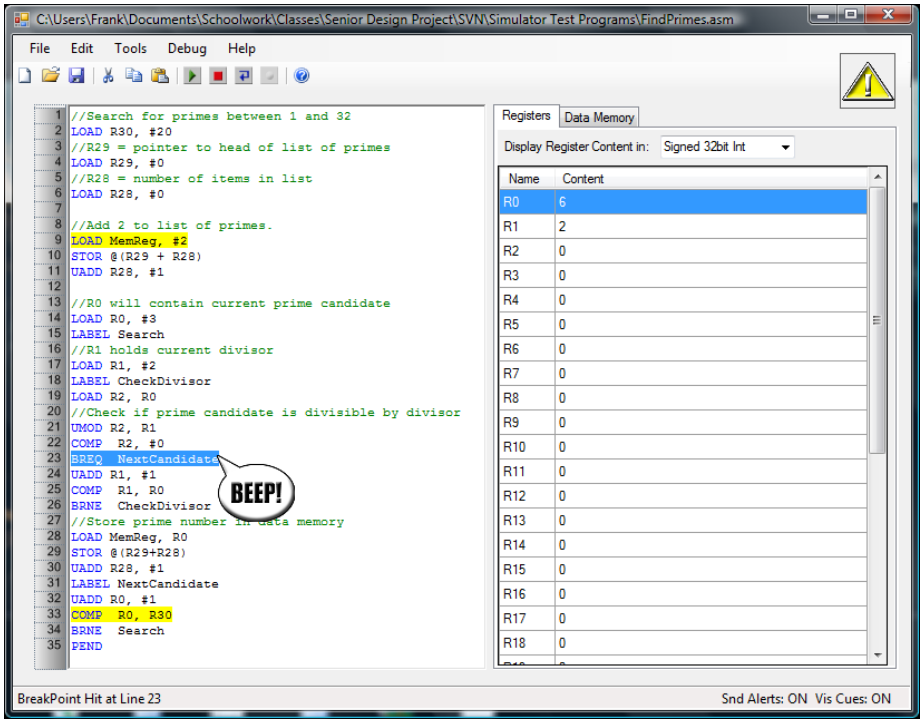

Figure 4: The SFI Simulator's main window.

Figure 4 above displays our simulator's layout and the following features of our simulator:

1. Built in code editor with the ability to load and save files.
2. Color coded keywords for instruction classes.

3. Tabbed register and data views with binary, hexadecimal, signed 32-bit integer, and unsigned 32-bit integer representations.
4. Unconditional breakpoints (yellow highlighting).
5. Visual cue ("Beep!" bubble) for hearing impaired users.

Along with the features listed above, for users who are colorblind our simulator offers a high contrast color scheme for the editing window so that the code is much easier to read. These major additions make it so that even users with impairments are able to comfortably work with our simulator.

Also included with the SFI simulator are a variety of debugging tools. The first of these features is active syntax checking, allowing the user to fix any unnoticed errors before they execute their program. Our simulator also includes two forms of breakpoints: unconditional and conditional. Unconditional breakpoints are used just like those in a vast majority of other IDEs, simply by marking the line to pause execution at. Conditional breakpoints, on the other hand, are used to watch registers. These breakpoints allow the user to designate a breaking condition for specific registers based on the stored data, making it possible to monitor data for unpredicted values. Included with the breakpoints is the ability to step as well, to see what is happening after each and every instruction without having to resume full execution.

Evaluation of the ISA and Simulator

In order to evaluate our architecture and simulator, our group coded and ran a prime number program. The program's objective was to find all prime numbers between 1 and 32. All coding was done within our simulator's IDE, and executed. The way the program works is that it incrementally checks all factors up to the number being assessed. If it is found that the number is divisible by anything other than itself and 1 the number being assessed is incremented. Both our instruction set and simulator were able to handle full execution of the prime number program. During the execution of the program, our simulators debugging tools were used to watch the registers as their data was modified. This success shows that both the instruction set and simulator are able to be used to code, execute, and debug.

Lessons Learned

Through the course of this project our team has learned a few things.

1. In major projects teamwork is essential. If the team cannot function, then the results will not be of good quality.
2. International competitions do make good senior design projects.
3. Contest entry and deadline dates should coincide with college semesters allowing them to be used in design classes.

Competitions provide students with a challenge that, while not rooted in industry, is still a worthwhile project for a student who perhaps would like to pursue further studies or students interested in the theory behind the scenes. It would be of great benefit if the separate organizations (Capstone, IEEE, colleges, etc.) all collaborated to produce a unified effort to bring these challenges to students so that they can be affected by the benefits that participating in such competitions can provide.

Summary and Future Work

In summary, both the SFI and our simulator are fully functional. The instruction set uses basic instructions, but is quite powerful and our simulator has an array of features including improved accessibility for those who have disabilities, a built in editor, and debugging capabilities. While both are functional, the following things are features we would like to add:

1. More complex instructions in our ISA to support more elegant code such as loops.
2. The ability to handle interrupts and message passing to support peripherals and other features.
3. The ability to run a modified Harvard Architecture without compromising security.
4. Optimize simulator code for better performance.
5. Use a different graphics library to support nicer user interface features.

References

1. K. Christensen and D. Rundus. "The Capstone Senior Design Course: An Initiative in Partnering with Industry," *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference*, pp. S2B12-S2B17, November 2003.
2. A. Clements. "Constructing a Computing Competition to Teach Teamwork," *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference*, pp. F1F1-F1F6, November 2003.
3. "IEEE Computer Society System Competition," http://www.computer.org/portal/web/competition
4. D. A. Patterson and J. L. Hennessy, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann Publishers Inc., 2003.
5. *US Code*, Title 29, section 794(d), http://www.section508.gov/index.cfm?FuseAction=Content&ID=12.