AC 2010-579: PRISM: A SIMPLE SIMULATION FOR INTRODUCTION OF ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE

Brian Peterson, United States Air Force Academy

Anne Clark, USAF Academy, CO

PRISM: A Simple Simulation for Introduction of Assembly Language and Computer Architecture

Abstract

One of the enduring problems in introducing computer architecture and assembly language to students is most systems are so complex that the students quickly lose sight of how the subsystems interrelate. To effectively teach how a system processes and executes instructions, most students must program in assembly language and observe how the processor handles the code through execution. Often, the students get lost in the myriad of instructions and registers and lose the basic concepts. This paper introduces a free visual simulation of a very simple 4-bit architecture computer, Programmable Reconfigurable Informational Simple Microcomputer (PRISM), to be used in an introductory digital logic course or the beginning of a computer organization or architecture course. It provides an assembler as well as a colorful block diagram of the overall system-indicating which Register-transfer-logic (RTL) units are being activated and which subsystems are accessed. Additionally, a state diagram graphic can be activated to illustrate the progress through the corresponding state machine as the students advance the clock. An interesting side effect of providing this tool to students was that, although understanding of the signal level workings of the computer increased, the mechanics and comprehension of hand-assembling a program dropped significantly.

Introduction

As students go through an introductory sequence in digital systems, there is a logical progression: combinational logic gates, sequential logic/state machines, Medium Scale Integrated (MSI devices) and then microprocessors. Most students have no problem understanding the relationship between combinational logic and state machines—that combinational logic is used to create the output and next-state logic within the state machine. Most also have no problem understanding that MSI devices are simply combinational/sequential logic packaged together to perform relatively simple operations. However, when the transition is made to microprocessors, most beginning students lose the connection to what they previously learned. Suddenly they have transitioned from gates/chips to programming in a new language and the hardware is just a computer, which they are already familiar with but separate from the digital design they had covered in the course. Even if the microprocessor is a simple 8-bit system, it is still more complex than many students can comprehend to understand that it is simply a state machine with combinational output and next state logic just like they had used in other state machines, just larger and more complex. Fewer still can understand what signal would be needed to control such a system.

For over thirty years, United States Air Force Academy has recognized the importance of training aids[1,2] and have used them to help students understand these concepts and visualize the computer architecture and their inner workings. Obviously, the aids have changed over the years as technology changed. The current aid is PRISM[3], a 4-bit hardware description language (HDL) microcomputer which is implemented on a Field Programmable Gate Array (FPGA). It is implemented on a Xilinx Spartan board with

I/O peripherals. Although this allows students to place a program into the virtual Read Only Memory (ROM) of the machine, it is still difficult for students to visualize and understand what is happening within the machine. Simulations of the system in software such as Aldec's Active HDL using a testbench allows a waveform to be created, but walking through many signals for each clock cycle is also very difficult.

To help students fully understand and visualize what happens to each signal during each cycle of each instruction and comprehend the activity of the state machine, The Department of Electrical and Computer Engineering at the United States Air Force Academy developed a simulation which visually displayed the entire microcomputer and state machine and creatively named it the PRISM Simulator.

PRISM Overview

The PRISM system is a 4-bit microcomputer with 16 assembly instructions, an 8-bit address bus, four input and four output ports organized as shown in Figure 1. The memory is broken into 176 nibbles of ROM and 80 nibbles of Random Access Memory (RAM). It uses port-mapped I/O with separate memory and input/output (I/O) select signals. Register-transfer-logic (RTL) design techniques were used to implement the controller and datapath. The controller is a relatively simple mealy state machine shown in Figure 2.

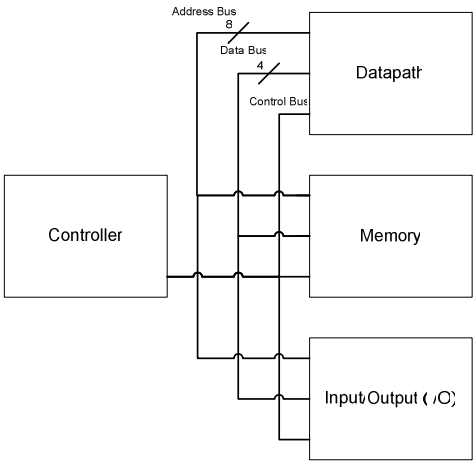

Figure 1. PRISM Top-Level Diagram.

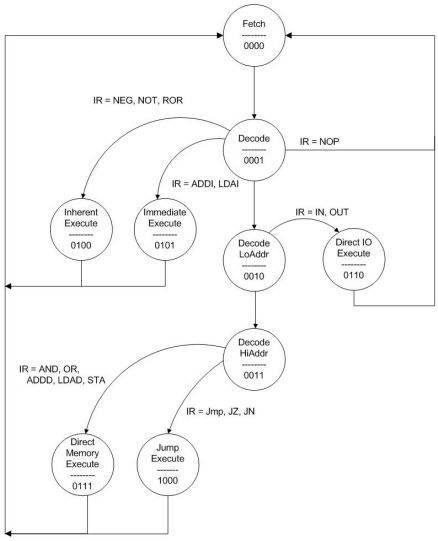

Figure 2. PRISM Controller State Diagram

The datapath, shown in Figure 3, uses both combinational and sequential logic components. An arithmetic logic unit (ALU) performs both arithmetic and logic operations when program instructions demand them. The datapath also contains multiplexers to determine if data is loaded into a variety of 4-bit and 8-bit registers or placed on the address and data buses. Control signals from the controller implement the instruction set in single clock cycle operations while status signals based on a single accumulator and instruction register help the controller decide to proceed.

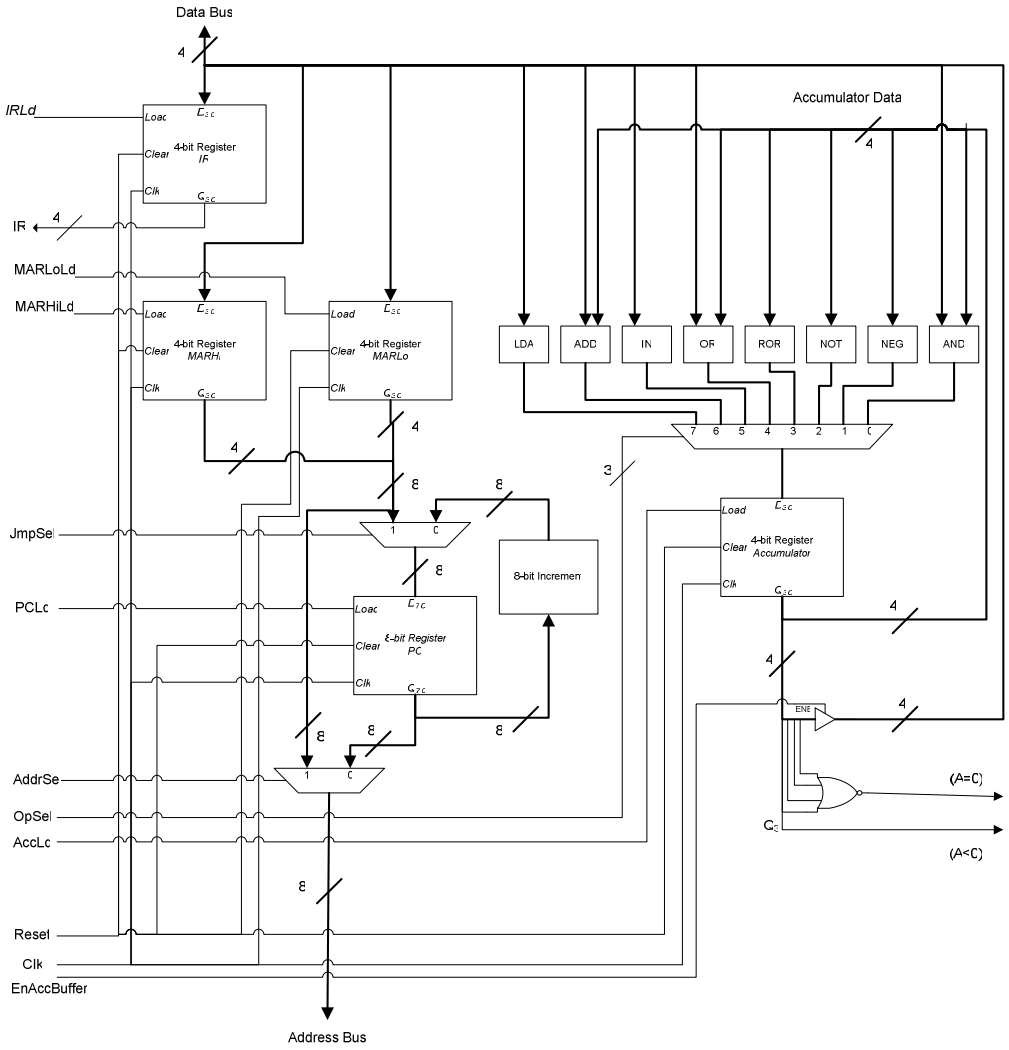

Figure 3. PRISM Datapath Block Diagram

PRISM Simulator

The intent of the PRISM Simulator was originally to make assembly of PRISM assembly programs easier for students. Originally, a Microsoft Excel spreadsheet was made to aid the students in writing their programs. However, making changes to programs required care to avoid misdirecting the formulas and links. Additionally, the students would have to manually enter the machine code into the .vhd file which governed the HDL that "loaded" the RAM. A student took the initiative to write a macro to format the machine code so that it would be pasted into the .vhd file, but this was still an unnecessary hassle for students. The plan to make an assembler grew into a visual simulation to help display what happened within the PRISM system. The final result of the main screen is shown in Figure 4.

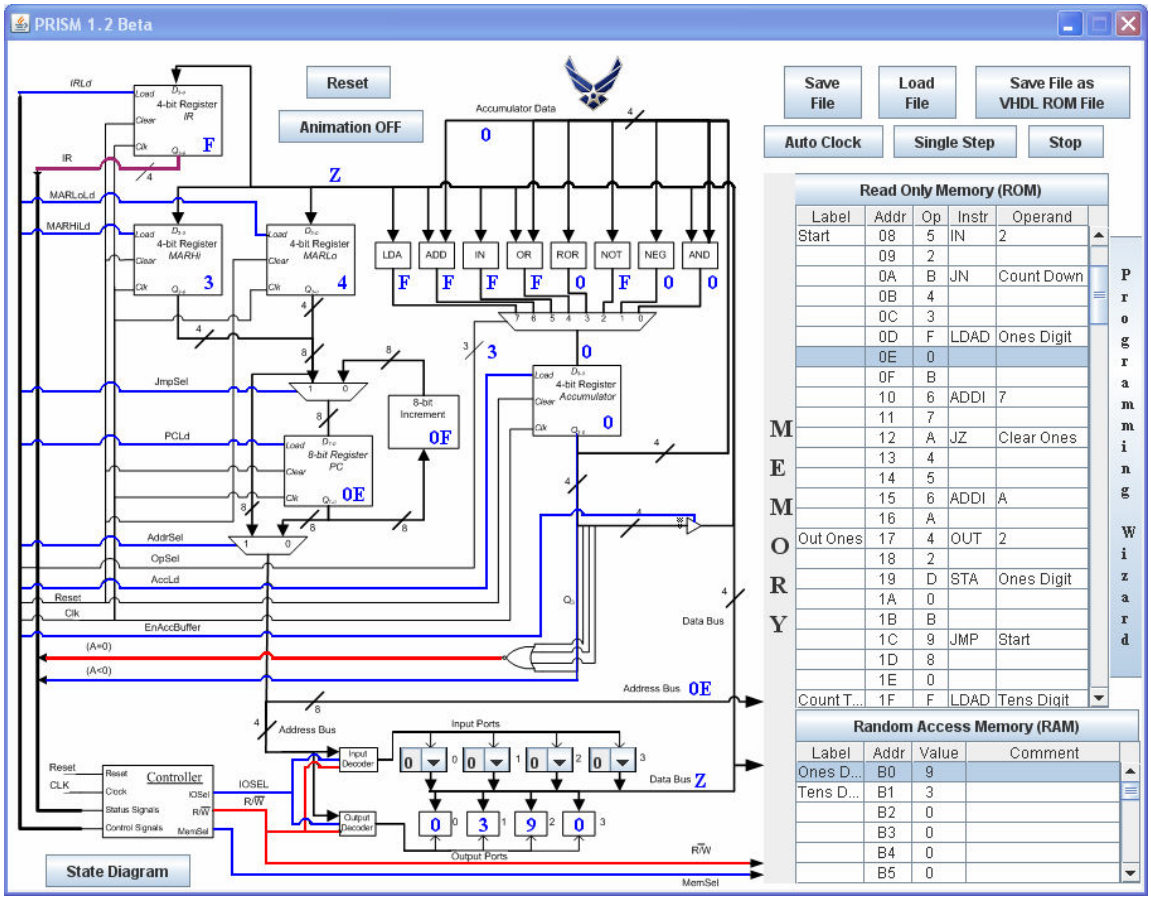

Figure 4. PRISM Simulator

Once a program has been loaded into ROM using the Programming Wizard (button on the far right), the user can press either “Auto Clock” or “Single Step” to begin advancing through the program. As the program counter advances, the location in memory that it refers to is highlighted. If RAM is accessed, that RAM location is highlighted. The default condition has “Animation ON.” This causes the simulator to show all relevant lines throughout the system as being a ‘1’ or ‘0’ but displaying them as red/blue, respectively. Additionally, the contents of all registers and busses are displayed and updated in real-time. This allows the students to see what is happening with all aspects of the system without having to refer to a table which shows the signal level for a given instruction and clock cycle, or position within the state machine.

To increase run speed, the animation can be turned off by pressing the “Animation OFF” toggle button. The Input ports allow the user to change inputs while the program is running while the contents of the Output ports are simply displayed.

In addition to the expected Reset for the system, buttons are provided to allow the user to save the program, load a program, or export the program to a .vhd file dictating the contents of ROM in the format expected by the rest of the PRISM system. Additionally, buttons are provided to display the controller state machine in a separate, resizable window and to spawn the Programming window, also in a separate window.

PRISM State Machine

In addition to having difficulty understanding which signals are active at a given time or what is present on one of the busses, students often have trouble correlating the block diagram of the system with the state machine. The State Machine window presents the controller state diagram and highlights the current state in real time as shown in Figure 5. The students can watch the state change as they step through a program as the circle for the current state turns red.

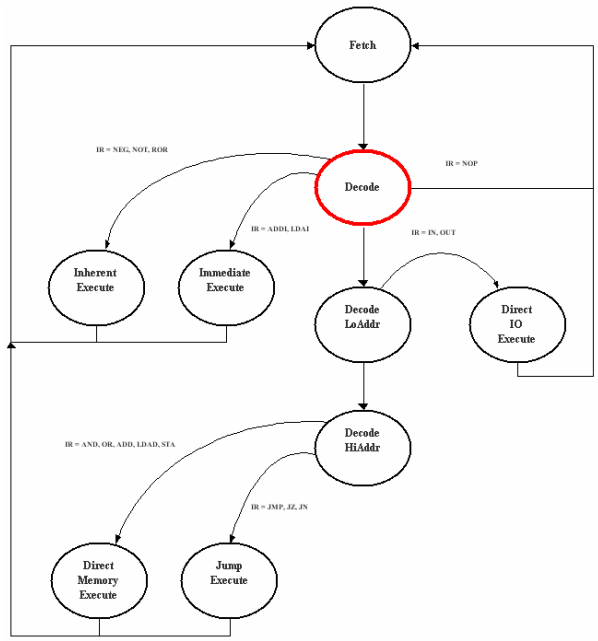

Figure 5. State Diagram Window

Programming Wizard

The Programming Wizard provides an almost error-proof template to walk beginning students through programming in assembly language. The Wizard allows the student to choose one of the sixteen instructions and automatically associates the machine opcode with the instruction as shown in Figure 6. For instructions that require an operand, it leaves bright red question marks indicating the user must input an operand as shown in Figure 7. If the user attempts to enter an invalid operand (for an I/O operation or immediately addressed operand) it will not permit the input and will give an error message in the messages box as shown in Figure 8. Immediately addressed instructions require a label either in the left column of the Programming Wizard (for jump operations) or in the RAM box on the main Simulator page. If no label name is given or the label does not yet exist, bright red Xs appear in the machine code column indicating an error (Figure 8) along with a "Label not found" error in the message box.

Standard editing options are also available for cutting and pasting sections of code. At the bottom of the Wizard, there is a checkbox to allow the user to edit values in RAM. Obviously, RAM cannot normally be modified by the user in such a simple system

except through the code running at the time, but this allows them to change memory values as they step through the code. It is on a separate page to make this point.

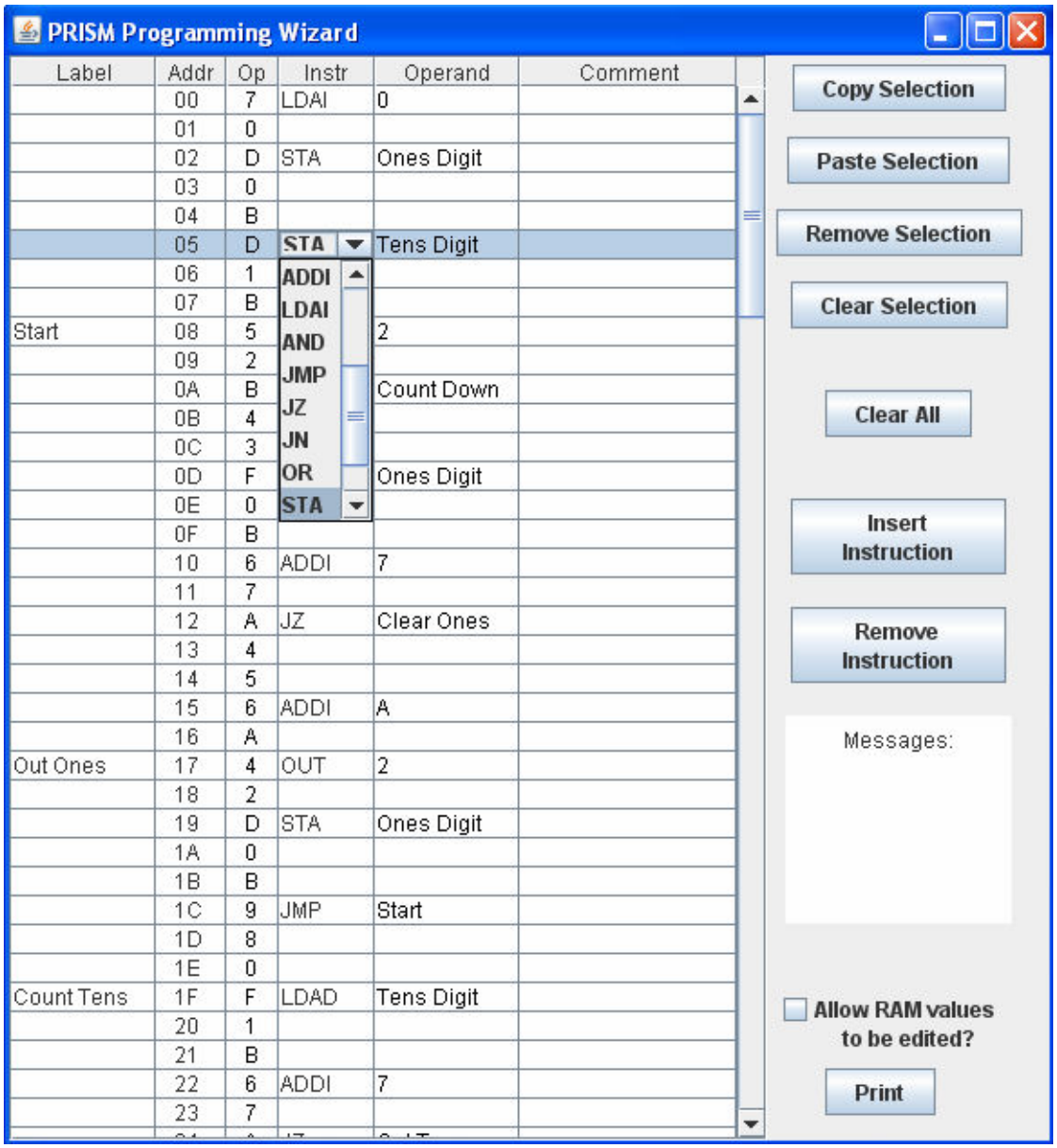

Figure 6. Programming Wizard

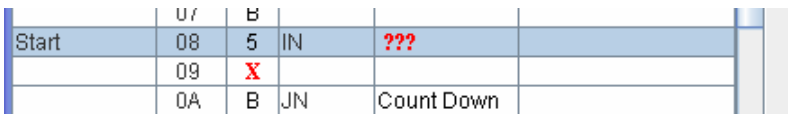

Figure 7. Prompt for Operand

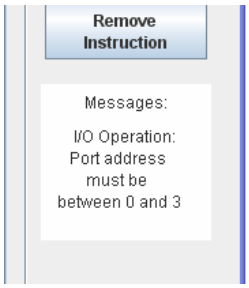

Figure 8. Error message

Figure 8. Prompt for Operand

Student Results and Feedback

The student response was overwhelmingly positive. Students who had completed the course before the Simulation was created stated they wished they had the simulation and told stories of "back in the old days when things were tough." More importantly, on the Final Exam, scores on the question involving understanding of signals associated with the microcomputer rose from 72% to 79%. Surprisingly, however, scores on the question requiring students to write code and assemble it themselves dropped from 87% to 79%, due almost entirely to their inability to assemble the code into machine language. Compared to previous classes, they trusted and relied on the Simulator to assemble the code instead of consistently having to check to see if the Excel spreadsheet had misaligned and corrupted their code. This resulted in less overall understanding of the machine code. As a result, an exercise explicitly requiring them to hand-assemble the code and interpret machine code has been added to the course.

Conclusion

The simple PRISM microcomputer is a wonderful aid to bridging the link between sequential machines and a micro computer. The limited number commands and signals allows them to understand what is actually needed within a computer. The PRISM Simulator enhances this learning by providing real-time insight into the status of all elements within the system as well as a visual link to the state diagram. As tools grow more efficient however, students are more likely to rely on tools to perform functions such as assembling code into machine language and are less motivated to fully understand how the tool works. Such concerns must be addressed whenever new tools and aids are introduced into a curriculum.

Note to Reviewer: The intent is to make PRISM and the Simulation available for anyone who would like a copy.

Bibliography

[1]Moser, A. T., "Animated Simulator for 68000 Microcomputer Architecture," ASEE Annual Conference Proceedings, June 1995, pg 179 - 181.

[2] Henderson, W. D., "Animated Models for Teaching Aspects of Computer Systems Organization," IEEE Trans. On Education, Vol. 37, No. 3, pp. 247 - 256, August 1994.

[3] Clark, A., and Peterson, B., "PRISM: The Reincarnation of the Visible Computer", submitted ASEE Annual Conference Proceedings, June 2010