AC 2010-1557: PRISM: THE REINCARNATION OF THE VISIBLE COMPUTER

Anne Clark, USAF Academy, CO

Brian Peterson, United States Air Force Academy

PRISM: The Reincarnation of the Visible Computer

Abstract

For over thirty years, the United States Air Force Academy (USAFA) has used training aids to help students in our introductory digital course visualize computer architecture concepts by showing the inner workings of a simple microprocessor made primarily of the SSI and MSI chips which they were already familiar with. The original "EDUcational COMPuter" or EDUCOMP was replaced with a more visual and improved version, the "VISIble COMPuter" or VISICOMP in 1996[3]. Today, we have transitioned these hardware training aids to a hardware description language (HDL) implementation called the Programmable Reconfigurable Informational Simple Microcomputer or PRISM. PRISM is implemented on a Field-Programmable Gate Array (FPGA) while still retaining the original strengths of the EDUCOMP/VISICOMP computer architecture.

PRISM is partitioned into the main subsystems of a computer (ALU, controller, memory, and input/output (I/O)). The controller is implemented as a simplistic, mealy state machine which allows students to see each step of the instruction cycle as an assembly language program is executed. PRISM's operation is visual to the student since: (1) each subsystem is built with small-scale-integration (SSI) and medium-scale-integration (MSI) components which the students have already learned in our course; (2) the status of the registers, signals, and busses are displayed directly on seven-segment displays; and (3) the students manually build and test each subsystem before integrating them into the final PRISM core to observe how each one works. After understanding the basics of the computer subsystems, the students then write their own assembly programs and translate them into machine code. They can then simulate their program running ahead of time or implement and run their programs in real time.

This paper will discuss the features of PRISM and how it is used in the classroom.

Introduction

Many years ago, the Department of Electrical Engineering at the United States Air Force (USAFA) and others[1,2] recognized the importance of training aids in teaching students the inner workings of a basic computer. As a result, professors developed first the "EDUcational COMPuter" or EDUCOMP and then a "VISIble COMPuter" or VISICOMP[3] as part of a concerted effort to introduce student-centered tools into our digital systems track of courses[4].

In the spring of 1996, VISICOMPs were introduced to students in the final portion of the sophomore introductory digital systems course after they had already been exposed to the design of combinational and sequential digital logic circuits. The underlying premise of both EDUCOMP and VISICOMP was that, by combining the small-scale-integration (SSI) and medium-scale-integration (MSI) chips that students were already familiar with into a simple computer, students would find it easier to understand how a computer works.

The VISICOMP, shown in Figure 1, was comprised of five subsystems, each on a separate printed circuit board: the controller, the arithmetic logic unit (ALU), the input/output (I/O)

subsystem, the memory subsystem, and a motherboard. Each board was populated with MSI integration components as well as light emitting diodes (LED) and seven-segment displays to help students follow the computer's operation. The VISICOMP featured a fully visible data bus, address bus, and control bus as well as visible indicators of key register contents, control signal, and the controller states. Finally, it contained 256 nibbles (4-bit data words) of memory and 16 operation codes.

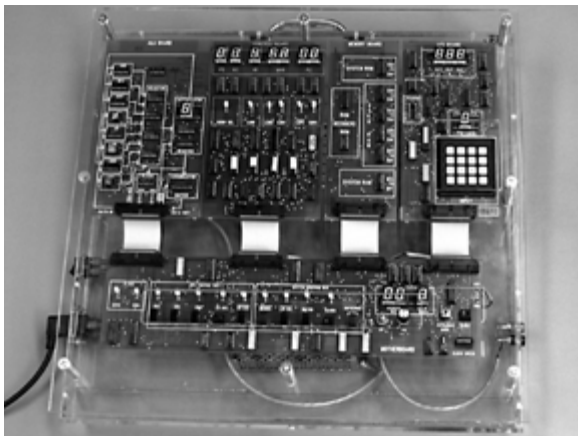

Figure 1. VISICOMP

Shortly after VISICOMP's introduction, USAFA added a computer engineering degree, graduating its first class in 2001, and the department became the Department of Electrical and Computer Engineering. In 2004, the IEEE Computer Society and the Association of Computing Machines (ACM) published guidelines for computer engineering curricula[5] which highlighted the need to introduce our students, particularly our computer engineers, to a hardware description language. The department decided that the best place to introduce this language was in our introductory digital systems course and in the fall of 2004, the course was revised to include VHDL programming as a core component. We began moving from VISICOMP to the Programmable Reconfigurable Informational Simple Microcomputer (PRISM) in the spring of 2005 and, by the spring of 2006, had settled on the version reported on in this paper.

PRISM's Design

PRISM was implemented using the same small and simple architectural design that had been so successful with EDUCOMP/VISICOMP. The distinct subsystems shown in Figure 2 allow students to see the orchestration required by even the most basic computer while the 16 instructions described in Table 1 provide a complete set of arithmetic, logic, and branching instructions without getting too complicated. As can be seen from the ADDI and ADDD, as well as the LDAI and LDAD instructions, there's even room to implement two addressing modes (immediate and direct).

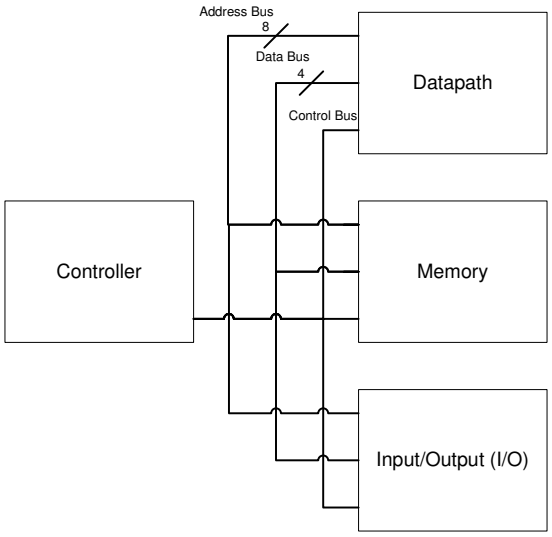

Figure 2. PRISM Top-Level Diagram.

Mnemonic	Encoding	Operation
NOP	0000	Do no operation.
NEG	0001	Takes 2's complement of the number in the accumulator.
NOT	0010	Takes 1's complement of the number in the accumulator.
ROR	0011	Rotates the accumulator data one bit to the right (with wrap-around).
OUT	0100 aaaa	Transfers the data from the accumulator to the selected output port.
IN	0101 aaaa	Loads the data from the selected input port into the accumulator.
ADDI	0110 vvvv	Adds the value of the operand to the number in the accumulator.
LDAI	0111 vvvv	Loads the value of the operand into the accumulator.
AND	1000 aaaa aaaa	ANDs the value stored at the operand address with the accumulator and stores the results in the accumulator.
JMP	1001 aaaa aaaa	Jump unconditionally to the operand address.
JZ	1010 aaaa aaaa	Jump to the operand address if the accumulator is zero.
JN	1011 aaaa aaaa	Jump to the operand address if the accumulator has a negative number.
OR	1100 aaaa aaaa	ORs the value stored at the operand address with the accumulator and stores the results in the accumulator.
STA	1101 aaaa aaaa	Store the contents of the accumulator into the operand address.
ADDD	1110 aaaa aaaa	Add the value stored at the operand address to the accumulator.
LDAD	1111 aaaa aaaa	Load the value stored at the operand address into the accumulator.

Table 1. PRISM Instruction Set.

Register-transfer-logic (RTL) design techniques were used to devise the controller's and datapath's functionality. The controller, shown in Figure 3, is a fairly simple mealy state machine which clearly demonstrates a standard instruction execution cycle of fetch-decode-execute while showing the complex orchestration between the datapath, memory, and I/O subsystems required to implement even a basic instruction set. A control bus, consisting of control signals from the controller and status signals from the datapath, allows students to view these signals as individual instructions are executed.

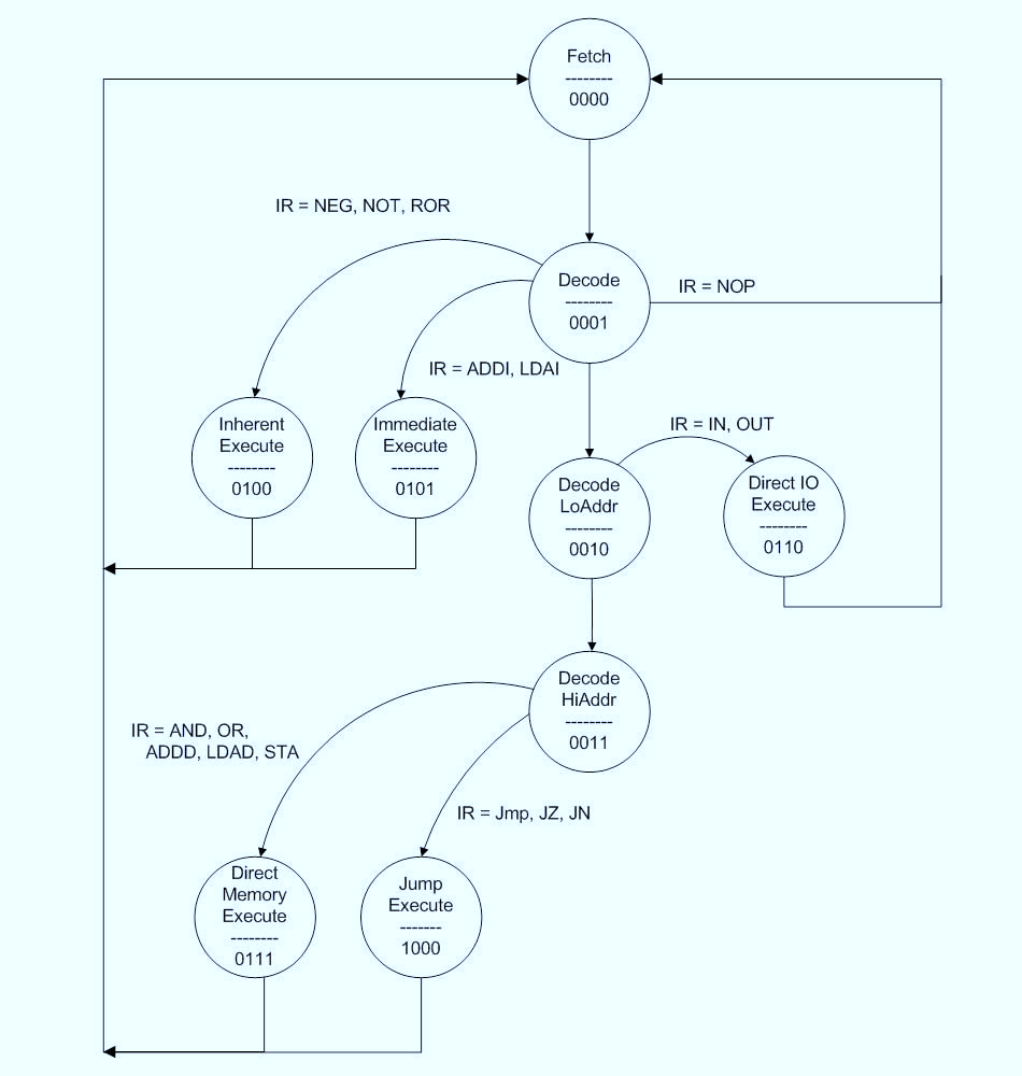

Figure 3. PRISM Controller State Diagram

The datapath, shown in Figure 4, uses both combinational and sequential logic components. An arithmetic logic unit (ALU) performs both arithmetic and logic operations when program instructions (and therefore the controller) demand them. The datapath also contains multiplexers to determine which 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 how to proceed.

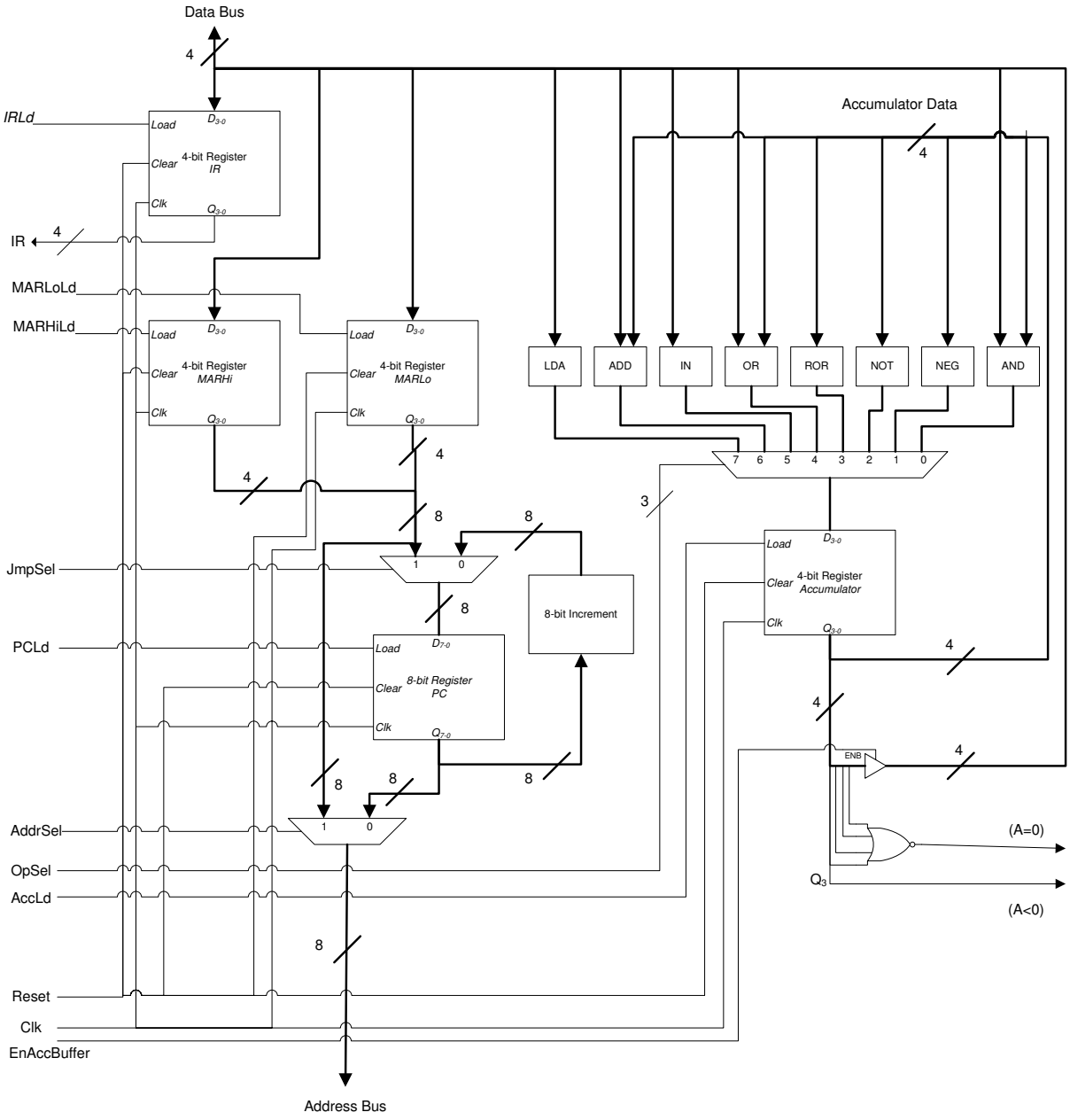

Figure 4. PRISM Datapath Block Diagram

PRISM's memory subsystem consists of one read-only memory (ROM) and five random-access memory (RAM) chips plus address decoding circuitry. With an 8-bit address bus, the PRISM address space consists of 2^8 (256) 4-bit wide memory locations. These correspond to hex addresses 00_{16} to FF_{16}. The system ROM is located at memory addresses 00_{16} to AF_{16} and holds the program to be executed by PRISM. Students are able to modify these memory locations through their HDL code before implementing their PRISM. RAM consists of five 16 x 4-bit RAM chips from $B0_{16}$ to FF_{16}.

PRISM's I/O subsystem provides the necessary interface to enter operands and return the results of program execution. PRISM has 4 input and 4 output I/O mapped addresses (known as ports).

The I/O map in Figure 5 lists the port addresses and their function. This subsystem provides our students' primary interface to PRISM after the computer is implemented onto FPGAs.

Address	Input	Output
0H	External Input Port	External Output Port
1H	External Input Port	External Output Port
2H	External Input Port	External Output Port
3H	External Input Port	External Output Port

Figure 5. I/O Address Map

How PRISM is Used in the Classroom

The goal of the last third of the introductory digital systems course is to pull the combinational and sequential design concepts introduced previously in the course into a single capstone example that will also help prepare students for follow-on digital courses. Through a series of lessons which combine lecture with lab time, the students begin to understand how PRISM works. In order to fully appreciate PRISM, students must first be introduced to assembly language programming and so they start by writing some simple assembly language programs which they then can run on a PRISM simulator (though this was originally a VISICOMP simulator since the architectures are the same).

Once they have mastered basic assembly language programming, students begin to implement PRISM in a series of structured lab events. They are first asked to write the hardware description language code for PRISM's ALU, using combinational circuit implementation skills learned earlier in the semester. They are then given a skeleton of PRISM's datapath and required to complete its functionality, including instantiating and interfacing their ALU code. This lab event culminates with a reverse-engineering exercise. A testbench is provided that exercises the datapath as if the controller was running a simple assembly language program that ends in an infinite loop. Students must simulate their datapath and use clues from incoming control signals and bus information to rewrite the assembly language program.

Students then begin an integrative exercise where they interface their datapath with provided code for the controller, memory, and I/O subsystems. They are asked to write a fairly complicated assembly language program implementing a stopwatch consisting of both minutes and seconds. This task mimics an earlier lab where students built the same stopwatch from modulo-16 counters. In order to complete the lab, they must fully implement PRISM with their program stored in the ROM memory segment, download it to Xilinx Spartan 2E demonstration boards, and demonstrate a working stopwatch using the switches and seven-segment displays provided on the boards.

Results

The introductory digital systems class that uses PRISM has a course objective that students be able to "apply the previous skills to analyze the various control signals and operation of a basic computer." We assess this course objective using questions on a final exam that remains relatively static from year to year. We do not have data going back to 1996 when VISICOMP

was introduced but students generally did very well in this area (with 2003 being an excursion). Initially, when PRISM was introduced in 2005, we saw a dip in the course objective's assessment with the course just barely reaching a "marginal" level (a rating assigned to assessment between a 70% and 77%). We believe this drop was due both to an immature implementation as well as the addition of HDL programming into the mix. Before PRISM, students were able to completely focus on the workings of a computer. The PRISM activities are much more stringent set of exercises requiring them to demonstrate competency in HDL programming and simulations before they can understand the details of how the computer is behaving. We have invested significant effort into upgrading the maturity of PRISM as well as building tools to help students more efficiently visualize what is happening. The results of this work are documented in a separate paper[6] but student performance has returned to fully meeting our course objective.

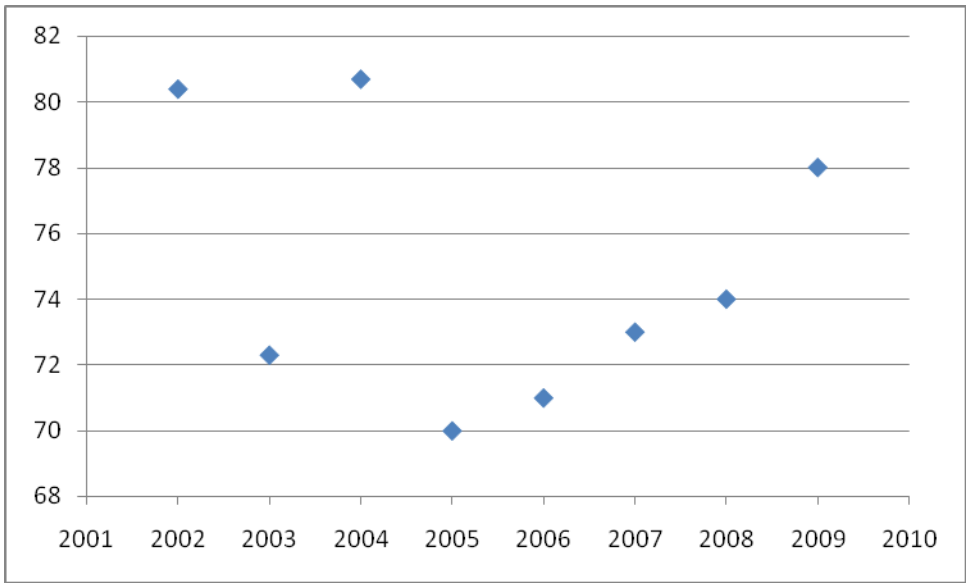

Figure 6. Student Performance on Computer Operations Course Objective Assessment

While it has taken a few years to gain a mature product that support students in learning how a basic computer works, the USAFA now has a training aid that provides a single capstone example that will also help prepare students for follow-on digital courses in more than just computer operation. PRISM exercises help to hone their HDL programming skills and give them added experience using simulation/implementation tools.

Conclusion

The USAFA continues to see the need for training aids to help students visualize and understand computer architecture concepts. PRISM provides a link between combinational and sequential logic circuits and computers while also giving students an integrative capstone experience in HDL programming and the use of simulation/implementation tools. Students' understanding of basic computer operations dropped off immediately after its introduction but has returned to fully satisfactory levels.

Bibliography

1. Moser, A. T., "Animated Simulator for 68000 Microcomputer Architecture," ASEE Annual Conference Proceedings, June 1995, pp 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. York, George, Fogg, Ruth D., "VISICOMP: The Visible Computer," ASEE Annual Conference Proceedings, June 1996.
4. Barrett, S. F., Pack, D. J., York, G. W. P., Neal, P. J., Fogg, R. D., Doskocz E. K., Stefanov, E. K., Neal, P. C., Wright, C. H. G. and Klayton, A. R., "Student-Centered Educational Tools for the Digital Systems Curriculum," ASEE Computers in Education Journal, Vol. IX, pp. 6 - 11, Jan - Mar 1999.
5. IEEE Computer Society, Association of Computing Machines (ACM), "Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering." 12 December 2004.
6. Peterson, B. and Clark, A., "PRISM: A Simple Simulation for Introduction of Assembly Language and Computer Architecture," submitted to ASEE Annual Conference Proceedings, June 2010.