

PSIM: A Processor SIMulator for Basic Computer Architecture and Operation Education

Michael Alexander Lusco and Charles E. Stroud

Department of Electrical and Computer Engineering
Auburn University, Alabama
luscoma@auburn.edu

Abstract — A Processor SIMulator (PSIM) for a basic stored program computer architecture is described which graphically displays the architecture while showing the detailed operation on a per clock cycle basis. The instruction set consists of twenty-five instructions that can be combined to execute many complex capabilities including conditional branching. The current version of PSIM includes an assembler for compiling assembly language programs to machine language code, the ability to display values in various formats including decimal and hexadecimal, and the ability to display and write to a file the contents of the program memory at any point in the simulation.¹

I. INTRODUCTION

A thorough understanding of computer architecture is essential to the greater understanding of software and computer science. Often it is difficult to understand the inner workings of a computer when studying large-scale or complex architecture systems. The motivation behind the Processor SIMulator (PSIM) is to help bridge the learning gap by providing a small-scale, limited instruction set architecture whose internal operations and components can easily be understood by individuals with little or no previous exposure to the internal operation of a processor or computer architecture.

The PSIM display is a graphical environment where the internal components and their interconnections are displayed so that the user can easily understand the signal paths that data and control take during operation. The simulator consists of several basic components including a simple arithmetic-logic unit (ALU), a number of user accessible and system registers, three multiplexors, an instruction read-only memory (ROM), and a data random access memory (RAM). In addition, the graphical user interface (GUI) provides additional options to the user; giving control over execution via control of the system clock as well access to the processor's input register and instruction assembler. The initial idea of the simulator is based on a DOS-based simulator described in [1] which was originally based on the simple computer architecture presented in [2]. The primary goal of the current effort was to improve upon the original idea in [1] by providing a Windows based GUI with greater user interactivity. However, additional instructions and architecture modifications were made as the project progressed to enhance the features and capabilities of the simulator in computer architecture and operation education.

This paper provides an introduction to the PSIM simulator software in Section II followed by an overview of the computer architecture used by the simulator in Section III. The instruction set is summarized in Section IV. PSIM is written in the .NET language C# and an overview of the organization and operation of the source code is given in Section V before the paper is summarized and concluded in Section VI. It should be noted that PSIM is currently maintained as an open source project located at <http://www.sf.net/projects/psimedu> and that it can be downloaded and run on any Microsoft Windows computer with Microsoft .NET Framework 3.5 installed.

II. THE SOFTWARE

The GUI is shown in Figure 1 and provides several options for the user. We begin with the button along the bottom of the GUI. The "Set Inputs" button allows the user to specify the contents of the INPUT register. The "Reset" button causes a global reset, resetting the program counter and all registers to 0. The remaining buttons provide user control of the clock. The "Run to Halt" button clocks the processor until it reaches a HALT command. In contrast, the "Clock" button causes the simulator to single step on a per clock cycle basis in order to provide a detailed look at how instructions are carried out over multiple clock cycles.

The menus along the top of the GUI provide several options including the ability to select the "Display Format" for all values on screen. By default all values are displayed in hexadecimal, but binary, octal, and decimal values are also available. The "Assembler" menu has a number of options in addition to the process of assembling an input program file. For example, there is an option that allows viewing the current contents of the instruction ROM or data RAM. There is also an option to save the contents of the instruction ROM as a binary file (in other words, the assembled machine language program). This file can then be loaded back into the instruction ROM by the assembler at any time. The "File" menu provides access to several additional options such as specifying contents of the INPUT register, specifying clock frequency, or to initiate one of the two additional dialogs in the GUI – the "Instruction Information Window" or "Bus Viewer". The "Instruction Information Window" provides the user a detailed look at the format and information associated with the instruction currently being executed. The "Bus Viewer", illustrated in Figure 2, is a very powerful tool for understanding the internal operation of the processor architecture. When the simulator is

¹ This work was supported in part by the National Science Foundation Grant CNS-0708962.

running, it gives a real time look at the data of user selected pins. Its view can be expanded from a twenty-five millisecond signal history up to a fifteen second signal history. Every signal selected to be viewed (by clicking the desired pin in the GUI) gets its own graph showing how it has changed over time. In Figure 2, one can see the clock oscillating and the program counter incrementing.

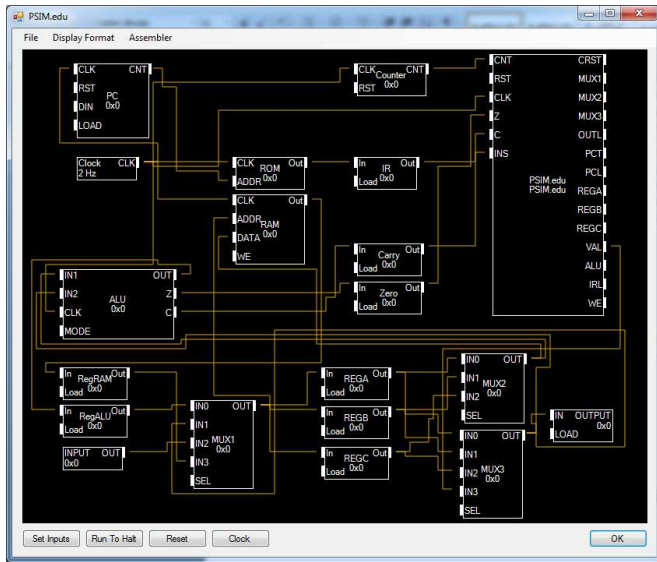


Figure 1 - Screen shot of PSIM GUI

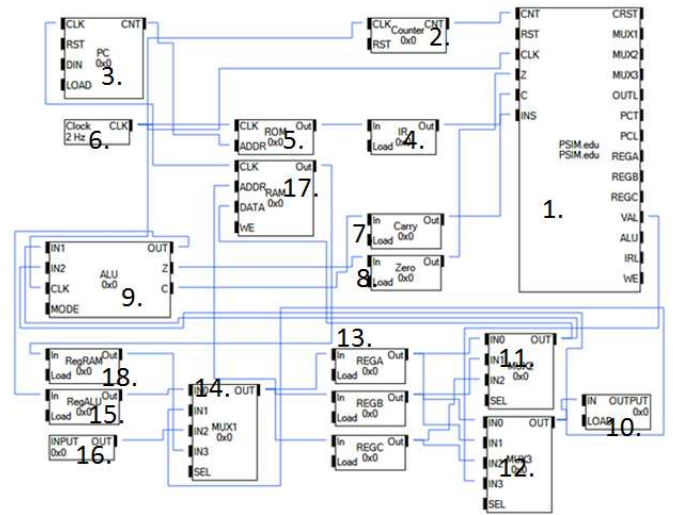


Figure 3 - The Simulator Layout

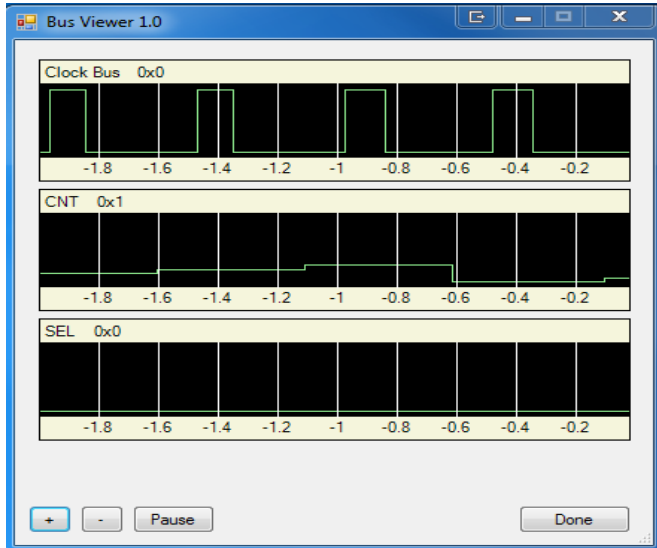


Figure 2 - Bus Viewer

III. THE ARCHITECTURE

The PSIM architecture is a multi-cycle architecture, meaning that every instruction takes multiple clock cycles to execute. It has several main components that make it a robust processor for simulation with many options. It includes the following major components:

- 256 × 16-bit Word Instruction ROM
- 256 × 16-bit Data RAM
- ALU with Zero and Carry flags provides ability to perform basic arithmetic and logic operations.
- Three user accessible 8-bit registers
- One 8-bit input register
- One 8-bit output register
- Support for JUMP and BRANCH instructions

A. Processor Architecture

Figure 3 shows the visual representation of the architecture used by the simulator where each component is labeled (numbered). A brief description is given related to each component's function in the architecture as follows:

1. PSIM Controller – Main PSIM state machine where each output controls a specific component allowing it to perform operations.
2. Clock Counter – This counter controls the instruction state and provides a way to track each step of an instruction being processed.
3. Program Counter (PC) – The PC provides the current memory address to be retrieved. The address holds the instruction to be processed by the controller. The PC can be incremented or updated by the PSIM controller.
4. Instruction Register (IR) – The IR register holds the last instruction retrieved from the ROM.
5. Instruction ROM – The Instruction ROM holds the compiled instructions to be executed.
6. Clock – A simple clock that outputs a 50% duty cycle wave form at a specified clock frequency.
7. Carry Register – The Carry Register holds the carry value from the last ALU operation.

8. Zero Register – The Zero Register holds the zero value from the last ALU operation.
9. ALU – The ALU handles all mathematic and logic operations.
10. Output Register – The Output Register holds the value on the output of the processor.
11. MUX 2 – Multiplexor (MUX) 2 selects a register output onto ALU input 1 depending on the instruction.
12. MUX 3 – MUX 3 selects the appropriate register or other component value to the Output Register.
13. Registers A, B and C – These are user accessible registers that hold user defined values.
14. MUX1 – MUX1 selects an input value for Registers A, B or C from the ALU, the INPUT register, the RAM, or a user value defined in the instruction.
15. ALU Register – The ALU Register holds the value from the last ALU operation.
16. Input Register – The Input Register holds any user specified value to be passed into the processor.
17. Data RAM – This memory can be written and read, and the user has the ability to save the contents.
18. RAM Register – This register stores the Data RAM output value.

B. Instruction Architecture

All PSIM instructions consist of a single 16-bit word that holds all the contents of the instruction and its operands as illustrated in Figure 4 (taken from the Instruction Information Window). The instruction is broken into two main bytes. The first byte holds the operation code (OP Code) or instruction code, the register to be operated on (REG) and a bit to denote if the value in the operand byte is a value (V) or an additional register address. All instructions fit this format which does sacrifice some size since it requires two bytes per an instruction and consequently a 16-bit instruction ROM; however, it is also makes the instruction much easier to understand from a complexity stand-point as it is clearly broken up and standardized among all instructions. This means it requires no special flags for instruction types making it easier for students to comprehend.

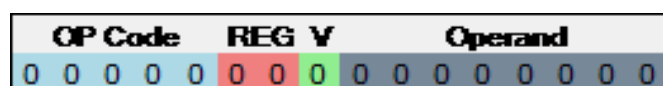


Figure 4 - Instruction Format

IV. THE INSTRUCTION SET

Though the maximum number of possible instructions is thirty-two instructions with the 5-bit OP Code, the simulator currently consists of twenty-five instructions ranging from basic operations to arithmetic operations to jump and branch operations. The complete instruction set of the PSIM architecture and simulator is summarized in Table 1.

Table 1 - Instruction Set

| Opcode | INST | Description | Example |
|--------|------|---|----------------|
| 0000 | HALT | Halts the processor | HALT |
| 0001 | LOAD | Loads a value or register into another register | LOAD %a %b |
| 1000 | LDCR | Loads the C flag into a register | LDCR %a |
| 1001 | LDZR | Loads the Z flag into a register | LDZR %a |
| 1011 | LDM | Loads a register with a value at a memory address | LDM %b 5 |
| 1100 | STM | Stores a register value into a memory address in a register | STM %a %b |
| 0010 | ADD | Adds a value to a register, storing it in the register | ADD %a 5 |
| 0011 | SUB | Subtracts a value from a register, storing it in the register | SUB %a 5 |
| 00100 | AND | Ands a value with a value from a register, storing it in the register | AND %a %b |
| 00101 | OR | Ors a value in a register | OR %a %b |
| 00110 | NOT | Inverts a value in a register | NOT %a |
| 00111 | XOR | XORs a value in a register | XOR %a 5 |
| 01000 | IN | Loads the value from the input register into a user register | IN %a |
| 01001 | OUT | Loads the value of a user register into the output register | OUT %a |
| 01010 | LSL | Logically shifts a value in a register left a number of times | LSL %a 5 |
| 01011 | LSR | Logically shifts a value in a register right a number of times | LSR %c 5 |
| 01100 | ROL | Rotates a value in a register left making its MSB its LSB | ROL %a 2 |
| 01101 | ROR | Rotates a value in a register right making its LSB its MSB | ROR %b 1 |
| 01110 | JUMP | Jumps to a memory address | JUMP @lbl |
| 01111 | BEQ | Branches to an address if two registers are equal | BEQ %a %, @lbl |
| 10010 | BNE | Branches to an address if two registers are not equal | BNE %a %b @lbl |
| 10011 | BRZ | Branches if the zero flag is 1 | BRZ @lbl |
| 10100 | BNZ | Branches if the zero flag is 0 | BNZ @lbl |
| 10101 | BRC | Branches if the carry flag is 1 | BRC @lbl |
| 10110 | BNC | Branches if the carry flag is 0 | BNC @lbl |

V. THE SOURCE CODE ORGANIZATION

The source code is organized into four main segments for ease of development as summarized below:

A. Main Executable

The main executable holds the GUI and the Instruction Information Window. It is the launch pad through which the rest of the program is referenced and called. All components and interconnections are defined in this main executable for graphics and communication.

B. Components

All of the basic logic components' behaviors are defined in the components module. Each component is defined by a list of input and output pins as well as a callback method which acts very similar to a process sensitivity list in VHDL. Any time an input pin in this "sensitivity list" changes, the callback is called and the component will update its output pins. In this manner, the components can easily mimic both sequential and combinational logic and are simple to understand as well as to design and implement.

C. Core

The core module defines the major components of the program. These include the assembler, bus viewer and memory viewer, as well as the major class objects such as pins, buses, and data types. All pins are connected together via busses. Data types such as Bit, Word, and Nibble are defined here for use across the entire program.

Currently data types are one of the more complex code segments of the program. When dealing with buses there is currently no "N-Bit" bus data type. This means anytime a bus is needed with a width that does not correspond to a data type that is already defined in the code; a new one must be created.

D. Drawing

The drawing module defines generic controls that can draw any logic component defined in the components module of the program. The drawing module consists of a control that wraps around a component and draws it, its ports, its name, and its current output value onto the screen. It also consists of the main view which accepts a number of these generic component blocks and draws them and their interconnections to the screen. All wires are drawn dynamically via the A* search algorithm. More information on A* can be found in [3]. For more information on the actual implementation used in PSIM see [4].

VI. OTHER SIMULATORS

For comparison purposes, both the "Relatively Simple CPU" simulator and the "Very Simple CPU" simulator were reviewed as educational tools for use in the classroom [6]. The "Very Simple CPU" is a simplistic 8-bit processor with a 6-bit address bus. It has a total of four instructions consisting of add, and, jump, and increment. It has a single accumulator register, and supports up to 64 bytes of memory [6]. The "Relatively Simple CPU" is a more robust system than the "Very Simple CPU" [7]. It is also an 8-bit processor but instead has a 16-bit address bus with support for up to 64 Kbytes of memory. In addition it has a total of sixteen instructions as well as an accumulator and a general purpose register. Its ALU also has a zero flag allowing for simple comparisons [7].

Compared to the PSIM program, these simulators are more simplistic in their simulations and capabilities. While the "Relatively Simple CPU" offers a much closer approximation to a realistic processor with its expanded instruction set, the PSIM still offers several advantages including support for shift and rotate instructions, an additional register, and support for an Input and an Output register for IO simulation. One advantage that both the "Relatively Simple CPU" and "Very Simple CPU" do have is that they are platform independent and can be run via any internet browser. Long term the PSIM can be converted to a Silverlight application which offers the same browser based functionality.

VII. SUMMARY

The PSIM program has evolved from its original inception in [1], but there are a number of improvements that can be made to the software so that its usefulness is expanded as an educational tool. First the architecture itself could be changed

so that it is easier to explain. Currently the system is setup such that the main controller is triggered on the clock edge while the remaining components are level sensitive. While this works well in the simulation environment, it is much more difficult to synthesize via a hardware description language such as VHDL or Verilog due to timing concerns. Therefore, future versions should be updated so that the controller is a combinational logic system with no reliance on the clock and all of the components are triggered on the same active clock edge.

Second a VHDL and/or Verilog implementation of the architecture would act as a supplemental aid to implementing the architecture, for synthesis in field programmable gate arrays (FPGAs) for example. This is dependent on the first modification being completed so that a simpler VHDL and/or Verilog model can be created, verified, and synthesized.

Long term goals are to expand the flexibility of the program into a more generic logic simulator. With this modification different architectures could easily be modeled and the differences between architectures could easily be demonstrated and/or evaluated. In addition, combinational logic circuits and other basic digital logic circuits could also be modeled. This would require some type of component descriptive script being added to the program.

PSIM is designed to be a classroom or self-learning tool. Its simplicity allows for a straightforward approach to learning the more complex aspects of computer architectures by allowing students to become familiar with the basic logic behind the more intricate computer systems. Furthermore, since it is an interactive tool, it allows students to quickly get acquainted with creating and executing assembly programs and how the underlying system executes those programs. By no means does the simulator exhaustively examine all possible aspects of computer architectures. Instead it demonstrates the basics to new people interested in the field and allows them to use PSIM as a platform for studying other more advanced architectures. PSIM is maintained as an open source project located at www.sf.net/projects/psimedu where more details can be found in the help manual [5].

REFERENCES

- [1] Stroud, C., "PSIM: Processor SIMulator (version 4.2)", July 23, 2003, available at www.eng.auburn.edu/~strource/ausim.html
- [2] Mano, M., *Computer Engineering Hardware Design*, pp. 280-292, Prentice Hall, 1988.
- [3] "A* Search Algorithm", available at en.wikipedia.org/wiki/A*_search_algorithm
- [4] Gustavo, F. "A-Star (A*) Implementation in C#" available at www.codeguru.com/csharp/csharp/cs_misc/designtechniques/article.php/c1252
- [5] Lusco, A., "PSIM.edu", available at www.sourceforge.net/projects/psimedu
- [6] Carpinelli, J. "The Very Simple CPU Simulator," Proc. Frontiers in Education Conf., 2002.
- [7] URL: www.awl.com/carpinelli; Companion web site for *Computer Systems Organization and Architecture*.