A Component-based Visual Simulator for MIPS32 Processors

Hessam Sarjoughian, Yu Chen, and Kevin Burger
sarjoughian@asu.edu, yu.chen8@hp.com, burgerk@asu.edu

Abstract - **Processor implementation and performance analysis are fundamental in computer architecture education. A processor can be described at different abstraction levels: a black box with inputs and outputs, the composition of RT (Register-Transfer) level components, the composition of gate level components, etc. Performance of a processor is impacted by factors such as clock cycle, programs, and components' propagation delays. With the traditional text-based educational material, teaching and learning of the processor implementation is difficult. Processor simulation offers an effective way for education through dynamic visualization and flexible experimentation. This paper presents a MIPS32 Processor Simulator that models the single-cycle, multi-cycle, and pipeline processors described in the classic textbook, "Computer Organization and Design: The Hardware/Software Interface" written by Patterson and Hennessy. The Simulator is developed in DEVSJAVA simulator, a realization of the Discrete Event System Specification with support for modeling parallel, hierarchical, and component-based systems. This simulator provides animation at RT-level during instruction execution, collects performance data (including cycle count, execution time, and instruction count), allows viewing components at desired abstraction levels, and is platform independent. The simulator can also be easily extended/reused to develop other processor types. Existing MIPS processor simulators do not provide sufficient support for the above mentioned features.**

Index Terms - Computer architecture education, MIPS processor simulation, discrete event modeling.

INTRODUCTION

Processor implementation and performance analysis are fundamental in computer architecture education. In many universities, MIPS (Microprocessor without Interlocked Pipeline Stages) processors are used to teach computer processors. A computer processor consists of many components that work concurrently with each other to execute a set of instructions. A processor can be described at different abstraction levels: a black box with inputs and outputs, the composition of RT (Register-Transfer) level components, the composition of gate level components, etc. Performance of a processor is impacted by factors such as clock cycle, programs, and components' propagation delays. With the traditional text-based educational material, teaching and learning of the processor implementation is difficult. Processor simulation offers an effective way for education through dynamic visualization and flexible experimentation.

A variety of computer architecture simulators exist. A simulator can be targeted for hardware components (e.g. [12]), instruction set architectures (e.g. [9] [15]), and computer systems including processors, memories, I/O, etc. (e.g. [10] [13]). At the same architecture level, different levels of details can be revealed. One simulator may only model the external behaviors of a system, while another may also model the interactions of the internal components. The focus of this work is on MIPS processor simulation. Existing MIPS processor simulators either do not model processors at the RT level [12] [14] [7], or lack details and accuracy for studying processor performance [1] [8].

In this paper, the concepts, design, and features of a MIPS32 processor simulator are presented. The simulator is implemented in Java using DEVSJAVA [6] environment, an implementation of the Discrete Event System Specification modeling theory [16][17]. It models the single-cycle, multi-cycle, and pipeline processor described in [11], provides RT level animation, collects performance data, supports multiple levels of abstraction, and can be easily reused or extended to model other MIPS processors.

BACKGROUND AND RELATED WORK

The MIPS Instruction Set Architecture (ISA) is widely used in embedded systems today. It belongs to Reduced Instruction Set Computer (RISC) family and has many versions, from the original MIPS I, through MIPS V, to the current MIPS32 and MIPS64. MIPS I, MIPS II, and MIPS32 support 32-bit registers and datapaths while the others are 64-bit implementations. Due to its simple and elegant design, it has been used in teaching undergraduate computer architecture courses at many universities.

In teaching computer architecture courses to undergraduate students, we have found that understanding of the processor implementation at RT level is challenging to most students. The purpose of this work is to create a simulator that helps students to understand different processor implementations and compare their performances. The simulator shall meet the following requirements: **(A)** It models the MIPS32 single-cycle, multi-cycle, and pipeline processor implementations described in [11] at the RT level, **(B)** It provides statistical data (including execution time, cycles per instruction (CPI), and cycle count) for performance comparison, **(C)** It provides run-time animation that demonstrates how the signals are passed among components at RT level during instruction execution, **(D)** It is platform independent so students can use the simulator on alternative hardware platforms. Before embarking on creating the DEVSJAVA MIPS32 simulator, we first examined the existing tools.

I. Existing MIPS Processor Simulators

WinMIPS64 [12], EduMIPS64 [14], and Simple MIPS Pipeline [7] [5] are simulators for pipeline processors. They focus on modeling functional aspects of pipeline stages instead of RT level logic components inside processors. EduMIPS64 is actually a re-design and re-implementation of WinMIPS64 in Java.

MiniMIPS [1] has the similar goal as this work. However, it models control units at a higher level instead of treating them as the composition of RT level components, such as shifters, Arithmetic Logic Unit (ALU) controls, and multiplexors. These lower level logic components are important for understanding processor implementations. Also, their attributes, such as delay, are needed to model processor performances. From the limited resources that we obtained from the authors, it seems that MiniMIPS does not provide animation, only provides cycle count as performance data, and is implemented in C and requires Unix machines.

WebMIPS [2] only models a pipeline processor. It models all the components inside the processor, and users can view each component's input and output data at a certain time by clicking on the component. However, the simulator does not show how the signals are sent and received among components during instruction execution. In terms of performance data, the total clock cycles are provided.

ProcessorSim [8] can be configured to model several datapath configurations for the MIPS32 single-cycle processor implementation [11], and provides animation that shows how instructions are executed inside processors. It provides good visualization but has some shortcomings. In ProcessorSim, only one component can send out messages at a time. However, components inside a real processor always work concurrently. ProcessorSim only shows effective execution paths for an instruction execution. That makes it easier for students to understand the processor implementations but hides some important details. ProcessorSim does not model component delays and thus can only support limited performance data. ProcessorSim is not based on any modeling and simulation theory, and therefore the simulator lacks a rigorous basis for defining the structures and behaviors of the MIPS components and their compositions. Thus, extending ProcessorSim to support other processor designs (e.g. MIPS64) is difficult.

Table I compares the existing MIPS processor simulators against the four criteria listed above as well as their implementations. Given the available literature, the list is not complete – for some simulators no code or executables are available and for others no paper or documentation can be found. In the table, the letter N means that the criterion is not met and the letter Y means the criterion is met. The table below shows that none of the simulators meets the all four criteria and only two of them satisfy two out of the four desired capabilities.

TABLE I
COMPARISON OF MIPS PROCESSOR SIMULATORS

Simulator	**A**	**B**	**C**	**D**	**Implementation**
WinMIPS64	N	Y	N	N	C++
EduMIPS	N	Y	N	Y	Java
SimpleMIPSPipeline	N	Y	N	N	HASE
MiniMIPS	Y	N	N	N	C
WebMIPS	N	N	N	Y	ASP, html
ProcessorSim	N	N	Y	Y	Java, XML

II. DEVSJAVA

DEVSJAVA [6] is a Java implementation of the Discrete Event System Specification (DEVS) modeling formalism [16]. The external view of a DEVSJAVA model is a component with input and output ports. A model receives messages through its input ports and sends out messages via its output ports. A message consists of a collection of port and value pairs, where port denotes the destination port and value encapsulates the data object. Ports and messages are the only means by which a model communicates with its external world. A DEVSJAVA model is either *atomic* or *coupled*. An atomic model is indivisible and can be used to build coupled models. An atomic model has eight essential elements: 1) input set, 2) output set, 3) state set, 4) internal transition function, 5) external transition function, 6) confluent function, 7) time advance function, and 8) output function. The internal transition function determines the state change upon the occurrence of internal events, while the external transition function determines the state change when external events arrive. Confluent function resolves the conflicts when internal and external events occur at the same time. The time advance function determines how long the system stays in the current state. The output function maps state set to output set. A coupled model consists of input and output ports, a finite number of (atomic or coupled) models, and couplings. The couplings link the ports together and are directed message channels.

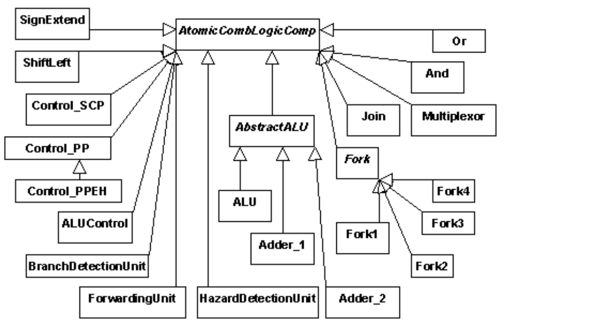
MIPS MODEL DESCRIPTION

In implementing the MIPS processor simulator, each *processor component* is modeled as a coupled model, and every *basic component* which is a part of the processor is modeled as an atomic model. These atomic models can be readily combined into coupled models. Additional atomic models are used to model special situations, such as when a bus is split into several buses and several buses are combined into one.

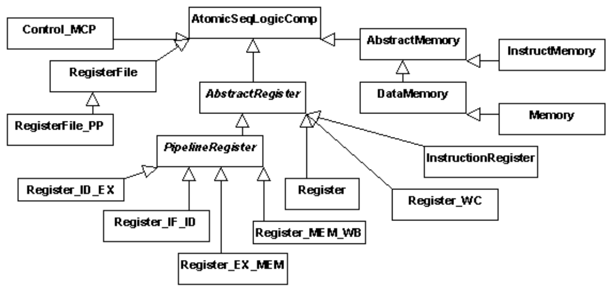
The major purpose of the simulator is to help students in learning processor implementations discussed in [11]. As such, the same assumptions from the textbook are used for the design of the simulator. For the single-cycle and multi-cycle processor, the lw, sw, add, sub, and, or, slt, beq, and j instructions are supported. For the pipelined processor, all these instructions are supported except j. For the beq instruction, the delayed branch is not modeled. Two types of exceptions are handled: arithmetic overflow and invalid instruction. Finally, edge triggered timing is used.

I. Basic Component Models

A computer processor consists of many logic components, which are either combinational or sequential. A combinational logic unit does not have memory elements, and its outputs solely depend on its current inputs. Some examples of combinational logic units are adders, shifters, and multiplexors. A sequential logic unit has memory elements, and its outputs depend on the current inputs and states. Registers, memories, and multi-cycle processor control are examples of sequential logic units. A generic logic component called AtomicLogicComp is developed as the basis for both combinational and sequential logic components.



(a) Family of the combinational logic model components



(b) Family of the sequential logic model components

FIGURE 1
BASIC COMPONENT MODELS

The class diagram for the basic combinational and sequential logic components are shown in Figure 1. These basic components are the specializations from the AtomicLogicComp model, which provides generic structures and behaviors for every basic element inside a MIPS processor. In most cases, the specification can be achieved by defining new input, output, and state sets and overriding the inherited functions (including external, internal, and confluent transition functions, and output as well as time advance functions). In the following, some models related to later discussions are briefly described.

AbstractALU component describes a general ALU with two operand inputs, one operation input, and three outputs: Zero, Overflow, and Results. The AbstractRegister models a register that updates its outputs upon the active clock edge, and has four sub-classes. The PipelineRegister is used for storing states of pipeline stages; The Register models a general register; Register_WC models a register with additional write control; and InstructionRegister models the register used for storing the current instruction. The RegisterFile models a set of 32 general registers. The Memory describes the memory shared by both data and instructions. The DataMemory and the InstrMemory model data memory and instruction memory, respectively.

II. Processor Component Models

Three processor models are developed for single-cycle, multi-cycle, and pipeline processor implementations. Due to the page limit, in the following, only the single-cycle processor model will be discussed. The SingleCycleProcessor model describes the single-cycle processor implementation [11] that executes each instruction within one clock cycle. Figure 2 shows its sub-models. A SingleCycleProcessor contains 7 atomic and 3 coupled models. The SCP_InstrFetcher handles instruction fetch and in turn has a Fork and an InstructMemory. SCP_AddrCalc computes the address for the next instruction. SCP_CtrlUnit provides control signals, and consists of a Control_SCP and an ALUControl. A coupled model can be displayed either as a black box or a white box as shown in Figure 3. Unlike a black box, a white box shows its parts and their couplings.

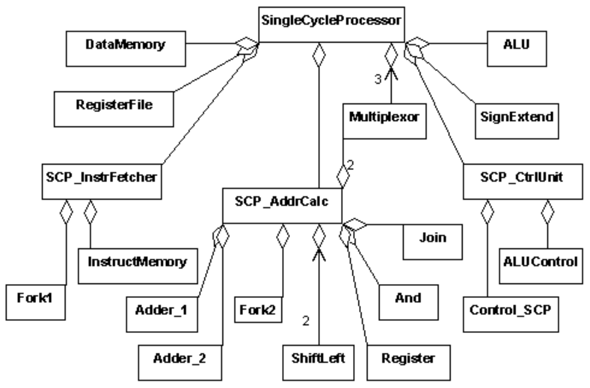


FIGURE 2
SINGLE CYCLE PROCESSOR MODEL

III. Experiment Setup

Besides the models mentioned above, several models are developed for setting up experiments. An ExperimentalFrame model is developed to allow different experiments to be conducted (see Figure 2). It reads simulation settings from a property file, which indicates the assembly file path, output file path, initial register values, and component timing parameters. The DEVSJAVA MIPS32 simulator with a set of assembly and property files is available [4]. An ExperimentalFrame has two sub-models, Clock and Transducer. Clock generates clock signals. Transducer observes the simulation and computes performance measurements, which include program execution time (ET), clock cycle time (CT), CPI, cycle count (CC), and instruction count (IC). Five coupled models, SCPEF, MCPEF, MCPEHEF, PPEF, and PPEHEF are developed for conducting experiments. Each of these models has one AbstractProcessor and one ExperimentalFrame instance. To set up and run a processor simulation, the following steps can be followed: (1) Create an assembly file and put its path in the property file, (2) Set initial register values, output file path, and component timing parameters in the property file, (3) Optionally, specify when a component should send outputs in the property file, upon receiving any effective inputs or when the new outputs are different from the old ones, (4) Start the simulator, choose which model (SCPEF, MCPEF, MCPEHEF, PPEF, or PPEHEF) to run, observe the component interactions, and adjust simulation speed when needed, (5) Check the results in the specified output file when the simulation ends.

DISCUSSION AND RESULTS

The DEVSJAVA MIPS32 processor simulator provides visualization, performance data, and reusability which have been partially supported by various processor simulators. Next, each of the features will be briefly described.

I. Visualization

A user can view the entire implementation of a processor through the simulator. Figure 3 shows the single-cycle processor experimental frame. A component is represented by a rectangle, with input ports on the left and output ports on the right. Couplings are represented by lines. The binary strings represent the outputs from the models. The current simulation time is displayed at the bottom, 200 picoseconds (ps), along with a slider that allows adjusting simulation speed at any time during the simulation. Three control buttons for stepping through, running through, and restarting the simulation are also provided. The simulator can be executed in soft real-time or logical time for many hundreds of millions of simulation steps.

Figure 3 has six coupled models. Three of them (SCEPF, ExperimentalFrame, and SingleCycleProcessor) are shown as white boxes, so their inner components can be viewed. The other three (AddressCalculator, InstrFetcher, and ControlUnit) are shown as black boxes. A user can set the view settings (black or white box) for each coupled model. Each individual coupled model, such as SCP_CtrlUnit, can also be simulated, as shown in Figure 4. A user can click on the input ports to inject one or more inputs into the model, and click "step" button to observe the change of the states and outputs. An input that is injected into a model consists of a port name, data value, and elapse time (e.g., port: Opcode, value: 100011, e:0.0). The elapsed time is a positive value that is used to schedule the input to be injected e units of time after the current time, shown by *simulator clock*).

Each basic component inside a processor, such as ALUControl, can be simulated as well, as shown in Figure 5. By momentarily placing the mouse on the component, a tooltip will display all user defined model states that are selected for viewing. Among the related work, only ProcessorSim provides visualization comparable to this work. However, the visualization provided by ProcessorSim does not reflect the concurrency accurately.

II. Performance Data

Our MIPS processor simulator can also be used in comparing processor performances. In the following, a simple example is considered for studying the performances of the single-cycle, multi-cycle, and pipeline processors. A processor's performance is determined by many factors, including processor implementations, component delays, clock cycle time, and applications used. In this experiment, we compared the single-cycle, multi-cycle, and pipeline processor, with the same timing parameters. Two assembly files were executed on each processor, and the simulation results are shown in Table II. Not to our surprise, the pipeline processor has better performance than the others. Contrary to the general belief, the single-cycle processor outperforms the multi-cycle processor in this case. In the experiment, the clock cycle time is a dominant performance factor. For the single-cycle processor, each instruction takes 1 clock cycle to finish (i.e., 600 ps). For the multi-cycle processor, an instruction takes between 3 and 5 clock cycles to finish, which equal 600 to 1000 ps. Hence, independent of what applications are used, the multi-cycle processor will always take no less time to execute the applications. Among the related work, only WinMIPS64, EduMIPS64, and Simple MIPS Pipeline provide similar performance data.

TABLE II
COMPARISON OF PERFORMANCE DATA

	Single Cycle Processor		**Multi-Cycle Processor**		**Pipeline Processor**	
	File1	File2	File1	File2	File1	File2
ET(ns)	120.6	2.4	153	3	48.8	1.8
CT(ps)	600	600	200	200	200	200
CPI	1	1	3.806	3.75	1.214	2.25
CC	201	4	765	15	244	9
IC	201	4	201	4	201	4

III Reusability

In our implementation, each RT-level physical component in the CPU is mapped to a DEVSJAVA model. Implementing a processor is simply developing the sub-models and composing them together. Thus, creating a DEVSJAVA processor model is similar to building a real processor using hardware. As real MIPS processors share many common components (e.g. ALU, multiplexor, and register), their corresponding DEVSJAVA models may also be reused in other DEVSJAVA processor models. These models are loosely coupled due to the DEVS component-based framework, which makes it easier to modify or extend the existing ones to model other MIPS processor implementations. Since most of the related simulators (such as those examined in Section II, except the Simple MIPS Pipeline) lack formal modeling and simulation principles and methods, they are difficult to reuse and/or extend.

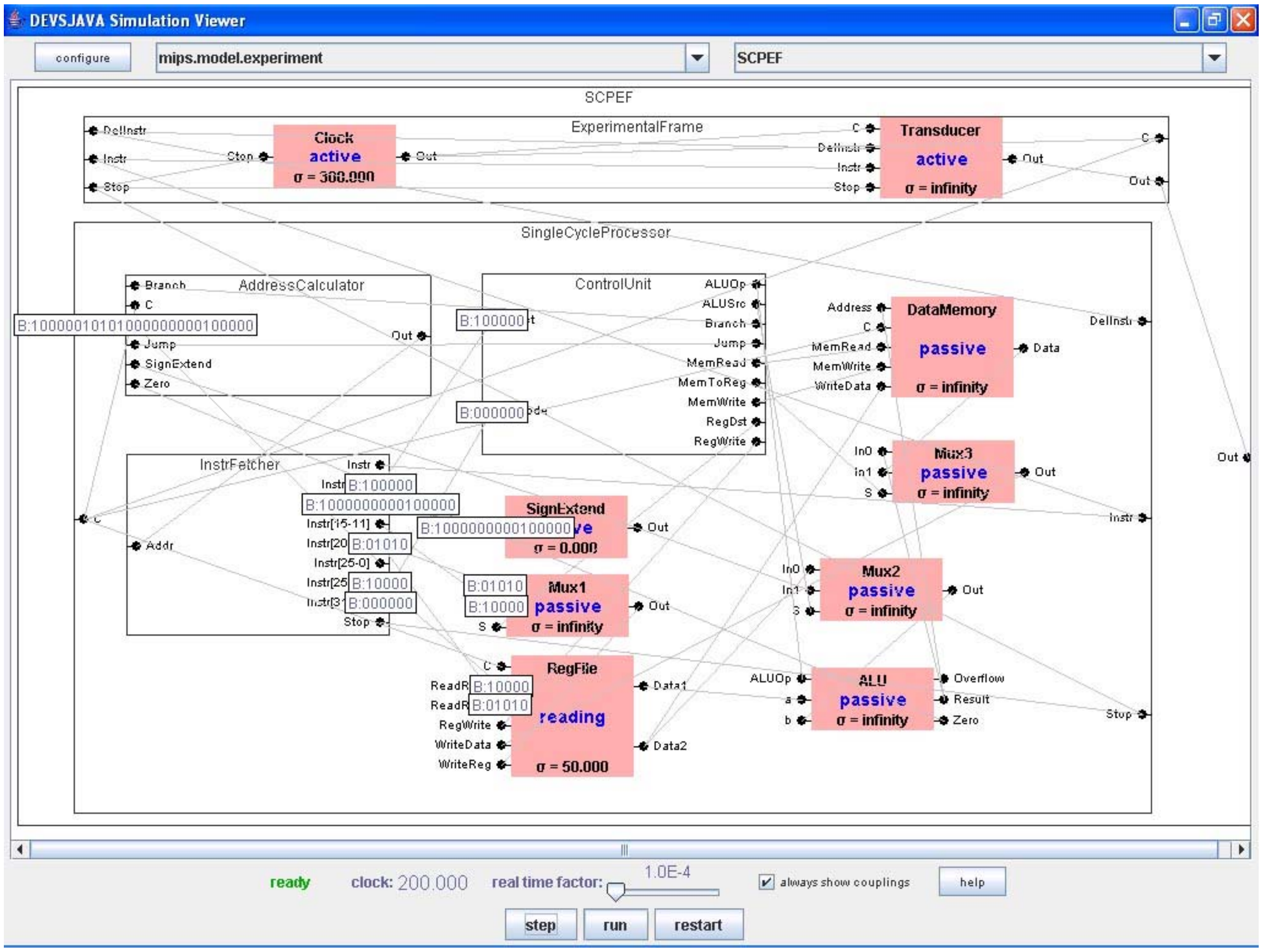
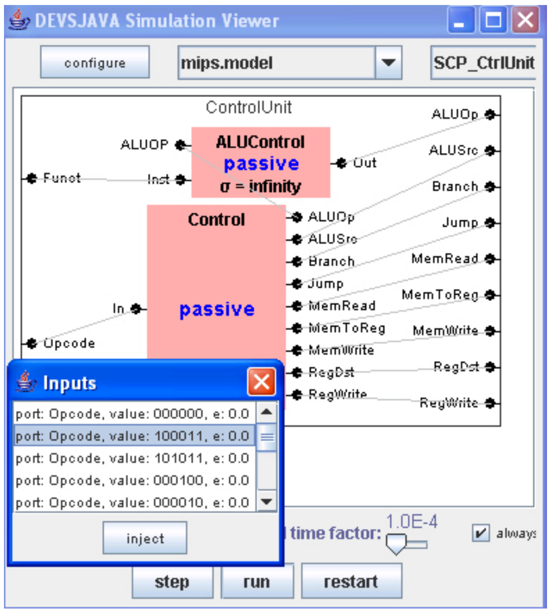
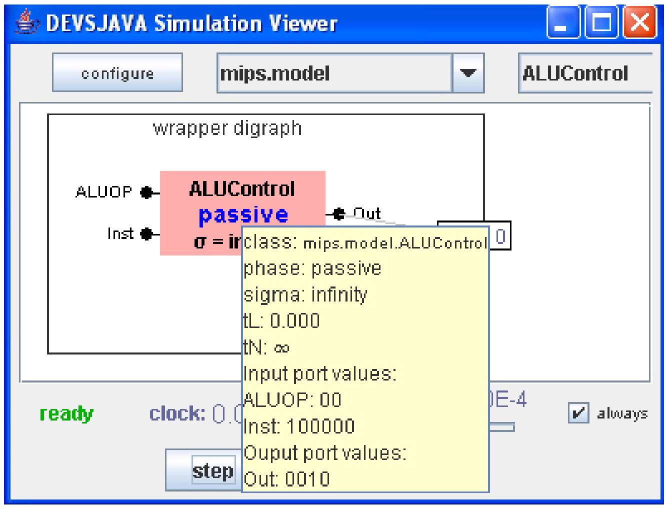


FIGURE 3
SINGLE-CYCLE PROCESSOR EXPERIMENTAL FRAME



port: Opcode, value: 000000, e: 0.0
port: Opcode, value: 100011, e: 0.0
port: Opcode, value: 101011, e: 0.0
port: Opcode, value: 000100, e: 0.0
port: Opcode, value: 000010, e: 0.0

FIGURE 4
SCP_CONTROLUNIT



class: mips.model.ALUControl
phase: passive
sigma: infinity
tL: 0.000
tN: ∞
Input port values:
ALUOP: 00
Inst: 100000
Ouput port values:
Out: 0010

FIGURE 5
ALUCONTROL

CLASSROOM USE

The main purpose behind the development of the DEVSJAVA MIPS32 simulator is to aid student learning and teacher instruction. The simulator's capability is being assessed in the "Computer Organization & Assembly Language Programming" course that is offered in the Computer Science and Engineering department at Arizona State University. In Spring 2008, the simulator is introduced to 43 students in this course. One course assignment is to ask students to predict the status of the single-cycle and multi-cycle control unit outputs for specific MIPS instructions, and then to use the simulator to validate their answers. At the end of the semester, a survey will be conducted to evaluate the impact of the simulator. The survey questions focus on (i) the visual support for understanding the logical and time-based dynamics of the processors, (ii) discovering errors in the components' operations, and (iii) comparing the different processor type performances. Given the simulator requirements described earlier, the survey also includes questions on simulator features that are found most useful, somewhat useful, or not useful in learning the MIPS processor principles. The survey also seeks to identify the features that could make the simulator more valuable. The results of the survey will be included in the final paper. A brief evaluation of the simulator by two instructors and teaching assistants is also planned to be included in the final paper.

CONCLUSION

Computer processor principles are essential to the computer science and engineering curricula. The MIPS processors serve as the primary means for teaching across many universities. Processor simulations can significantly improve the learning and teaching experiences of the students and instructors. The Java WebStart enabled DEVSJAVA MIPS32 processor simulator [4] [6] presented in this paper has a theoretical foundation and supports the single-cycle, multi-cycle, and pipeline processors described in [11]. It encourages virtual hands on experience through animation and provides simple ways to set up experiments and collect performance data. This simulator is publicly available and the full source code is free for education and research purposes.

REFERENCES

[1] E. Z. Bem and L. Petelczyc. MiniMIPS: a simulation project for the computer architecture laboratory. In SIGCSE '03, pages 64–68, NY, USA, 2003. ACM Press.

[2] I. Branovic, R. Giorgi, and E. Martinelli. WebMIPS: A new web-based MIPS simulation environment for computer architecture education. In Workshop on Computer Architecture Education, 31st International Symposium on Computer Architecture, pages 93–98.

[3] WEBMIPS, 2004, http://www.dii.unisi.it/˜giorgi/WEBMIPS/, June.

[4] Y. Chen and H. S. Sarjoughian. DEVS MIPS Processor Simulator. 2007, http://acims1.eas.asu.edu/WebStarts/MIPSProcSim.html, Dec.

[5] P. S. Coe, F. W. Howell, R. N. Ibbett, and L. M. Williams. Technical Note: A hierarchical computer architecture design and simulation environment. ACM Trans. Model. Comput. Simul., 8(4):431–446, 1998.

[6] DEVSJAVA. Arizona Center for Integrative Modeling and Simulation. http://www.acims.arizona.edu, 2003.

[7] D. Dolman. Computer architecture: Visualization of a simple MIPS pipeline. http://www.icsa.inf.ed.ac.uk/research/groups/hase/models/mips/, March 2007.

[8] J. Garton. ProcessorSim - A visual MIPS R2000 processor simulator. http://jamesgart.com/procsim/, August 2005.

[9] J. Larus. SPIM - A MIPS32 simulator. http://pages.cs.wisc.edu/˜larus/spim.html, June 2007.

[10] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. SIGARCH Comput. Archit. News, 33(4):92–99, 2005. http://www.cs.wisc.edu/gems/.

[11] D. A. Patterson and J. L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, San Fransisco, CA, USA, 3rd edition, August 2004.

[12] M. Scott. WinMips64. http://www.computing.dcu.ie/˜mike/winmips64.html, June 2006.

[13] SimpleScalar. http://www.simplescalar.com/, Aug 2004.

[14] The EduMIPS64 Team. EduMIPS64. http://www.edumips.org/, June 2007.

[15] K. Vollmar and P. Sanderson. MARS: An education-oriented MIPS assembly language simulator. In SIGCSE '06, pages 239–243, NY, USA, 2006. ACM Press.

[16] B. P. Zeigler, H. Praehofer, and T. G. Kim. Theory of Modeling and Simulation. Academic Press, New York, 2000.

[17] B. P. Zeigler and H. S. Sarjoughian. Introduction to DEVS modeling & simulation with JAVA: Developing component-based simulation models. Retrieved Jul, 21, 2006, from http://www.acims.arizona.edu/, Aug 2003.

AUTHOR INFORMATION

Hessam Sarjoughian, Assistant Professor, School of Computing and Informatics, Arizona State University, Tempe, Arizona, sarjoughian@asu.edu.

Yu Chen, Software Engineer, Indigo Digital Press Division R&D, Hewlett Packard, Boise, Idaho, yu.chen8@hp.com.

Kevin Burger, Lecturer, School of Computing and Informatics, Arizona State University, Tempe, Arizona, burgerk@asu.edu