An Open and Platfom-independent Instruction-Set Simulator for Teaching Computer Architecture

VINCENZO CATANIA, DAVIDE PATTI, MAURIZIO PALESI, ANDREA SPADACCINI
University of Catania
Dipartimento di Ingegneria Elettrica, Elettronica e Informatica
CATANIA, ITALY
{vcatania,dpatti,mpalesi,aspadaccini}@dieei.unict.it

FABRIZIO FAZZINO
NVIDIA Inc.
BRISTOL, UNITED KINGDOM
ffazzino@nvidia.com

Abstract: Instruction-set Simulators (ISS) are commonly used in any computer architecture course as primary tools for supporting the teaching activity. Although there are several simulation platforms for educational purposes, the lack of an unified and integrated platform often forces educators to use a range of heterogeneous tools to cover the different topics of the syllabus. This paper presents EduMIPS64 a free, visual, and platform-independent MIPS64 Instruction-Set Simulator designed as a learning aid for topics like instruction pipelining, hazard detection and resolution, exception handling, interrupts, and memory hierarchies. Its dual execution mode – stand-alone application and web applet – allows for inclusion in distance learning courses.

Key–Words: Instruction-set Simulator, Graphic User Interface, MIPS64, Assembly Interpreter, Pipeline, Stall, Hazard, Cache, Delay slot, Exception, Interrupt

1 Introduction

Computer Architecture topics represent one of the most important disciplines which characterize the curriculum of a computer engineer. The canonical topics treated in any Computer Architecture course include instruction-set architecture (ISA), pipelining, memory hierarchies and input/output (I/O). All of them represent the main basis which is required to be understood for anyone working with architecture or hardware, including architects, chip and computer system engineers, and compiler and operating system engineers.

The MIPS64® architecture [1] represents one of the most widespread reference architecture in several computer architecture courses. The simplicity which characterizes such an architecture along with its flexibility in supporting a range of architectural and microarchitectural features for performance improvement, make it an ideal platform for understanding basic principles of computer architecture like ISA, pipelining, memory hierarchies, *etc.*. [2].

To better metabolize such topics, the theoretical study must be coupled with practical activities which are usually done in a laboratory using Instruction-set Simulators (ISSs). Such activities are mainly directed to demonstrate specific phenomena or behavior, and provide experiences with measuring and studying desired characteristics.

Many ISSs, both free and commercial, are commonly used in both academia and industry. A good survey on simulators currently available and suitable for teaching courses in computer architecture and organization is presented by Nikolic *et al.* [3]. Many of the simulators discussed in [3] are mainly designed for application evaluation purposes and architecture customization through instruction-set specialization/extension and micro-architecture parametrization. That is, using such simulators the user can assess the impact on application performance when several architectural parameters (*e.g.*, cache configuration, number of functional units, branch prediction schemes, *etc.*) are changed.

This paper focuses on a different aspect which characterizes any ISS, that is its ability of making practical the theoretical topics dealt in the course. We think that there is a lack of tools (ISSs in particular) which have been specifically designed with the aim of supporting students of computer architecture course in understanding the fundamentals of the basic topics of assembly programming, basic pipeline, and memory

hierarchies. In this context, visual representation of the concepts introduced in the course, platform independence, distance learning support and freely availability of the tools are key factors which determine the effectiveness of the teaching armaments.

In this paper we present a MIPS64 simulator, called EduMIPS64 [4]. The project was originally started in order to make a simulator similar to WinMIPS64 [5] available to users of every platform, but it progressively went beyond the original idea, growing up quickly and extending the original functionalities of WinMIPS64 with new and important (under the educational viewpoint) features like the implementation of an interrupt and exception handling system. Further, the choice of implementing the simulator in Java gave to the EduMIPS64 project two important advantages over WinMIPS64: first, the possibility of using the simulator across different operating systems and platforms, feature of fundamental importance when dealing with heterogeneous environments (different universities, teaching laboratories, students notebooks, *etc.*) with no deployment issues. Second, the availability of a Java-applet web interface to the simulator enables the support for new approaches in teaching courses such as distance learning. Moreover, thanks to its licensing policy (GNU General Public License) EduMIPS64 itself becomes an experimental laboratory, where every user is able to study the source code, do research activities and experiments, modify it and share the contributions back with the community. Moreover, the application is available in multiple languages and it is easy to add support for new languages, making it a good teaching aid for courses held in countries in which English is not the major spoken language.

The rest of this paper is organized as follows. Section 2 provides a brief overview on ISSs mainly used in educational scenarios and describes how EduMIPS64 compares to them. The basic features of EduMIPS64 and its user interface are presented in Sections 3 and 4 respectively. The assessment survey and evaluation is reported in Section 5. Finally, in Section 6 we draw our conclusions and discuss possible future developments.

2 Comparison with Other Simulators

This section provides a brief overview of the most common simulation platforms used for teaching Computer Architecture topics and analyzes how EduMIPS64 compares to them. In particular, we focus on simulators designed for beginners courses on computer organization and architecture. Thus, we will exclude simulators addressing topics such as microarchitecture design at RT-level, custom instruction-set, or tools strictly focused on architectures that demonstrated to be not so effective for teaching the basic principles of a computer architecture (e.g. not RISC-like architectures like x86).

Table 1: Coverage of Computer Engineering Topics

Knowledge Unit	Coverage
Fundamentals of Computer Architecture	87.50%
Memory system organization	14.28%
Interfacing and communication	0.00%
Device subsystems	0.00%
Processor systems design	10.00%
Organization of the CPU	72.72%
Overall	30.75%

The authors of [3] analyze 28 simulators, classifying them into 2 different groups: those that allow the user to design a computer environment and those that allow the simulation of a target architecture. EduMIPS64 fits in the second category, so in this section we will compare it to the most representative simulators described in the survey. The evaluation criteria proposed by them are of two types: teaching topics coverage and simulation features. The first kind of criteria is meant to assess how much the simulators are suited to teach the topics of Computer Engineering, according to the IEEE Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [6]; the second kind is meant to compare some specific features of the simulators not strictly related to teaching topics. Each key area defined in the first group of criteria is composed by several topics, and each simulator is awarded 1 point if it supports that topic directly, 0 points if it is not supported and 0.5 points if it is not directly supported but support can be achieved in some way.

We followed the subjective evaluation protocol described in [7] to see how EduMIPS64 compares to the other simulators presented in the survey, and the results are shown in Table 1. EduMIPS64 is not meant to be a comprehensive tool for Computer Engineering education, but it aims to be a valuable aid for education in Computer Architecture topics. This means that some of the topics contained in [6] are not covered at all (Interfacing and Communication, Device subsystems) or very briefly (Processor systems design). Instead, the simulator is focused on topics related to the basic concepts of computer architecture and CPU organization, usually taught in introductory Computer Architecture courses, and the coverage values for those areas obtained through this evaluation suggest us that EduMIPS64 is a solid teaching aid for

Table 2: Evaluation results for the Simulation Features

Feature	Evaluation result
Scope and complexity	Basic Architecture
Design Support	No
Visual Presentation	Yes
Simulation Flow	Interactive
Simulation Level	Clock level
Implementation Details	Yes
Distance Learning	Yes

those courses. The survey also compares simulators according to their features. Table 2 shows the results of the evaluation of EduMIPS64 according to those criteria.

In addition to the simulators compared in [3], there are three other simulators that can be directly compared to EduMIPS64. The first one is SPIM [8], an open source MIPS32 simulator with a simple user interface for interactive execution and debugging, called QtSpim.

The other simulators that have been a great source of inspiration for the design and development of EduMIPS64 are WinDLX [9] and WinMIPS64 [5]. They have a very good user interface, with a visual dynamic representation of the CPU pipeline, that EduMIPS64 borrowed. EduMIPS64 tries to take the best from those three simulators, implementing an unique feature set: it is a cross-platform visual MIPS64 CPU simulator released under a free license (GNU General Public License version 2) and suitable for web-based distance learning. SPIM implements the MIPS32 ISA, is open source and is cross-platform, even if a recompilation of the source code might be required, but its GUI does not offer a graphical representation of the execution of the program apart from the memory and register file contents and input/output. WinDLX and WinMIPS64 offer a richer visual representation of the CPU state, but they are not cross-platform and they are not released under a free license.

Finally, none of the three simulators supports distance learning activities, an important requirement that also has the nice side effect of making the simulator's installation almost effortless. In fact, the only dependency to satisfy in order to use EduMIPS64 is the presence of a Java Runtime Environment that supports Java 5 or later. There are some features of each of those three simulators that have not yet been implemented in EduMIPS64, but as described in the Computer Engineering topics coverage analysis, all the basic features are already implemented and the additional benefits described in the latter part of this section are compelling enough to justify its adoption in introductory Computer Architecture courses.

3 Features of EduMIPS64

In this Section the main features of EduMIPS64 will be discussed.

3.1 Pipeline architecture

EduMIPS64 implements a parallel pipeline, composed by five stages: Instruction Fetch (IF), Instruction Decode (ID) Execution / Address Calculation (EX), Data memory access (MEM), Write Back (WB).

A colour is associated to each stage of the pipeline, so during the execution it is easy to visually recognize the parts of the user interface that represent information associated to each stage.

The status of the pipeline during program execution is represented in the Pipeline frame of the simulator interface, described in Section 4.3.

3.2 The Instruction Set

EduMIPS64 offers to users a representative and large subset of the MIPS64 ISA. Table 3 presents an overview of the instructions implemented in EduMIPS64, according to the taxonomy presented in [1].

Some instructions have been implemented even if they do not belong to the MIPS64 instruction set because other simulators implemented them, and we wanted to make it easier for users of other simulators to adopt EduMIPS64. Examples of such instructions are: `DADDUI`, `BNEZ`, `BEQZ` and `HALT`.

Instructions are encoded by the simulator with bit-level accuracy, and follow the formats described in [1] for I-Type, J-Type and R-Type instructions.

3.3 Forwarding

Forwarding is a technique that allows the reduction of stalls introduced by dependencies in the data used by the instructions. EduMIPS64 allows users to run their programs with or without forwarding, so that they can experiment with it and understand its impact to the program flow.

3.4 Exceptions

The simulator supports synchronous exceptions, namely Division by zero and Integer overflow, and the user can choose the exception handling policy in the Settings dialog (Section 4.3). Exceptions can be silently ignored (masked); if they are not masked, a pop-up window notifies the user. The user can further

Table 3: The EduMIPS64 Instruction Set

Type	Implemented instructions
Aligned CPU Load/Store Instructions	LB, LBU, LD, LH, LHU, LW, LWU, SB, SD, SH, SW
ALU Instructions with an Immediate	ADDI, ADDIU, ANDI, DADDI, DADDIU, LUI, ORI, SLTI, SLTIU, XORI
Three-operand ALU Instructions	ADD, ADDU, AND, DADD, DADDU, DSUB, DSUBU, OR, SLT, SLTU, SUB, SUBU, XOR
Shift Instructions	DSLL, DSLLV, DSRA, DSRAV, DSRL, DSRLV, SLL, SLLV, SRA, SRAV, SRL, SRLV
Multiply/Divide Instructions	DDIV, DDIVU, DIV, DIVU, DMULT, DMULTU, MFHI, MFLO, MULT, MULTU
PC-region Unconditional Jump	J, JAL, JALR, JR
PC-relative Conditional Branch	BEQ, BNE, BGEZ
Miscellaneous	SYSCALL, BREAK, NOP

Table 4: File Opening Flags

Name	Value	Effect
O_RDONLY	$0x01$	Open the file in read-only mode
O_WRONLY	$0x02$	Open the file in write-only mode
O_RDWR	$0x03$	Open the file in read/write mode
O_CREAT	$0x04$	Create the file if it does not exist
O_APPEND	$0x08$	In write mode, append written text at the end of the file
O_TRUNC	$0x10$	In write mode, delete the content of the file as soon as it is opened

decide if synchronous exception must be considered fatal; in this case, the program will terminate when such an exception occurs.

3.5 System calls

EduMIPS64 implements six system calls, that can be accessed by the programmers through the SYSCALL instruction. The system calls that need input parameters expect their address to be stored in the register R14. All the system calls return -1 in case of failure, except for SYSCALL 0, that cannot fail; their return value is stored in register R1.

Standard input, standard output and standard error are represented, respectively, by file descriptors 0, 1 and 2; writing to standard output or to standard error is implemented by the GUI Input/Output window, while reading from standard input is implemented by using an input dialog (see Section 4.1). A short description of each SYSCALL is as follows:

- SYSCALL 0, *exit()*, stops the execution of the simulator;
- SYSCALL 1, *open()*, opens a file, according to the user-specified combination of the flags shown in Table 4, and returns the file descriptor of the opened file;
- SYSCALL 2, *close()*, closes an open file;
- SYSCALL 3, *read()*, reads data from a file and stores it in memory;
- SYSCALL 4, *write()*, writes data to a file;
- SYSCALL 5, *printf()*, prints formatted data to standard output, replacing each placeholder (%s for string data, %i or %d for integers) with a parameter stored by the user in memory following the format string.

For more details on the SYSCALL syntax, please refer to [10].

3.6 Cache simulation

EduMIPS64 does not provide a built-in cache simulator. Instead, it offers two features that make it easier to work with external cache simulators:

- *Trace file export*, that saves to an external trace file all the memory accesses attempted by the executed program;
- *DineroIV front-end*, that makes easier to feed the trace file to DineroIV [11], a popular cache simulator, and to get the simulation results without leaving the EduMIPS64 user interface.

3.7 Source files format

There are two sections in a source file, the *data* section and the *code* section, introduced respectively by the .data and the .code directives.

Multiple spaces and tabs can be used throughout the source code to improve its readability, as the parser ignores multiple spaces and considers them as a single

space. Comments are introduced using the ";" character; everything that follows that character will be ignored, so comments can be appended to a code line or inserted on a stand-alone line.

Labels can be used in the code to reference a memory address or an instruction. They are case insensitive. Only a label for each source code line can be used.

3.7.1 The `.data` section

The *data* section contains commands that specify how the memory must be filled before program execution starts. The general form of a `.data` command is:

```
[label:] .datatype val1 [, val2 [, ...]]
```

where `label` is the optional data label, and `valN` are the N values that need to be stored in memory. EduMIPS64 supports the numeric data types represented in Table 5.

Table 5: Basic data types

Type	Directive	Size (bits)
Byte (*B*)	`.byte`	8
Half word (*H*)	`.word16`	16
Word (*W*)	`.word32`	32
Double Word (*D*)	`.word`	64

In addition to the basic data types, three other directives can be used in the `.data` section. The `.space` directive is used to allocate some empty space that can later be used by the program; the `.ascii` directive is used to put in memory character strings containing any printable ASCII characters and C-like escaping sequences; the `.asciiz` directive behaves exactly like the `.ascii` command, with the difference that it automatically ends the string with a null byte.

3.7.2 The `.code` section

The *code* section contains the symbolic instructions that the simulator will encode and then execute when the program is started. The general form of a `.code` command is:

```
[label:] instr [p1 [, p2 [, p3]]]
```

where `instruction` is the name of the instruction and `p1`, `p2`, and `p3` are the three parameters. The number and type of parameters is specified by the syntax of each instruction.

Instructions can take three types of parameters:

- *Registers* a register parameter is indicated by an uppercase or lowercase "r", or a $, followed by the number of the register (between 0 and 31), as in `r4`, `R4` or `$4`;
- *Immediate values* an immediate value can be a number or a label; the number can be specified in base 10 or in base 16, using in the latter case the prefix `0x`;
- *Address* an address is composed by an immediate value followed by a register name enclosed in brackets. The value of the register will be used as the offset.

The general purpose registers can also be addressed using the standard MIPS32 aliases, like `$zero` for `R0`, `$t0` for `R8` and so on.

The size of immediate values is limited by the number of bits that are available in the bit encoding of the instruction. Please see [12] for more details about how the instructions are actually encoded.

4 The user interface

We have been using WinMIPS64 during the laboratory sessions of the Computer Architecture course, and we found its interface to be functional and useful for making students understand the basic concepts of the course. So we modelled the Graphical User Interface (GUI) of EduMIPS64 after the interface of WinMIPS64. This has also the side-effect to make the transition from WinMIPS64 easy.

The interface of EduMIPS64 follows the Multiple Document Interface (MDI) paradigm: the main window contains several children frames, and the user can choose which frames he wants to display, moving them or changing their size.

By default, the simulator starts with a full-screen main window that contains 6 frames (Cycles, Registers, Statistics, Pipeline, Memory, Code), layed out in two rows of three frames; a 7th frame (Input/Output) is hidden by default. The main window also contains a menu bar and a status bar.

In the rest of the section we will discuss the purpose of the main elements of the user interface of the simulator. All the screenshots shown in this Section refer to the same execution cycle of a program, except for the Input/Output frame, whose screenshot represents the output presented at the end of the program.

4.1 Frames

The program execution is presented to the user through seven frames:

- *Cycles*: the Cycles frame (Figure 1) shows the evolution of the program flow over time. The left section of the frame contains a list of instructions, updated as they enter in the pipeline, while the right section contains a plot of the position of instructions as time goes by.

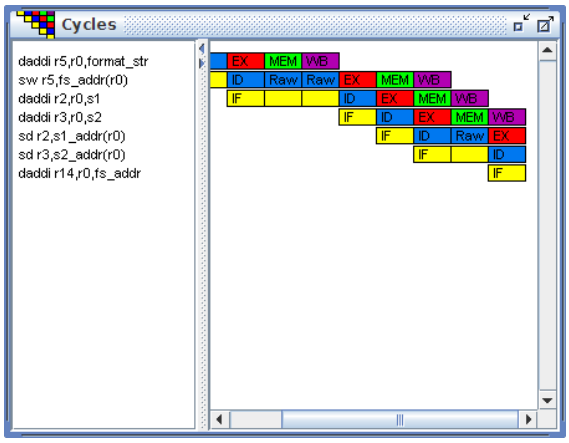

Figure 1: The Cycles Frame

- *Registers*: the Registers frame (Figure 2) shows the content of the 32 General Purpose Registers (GPRs), floating point registers and the special registers LO and HI. In the current stable version of the simulator, floating point registers are not active. A single click on a register has the effect to display its (signed) decimal value in the status bar, while a double click pops up a dialog that allows to change its value at runtime.

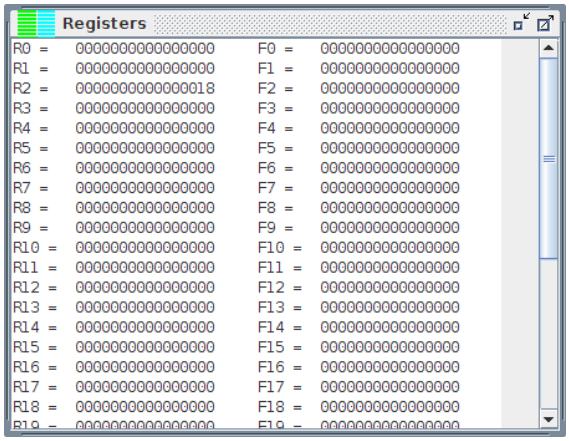

Figure 2: Registers Frame

- *Statistics*: the Statistics frame (Figure 3) shows the following real-time statistics about program execution: cycles ran by the CPU, instructions executed during those cycles, the number of Cycles Per Instruction (CPI), stalls (RAW, WAR, WAW, structural, branch taken, branch mispredicted), code size.

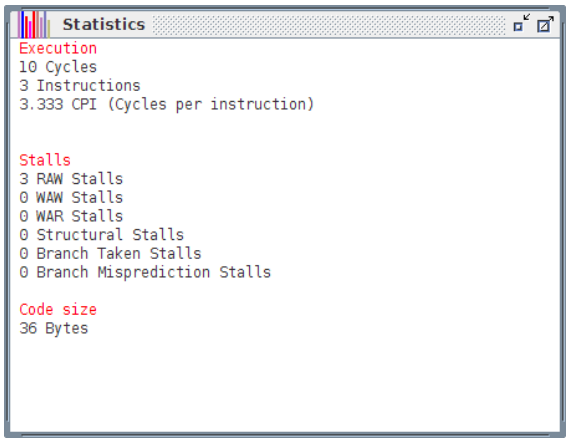

Figure 3: The Statistics Frame

- *Pipeline*: the Pipeline frame (Figure 4) shows a graphical model of the 5-stages CPU, showing which instruction is in any of the stages. The FP-related stages are not active in the current stable version, but are active in the development version.

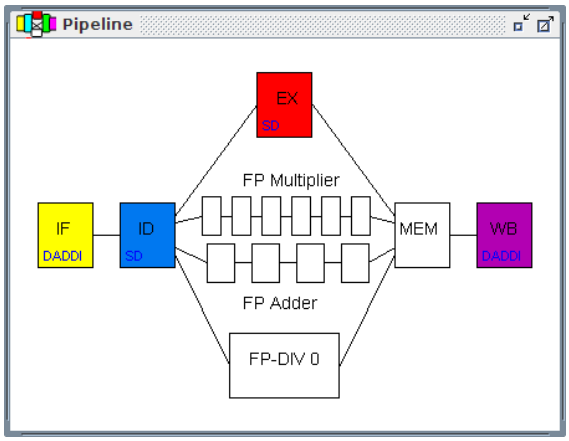

Figure 4: The Pipeline Frame

- *Memory*: the Memory frame (Figure 5) shows the address and contents of memory cells, along with labels and comments taken from the source code. The same mouse actions that are available for registers are enabled also for memory cells.
- *Code*: the Code frame (Figure 6) shows a representation of the source code of the current program, with a layout similar to the Memory frame.
- *Input/Output*: the Input/Output frame (Figure 7) shows the output of the program when system

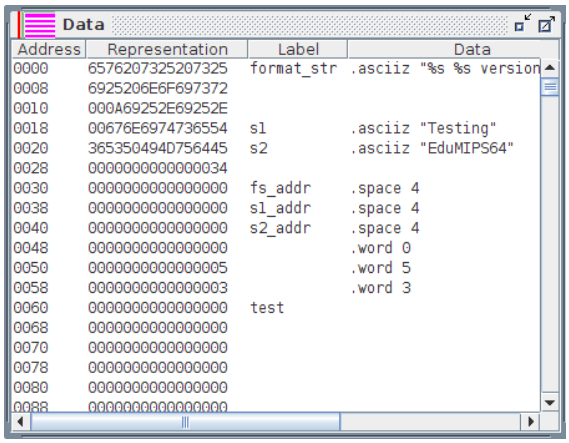

Figure 5: The Memory Frame

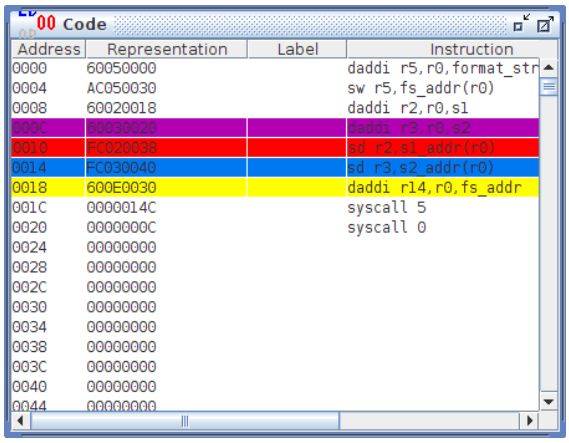

Figure 6: The Code Frame

calls 4 and 5 are used. It is not actually used for input, as there is a dialog that pops up when a `SYSCALL` 3 tries to read data from the standard input.

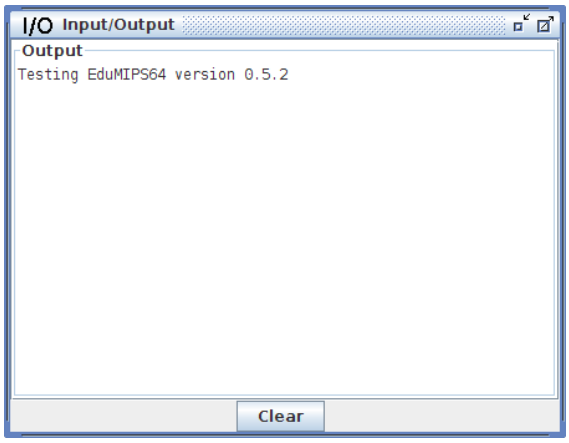

Figure 7: The Input/Output Frame

4.2 Menus

The user can interact with EduMIPS64 via six menus:

- *File*: contains menu items for opening a new file, resetting the simulator, writing a trace file in DineroIV format and exiting the simulator.
- *Execute*: contains menu items for starting and stopping the simulation, running the program step-by-step or all in one execution step.
- *Configure*: contains menu items for opening the Settings dialog, that is described in 4.3 and for changing the language of the interface of the simulator. Currently the English and Italian languages are supported.
- *Tools*: allows the user to run the Dinero Frontend dialog, described in 4.3.
- *Window*: contains menu items to show, hide and arrange the frames.
- *Help*: contains an option (Manual...)that runs the Help dialog, that contains the User Manual, and an option (About us...) that shows the names of the members of the development team.

4.3 Dialogs

Here is a summary of the most important dialogs:

- *Settings*: this dialog allows the configuration of many aspects of the simulator, like enabling forwarding, multi-step execution mode parameters (synchronization with GUI, number of steps, refresh interval), exception-related settings and GUI-related options.
- *Dinero Frontend*: this dialog allows to execute the DineroIV cache simulator [11] with the trace file containing the memory accesses generated by the execution of the current program.
- *Parsing errors*: if the parser finds any errors in the input file, the Parsing Errors dialog appears (see Figure 8) and presents to the user a list of all the errors that were found, like missing labels, incorrect syntax, *etc.*If the user turns warning reporting on, this dialog will also be used to notify minor and non-blocking issues in the code, like the usage of non-standard MIPS64 instruction

4.4 Running EduMIPS64

EduMIPS64 is distributed both as a source package and as a binary Java Archive (`.jar`) file. Typical

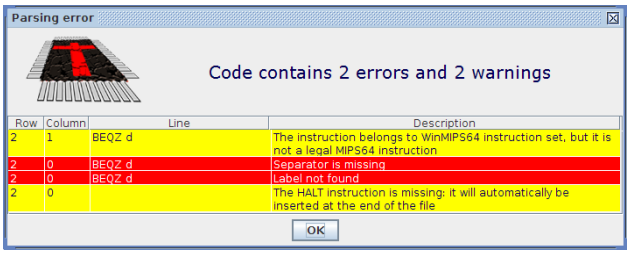

Figure 8: The Errors Dialog

users will not need the source package and will use only the `.jar` file.

The EduMIPS64 `.jar` file can be used both as a desktop application and as a web applet. Both methods need the Java Runtime Environment, version 5 or later.

Running as a web applet allows the simulator to be seamlessly embedded in distance learning environments, enabling users to use it without having to download it. An example of this set-up is available in the EduMIPS64 web site [4].

5 Assessment

In order to assess the effectiveness of EduMIPS64, a survey was conducted to obtain feedback from a set of students after completion of their course.

The aim of the survey was to investigate two main areas:

- **Cross-platform usage:** investigate to which extent the possibility of running EduMIPS64 on different Operating System flavours is perceived as an added-value.
- **Usability:** issues regarding the easiness of use, the user interface, the learning curve to face when learning the EduMIPS64 environment, the consistency and readability the simulator behaviour.

A form was created online and distributed via email. At the time this paper is being written, a total of 93 students belonging to 3 different courses answered to the survey.

5.1 Cross-platform usage

Figure 9 shows the Operating System distribution among students, comparing the OS used at home against the platform on which they have been using EduMIPS64. It can be noticed that the OS fragmentation among students of a computer architecture course is more equally distributed over different platform as compared to the typical OS diffusion on

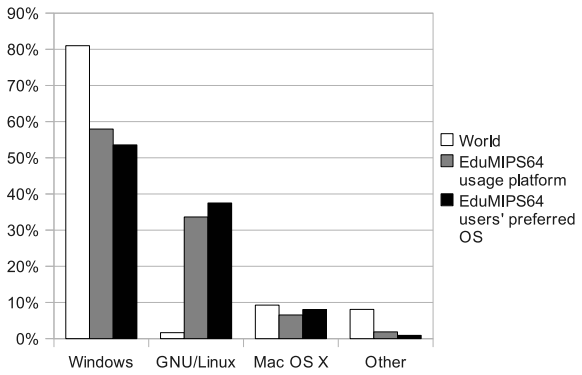

Figure 9: Operating Systems distribution among students

generic, non-academic users. So the cross-platform nature of EduMIPS64 is certainly an added-value for GNU/Linux and MacOS X users. Note the slight difference between EduMIPS64 usage OS and users' home OS (grey and black bars), showing that not all GNU/Linux and MacOS X home users could perform their examination running EduMIPS64 on their favourite OS, most likely for lacking of support and installations of these platforms on laboratory equipments. As for what concerns the cross-platform web applet usage, 17% of users did know about the web version of EduMIPS64 and 84% of them found it useful.

5.2 Usability

Usability and easiness of use are one of the main reasons behind the EduMIPS64 project. A first aspect we investigated was the general speed/responsiveness of the simulator that, because of running on an high-level Java virtual machine could suffer some performance/memory usage issues. However, 19% of users evaluated as excellent the performance of the simulator and 60% of them considered it good, while 18% think that it was sufficient and 3% considered it poor. The graphical user interface of EduMIPS64 was evaluated excellent from 60% of users, 20% found it good while the remaining 19% and 1% sufficient and poor respectively. Summarizing these results, we can definitely assume that the choice of using Java as a cross-platform technology did not penalize the user-experience from both the performance and visual points of view, since only a neglegible fraction of the users rated EduMIPS64 less than sufficient.

6 Conclusions

As stated before, EduMIPS64 is still actively developed, and while some features are not available in the current stable version, they are being developed and/or tested and will be released to the public.

The most important feature that is only in the development release is the support the for the MIPS64 Coprocessor 1, the Floating Point Unit (FPU). It adds to EduMIPS64 the support for the 32 floating point registers (FP0 - FP31), adopting the IEEE 754 representation of floating point numbers and including special values like ± 0, $\pm \infty$, Not a Number (NaN). Floating point exceptions are represented via a bit-accurate implementation of the Floating point Control and Status Register (FCSR). The FPU support adds 23 instructions to the EduMIPS64 instruction set and adds three multi-level functional units to the pipeline, namely the floating point adder (4 sub-stages), the floating point multiplier (7 sub-stages) and the floating-point divider (1 sub-stage).

Another important feature that is being tested is the support for the branch delay slot.

Acknowledgments: The authors hereby acknowledge the fundamental contribution of all the developers of EduMIPS64, whose names are listed in the About Us dialog of the simulator itself.

References:

[1] MIPS Technologies Inc. *MIPS64 Architecture For Programmers Volume I: Introduction to MIPS64 Architecture*, July 2005.

[2] John L. Hennessy and David A. Patterson. *Computer Architecture*. Morgan Kaufmann, third edition, 2002.

[3] Boško Nikolic, Zaharije Radivojevic, Jovan Djordjevic, and Veljko Milutinovic. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *IEEE Transactions on Education*, 52(4):449–458, November 2009.

[4] EduMIPS64 official web site.

[5] Michael Scott. WinMIPS64 home page.

[6] Computing Curricula - Computer Engineering (CCCE) Task Force. IEEE Computer Society / ACM Computing Curriculum - Computer Engineering, 2005.

[7] Boško Nikolic, Zaharije Radivojevic, Jovan Djordjevic, and Veljko Milutinovic. Survey of Simulators 2008.

[8] James Larus. SPIM – a MIPS32 simulator.

[9] Herbert Grünbacher. WinDLX home page.

[10] Andrea Spadaccini and the EduMIPS64 Team. *EduMIPS64 User Manual*, June 2007.

[11] Mark Hill. DineroIV trace-driven uniprocessor cache simulator.

[12] MIPS Technologies Inc. *MIPS64 Architecture For Programmers Volume II: The MIPS64 Instruction Set*, July 2005.