Supporting Undergraduate Computer Architecture Students Using a Visual MIPS64 CPU Simulator

Davide Patti, *Member, IEEE*, Andrea Spadaccini, *Student Member, IEEE*, Maurizio Palesi, *Member, IEEE*, Fabrizio Fazzino, *Member, IEEE* and Vincenzo Catania

***Abstract*—The topics of computer architecture are always taught using an Assembly dialect as an example. The most commonly-used textbooks in this field use the MIPS64 Instruction Set Architecture (ISA) to help students in learning the fundamentals of computer architecture, because of its orthogonality and its suitability for real-world applications.**

This paper shows how to use the EduMIPS64 visual CPU Simulator as a supporting tool for teaching the standard topics covered by an undergraduate course in computer architecture. The proposed approach is first compared with other similar works in the field, then, after a short description of the simulator, the paper focuses on how it can be used for teaching specific topics in an undergraduate computer architecture course.

This discussion is then followed by a quantitative assessment of the suitability of the simulator, by means of a survey compiled by students themselves; the results show that EduMIPS64 is suitable for the purpose for which it was built, that is, supporting the learning process of computer architecture topics.

***Keywords*-Assembly programming, distance learning, instruction-set simulator, teaching methodology, visual simulation**

I. Introduction

Computer architecture and organization is one of the key elements which characterizes the curricula of electrical and computer engineers [1]. One of the most critical aspects on teaching this discipline is how to support the theoretical concepts of the subjects with appropriate practical experience, usually organized as laboratory assignments. Such experience is mainly aimed at performing a quantitative analysis on the system parameters related to both the *computer architecture* (which encompasses the programmer's abstract view of the machine) and the *computer organization* (which deals with implementation details).

Instruction-Set Simulators (ISSs) represent the basic tool used for supporting teaching of the topics related to the computer architecture part of the course. In this context, visual representation of the concepts introduced in the course, platform independence, distance learning support, free availability of the tools and full coverage of the course topics are key factors which determine the effectiveness of the teaching resources.

D. Patti, A. Spadaccini and V. Catania are with the Dipartimento di Ingegneria Elettrica, Elettronica ed Informatica, University of Catania, Catania, Italy (email: {davide.patti,andrea.spadaccini,vincenzo.catania}@dieei.unict.it).

M. Palesi is with the Kore University, Enna, Italy (email: maurizio.palesi@unikore.it).

F. Fazzino is with NVIDIA Inc., Bristol, United Kingdom (email: ffazzino@nvidia.com).

Unfortunately, some of the available educational ISSs focus on specific aspects of a computer system, like assembly language [2], pipelining [3] or memory hierarchy [4]; the heterogeneous nature of the tools used to cover the different topics of the course, could lead to a fragmentation of the knowledge acquired. In fact, analyzing the different subsystems of the computer system separately makes it difficult for the student to understand how those parts interact, with a consequent blinding of student to the system-level view. Moreover since most ISS do not offer the other capabilities mentioned above, such as platform independence or distance learning support, it is difficult to base a solid and compelling computer architecture course on them.

This paper is focused on the description and assessment of a computer architecture undergraduate course in which the EduMIPS64 simulator [5] was employed as the primary tool for explaining the topics, for the students' homework and for the final exam. EduMIPS64 is a free visual MIPS64 CPU simulator built over the last five years at the University of Catania, Italy.

The rest of the paper is organized as follows. Section II provides a brief overview on available tools and methodologies aimed at enhancing the learning process of the topics in an undergraduate course on basic computer architectures. Section III presents the main features of EduMIPS64 used as the reference simulation platform in this paper. The teaching methodology based on the use of EduMIPS64 is presented in Section IV. The suitability of EduMIPS64 is discussed in Section V by means of a quantitative assessment carried out by survey compiled by students themselves. Finally, Section VI draws conclusions and discusses possible future investigations.

II. Related Work

Various tools and methodologies have been adopted to enhance the learning process of the basic concepts of CPU organization usually taught in introductory computer architecture courses. Some approaches focus on the framework adopted (e.g. simulators, web-based learning, games), while others try to exploit the positive effects of collaborative projects in improving students' motivation.

In [6] the authors propose a method and an activity designed to improve student motivation in computer architecture courses by showing them how to successfully optimize the assembly code produced. An educational experience of teaching computer architecture topics using a project-based learning is described and evaluated in [7]. The main idea was that of using a collaborative approach where students were assigned to different sub-projects but still had to exchange some acquired knowledge in order to complete their tasks. [8] presents a Web-based environment to support the learning process in an introductory computer science course using a customizable e-learning approach.

As regards tools-based approaches, computer architecture topics are addressed at different levels of abstraction and granularity. [9] introduces an educational computer system specifically designed to help teach architecture topics from the designer's perspective. In [10] the authors propose a pedagogical and fully-configurable simulator of the MC88110 32-bit microprocessor, with the aim of reducing the effort of using many different simulators to cover the topics of a typical computer architecture course. [11] shows how an educational framework was used in introductory computer architecture and engineering courses to teach some topics related to the tuning of cache parameters to optimize performance and energy consumption. [12] presents a pedagogical tool for dealing with code-based exercises expressly focused on cache memory related topics.

In [13] the authors compare 28 pedagogical tools, classifying them in two categories: those that allow the design of a complete computer environment and those that instead focus on the simulation of a target architecture. EduMIPS64 fits in the second category; while it does not try to implement all the architectural features offered by many of those simulators, it instead focuses mainly on topics related to the basic concepts of computer architecture and CPU organization, making it suitable for undergraduates. Finally, two simulators that in particular have been a great source of inspiration for the design and development of EduMIPS64 are WinDLX [3] and WinMIPS64 [14]. EduMIPS64 tries to take the best from those simulators, adding platform independence and a support for the DineroIV [4] cache simulator.

The main advantage of EduMIPS64 over these tools is the combination of features not strictly related to simulation: it is open source, suitable for web-based distance learning activities and cross platform; this constitutes a unique feature set in the field of didactic CPU simulators.

III. Description of the Simulator

As stated above, EduMIPS64 is a visual MIPS64 Instruction Set Simulator; in this section, the main features of the simulator and of its graphical user interface will be briefly described. For more details on the simulator, please refer to [5]. Fig. 1 shows an overview of the user interface of the simulator.

A. Architectural Features

1) Pipeline: EduMIPS64 implements a 5-stages pipeline that allows for instruction-level parallelism. As each virtual CPU cycle is executed, the user interface updates all its components to offer an appealing and informative representation of the current program execution state. The five stages of the pipeline are drawn in one of the sub-windows of the simulator and each stage of the pipeline contains a representation of the instruction that is being executed.

EduMIPS64 implements a rich subset of the MIPS64 ISA, comprising 65 instructions.

2) Forwarding: The simulator provides built-in support for instruction forwarding, so that students can evaluate the performance of their programs with or without forwarding.

3) Cache simulation: EduMIPS64 generates trace files containing the memory accesses that are performed by the simulated program in a format compatible with the DineroIV [4] cache simulator; there is also a built-in interface for invoking the command line of the DineroIV cache simulator using the trace file produced. This allows a student to test different cache scenarios without having to exit the EduMIPS64 interface.

4) Interrupts and System Calls: Via the `SYSCALL` instruction the simulator gives to the student an interface for system calls, that simulates basic console and file I/O operations, and provides support for maskable synchronous exceptions.

B. The User Interface

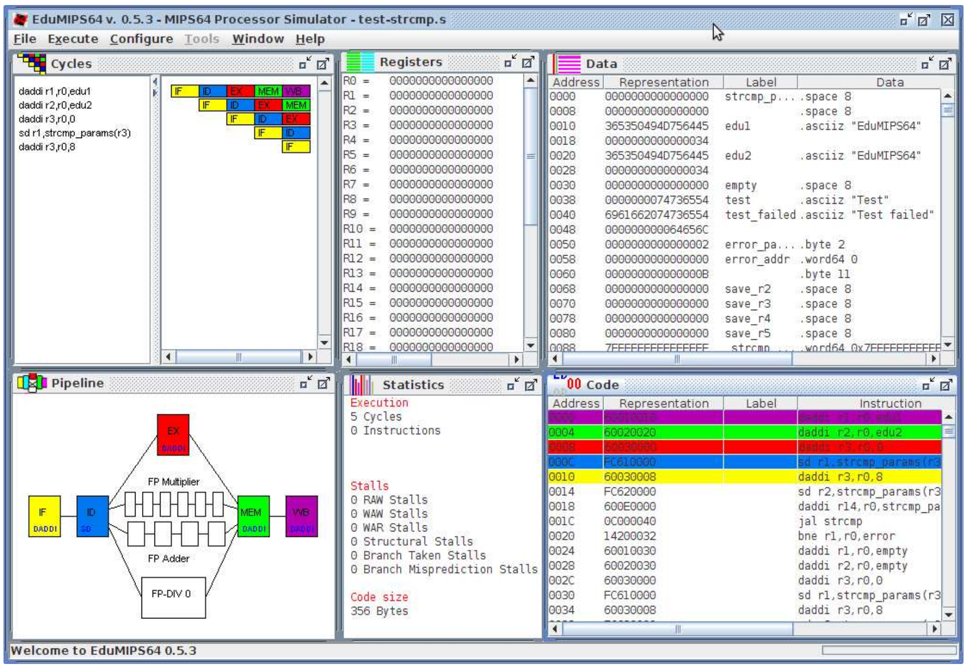
The user interface of EduMIPS64 is similar to that provided by WinMIPS64 [14] and WinDLX [3]. It adopts the Multiple Document Interface paradigm, with a main window that contains the individual frames that represent a given aspect of the simulation. Each stage of the pipeline is associated with a given (customizable) color, that is used throughout the user interface to represent the events and data associated with that stage, allowing the student to easily correlate the various sections of the interface with what happens in a given pipeline stage.

The seven frames of the simulator interface give at a glance many details of the current and past status of the simulation: the current content of memory and General Purpose Registers; a graphical representation of the pipeline stages and the instructions in each stage; the program code; and some statistics on the execution, such as the number of Cycles Per Instruction (CPI) and the number and class of stalls.

The user interface allows the student to execute the program either step-wise or at once, and to choose whether during multi-step execution the interface should display each intermediate step or if it should only show the final stage, which is useful for the execution of long programs.

IV. Teaching Methodology

EduMIPS64 is tailored for use in introductory computer architecture courses, usually taught in Electrical Engineering or Computer Science curricula. These courses usually start with an "onion view" of a computer system seen as a stack of layers, and where a usual level of abstraction is given by the bare-metal CPU able to execute assembly sources translated ("assembled") into binary machine code. EduMIPS64 can be productively used to test and improve student understanding of: instructions and data encoding; the MIPS instruction set; pipelining; I/O and system calls; the memory hierarchy; and Amdahl's law. More advanced topics, such as scoreboard, Tomasulo scheduling or virtual memory, are not covered since they were not on the syllabi of the courses at which this work was directed.


EduMIPS64 v. 0.5.3 - MIPS64 Processor Simulator - test-strcmp.s
File Execute Configure Tools Window Help
Cycles
IF ID EX MEM WB
Registers
Data
Pipeline
IF DADDI
ID SD
EX DADDI
FP Multiplier
FP Adder
FP-DIV 0
MEM DADDI
WB DADDI
Statistics
Code
Welcome to EduMIPS64 0.5.3

```
daddi r1,r0,edu1
daddi r2,r0,edu2
daddi r3,r0,0
sd r1,strcmp_params(r3)
daddi r3,r0,8
```

```
R0  =   0000000000000000
R1  =   0000000000000000
R2  =   0000000000000000
R3  =   0000000000000000
R4  =   0000000000000000
R5  =   0000000000000000
R6  =   0000000000000000
R7  =   0000000000000000
R8  =   0000000000000000
R9  =   0000000000000000
R10 =   0000000000000000
R11 =   0000000000000000
R12 =   0000000000000000
R13 =   0000000000000000
R14 =   0000000000000000
R15 =   0000000000000000
R16 =   0000000000000000
R17 =   0000000000000000
R18 =   0000000000000000
```

```
Address  Representation     Label        Data
0000     0000000000000000   strcmp_p... .space 8
0008     0000000000000000               .space 8
0010     365350494D756445   edu1        .asciiz "EduMIPS64"
0018     0000000000000034
0020     365350494D756445   edu2        .asciiz "EduMIPS64"
0028     0000000000000034
0030     0000000000000000   empty       .space 8
0038     0000000074736554   test        .asciiz "Test"
0040     6961662074736554   test_failed .asciiz "Test failed"
0048     000000000064656C
0050     0000000000000002   error_pa... .byte 2
0058     0000000000000000   error_addr  .word64 0
0060     000000000000000B               .byte 11
0068     0000000000000000   save_r2     .space 8
0070     0000000000000000   save_r3     .space 8
0078     0000000000000000   save_r4     .space 8
0080     0000000000000000   save_r5     .space 8
0088     7FFFFFFFFFFFFFFF   strcmp_...  .word64 0x7FFFFFFFFFFFF
```

```
Execution
5 Cycles
0 Instructions


Stalls
0 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
0 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
356 Bytes
```

```
Address  Representation  Label  Instruction
0000     60010010               daddi r1,r0,edu1
0004     60020020               daddi r2,r0,edu2
0008     60030000               daddi r3,r0,0
000C     FC610000               sd r1,strcmp_params(r3
0010     60030008               daddi r3,r0,8
0014     FC620000               sd r2,strcmp_params(r3
0018     600E0000               daddi r14,r0,strcmp_pa
001C     0C000040               jal strcmp
0020     14200032               bne r1,r0,error
0024     60010030               daddi r1,r0,empty
0028     60020030               daddi r2,r0,empty
002C     60030000               daddi r3,r0,0
0030     FC610000               sd r1,strcmp_params(r3
0034     60030008               daddi r3,r0,8
```

Fig. 1. Overview of the EduMIPS64 User Interface

In the next sections the teaching methodology adopted for each of the areas listed above will be briefly described, followed by some proof-of-concept assignments that show concrete examples of how EduMIPS64 can be exploited to assess student knowledge. These can be used in the course final examination, but also during lectures and as homework assignments, in order to receive immediate feedback on the learning process.

A. Understanding the Instructions and Data Encoding

The initial part of the syllabus of the computer architecture course addressed in this paper requires the teacher to explain not only the fundamental concepts of instruction-set architectures and assembly language, but also hexadecimal number conversions, instruction encoding (with the three main types: R-type, I-type and J-type) and even the concept of endianness (ordering of bytes within words into memory).

So a first step with EduMIPS64 should be to show how an assembly source file written following a MIPS-like syntax is loaded into memory and how the resulting binary content, shown as hexadecimal, appears in the *data* and *code* windows. At this stage of the course students have no idea of how assemblers and linkers work, so they will see that their data and code (complete with labels and mnemonics) magically appear in the memory map shown in the two corresponding EduMIPS64 *memory* and *code* windows described in Section III-B. Note that the bit-accurate encoding of instructions of EduMIPS64 allows students to verify how the three different instruction formats are represented in memory. At this stage some of the windows inside EduMIPS64 can safely be ignored (namely Cycles, Pipeline and even Statistics), and the students should focus on the remaining ones (Registers, Data and Code).

Sample assignments:

- *Before loading a given program into memory, try to work out how some lines of the* `.data` *section will appear in the EduMIPS64 memory window.*
- *Maximize the memory encoding window and highlight some lines, then try to find the corresponding snippet of code in the text editor.*
- *Choose one of the lines in the memory or code window of EduMIPS64 and try to explain the meaning of the hexadecimal value associated with them.*

B. Understanding the MIPS Instruction Set

The behavior of each instruction of the MIPS instruction set, and its effect on registers, memory and code execution, are certainly some of the most tricky concepts to deal with, from both the teaching and learning perspectives. In an introductory computer architecture course, showing complex algorithms and data structures is not the most appropriate approach, despite being more representative of the "real world". Rather, EduMIPS64 can be used to effectively show various implementations of several simple small algorithms, visually and intuitively, so that students can capture the real meaning of each assembly instruction in the repertoire.

Sample assignments:

- *Write a MIPS assembly program implementing an algorithm that loads a set of values previously encoded in the* `.data` *section and then copies them in a different area of memory, according to the following* `<condition>`.

Note: further assignments can be derived from a template like that presented above, depending on the chosen `condition`, e.g., *being even but not a multiple of 4*, *being odd and less than 7*, *being divisible by 9* and so on.

C. Understanding the Pipeline

The teaching of the pipelining mechanisms and the related micro-architectural events involved can derive great benefit from the visual representation of the CPU described in Subsection III-B. By executing code using EduMIPS64 in the cycle-by-cycle mode, the teacher can show how the various elements of the pipeline deal with data (e.g. using forwarding, creating bubbles), what the current stage of execution is for each instruction, which signal controls are required, and so on.

Sample assignments:

- *Try to enable (disable) the forwarding settings of EduMIPS64 and identify which instructions of your code are affected by this micro-architectural feature.*
- *When possible, try to rearrange your code so that the number of data hazards shown in the statistics window is minimized. If you think that your code is already optimal, move some instructions in order to intentionally create data hazards while preserving the execution semantics.*
- *Execute the program cycle-by-cycle, until the n^{th} clock cycle, and describe the nature of the next stall encountered, showing the involved pipelining elements.*

D. Understanding the Memory Hierarchy

The ability to interface EduMIPS64 with the well-known DineroIV cache simulator allows both teacher and students to experiment quantitatively with the impact of different cache configurations on code execution and performance.

Sample assignments:

- *Assuming a CPU clock frequency $f = 1GHz$, compare the ideal time of execution T_{CPU_ideal} of your code against a system with unified, single-port, direct-mapped 4K data/instruction L1 Cache memory, with block size 16 byte and miss delay $L1_{delay} = 50$ cycles.*
- *Change the proposed cache configuration so that the number of demand misses is increased/decreased, showing the reasons for this different behavior.*

Note: Students can get both the total CPU cycle count N_{cyles} and the number of structural hazards $N_{hazards}$ from EduMIPS64 statistics window. Then, using the specified cache configuration they get the number of L1 misses $L1_{misses}$ and can compute the total time T_{CPU} as:

$$T_{CPU} = (N_{cycles} + N_{hazards} + L1_{misses} \times L1_{penality}) \times T_{clock}$$

where $T_{clock} = 1/f$, remembering that when the single-port unified cache is used each load/store causes a structural hazard. Further, since DineroIV also supports separate L1 data and instruction caches, different versions of this assignment can be designed in order to check the students' ability to deal properly with separate data and instruction cache parameters.

E. Understanding I/O and System Calls

These concepts are often addressed in the second part of the course, depending on the overall course organization. Nevertheless they can be easily accessed using the `SYSCALL` instruction of EduMIPS64. There are two main classes of assignments that involve the usage of the `SYSCALL` instruction:

- Assignments that test the students' ability to put the inputs of the `SYSCALL` in the `.data` section properly and then get the outputs when the `SYSCALL` has ended.
- Assignments that stress the usage of `SYSCALL` in combination with an external routine, called `input_unsigned`, provided with EduMIPS64 as part of sample source code.

The motivation behind the second class of assignments is to enrich the complexity of the algorithms that can be developed. This class exploits the usage of the `input_unsigned` routine to facilitate the interactive input of unsigned integer values. The students must include this in their programs using the `#include` directive and then call it when necessary by jumping to the routine using the `jal input_unsigned` instruction. When the routine execution has ended, the read unsigned integer value can be found on register R1. In addition, the teacher should consider the possibility of dedicating a whole lesson to the contents of the `input_unsigned` routine itself, which is a very good example of how simple system calls work and how they can be used to build an higher abstraction routine similar to the C-like `scanf(...)` function that students will have used in previous courses.

Sample assignments:

- *Properly setup a `.data` section so that two bytes are read from keyboard using the `SYSCALL 3` and their arithmetic sum is printed on screen using `SYSCALL 5`.*
- *Use the `input unsigned` routine to read a set of ten integers from the keyboard and then print them on screen in reverse order.*

F. Understanding Amdahl's Law

When teaching computer architecture from the performance analysis perspective, Amdahl's law is probably the most important principle for students to learn. In short, Amdahl's law is used to find the maximum expected improvement to an overall system when only one part of that system is improved, that is:

$$S_{overall} = \frac{1}{1 - F_e + \frac{1}{S_e}}$$

where $S_{overall}$ is the overall speedup, F_e is the fraction of the total time affected by the enhancement and S_e the speedup of the improved part of the system.

Using EduMIPS64 in conjunction with DineroIV, simple assignments can be designed in order to help students understand how the overall performance of the system is governed by the slowest component. There are two main categories of assignment that can be proposed to students:

- Assignments that ask them to compute the speedup of the system if a particular component improves its performance, in other words, the direct usage of Amdahl's law in order to compute $S_{overall}$ from a given F_e and S_e.
- Assignments that ask students to find which would be the required S_e of a particular component in order to achieve a global overall system speedup. This is an inverse usage of Amdahl's law, that is, finding S_e from a given desired $S_{overall}$ and known F_e.

Sample assignments:

- *Starting from the results obtained when executing a given MIPS64 program code on separated instruction and data caches, compute the overall speedup that would be achieved if the L1 data cache had only half of the previously-reported demand misses.*
- *Starting from the results obtained when executing your program code on unified L1 cache memory, find the miss penalty that would be needed to get an overall system speedup of* 30% *in performance.*

Both assignments stress students' ability to compute quantitatively the correct F_e from execution statistics. A very interesting feature is present in the second category, which can result in a negative value for S_e; this is useful to make the students understand that sometimes a desired speedup cannot be obtained, no matter how much the S_e of the component is improved.

V. Assessment Survey and Evaluation

A student survey was conducted at the end of the courses to obtain feedback on the effectiveness of the adopted methodology. The survey was designed to investigate the following areas:

- **Simulation environment:** issues regarding the ease of use of the adopted tool, i.e. the user interface, the learning curve faced when learning the EduMIPS64 simulator, and the consistency and readability of the whole learning environment.
- **Impact on learning:** the extent to which EduMIPS64 was effective in helping students to understand the course topics and increasing their motivation to learn.

An online form [15] was completed by a total of 93 students enrolled in undergraduate[1] computer architecture courses [16], [17] in the period 2008-2010. The main goal of the questions posed was to capture the perceived effectiveness of the EduMIPS64 tool and the adopted methodology, the usage frequency, and impact on the learning of several key topics of computer architecture and organization. The results collected [15] have been summarized in the Subsections V-A and V-B.

A. Simulation Environment

Usability and ease of use are the prime reasons behind the adoption of the EduMIPS64 project as the course's simulation environment. A first aspect investigated was the general speed/responsiveness of the simulator that, since running on an high-level Java virtual machine could cause some performance/memory use issues. However, 19% of the users rated the performance of the simulator as excellent and 60% of them considered it good, while 18% thought it was adequate and 3% considered it poor. The graphical user interface of EduMIPS64 was rated as excellent by 60% of users, 20% of them found it good and the remaining 19% and 1% found it adequate and poor respectively. Summarizing these results, it can definitely

[1]Students who are going to take the Bachelor's degree and are familiar with basic computer science concepts.

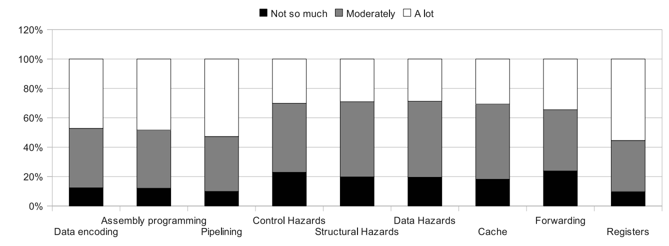


Fig. 2. Impact of EduMIPS64 on student understanding of topics

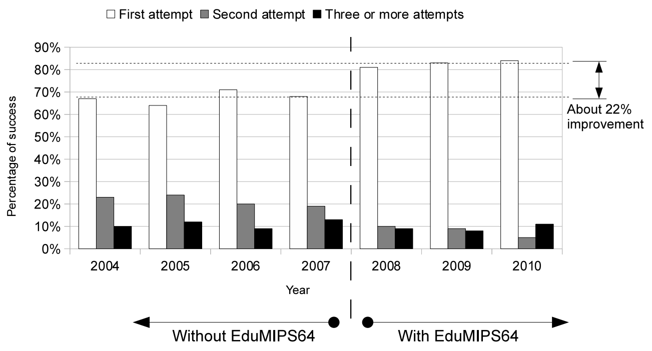


Fig. 3. Distribution of the percentage of success across the number of attempts to pass the final examination

be concluded that the choice of using Java as a cross-platform technology did not degrade the user-experience from both the performance and visual points of view, since only a negligible fraction of the users rated EduMIPS64 as less than adequate.

B. Impact on Learning

The percent of the time spent by the students on EduMIPS64 is an indicator of the role played by the laboratory-based methodology in the course. Less than 8% of the students used EduMIPS64 for less than 10% of the study time allocated for this course. About 20% of them used EduMIPS64 for 20% – 30% of the time, whereas the remaining 70% claimed to have spent more than 40% of the time on EduMIPS64.

An interesting issue to investigate was the impact of the adopted methodology on student understanding of the different topics encountered during the course. Note that some topics that are usually strictly related to the theoretical part of a computer architecture course were deliberately included, in order to capture more accurately to what extent, even indirectly, the adoption of EduMIPS64 can affect the learning process and student motivation. Fig. 2 summarizes the results obtained, and shows that the experience with EduMIPS64 seems to influence to a greater or lesser extent all the topics presented in the survey; topics particularly impacted included registers and data encoding — due to bit-accurate encoding — and assembly programming and pipelining, because of the visual representation of the CPU during the execution flow of the instructions.

Finally, Fig. 3 shows how the average percentage of success is distributed across the number of attempts required to pass the final examination. As can be observed, although those students requiring three or more attempts were not particularly affected, the adoption of the methodology since 2008 seems to increase the number of students who pass the first examination at the first attempt (about 22% improvement on average). Note that the second attempt shows a lower result with EduMIPS64, because an increasing number of students have no need to take the exam a second time.

VI. Conclusions

Computer architecture and organization is one of the core topics that a well-educated computer engineer needs to master in order to be successful. This paper presented the outline of an undergraduate-level computer architecture course that adopted as its main educational tool a visual, free and platform-independent MIPS64 Instruction-Set Simulator called EduMIPS64, for its various features and its user-friendly interface. Several assignments and teaching processes based on EduMIPS64 have been described.

The impact of the adoption of the simulator on the teaching process was evaluated by means of a survey completed by students after the completion of the course; this feedback confirmed the authors' hypothesis on the suitability of EduMIPS64 for the course, and also contributed to an objective analysis of the simulator's strengths and weaknesses, that will be useful in planning future development work.

Acknowledgments

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

References

[1] Computing Curricula - Computer Engineering (CCCE) Task Force. (2005) IEEE Computer Society / ACM Computing Curriculum - Computer Engineering. [Online]. Available: http://www.eng.auburn.edu/ece/CCCE

[2] J. Larus. SPIM – a MIPS32 simulator. University of Wisconsin-Madison. [Online]. Available: http://www.cs.wisc.edu/~larus/spim.html

[3] H. Grünbacher. WinDLX home page. [Online]. Available: http://www.soc.tuwien.ac.at/intern/RA/

[4] M. Hill. DineroIV trace-driven uniprocessor cache simulator. [Online]. Available: http://www.cs.wisc.edu/~markhill/DineroIV/

[5] A. Spadaccini, D. Patti, M. Palesi, and F. Fazzino, "EduMIPS64, a Visual CPU Simulator for Teaching Computer Architecture," *submitted for publication*.

[6] M. Anguita and F. J. Fernandez-Baldomero, "Software Optimization for Improving Student Motivation in a Computer Architecture Course," *IEEE Transactions on Education*, vol. 50, no. 4, pp. 373–378, Nov. 2007.

[7] A. Martinez-Mones, E. Gomez-Sanchez, Y. Dimitriadis, I. Jorrin-Abellan, B. Rubia-Avi, and G. Vega-Gorgojo, "Multiple Case Studies to Enhance Project-Based Learning in a Computer Architecture Course," *IEEE Transactions on Education*, vol. 48, no. 3, pp. 482–489, Aug. 2005.

[8] I. Verginis, A. Gogoulou, E. Gouli, M. Boubouka, and M. Grigoriadou, "Enhancing Learning in Introductory Computer Science Courses Through SCALE: An Empirical Study," *IEEE Transactions on Education*, vol. 54, no. 1, pp. 1–13, Feb. 2011.

[9] J. Djordjevic, B. Nikolic, and a. Milenkovic, "Flexible Web-Based Educational System for Teaching Computer Architecture and Organization," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 264–273, May 2005.

[10] M. I. Garcia, S. Rodriguez, A. Perez, and A. Garcia, "p88110: A Graphical Simulator for Computer Architecture and Organization Courses," *IEEE Transactions on Education*, vol. 52, no. 2, pp. 248–256, May 2009.

[11] R. Quislant, E. Herruzo, O. Plata, J. I. Benavides, and E. L. Zapata, "Teaching the Cache Memory System Using a Reconfigurable Approach," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 336–341, Aug. 2008.

[12] J. Sahuquillo, N. Tomas, S. Petit, and A. Pont, "Spim-Cache: A Pedagogical Tool for Teaching Cache Memories Through Code-Based Exercises," *IEEE Transactions on Education*, vol. 50, no. 3, pp. 244–250, Aug. 2007.

[13] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 449–458, Nov. 2009.

[14] M. Scott. WinMIPS64 home page. [Online]. Available: http://www.computing.dcu.ie/~mike/winmips64.html

[15] EduMIPS64 survey - Questions and Results. http://www.diit.unict.it/users/spadaccini/edumips64-survey.html.

[16] Laboratorio di Calcolatori - Universit of Catania. [Online]. Available: http://hal9000.diit.unict.it/dokuwiki/doku.php?id=labce

[17] Calcolatori Elettronici - Kore University of Enna. [Online]. Available: http://www.unikore.it/mpalesi

Davide Patti (M'06) Davide Patti received the Laurea degree and the Ph.D. degree in Computer Engineering at University of Catania, in 2003 and 2007, respectively. He is currently a research assistant at the same university and his research focuses on platform based system design, design space exploration, low-power techniques for embedded systems, and Network-on-Chip architectures.

Andrea Spadaccini (GSM'11) received the M.S. degree with honors in Computer Engineering in 2008 from the University of Catania, Italy, where he is currently a Ph.D. candidate. He is the project leader of EduMIPS64 and one of its core developers. His research activity is focused on biometric recognition, novel biometrics, speaker recognition and signal processing.

Maurizio Palesi (M'06) received the M.S. and Ph.D. degrees in Computer Engineering from the University of Catania, Italy, in 1999 and 2003, respectively. Since November 2010 he has been Assistant Professor at Kore University, Enna, Italy. He has served as a Guest Editor for the VLSI Design Journal, the International Journal of High Performance Systems Architecture, Elsevier MICPRO Journal and the ACM Transactions on Embedded Computing Systems.

Fabrizio Fazzino (M'08) received the M.S. degree in Computer Engineering from the University of Catania, Italy, in 1997. Until 2001 he was responsible for the functional verification of 32-bit lines of microprocessors at STMicroelectronics. Since 2004 he has collaborated with the Department of Computer and Telecommunications Engineering. He is a silicon engineer at NVIDIA Inc, Bristol (UK).

Vincenzo Catania received the M.S. degree with Honors in Electrical Engineering from the University of Catania, Italy, in 1982. Since 1985, he has cooperated in research on advanced computer architectures and computer networks with the Department of Computer Science and Telecommunications Engineering, University of Catania, where he is currently a Full Professor of Computer Science where since November 2006, he has also been the Director. Currently, his research focuses on pervasive embedded systems, network-on-chip architectures, and mobile terminal platform and services.