2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)

Tool to Support Computer Architecture Teaching and Learning

Bruno Nova[1], João C. Ferreira[2], António Araújo[2]
[1]Faculdade de Engenharia da Universidade do Porto, Porto, Portugal
[2]INESC TEC and Faculdade de Engenharia da Universidade do Porto, Porto, Portugal

***Abstract*—Computer architecture is an important subject for informatics and electrical engineering courses. However, students display some difficulties in this subject, mainly due to the lack of educational tools that are intuitive, versatile and graphical. Existing tools are not adequate enough or are very specific. In this paper, an educational MIPS simulator, DrMIPS, is described. This tool simulates the execution of an assembly program on the CPU and displays the datapath graphically. Registers, data memory and assembled code are also displayed and a "performance mode" is also provided. Both unicycle and pipeline implementations are supported and the CPUs and their instruction sets are configurable. The tool is currently available for PCs and Android tablets, and is fairly intuitive and versatile on both platforms.**

***Keywords*—MIPS; Simulation; Computer Architecture teaching**

I. Introduction

A. Motivation

Computer architecture is an important subject in the syllabus of Informatics and Electrical Engineering courses, such as the MIEIC (*Mestrado Integrado em Engenharia Informática e Computação*) and MIEEC (*Mestrado Integrado em Engenharia Electrotécnica e de Computadores*) of FEUP (*Faculdade de Engenharia da Universidade do Porto*) . Here, students get to know the basics of how processors and computers work, learning topics like data representation on the computer, digital circuits, conceptual composition of a Central Processing Unit (CPU), assembly programming and processor performance.

However, many students exhibit difficulties understanding various topics on this subject, such as pipelined processors and calculating processor performance. Teachers and researchers from FEUP have concluded that these difficulties exist mostly due to the absence of tools on an integrated environment that are geared towards education. More specifically, tools that allow students to view the composition of a CPU's datapath graphically and consult detailed information about the data in each functional block, data being transmitted on the buses and control signals for each instruction on a set of instructions executed by the CPU.

There are already many tools created to simulate the operation of a CPU, and even some of them have graphical interfaces to show the CPU datapath. However, most of them are not very adequate for educational purposes, are difficult to use and understand or are too specific for some problem and not very versatile.

978-1-4799-1221-6/13/$31.00 ©2013 IEEE

B. Objectives

The main objective of the work presented in this paper was to create a tool to support computer architecture teaching and learning. This educational tool is a simulator the MIPS processor [1], which is a well-known processor in the computer architecture academic community and also one of the most used processors for teaching computer architecture courses in universities [2]. This simulator was developed under the MIEIC Master's Dissertation at FEUP .

Its development was based on the following requirements:

- Allow the configuration and parametrization of the CPU.
- Allow step-by-step simulation of MIPS assembly programs.
- Graphically display the datapath and the values of inputs and outputs at each component.
- Simulate both unicycle and pipeline versions of the CPU.
- Have a "performance mode" showing the latencies and critical path of the CPU.
- Be versatile but intuitive and simple to use.

And it seeks to help students better understand:

- The composition and operation of a "simple" datapath.
- How instructions are encoded.
- The values of the signals in the datapath.
- Relevant blocks and signals for each instruction.
- Pipelining, hazards, forwarding and stalls.
- Performance measuring and critical path identification.

The tool was developed mainly for personal computers, but a version for Android devices, especially tablets, was also created.

The rest of this paper is organized as follows. Section II discusses the most relevant educational simulators. Section III explains how the code of the tool was structured and how the simulation logic was implemented. Section IV presents and details the implementation of the user interface. Finally, Section V draws conclusions and discusses possible future work.

II. Related Work

As mentioned before, several tools to simulate the operation of a CPU already exist, and some of them even display the composition of the datapath visually. Most of them, however, are either not very suitable to teach students of computer architecture courses, are too difficult for a student to use or have a very specific objective. This section presents the most relevant educational simulators, as well as their strengths and weaknesses.

QtSPIM (formerly SPIM) [3] is an open-source simulator, written in C++ and Qt, that runs MIPS32 programs. It was widely used, both for education and for the industry [4], and supports a large number of MIPS instructions, including syscalls and some floating point operations [5]. The tool is good for debugging MIPS assembly programs and is reasonably intuitive but only simulates the unicycle version of the CPU. It also doesn't have a graphical view of the datapath neither integrates a code editor.

The MIPS Assembler and Runtime Simulator (MARS) simulator [4], developed in Java, is used in computer architecture courses in many universities all over the world. It simulates the execution of a MIPS assembly program, showing the results in the registers and memory on the screen. The simulation can be executed at once or step-by-step. The supported instruction set includes floating-point operations and various pseudo-instructions. MARS is also an Integrated Development Environment (IDE) that includes an editor with syntax highlighting and many help topics. The tool is very good for simulating and debugging MIPS assembly programs but, however, only simulates the unicycle version of the CPU and doesn't display the datapath visually. MARS supports plugins. One on these is the MIPS X-Ray [6], which displays the MIPS unicycle datapath graphically and the relevant wires and components for the instructions being executed using animations.

ProcSim [7] is a tool developed in Java that simulates the MIPS R2000 unicycle MIPS CPU. Assembly code is executed and displayed graphically as an animation in the datapath. The tool includes several different datapaths and the user can create more. It also provides a very simple code editor. ProcSim provides a good visualization of the datapath. However, it supports only a small set of MIPS instructions and only one component can send messages at a time during the simulation, displaying the animations sequentially by component, whereas in a real processor the components work concurrently [8]. Furthermore, it doesn't support pipelined datapaths.

MIPS-Datapath [9], developed in C++, is an open-source software that simulates a set of MIPS instructions and displays the execution in the datapath graphically. It can simulate not only a unicycle datapath but also a pipelined one, with or without data forwarding. The instructions are executed step-by-step and the relevant wires for the selected instruction are highlighted. A very simple code editor is also provided. MIPS-Datapath allows a person to see how each instruction is executed by the processor. However, it supports a very limited instruction set, doesn't support pipeline stalls and doesn't allow the datapath to be configured.

TABLE I. Comparison of the presented tools

	SPIM	MARS	PS	MD	WM	EM
Open-source	Yes	Yes	No	Yes	Yes	Yes
Code editor	No	Yes	Yes	Yes	Yes	No
Syntax-highl.	No	Yes	No	No	No	No
Unicycle	Yes	Yes	Yes	Yes	No	No
Pipeline	No	No	No	Part.	Yes	Yes
Floating point	Yes	Yes	No	No	No	Yes
Syscalls	Yes	Yes	No	No	No	Yes
Edit data in exec	Yes	Yes	No	No	No	Yes
Visual datapath	No	No	Yes	Yes	Part.	Simple
Datapath conf.	No	No	Yes	No	No	No
Written in	C++,Qt	Java	Java	C++	ASP	Java

WebMIPS [10] is an educational MIPS simulator that can be executed from a Web browser and, thus, from any system without installation. It was written in Active Server Pages (ASP) and simulates a five-stage pipeline with hazard detection, having been used in an introductory computer architecture course in Italy. The application provides a simple code editor and the code is executed step-by-step, also showing a graphical representation of the datapath It is a good educational simulator but has some important shortcomings. Simulation of the MIPS unicycle version is not possible and the datapath visual representation is static, only displaying input/output data on click.

EduMIPS64 [11] is an educational simulator that runs MIPS64 programs. The tool was used in some undergraduate courses to evaluate it and the results were positive, both in terms of percentage of success [12] and in terms of appreciation from the students [13]. It was based on WinMIPS64 [14] and WinDLX and was developed in Java. The tool simulates a 5-stage MIPS64 CPU pipeline and includes floating point operations, syscalls, hazard detection and supports a quite reasonable number of instructions. The interface is intuitive, but no code editor is provided, doesn't support unicycle simulation and doesn't show a detailed datapath representation.

Table I summarizes the tools presented here. *PS* stands for ProcSim, *MD* for MIPS-Datapath, *WM* for WebMIPS and *EM* for EduMIPS64. Also, *Part.* means that the tool supports the feature partially and *Simple* for the *Visual datapath* feature means that the tool presents a very simple block diagram. The table shows that it would be necessary to use more than one tool to cover the most computer architecture topics, which would be unintuitive and cumbersome for students and teachers.

None of the presented tools has an Android version. As such, a version of the developed simulator for Android tablets is something innovative, considering how Android tablets are becoming very popular [15], [16].

III. Simulator Implementation

As described before, the main objective of this work was to create an educational MIPS simulator. This simulator, called DrMIPS, lets the user create or load an existing assembly program and then simulate its execution on the 32 bit MIPS CPU, step-by-step, while visualizing what happens inside the processor. In each step, the user can see the contents of the registers and data memory and, more importantly, the

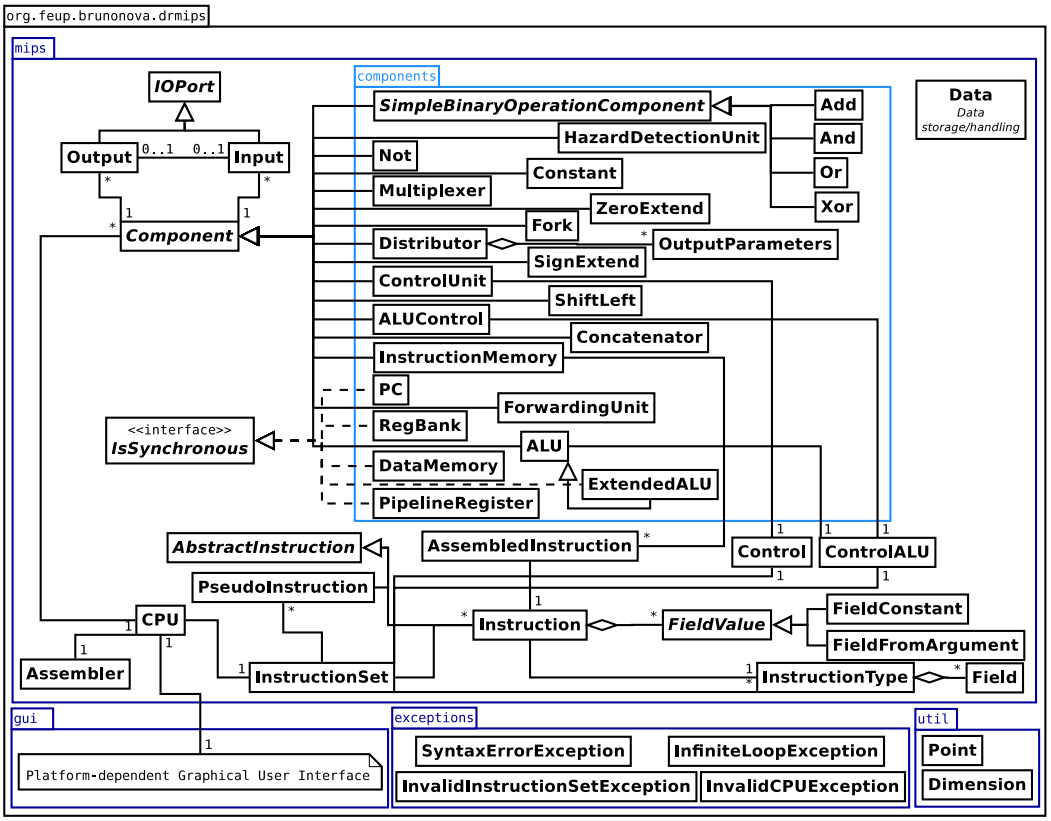

Fig. 1. Simulator Unified Modeling Language (UML) class diagram

composition and state of the unicycle or 5-stage pipeline datapath. It also shows the values at each input and output of each component, which wires are relevant for the execution of the current instruction and, in the case of the pipelined datapath, what instructions are in each stage. Besides seeing how the data flows in the datapath, the user can also view the latencies of the components and the critical path of the circuit using the "performance mode".

DrMIPS provides several different MIPS CPU datapaths, based on [1], including the unicycle datapath with some simplified variants and the pipelined datapath, with or without hazard detection and resolution. These CPUs can be created and configured by specifying in a file all the components and their properties and the wires connecting them. But, besides the datapaths, the instruction sets used by them can also be configured and new ones can be created, by specifying the properties of the different instruction types, instructions and pseudo-instructions and what they do. Floating point operations and syscalls are currently not supported.

The simulator was implemented not only for the Personal Computer (PC) but also for Android devices, especially tablets. For that reason, the simulator was developed in Java. This makes porting code from the PC version to the Android version and vice-versa easy, as Android also uses Java, and makes the PC version runnable from most operating systems without much trouble. The user interface also supports multiple languages. At the moment only Portuguese and English are available. As for the IDEs, Netbeans was chosen for the development of the PC version, as it makes the creation of user interfaces very easy, while for the Android version the chosen one was Eclipse, as it is the one recommended by Google.

To ease the development of both versions, the code was divided in two parts: the simulation logic and the user interface. With this division, only the user interface part is dependent on the platform (PC or Android), while the simulation logic part is exactly the same for both platforms. Figure 1 shows a simplified UML class diagram of the simulator.

The code consists of the following Java packages:

- `mips`: contains the simulation logic.
- `mips.components`: inner package that defines all the types of components.
- `gui`: contains the platform dependent user interface.
- `util`: contains some utility classes.
- `exceptions`: contains the exception classes.

The rest of this section discusses the implementation details of the simulation logic, while section IV discusses the details of both versions' user interfaces.

A. CPU Definition

Each CPU is defined in a JSON file. JavaScript Object Notation (JSON) is a file format that can be parsed easily in

Java and in Android. It is also easy for humans to read/write and creates smaller files than other formats like eXtensible Markup Language (XML). A CPU file lists the components and their properties, the wires connecting the components, the "friendly" names of each register and the used instruction set (described in the next sub-section).

In the code, as seen in Figure 1, the `CPU` class is the central part of the simulator. It grants access to all its components, the instruction set and the assembler and is the "interface" that the user interface uses to assemble programs, execute cycles, get values and latencies, etc. It is also where the CPU JSON files are loaded and parsed.

A CPU is comprised of several components. The `Component` class is the base class for all the components. Each component must extend from this class, set the required properties and implement `execute()`, where its behaviour is defined, using the values of the inputs to set the correct values of the outputs. And, of course, the component has inputs and outputs, defined by `Input` and `Output` respectively. Each output can be connected to an input, which represents a wire. The CPU can, therefore, be represented as a graph.

Some of the components are synchronous, like the program counter and register bank. These components have an internal state that can be changed only during a clock transition. In terms of code, these components implement the `IsSynchronous` interface, and must then implement the `executeSynchronous()` method, where the synchronous behaviour that changes the component's internal state is defined. Furthermore, to allow the user to return to previous states during execution (i.e. back step), the internal states are saved in each clock cycle in a stack and some additional methods to save and restore these states were also implemented.

When executing code, each clock cycle starts by executing the synchronous behaviour of the synchronous components, and then proceeds to execute the "normal" behaviour of the components, starting by the synchronous ones. During this process, the outputs of the components usually have their values changed. When this happens, the new value is propagated to the connected input, which then executes the "normal" behaviour of the input's component and, possibly, continues the data propagation. This only occurs when the new value is changed (i.e. the new value is different from the previous one), avoiding infinite loops that would have resulted from this solution. This also means that the synchronous behaviour of the components cannot cause any data propagation.

As for accumulated latencies and critical path, they are calculated using propagation too, but are only calculated when the CPU is loaded or when a latency is changed by the user. Each component has its individual latency and an accumulated latency. Each input also stores the accumulated latency. The calculation of the accumulated latencies starts on the synchronous components and is propagated up to inputs that are only used by synchronous behaviours (like the `WriteData` input of the register bank). The critical path is, then, calculated backwards from the input or inputs with the highest accumulated latency. It is worth noting that a CPU can have multiple critical paths.

Regarding the pipeline version of the MIPS CPU, it must have exactly five stages and, thus, four pipeline registers that separate them. These registers, and the program counter, are used to determine what instructions are in each stage of the pipeline by checking their `Write` and `Flush` control signals. This is necessary due to the hazards that can occur. Both an hazard detection unit and a forwarding unit were implemented, and their behaviours were based on [1].

In each clock cycle, each wire can be marked as "irrelevant" (gray on the user interface) or relevant. This decision is made based only on the values in the wires and components, and not on the instruction, as it would be fairly difficult to determine the relevant wires and components for each instruction when the CPU and even the instruction set is very generic and configurable. That said, the conditions to mark a wire as irrelevant are very simple: the wire carries one bit with the value zero, a stall is occurring, the wire is not selected by a multiplexer, etc.

B. *Instruction Set Definition*

Like the CPU, each instruction set is defined in a JSON file. An instruction set file lists the instruction types, the instructions, the pseudo-instructions and their properties and also specify how the control unit, the Arithmetic and Logic Unit (ALU) and ALU control work.

As shown in Figure 1, the main class that defines the instruction set is `InstructionSet` and is accessible from the `CPU`. This class loads and parses the instruction set from the file specified in the CPU file, and grants access to all of the instruction types, instructions, pseudo-instructions and control definitions.

An instruction set has several different instructions. In the MIPS case, which is a Reduced Instruction Set Computer (RISC) architecture, all of the instructions have the same size. Each instruction belongs to a type and, for the MIPS, that means one of `R`, `I` or `J`. The 32 bits that comprise the instruction code are split in fields. The fields are different for each type, except the `opcode` field, which is always the first field and has always the same size. `InstructionType` is the class that represents this information.

The most complete implemented datapath supports the following instructions: `nop`, `add`, `sub`, `and`, `or`, `slt`, `addi`, `lw`, `sw`, `beq`, `j`, `nor`, `xor`, `mult`, `div`, `mfhi` and `mflo`. This datapath also includes the following pseudo-instructions: `li`, `la`, `move`, `subi`, `sgt`, `neg`, `bge`, `ble`, `b`, `not`, `mul` and `rem`.

The instructions and pseudo-instructions are represented by the `Instruction` and `PseudoInstruction` classes respectively. Each instruction belongs to a type, has a mnemonic and defines the number and type of arguments, the values of each field (that may come from an argument) and a symbolic description that can be viewed by the user. Each pseudo-instruction has a mnemonic and defines the number and type of arguments, the instructions it is converted to when assembling and a symbolic description. In the developed simulator, each mnemonic can only belong to one instruction or pseudo-instruction.

An instruction set also has to define what the instructions do. That is defined in the `Control` and `ControlALU` classes. The `Control` class controls the behaviour of the control unit, and contains the information necessary to produce the right control signals for each instruction `opcode`. The `ControlALU` class controls the behaviour of both the ALU and ALU control components, and contains the information necessary for the ALU control to produce the right outputs for each `ALUOp` and instruction `func` field combination, and also the correspondence between each possible ALU control input and operation.

C. The Assembler

The assembler is represented by the `Assembler` class, and is accessible from the `CPU`, as visible in Figure 1. When the code is to be assembled, the user interface uses the `CPU`'s `Assembler` to parse the code, consulting the CPU's `InstructionSet` and converting the instructions and pseudo-instructions in the text segment to assembled instructions, represented by the `AssembledInstruction` class, which are then loaded into the CPU's instruction memory. The assembling process also involves parsing the data segment in the code to initialize the CPU's data memory with the specified values.

The code entered by the user can have errors, which must be shown to the user. When an error is found in an instruction, the assembler throws an exception. However, instead of stopping the assembler here, the exception is caught and added to a list of exceptions. The assembler then resumes in the next instruction. With this technique, all the errors present in the code can be shown to the user. Only one error can be displayed per line, though.

The implementation of the assembler is fairly simple, and the assembling process is done in two steps:

1) The assembler parses the data segment and loads the values to the data memory, while also keeping track of labels and converting the pseudo-instructions into instructions in the text segment.
2) All instructions are converted into the corresponding instances of `AssembledInstruction` and assembled to machine code, being then loaded into the CPU's instruction memory.

In terms of assembler directives, four directives are currently supported:

- `.data`: starts the data segment, where the data memory is initialized
- `.text`: starts the text segment, where the code is defined
- `.word`: declares one or more values to be stored in the data memory as 32 bits words
- `.space`: reserves some bytes in the data memory

IV. Interface Implementation

The simulator was developed for both computers and Android devices, especially tablets. This section discusses some details about the implementation of their user interfaces. Looking at the UML class diagram in Figure 1, the user interface code is defined in the `gui` package, and is the only package that differs between the PC and Android versions.

A. PC Version

The PC version is the most complete. By default, the interface is shown with a light theme and with the contents split in five tabs, as shown in Figure 2. Two tabs can be viewed at once, as the window is split in two sides horizontally, and the user can move any tab from one side to the other by right clicking on the tab title and selecting the only option from the pop-up menu. However, the user can choose to use a dark theme and can also choose to use internal windows instead of tabs, as shown in Figure 3. Using internal windows is useful in large screens, and the windows' positions and sizes are remembered on exit. The interface supports multiple languages, Portuguese and English at this stage, and they can be changed in the menus.

The different tabs or windows are:

- *Code*: contains the code editor.
- *Assembled*: displays the assembled instructions and resulting machine code.
- *Datapath*: displays the datapath and the instruction(s) being executed.
- *Registers*: lists the registers and their values.
- *Data memory*: shows the values in the data memory.

DrMIPS provides a code editor with syntax-highlighting, auto-complete, search/replace, line numbers, undo and redo thanks to the RSyntaxtTextArea component and AutoComplete library [17]. The syntax-highlighting and auto-complete rules are not "static" and depend on the loaded CPU datapath. The auto-complete provided is mostly for help and is activated by `<Ctrl>+<Space>`. All of the errors present in the code are indicated close to the line numbers when the user assembles the code.

The assembled code table displays the resulting machine code and highlights the instruction or instructions being executed by the CPU. Hovering the mouse cursor over each instruction also displays the type of the instruction and the values of the instruction's fields, as shown in Figure 3. The registers and data memory tables display their respective values but also highlight the registers or addresses being accessed. Registers and values in the data memory can be changed by double-clicking them in the table. All these values can be displayed in decimal, binary and hexadecimal formats.

The datapath is probably the most important part of the simulator. The components are all displayed as rectangles and represented internally as `JPanels`. The component's name, description, input and output values, or latency are shown as a tooltip when the user hovers the mouse cursor over it, as shown in Figure 2. The wires are drawn on the background of the datapath and have different colors when they are irrelevant or belong to the control path. Some of the components' inputs and outputs display a small permanent tip with the current value of

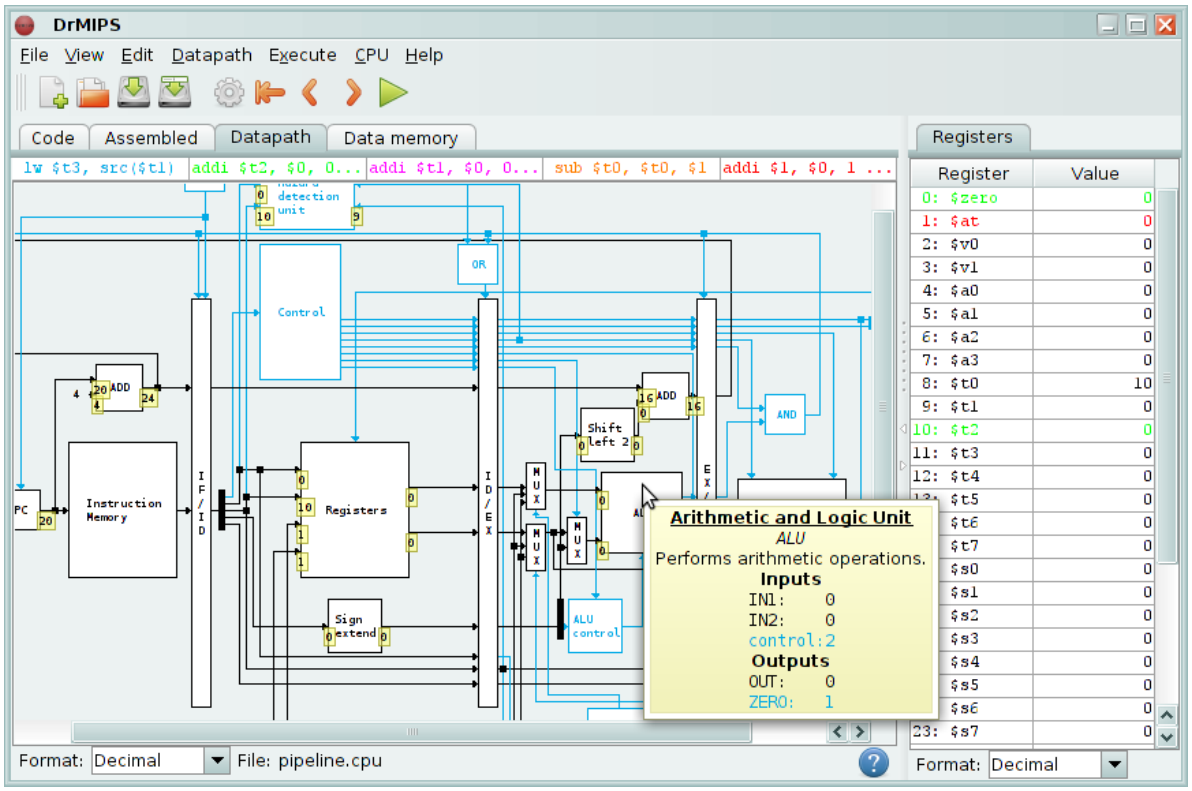

Fig. 2. DrMIPS for the PC with the default options, with the tooltip of a component being shown

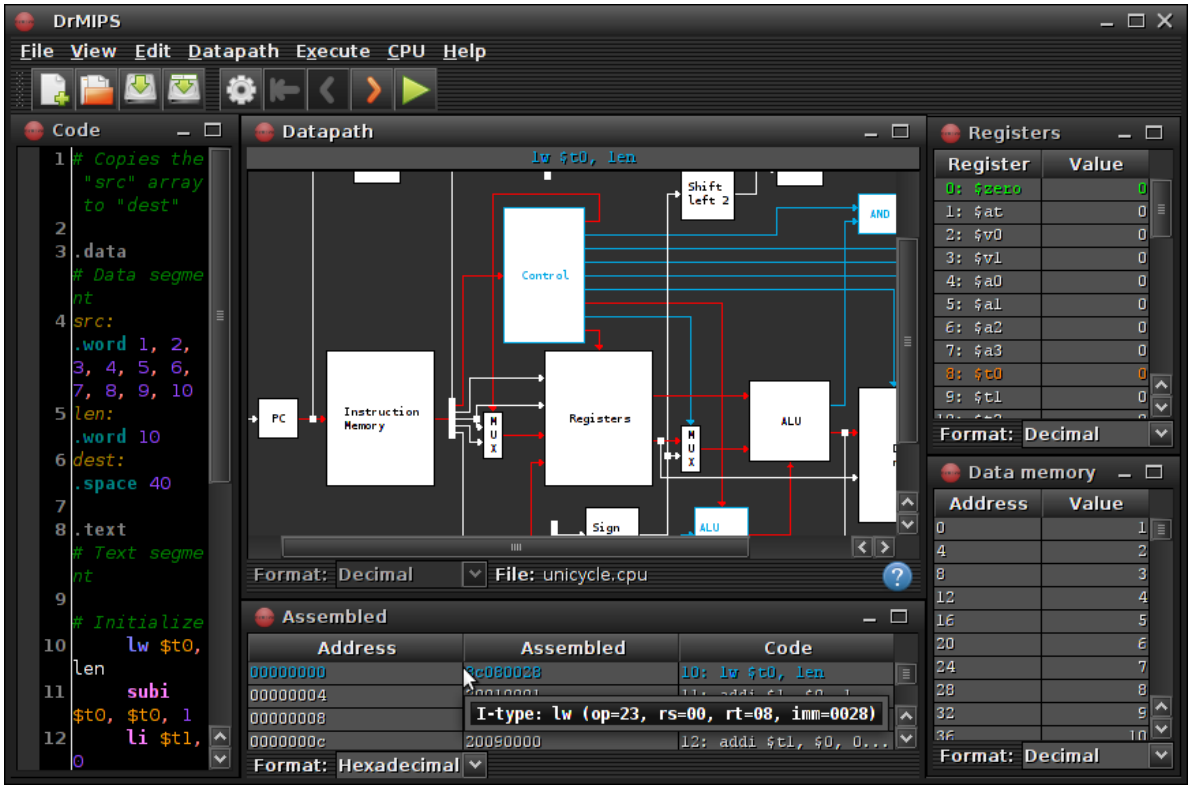

Fig. 3. DrMIPS for the PC using internal windows and dark theme while in “performance mode”

the input or output. That data tip is always placed below the input or output's point of entry or exit, and is represented by a `JLabel`. The datapath is a `JLayeredPane` to allow the data tips to be always on top. The user can hide the control path, the data tips and the arrows in the wires.

It was decided to make the graphical datapath as small as possible, but still readable. This way, a larger portion of the datapath, or even the whole datapath, will be visible. Because of this some informations, like the names of the inputs and outputs, must be omitted.

The datapath can also be displayed in a "performance mode". In this mode, the wires that belong to the datapath are shown in red, the data tips are hidden and the components' tooltips display their latencies and accumulated latencies instead of values. Also, double-clicking a component while in this mode allows the user to edit the latency of the component. The new value is not stored in the CPU file.

The instruction or instructions currently being executed are displayed above the datapath in a table with just one line and no header. Using a table like this means that the columns will probably not be aligned with the corresponding pipeline registers in the datapath, but it also means that they are always visible even if the whole datapath doesn't fit in the screen, as can be seen in Figure 2.

B. Android Version

The Android version is very similar to the PC version. Like in the PC version, the application uses a light theme by default, but a dark theme is also available. Figure 4 shows the application running on a tablet with Android 4.0.3 using the light theme, while Figure 5 shows it running on a smartphone with Android 4.1.2 using the dark theme.

The application contains only one activity and its contents are split in tabs using a `TabHost`. The contents of the different tabs are defined in different layout files which are included in the activity layout, without using the new Android fragments feature. The application supports screen rotations without losing its state. Currently, the application can be shown in Portuguese and English.

The code editor of this version is a simple `EditText`. Also, due to the fact that MIPS assembly programs rely heavily on the dollar sign (`$`) to reference registers, using the default on-screen keyboard may be very annoying, as the dollar sign is usually not on its first page.

The assembled code, registers and data memory tables are similar to the ones in the PC version. However, to view an instruction "tooltip" in the assembled code, the user must press on the instruction instead of hovering the "cursor" over it, as mobile devices don't have a cursor nor a mouse. Also, to change a register or a value in the data memory, the user must long-press it in the table instead of double-clicking. This

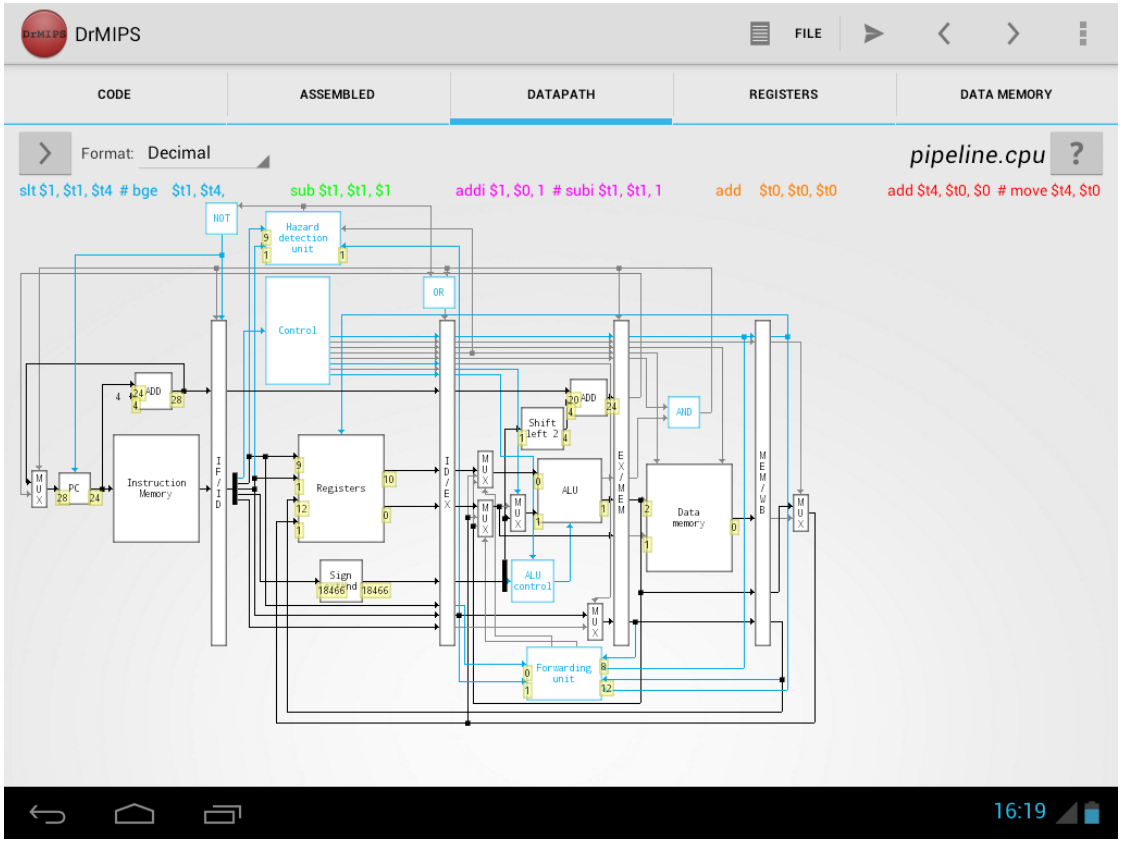

Fig. 4. DrMIPS for the Android using the light version on a tablet

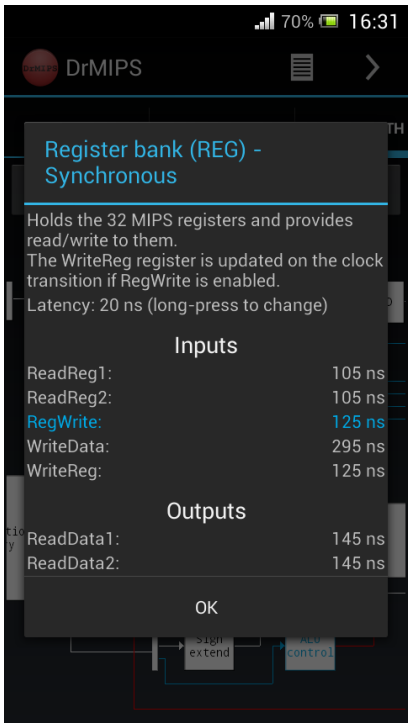

Fig. 5. DrMIPS for the Android using the dark version on a smartphone, displaying the details of a component while in "performance mode"

behaviour is used in other parts of the interface.

The datapath is displayed in the same way as in the PC version. It uses a `RelativeLayout` to display the components in the specified positions and with the specified sizes, all measured using density-independent pixels, which is a unit that makes the graphical components have approximately the same real size in all devices. Each component is a `TextView` and its name, description, input and output values, or latency can be displayed by pressing the component, as shown if Figure 5. Long-pressing it while in performance mode lets the user change the latency. The data tips are also displayed in the same way as in the PC version and, because they are the last user interface components added, they are always shown on top.

The CPU, instruction set and code files are stored in the application's data directory, preferably in the external memory. This memory, usually mounted in `/mnt/sdcard`, in many devices is actually a partition in the device's internal memory and not the external memory card. But, in case it is the external memory card and it is not available, the application stores the files on the application's private directory in the device's internal memory. Storing the files in the external memory allows the user to access them using another application like a file explorer. To know the path to the loaded CPU or code file, the user can press on the file's name in the user interface.

V. Conclusion and Future Work

A tool to aid computer architecture students and teachers was presented. This tool, a MIPS simulator, is fairly versatile, intuitive, configurable and aggregates several different features that are found scattered through similar tools. It supports both unicycle and pipeline versions of the CPU, displays the datapath graphically and has a "performance mode". This tool is also available for Android, while none of the similar tools has an Android version. However, due to lack of time, some intended features where not implemented, like timing diagrams and a graphical CPU editor, besides some additional polishing. But the end result is very positive.

Acknowledgements

The author would like to thank the DEI of FEUP and professors António Araújo and João Canas Ferreira from the DEEC for the given orientation and advices, and also for proposing this work .

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design - The Hardware/Software Interface*, 3rd ed. Morgan Kaufmann, 2005.

[2] J. L. S. C. Pereira, "Educational package based on the MIPS architecture for FPGA platforms," Master Thesis, Faculdade de Engenharia da Universidade do Porto, June 2009, accessed on June 6, 2013. http://repositorio-aberto.up.pt/bitstream/10216/59975/1/000135086.pdf.

[3] J. Larus, "SPIM: A MIPS32 Simulator," accessed on June 6, 2013. http://spimsimulator.sourceforge.net.

[4] D. K. Vollmar and D. P. Sanderson, "MARS: An Education-Oriented MIPS Assembly Language Simulator," March 2006, accessed on June 6, 2013. http://www.cs.missouristate.edu/~vollmar/MARS/fp288-vollmar.pdf.

[5] J. Larus, "SPIM S20: A MIPS R2000 Simulator," Computer Sciences Department, University of Wisconsin, Tech. Rep., 1990, accessed on June 6, 2013. http://phoenix.goucher.edu/~kelliher/f2005/cs220/spim.pdf.

[6] G. C. R. Sales, M. R. D. Araújo, F. L. C. Pádua, and F. L. C. Júnior, "MIPS X-Ray: A Plug-in to MARS Simulator for Datapath Visualization," 2010.

[7] J. Garton, "ProcessorSim – A Visual MIPS R2000 Processor Simulator," 2005, accessed on June 6, 2013. http://jamesgart.com/procsim.

[8] H. Sarjoughian, Y. Chen, and K. Burger, "A Component-based Visual Simulator for MIPS32 Processors," *38th Frontiers in Education Conference*, pp. F3B–9 – F3B–14, October 2008.

[9] A. Gascoyne-Cecil, "MIPS-Datapath," accessed on June 6, 2013. http://mi.eng.cam.ac.uk/~ahg/MIPS-Datapath.

[10] I. Branovic, R. Giorgi, and E. Martinelli, "WebMIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education," *Workshop on Computer Architecture Education, 31st International Symposium on Computer Architecture*, 2004, accessed on June 6, 2013. http://www4.ncsu.edu/~efg/wcae/2004/submissions/giorgi.pdf.

[11] T. E. Team, "EduMIPS64," accessed on June 6, 2013. http://www.edumips.org.

[12] D. Patti, A. Spadaccini, M. Palesi, F. Fazzino, and V. Catania, "Supporting Undergraduate Computer Architecture Students Using a Visual MIPS64 CPU Simulator," *IEEE Transactions on Education*, vol. 55, no. 3, pp. 406 – 411, August 2012, accessed on June 6, 2013.

[13] "EduMIPS64 Students Questionnaire," accessed on June 6, 2013. http://www.diit.unict.it/users/spadaccini/edumips64-survey.html.

[14] M. Scott, "WinMIPS64," April 2012, accessed on June 6, 2013. http://indigo.ie/~mscott.

[15] M. Butler, "Android: Changing the Mobile Landscape," *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4 – 7, January-March 2011.

[16] R. Shim, "Tablets Impact the Notebook Market: Enter the Ultrabook," *Information Display*, vol. 28, no. 2 and 3, pp. 12 – 14, February/March 2012, accessed on June 6, 2013. http://sid.calcey.net/Portals/InformationDisplay/IssuePDF/03_2012.pdf#page=14.

[17] Fifesoft, "RSyntaxTextArea — Fifesoft," 2013, accessed on May 27, 2013. http://fifesoft.com/rsyntaxtextarea.