Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

QtARMSim: an ARM simulator for teaching Computer Architecture

Sergio Barrachina* | Germán Fabregat | Juan Carlos Fernández | Germán León

[1]Department of Computer Engineering and Science, Jaume I University, Castellón, Spain

Correspondence
*Sergio Barrachina. Email: barrachi@uji.es

Present Address
Department of Computer Engineering and Science. Jaume I University. Avda. Vicente Sos Baynat, s/n, 12071, Castellón de la Plana, Spain

Summary

Many of the teaching objectives of introductory Computer Architecture courses are rooted in the perception of the computer held by the programmer of assembly language. As a general rule, the teaching framework presumes a specific computer architecture, which normally observes two criteria: it should be simple, and, simultaneously, it should inspire motivation in the students.

The ARM architecture is an ideal training ground for teaching Computer Architecture. On one hand, as it is based on the Reduced Instruction Set Computer (RISC) architecture it is relatively simple. On the other hand, it is a modern and widely used architecture, especially in mobile phones, smartphones and tablets, which provides motivation for the students.

For practical work assignments based on ARM, either an ARM simulator or a development tool on an ARM machine should be used. Therefore, when designing a first year undergraduate course, we believe that the selected tool should be easy to use, multiplatform, free, and open.

After evaluating existing options, and aligned with the above objectives, we decided to develop and release an ARM simulator, ARMSim, and a graphic interface for that simulator, QtARMSim. This software has already been used in introductory Computer Architecture courses in 2014 and 2015, and the feedback received from both students and laboratory teachers has been very positive.

KEYWORDS:
Computer Assisted Education, Computer Architecture, ARM Thumb, Simulator, Assembly Language

1 | MOTIVATION

Historically, the content of courses on Computer Architecture has followed the frenetic pace set, first, by the technological advances in the design of large computers and, later (from the 1980s on), by the evolution in the design of microprocessors. The teaching of Computer Architecture may be seen to have passed through six main stages, based on: mainframes, minicomputers, the first microprocessors, microprocessors, RISC and post RISC. At each stage, and subject to cost constraints, universities have accessed specific hardware, including the PDP-11, the Motorola 68000, the Intel 80x86, the MIPS and the SPARC, which are amongst the machines and microprocessors that have enjoyed most popularity in the teaching of Computer Architecture. In some instances, however, hypothetical computers dedicated exclusively to the teaching of architectural concepts have been employed[1].

The Computer Architecture courses at the University of Jaume I, Spain, have also passed through the various stages mentioned above. The first system to be used was the 68000 Motorola; later a hypothetical computer was briefly used; subsequently, this was replaced by the MIPS architecture, which was used until 2013.

Early in 2013 the lecturers involved in teaching these courses proposed a change from MIPS to ARM, for the following reasons. On one hand, while the ARM architecture has many characteristics that distinguish it from other contemporary architectures, it is relatively simple since it is based on RISC[2]. On the other hand, ARM is currently widely used, especially in mobile phones, smartphones and tablets, which substantially contributes to the motivation of the students[3]. In fact, over the past two decades, the ARM architecture has exploded in popularity because of its efficiency and rich ecosystem. More than 50 billion ARM processors have been sold and distributed, and more than 75% of the population of the world use products with ARM processors[4].

Once the decision to make this change had been taken, all the instruction materials were updated in accordance, for both the theoretical and the laboratory courses concerned with the subject of Computer Architecture. In the case of the Computer Organization course, which is taught in the first semester of the Computer Engineering and Computational Mathematics degrees at the Jaume I University (Spain), one of the primary objectives was to select an ARM simulator or framework that could be used in the laboratory and was commensurate with the Arduino Due platform[5] used in other courses, which is equipped with a microprocessor based on the ARM Thumb 2 architecture.

In previous years, a MIPS simulator, `xspim`, was used in the laboratory sessions of this course (`xspim` is a previous version of the current `QtSpim`[1] simulator)[6]. The `xspim` simulator was a suitable platform for the practical work in the Computer Organization course for the following reasons: i) it permitted step-by-step simulation of the execution of assembly programs; ii) it visualized the content of the registers and memory in a simple way; and iii) it was a free, open and multiplatform software package. Given that `xspim` was sufficiently simple, the students were able to concentrate on the development and simulation of programs, rather than on operating the simulator. In addition, since it was a free multiplatform software package, the students were able to download it and practice in their own time and on their own computers.

Although there are quite a few simulators suitable for teaching courses in Computer Architecture and in Computer Organization (see, for example, the survey in[7]), we were particularly interested in ARM simulators/frameworks that focused on bestowing the fundamentals of Computer Architecture as defined in the Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering[8].

The first option considered for replacing `xspim` with an ARM framework was the µVision[2] integrated development framework, from the Keil company. This framework is the most widely used in the programming of microcontrollers based on ARM (the Keil software company was bought by ARM in 2005) and is cited in books about ARM such as[9] and[10]. It includes C and C++ compilers, assemblers, linkers, filters, debuggers and simulators of various microcontrollers based on ARM.

While adopting µVision would have provided a realistic framework for embedded systems programming, its operation would have been slightly more complex than was suitable for a first year undergraduate course that was intended to explain the fundamentals of Computer Architecture, rather than to teach how to program embedded systems. Further, there is only a Microsoft Windows version, which is not open software, and while there is a free of charge version there are certain limitations - 32 KiB of code (which would not, in fact, have caused any problem in the case under discussion), and the registration requirement for its download (under which the students would need to submit personal information prior to its use).

Another option considered, related to the µVision IDE, was a product called Lab-in-a-Box[3] from ARM University. This product includes: i) licences of Keil MDK-ARM Professional Software Tool (which forms part of µVision IDE); ii) Freescale cards based on the ARM Cortex-M0+ processor; and iii) teaching material for both theoretical and laboratory classes. However, having ascertained that the course was based exclusively on the use of C for programming embedded systems, this alternative had to be discarded as a valid option for this course.

The next alternative considered was the use of Eclipse DS-5. In the same way that µVision is related to embedded systems, Eclipse DS-5 is the development framework provided by ARM for ARM System-on-a-Chip processors. It allows the programmer to use C/C++ and assembly, as well as to debug the code which is being executed on the computer for which it is being developed. Although the usual version of Eclipse DS-5 is neither open nor free, there is a version called Community Edition which is both. Furthermore, since it is actually an Eclipse plugin, it is a multiplatform solution. Eclipse DS-5 does not provide a true simulation environment, although the debugging options can nevertheless be used to mimic a simulation environment. This can be achieved by creating a virtual ARM machine (e.g. with QEMU), installing a GNU/Linux operating system on the virtual machine and

---

[1] QtSpim can be downloaded from: http://spimsimulator.sourceforge.net/
[2] µVision Getting Started book: http://www.keil.com/product/brochures/uv4.pdf
[3] ARM Course/Lab: http://arm.com/support/university/educators/embedded/

configuring it so that a GDB server can be accessed from Eclipse DS-5 through SSH. In this way, the debugging capacity of Eclipse DS-5 serves to permit the observation of the execution of a program without the need to use specific hardware. The same procedure can be followed using actual hardware, such as a Raspberry Pi. Although we were able to debug code using both methods (with a virtual machine and with a Raspberry Pi) the use of the Eclipse debugger as a simulation environment, as well as its setup, were far too complex for a first year undergraduate course.

Lastly, the ARMSim#[11] simulator was evaluated. This application facilitates simulation of the execution of assembly programs in a system based on the ARM7 processor. It is written in C# and works in Microsoft Windows; supposedly, it can also be implemented with Mac OS X and GNU/Linux using the Mono implementation of the .NET environment. While this simulator appeared to meet the specifications outlined for this course, and while it displayed interesting features such as the simulation and visualization of peripherals, and offered the possibility of developing plugins that could extend its operation to new I/O devices, it does not currently support the Thumb variant of ARM. This factor is necessary in order to link the simulator-based practical classes in this course with subsequent practical work in this and future degree courses, where the student is required to program an Arduino Due card (which uses an ARM Thumb2 processor).

Since none of the systems considered met the course requirements, we decided to develop our own ARM simulator[4], with the intention of releasing it for use by other universities in their introductory courses on Computer Architecture.

The simulator developed consists of a simulation engine, ARMSim, and a graphic interface, QtARMSim. The two applications are independent, so that a user may be interested only in the simulator and could develop his/her own graphic interface; conversely, the graphic interface could be modified in order to match it with another simulator. In addition, a textbook[5] has been written (in Spanish), in which the simulator is described in detail and a series of practical introductory exercises on the ARM Thumb architecture are offered.

An early version of the ARMSim and QtARMSim simulator was described in[12] (in Spanish). Later, in[13] (also in Spanish), the simulation engine was described in more detail, some improvements on the graphic interface were presented, and feedback obtained from the students, as well as an assessment of their academic progress before and after using the simulator, were outlined. This article presents the current version of the simulator, in which the major improvements and enhancement are: i) a color trace of the instructions that have been simulated; ii) a memory dump at the byte level; iii) a basic LCD device simulation; and iv) a bundled GCC assembler. These enhancements aim to provide the student with a better understanding of the execution flow and the memory organization, and to make the installation of the simulator easier.

The rest of the article is organized as follows: The second section describes the objective of the simulator. Section 3 describes the simulation engine, ARMSim. Section 4 describes QtARMSim, the graphic interface of the simulator. Section 5 presents the results obtained. And finally, in Section 6, conclusions are presented and future work suggested.

2 | OBJECTIVE

The main objective guiding the development of the simulator has been to provide the students with an ARM Thumb simulator that would facilitate their acquisition of the fundamentals of Computer Architecture. In order to achieve this teaching objective, the simulator had to be as simple as possible, free, multiplatform, and open, and it had to cover the entire process of development, assembly, and simulation of ARM Thumb assembly code. In addition to these requirements, we believed that it would be convenient for the simulation engine to be disengaged from the simulator graphic interface.

Poorly designed interfaces compel students to waste time that could be better spent trying to understand the actual study material, and they ultimately reduce the efficacy of the learning and the retention of subject matter. The user should be involved in the learning process without being overwhelmed[14]. Therefore, the simulator should be as easy to use as possible, so that the students may concentrate on the development and simulation of assembly programs, rather than on the operation and configuration of the simulator. Furthermore, the simulator should be both free and multiplatform, so that the students can install it at no cost on their own computers, regardless of the specific operating system they use.

The objective of releasing the simulator as open source software was to make it possible for the students to inspect the code of both the simulation engine and the graphic interface, as well as to enable third-party individuals or groups to become involved in the simulator's development or adapt it for their own needs.

[4] QtARMSim can be downloaded from: http://lorca.act.uji.es/projects/qtarmsim/
[5] The textbook can be downloaded from http://lorca.act.uji.es/libro/introARM/

As stated, we envisaged the simulator as an end-to-end solution, covering the processes of code editing, assembly, and simulation as simply as possible, so the student would not need to alternate between an editor (to write the assembly code), and a simulator (to assemble and simulate the code), as is commonly found in other simulation environments.

Lastly, we tried to disengage the simulation engine and the graphic interface as much as possible so that the simulator could be used through different (including remote) graphic interfaces, or the graphic interface could be used with different simulation engines. Due to this characteristic, the simulator actually consists of two applications: ARMSim, a simulation engine for the ARM Thumb architecture, described in the next section; and QtARMSim, a graphic interface for the simulator, described in Section 4.

3 | ARMSIM

ARMSim is a simulation engine for the ARM Thumb architecture. It provides a model of the processor, which includes registers and state indicators, and a memory model (both RAM and ROM). The model of the processor is able to decode and execute the entire set of instructions from the ARM Thumb architecture. The memory model makes it possible to set the amount of available memory of a given type, as well as to initiate, consult and modify its contents.

Apart from modelling the above-mentioned components, the simulator also allows an assembly source file to be indicated. Instead of implementing its own assembler, a more versatile and powerful option has been chosen: ARMSim executes an ARM assembler (e.g., the GNU GCC compiler); then, it decodes the object code file generated by the ARM assembler to initialize the ROM and the RAM models; concurrently, it extracts the information regarding which lines of the source code have generated each machine instruction. We note that ARMSim follows the specifications and execution algorithms described in the ARMv7-M Architecture Reference Manual [15].

As stated previously, ARMSim does not provide a graphic interface. Instead, when ARMSim is executed, it listens on the port indicated in the command line. In order to interact with ARMSim, it is necessary to establish a connection with that port (e.g., using the `telnet` application) and to use text commands to indicate which actions should be taken.

The syntax of the commands recognized by the ARMSim simulation engine is described in Figure 1. It can be seen that the commands accepted by ARMSim allow the user to: i) assemble a file; ii) consult and modify the content of registers and memory addresses; iii) disassemble memory addresses; iv) define, remove, and consult breakpoints; v) execute code from a given position until the program is finished or until a breakpoint is reached; and vi) execute the program step by step (with or without entering subroutines).

4 | QTARMSIM

As has already been mentioned, QtARMSim is the graphic interface for the ARMSim simulation engine. The relationship between QtARMSim, which is described in this section, ARMSim, described in the previous section, and the GNU GCC compiler are schematically presented in Figure 2. It can be seen that QtARMSim takes care of editing the assembly source file and delegates the simulation of the object code to ARMSim. The communication between the graphic interface and the simulation engine is done using the telnet protocol. The graphic interface sends the actions which the simulation engine has to undertake and, in reply, receives the results of these same actions. A second interaction occurs between the simulation engine and the GNU GCC compiler. When required, the simulation engine executes the `gcc` command in a suitable way in order to assemble the source code indicated by the graphic interface. Once the the source code has been assembled, the simulation engine loads the resulting object code into the memory, informs the graphic interface of its new state and then waits for further requests.

In addition, the graphic interface is also responsible for starting the simulation engine and for managing its configuration (which port it should listen to, the `gcc` command path, and its compilation parameters).

Clearly, these interactions are hidden from the graphic interface user. When QtARMSim is executed, the user simply interacts with a graphic window like the one shown in Figure 3. QtARMSim presents two working modes: the edit mode and the simulation mode. Figure 3 displays a sample QtARMSim main window when editing a brief code (i.e., in edit mode). In this mode, the central part of the window corresponds to the editor of the assembly source code. A number of panels are distributed around this central part. In the default layout, the registers panel is located to the left of the editor, and the memory panel to its right. Below, another section is displayed with three tabbed panels: a message panel, a memory dump panel, and an LCD panel. The last two have been added to the previous design described in [13].

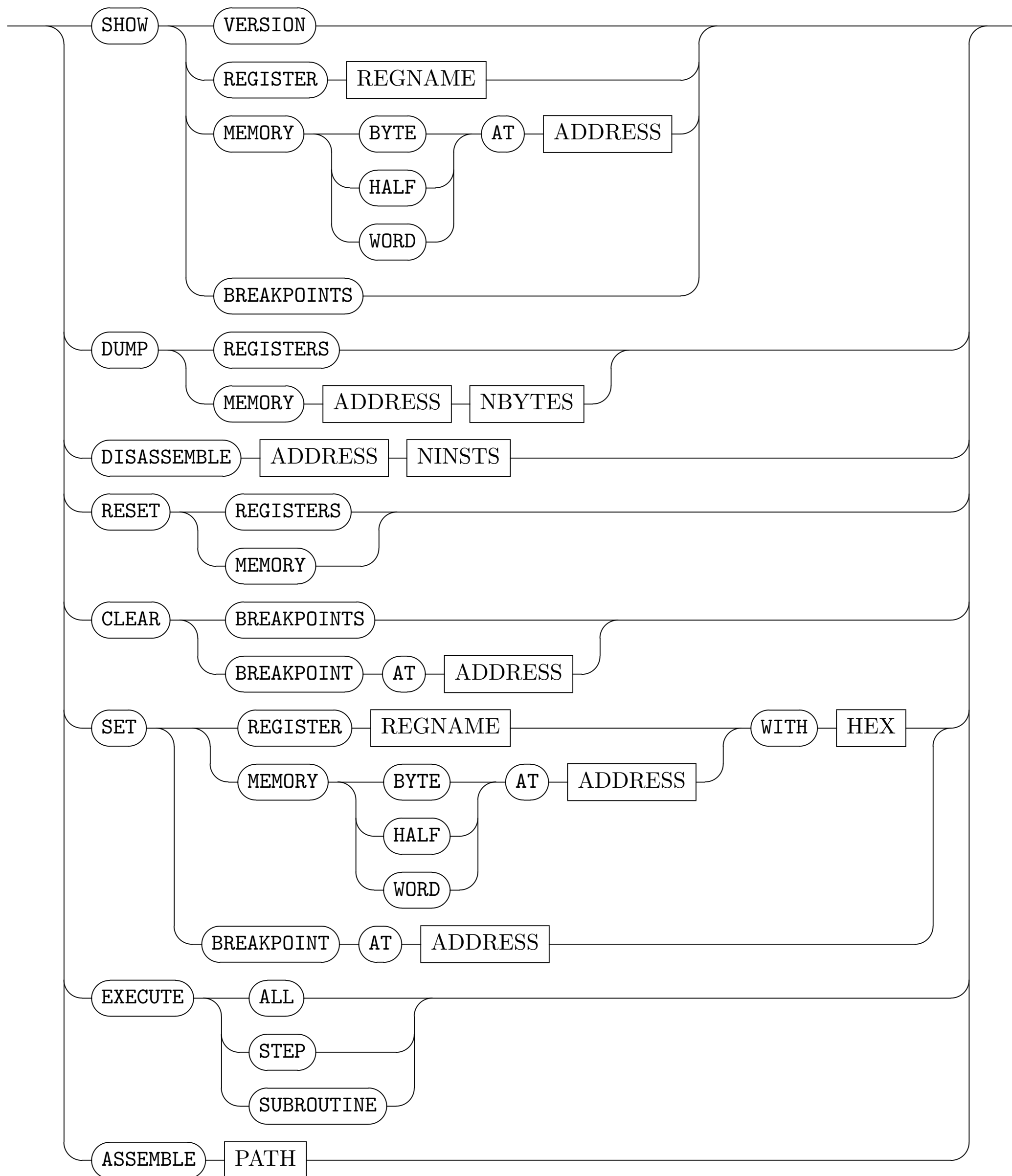

FIGURE 1 Syntax of the commands recognized by the ARMSim simulation engine

As can be seen in Figure 3, the memory and registers panels are disabled when in the edit mode; these will be described in Section 4.3. As far as the message panel is concerned, it has the function of showing those messages related to the actions which are being done. For example, whether the source code has been correctly assembled or not, which line has just been executed, etc. Finally, the memory dump and LCD panels, which are used when in simulation mode, will be described in Section 4.3.

As all the panels are implemented as dockable widgets, it is possible to show, hide, resize, detach or stack them. This feature allows the user to fully customize the presentation of the information at any time. As an example, a different layout, in which the registers and memory panels have been closed, can be seen in Figure 6. In addition, a convenient *Restore Default Layout* menu entry is provided to restore the default layout. This option is especially useful when a student has accidentally wrecked the simulator layout in a laboratory session, as it enables the teacher to easily restore the default layout.

The current version of QtARMSim is an improvement on the version described in [13], in that it is bundled with the GNU GCC ARM binary files required to assemble an ARM source code on GNU/Linux, Microsoft Windows, and Mac OS X. This bundling simplifies both the QtARMSim installation process and the automatic GNU GCC path configuration (see Section 4.1). Further,

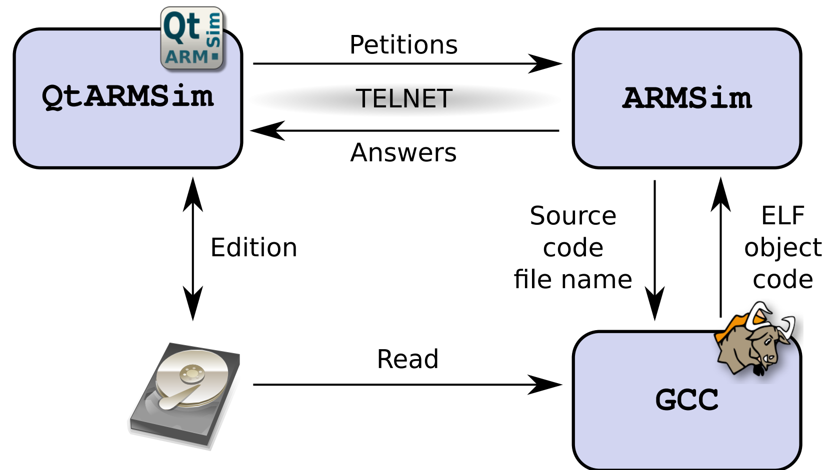

FIGURE 2 Components of the simulator and their interactions

as only those files of the GNU GCC toolchain that are strictly required to assemble are included, much less space is required than would be needed if the whole GNU GCC for the ARM toolchain were to be manually installed by the user.

In the next subsections, the QtARMSim configuration and its edit and simulation modes will be described.

4.1 | Configuration

In its first execution, QtARMSim will automatically try to find out all its necessary parameters. Therefore, in most cases, the user will not need to manually modify the QtARMSim configuration. Nevertheless, it is possible to manually configure how to execute the ARMSim simulation engine or where the GNU GCC cross compiler for ARM is installed. These options are entered in QtARMSim's preferences dialog box (see Figure 4). The top frame of the configuration dialog box corresponds to parameters related to the ARMSim simulation engine and enables configuration of: the server and port to which the graphic interface must be connected; the command line for executing the simulation engine (in the case where it is executed on the same machine as the interface); and its working directory. The bottom frame manages the configuration related to the GNU GCC compiler for ARM. This frame can be used to indicate the route to the GNU GCC compiler for ARM executable and the command line options that should be used to assemble the source code.

It should be noted than the configuration dialog box provides a *Restore Defaults* button that makes it possible to restore the automatically obtained configuration. As mentioned above with regard to the *Restore Default Layout* menu option, this button is especially useful in the laboratory class context, as it provides the teacher with a rapid way to recover the QtARMSim default configuration in the case where a student has inadvertently wrecked it.

4.2 | Edit mode

In edit mode, as mentioned, the central part of the window is an ARM assembly source code editor, in which the student may write the assembly program he wishes to simulate. Figure 3, above, shows a QtARMSim window in which a small assembly program is being edited. It can be seen that the editor recognizes the ARM Thumb assembly language and conveniently highlights the source code. In addition to this syntax highlighting, the source code editor also offers the following features: i) it allows the font size to be changed, which is very helpful when doing classroom presentations; and ii) both registers and labels in the source code are recognized, so that other occurrences of the currently focused register or label are highlighted, which makes it easier to follow data dependencies during the development or debugging of the code. One example of this last characteristic can be seen in Figure 3; as the text cursor is placed above an appearance of the `r1` register, every occurrence of this register in the source code is highlighted in yellow. The same will happen when the text cursor is on a label.

introsim-cubes.s [modified] - QtARMSim

File Edit View Run Help

Registers

Register	Value
General	
r0	0x00000000
r1	0x00000000
r2	0x00000000
r3	0x00000000
r4	0x00000000
r5	0x00000000
r6	0x00000000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13 (SP)	0x00000000
r14 (LR)	0x00000000
r15 (PC)	0x00000000

```
1          .text
2  main:   mov  r0, #0        @ Total to 0
3          mov  r1, #10       @ Initialices n to 10
4  loop:   mov  r2, r1        @ Copies n to r2
5          mul  r2, r1        @ Stores n^2 in r2
6          mul  r2, r1        @ Stores n^3 in r2
7          add  r0, r0, r2    @ Adds r0 and n^3
8          sub  r1, r1, #1    @ Substracts 1 to n
9          bne  loop          @ Jumps to loop if n != 0
10 stop:   wfi
11         .end
12
```

Source Code | ARMSim

Memory

Address	Value

Messages

QtARMSim version 0.3.4
(c) 2014-16 Sergio Barrachina Mir
Developed at the Jaume I University, Castellón, Spain.

Messages | Memory Dump | LCD Display

Ready Flags: - - - -

FIGURE 3 QtARMSim main window in edit mode

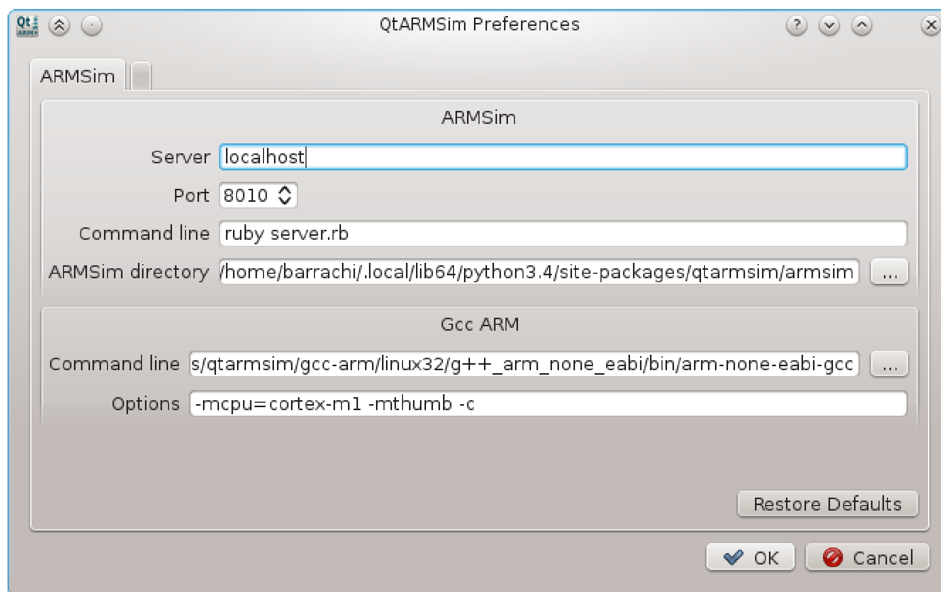

FIGURE 4 QtARMSim preferences dialog box

4.3 | Simulation Mode

In order to enter simulation mode it is only necessary to click on the tab labeled "ARMSim"; this tab can be found below the central section of the main window. The appearance of QtARMSim in simulation mode is illustrated in Figure 5.

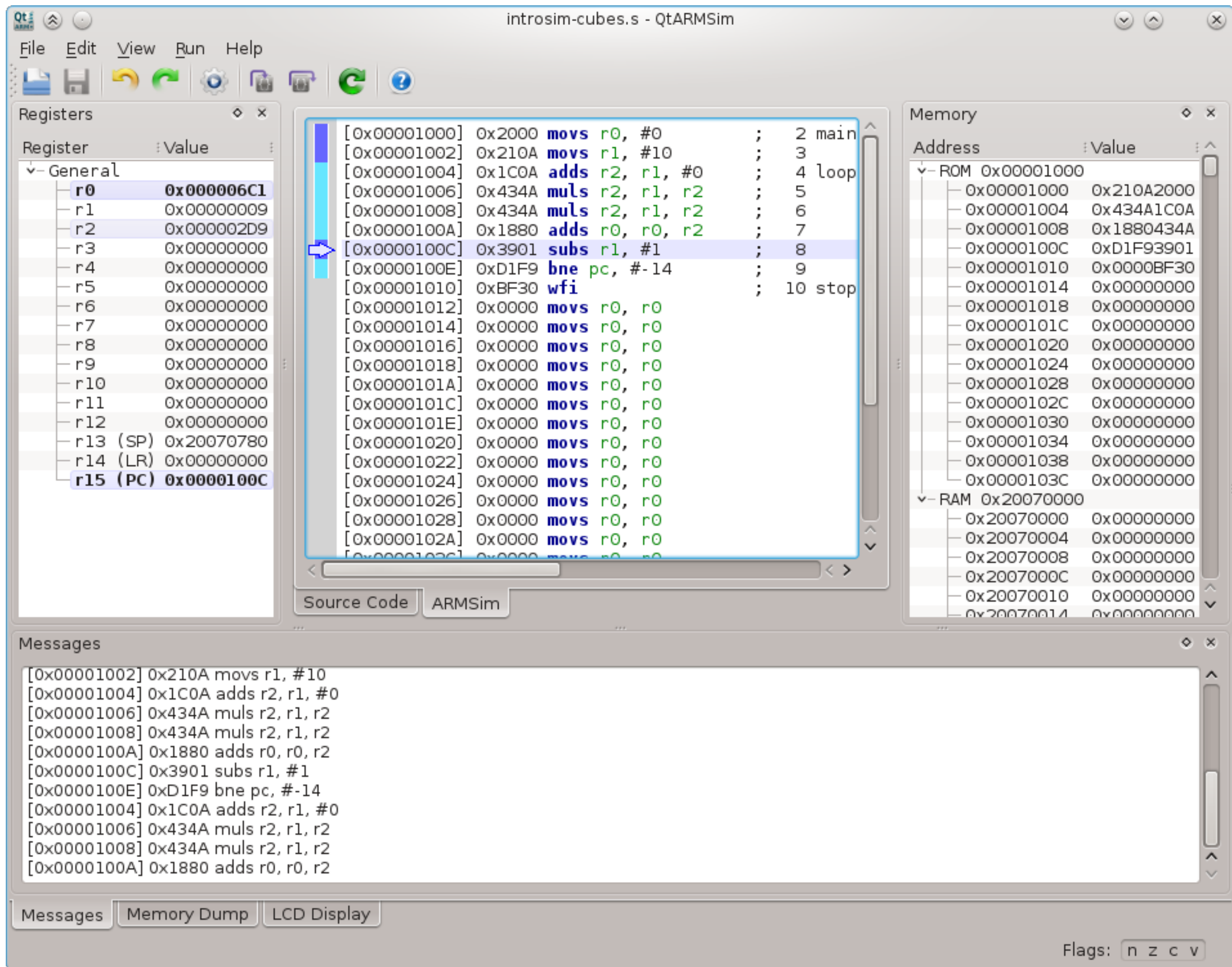

FIGURE 5 QtARMSim in simulation mode after executing some instructions

When changing from edit to simulation mode, behind the scenes the graphic interface instructs the ARMSim simulation engine to perform the following actions: i) to execute the cross assembler to assemble the source code and create an object code file; ii) to create memory regions as indicated by the object code file; iii) to initialize the ROM memories with the machine instructions indicated by the object code file; iv) if this is the case, to initialize the RAM memories with the data indicated by the object code file; and, lastly, v) to initialize the registers of the simulated processor. If no error occurs, the change to the simulation mode is completed. Otherwise, a dialogue box that informs that an error has occurred is shown, detailed error information is printed on the messages panel, and the change to the simulation mode is aborted.

Once in the simulation mode, every line of the central window shows the information corresponding to a machine instruction (see Figure 5). For each machine instruction, the following is shown (from left to right):

1. The memory address where the machine instruction is stored.
2. The machine instruction expressed in hexadecimal.
3. The machine instruction decoded in assembly language.

4. The original line in the assembly source code which has produced this machine instruction.

Thus, for example, the information shown in the first line of the simulator window in Figure 6 indicates that:

1. The machine instruction is stored in the memory address `0x0000 1000`.
2. The machine instruction expressed in hexadecimal is `0x2000`.
3. The machine instruction decoded in assembly language is "`movs r0, #0`".
4. The instruction has been generated from the second line of the assembly source code, which is: "`main: mov r0, #0 @ Total to 0`".

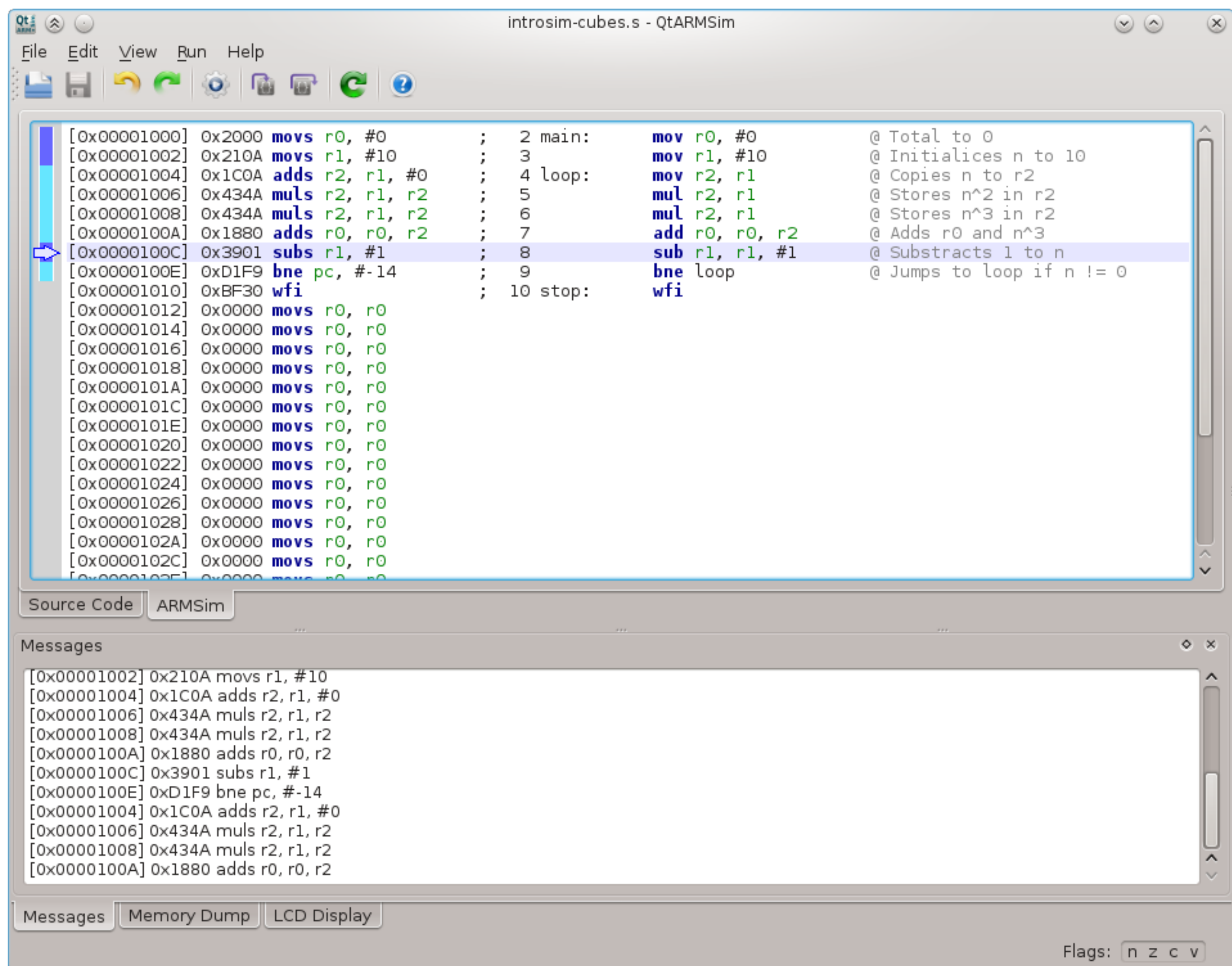

FIGURE 6 QtARMSim in simulation mode, with the registers and memory panels closed

The contents of the `r1` to `r15` registers are shown in the registers panel (see left part of Figure 5). The `r15` register requires special mention since in the ARM architecture it is the *Program Counter* (`PC`). QtARMSim highlights the line that corresponds to the address indicated by the `PC`, in order to help the student to identify which line of code should be executed next.

Furthermore, the memory content of the simulated computer is shown in the memory panel, as can be seen on the right part of Figure 5. In this example, it is apparent that the simulated computer has two memory blocks: a ROM memory block, which begins at address `0x0000 1000`, and a RAM memory block, which begins at address `0x2007 0000`. It can also be seen that some

ROM memory cells have nonzero values, which correspond to the machine code generated from the assembled source code, and that in this particular example all the RAM cells are set to zero. QtARMSim allows the student to manually edit the contents of any RAM cell by double clicking over it and entering a numeric value (in decimal, hexadecimal, or binary), or a string. If the value entered exceeds the word capacity, an error is generated and an explanation is given to the student.

The current version of QtARMSim incorporates two new panels: a memory dump panel and an LCD panel. The memory dump panel shows a more compact representation of the memory (see Figure 7). In this panel, each memory block is displayed in a different tab, and in each tab the contents of the corresponding memory block are represented at the byte level, with 16 bytes per row. In addition to the hexadecimal representation of each byte, an ASCII representation of all the bytes in a row is also shown in the last column. This panel is particularly useful for understanding the Little Endian memory organization, since it makes it possible to compare the memory contents represented at a byte level with the same contents grouped by words in the memory panel. It also helps the students to understand that the same binary value can be used to represent different types of data. Finally, it is useful in showing how to work properly with examples in which vectors, strings or characters are involved. As was the case in the memory panel, the student can manually edit the contents of any RAM byte by entering a numeric value (in decimal, hexadecimal, or binary), or a character. Again, if the value entered exceeds the byte capacity, an error is generated and an explanation is given to the student.

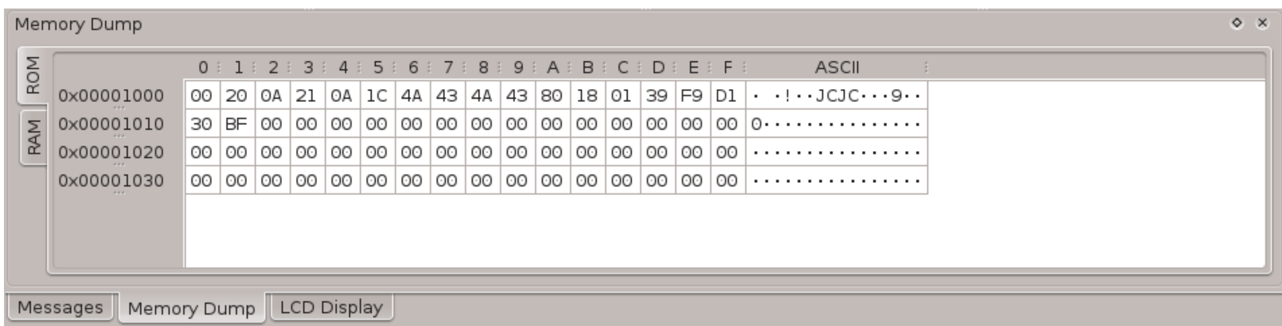

FIGURE 7 QtARMSim memory dump panel

The second panel that has been added to QtARMSim is an LCD (see Figure 8). This panel mimics a 40×6 LCD which is mapped to a RAM memory region. It displays each byte in that region as if it were an ASCII character. The purpose of this LCD is to provide the students with a first approximation to the input/output problem and to enable their programs to display results in a visible way. Moreover, as each LCD character is actually a byte of a matrix that is stored in memory, it also provides them with a training ground to check how the address of a matrix element should be computed.

FIGURE 8 QtARMSim LCD panel

Now that the different graphic elements available on the simulation mode have been described, the different actions that the student can perform will be described in the next subsections: i) to execute the whole program; ii) to execute the program step by step; iii) to edit the contents of registers and memory; and iv) to define execution breakpoints.

4.3.1 | Full program execution

The simplest simulation action is to execute the whole program. Figure 9 displays the QtARMSim window after executing the machine code corresponding to the source file “add.s”, which adds the bytes at addresses `0x2007 0000` and `0x2007 0001`, and writes the result on the byte at `0x2007 0002`. As can be seen in the figure, QtARMSim highlights those registers and memory positions which have been written at any time of the code execution. Thus, as can be seen in Figure 9, after executing the shown machine code, the following elements that have been written by the program are in bold and have a blue background: i) the `r0`, `r1`, and `r15` (`PC`) registers; ii) the word at `0x2007 0000` in the memory panel; and iii) the byte at `0x2007 0002` in the memory dump panel. This highlighting feature has been implemented in order to make it easier for the students to identify those registers and memory positions that have been written during the code execution.

As the current version of QtARMSim (compared to the version described in [13]) provides a new visualization of the memory at the byte level, which is shown on the new memory dump panel, the graphic interface memory model was completely rewritten to work at this lower level. Within the framework of this major modification, the history of preceding memory positions that have been written has also been narrowed down to the byte level. In this way, although the memory panel highlights those words in which any of their bytes have been rewritten, the memory dump panel can be used to see which bytes of those words have actually been rewritten. This is illustrated in Figure 9. As stated before, when the machine code is executed the byte at address `0x2007 0002` is written with the computed result. Therefore, the whole word at `0x2007 0000` is highlighted on the memory panel, whereas only the byte at address `0x2007 0002` is highlighted on the memory dump panel.

4.3.2 | Step-by-step execution

Although the entire execution of a program, described in the previous subsection, can serve to test if the program does what it is supposed to do, it does not allow the student to see in detail how the final state has been reached. It only makes it possible to compare the initial state of the simulated computer with its final state after executing the whole program.

In order to evaluate what happens when executing each machine instruction, the simulator provides an option to execute the machine code step by step (i.e., machine instruction by machine instruction). This option enables the student to become familiar with what each machine instruction actually does. It is also used in practice to examine why a program, or part of a program, is not doing what is expected of it. Moreover, this option makes it possible for the student to see how a modification of either a register or a memory position alters the subsequent execution at any given moment of the execution.

The simulator provides two step-by-step execution modes. The first one, called *step into*, executes only one machine instruction at each step. The second one, called *step over*, takes into account that a program is usually structured using subroutines, and executes one machine instruction at each step, except when a call to a subroutine is found. In this latter case, the simulator executes the whole subroutine as if it was a unique machine instruction. This second option helps the student to compare between the state of the simulated computer before calling a subroutine and after executing it.

A new feature of the current version of QtARMSim is that each time a step-by-step execution is done a colored ribbon is drawn at the left margin of the last machine instruction executed. While the instructions are executed in sequential order, the color of the ribbon is maintained, but when a jump is made, a new color for the ribbon is chosen; this color will remain the same until another jump is made. In this way, the student can visually follow which machine instructions have already been executed, and can visually see the effect of control flow instructions (i.e., conditional and unconditional branches, for and while structures, etc.). This colored ribbon can be seen in Figures 5 and 6. In these two figures, the machine code has been executed until the moment when its loop will end for the second time. Figure 10 illustrates how the margin decoration changed when the machine code shown in Figures 5 and 6 was executed step by step. The left snapshot of Figure 10 shows the left margin at the start of the simulation, when no instruction had yet been executed, and the PC was pointing at the first instruction. The right snapshot corresponds to the margin as depicted in Figures 5 and 6, where twelve instructions have been executed. As can be seen, when the eighth instruction, `bne pc, #-14`, was executed, and because the sequential order execution was broken, the ribbon color changed from dark to light blue. It can also be seen that the new light blue color remained while the next instructions were executed in sequential order.

FIGURE 9 QtARMSim in simulation mode after executing the whole program

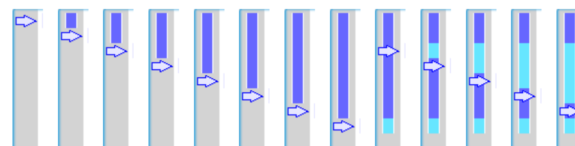

FIGURE 10 Detail of how the left margin decoration changes as the machine code shown in Figures 5 and 6 is executed step by step; whenever a jump is made, the ribbon color changes

4.3.3 | Edit the contents of registers and memory

As mentioned in the previous subsection, one of the uses of the step by step execution is to manually modify the contents of a register or a memory position and to see how this modification alters the subsequent execution.

QtARMSim makes it possible to modify the contents of a register or a memory position by double clicking on the cell whose content is to be modified, and writing the new contents, followed by the `Enter` key. As an example, Figure 11 shows how the contents of the `r1` register will be replaced by the number 3. The new contents can be either a numeric value or a string. If it is a numeric value, it can be entered either in decimal, as is, in hexadecimal, using the "0x" prefix, in octal, using the "0" prefix, or in binary, using the "0b" prefix. On the other hand, if the new contents is a string or a character it should be entered either quoted or double quoted (e.g., 'Hi' or "Hi").

Another new feature of the current version of QtARMSim is that when the entered value cannot be represented in the chosen register or memory position, due to its binary size being greater than that of the chosen holder (either a word or a byte), the user gets a message pointing out this error. This feature can be used to stress that a given value can only be stored if there are enough bits to represent it.

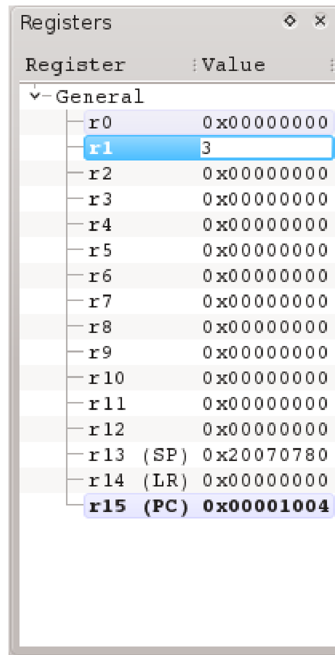

FIGURE 11 Manually editing the contents of the `r1` register

4.3.4 | Breakpoints

The previously described step-by-step execution allows us to look carefully at what is actually happening at one specific point of the code. However, in order to get to an area that we want to inspect in detail, we may have to execute several instructions first. For example, we might be interested in a part of the code that can only been reached after a loop with hundreds of iterations. It would not make sense to go step by step until we manage to get out of the loop and arrive to the point of the code that we really want to see in detail. This is one case in which using a breakpoint to instruct the simulator where it should stop executing instructions is useful. Setting a breakpoint will allow the simulator to execute those parts of the code that we are not interested in seeing in detail and stop the execution at the machine instruction from which the step-by-step execution will be done.

Another situation in which setting breakpoints can also be useful is when one wishes to easily observe the simulator state evolution at different points of a program.

QtARMSim makes it possible to set and unset breakpoints with ease. To set a breakpoint, it is enough to click on the left hand margin of the line at which we want to set the breakpoint (in the simulation window); a red symbol will appear in the margin to indicate that a breakpoint has been set at that line. To illustrate this, Figure 9 shows a breakpoint set at the memory address `0x0000 1004`. To unset a previously created breakpoint, it is only necessary to click on the breakpoint mark that has to be unset.

5 | RESULTS

An ARM simulator that complies with the objectives prescribed in Section 2 has been developed and can be downloaded free of charge. The version of the simulator presented in this paper is an improvement on the version published in Spanish in [13] with the following enhancements: i) a color trace of the instructions that have been simulated; ii) a memory dump at the byte level; iii) a basic LCD device simulation; and iv) a bundled GCC assembler. These new enhancements provide the student with a better

understanding of the instruction execution flow and the memory organization, as well as easier installation and configuration of the simulator.

The pertinent details have also been published (in Spanish) under a *Creative Commons* license as a textbook, which has been used and updated during the 2014 and 2015 academic years. In this textbook, the simulator is described in more detail, and laboratory exercises are provided that make extensive use of the proposed simulator. In particular, the textbook relies on ARMSim and QtARMSim to cover the following topics of the *Fundamentals of Computer Architecture* knowledge unit defined in the Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [8]:

- Organization of the von Neumann machine.
- Instruction formats.
- The fetch/execute cycle; instruction decoding and execution.
- Registers.
- Instruction types and addressing modes.
- Subroutine call and return mechanisms.
- Programming in assembly language.

Since the courses given in 2014 and 2015 were the first in which the simulator was used, students were constantly asked for their opinions on its usability, so we received invaluable feedback on which aspects should be improved, which enabled us to constantly improve the simulator and reach its current state.

In general, the feedback has been very positive; the sharpest criticism was regarding the installation process, mostly due to the fact that the simulator relies on a series of dependencies that can be awkward to install on one operating system in particular. To address this problem, we modified the installation process in two ways. The first step consisted of migrating the graphic interface dependencies from PyQt and QScintilla to PySide and our own code editor implementation. This alleviated the installation procedure because the PySide dependency is easier to install than the PyQt and QScintilla ones. The second step was to bundle the indispensable GNU GCC binary files in our package, so that this dependency does not have to be manually installed by the student, and the GNU GCC path configuration option can be automatically set.

It should be highlighted that the feedback from students who repeated the course, where the first time they used MIPS and the second time they had to change to ARM, has been especially gratifying. Their perception of the usability of the simulator was so good that they commented that the assembly language of ARM was much simpler than MIPS (when, in fact, we do not share this opinion).

In addition to the previous points, in the first year the simulator was used we conducted a survey based on the veracity of the following statements and on two open questions:

1. I have installed QtARMSim in my computer without any problem.
2. The graphic interface of QtARMSim seems adequate.
3. The code edition has been easy for me.
4. The debug options (step-by-step execution, breakpoints, etc.) have been useful for me.
5. The information displayed on the different panels of QtARMSim is adequate.
6. I consider that the simulator has helped me understand better how the ARM processor works.
7. I would recommend that we continue using QtARMSim in Computer Architecture teaching.
8. I consider that this year's course using QtARMSim has been much more interesting than last year's with the xspim simulator.
9. What positive features would you highlight about QtARMSim?
10. What do you think needs to be done to improve QtARMSim?

The first eight questions were Likert items ranging from 1 to 5, where 1 “disagree strongly”, and 5, “agree strongly”. The last two question were open questions. This survey was answered by 28 students (19% of the students enrolled in the course).

The first five Likert items of the survey were about the technical aspects of the simulator (installation, interface, debugging and provided information). In those questions, a mean score of 4 was obtained for statements 2 - 8, and a mean score of 3 for statement 1.

The following three Likert items obtained a score of 5: *“I consider that the simulator has helped me understand better how the ARM processor works”*, *“I would recommend that we continue using QtARMSim in Computer Architecture teaching”*, and *“I consider that this year’s course using QtARMSim has been much more interesting than last year’s with the xspim simulator”*. Note that these statements related to the teaching objective of the simulator and to student satisfaction.

The answers given to questions 9 and 10, (*“What positive features would you highlight about QtARMSim?”*, and *“What do you think needs to be done to improve QtARMSim?”*) stressed, on one hand, the ease of use of the simulator and its efficacy in the learning process. On the other hand, they also pointed out that the installation process needed to be simplified. Some of the responses to these two questions are reproduced in Tables 1 and 2, respectively. It should be noted, as previously stated, that the survey was conducted during the first year that the simulator was in use. In particular, the suggestions for making the installation easier and for a more automatic configuration have already been addressed, as mentioned above.

TABLE 1 Some of the answers to the question “What positive features would you highlight about QtARMSim?” obtained after the first year the simulator was used

Simple and easy to understand.
Easy to use. Detailed information.
It is well designed in all its aspects.
The use of the step-by-step procedure and the clear position of the flags.
Ease of the interface and its usage. Very useful for studying.
Being able to execute ARM assembly code and to see the contents of the registers and the memory.
It provides a very visual way to see the how the memory and the registers are modified while each instruction is being executed.
It is clear what kind of information, data or a memory address, is stored in each register. It is also very graphic and not confusing, like the previous `xspim` simulator, and it is a lot easier to understand what is going on.
The simulator graphically represents very well the execution of ARM programs, where the data is stored, from where it is loaded, what is done with it, etc. Once you understand the theory, the simulator is very visual and helps to eventually understand the course related concepts.

TABLE 2 Some of the answers to the question “What do you think needs to be done to improve QtARMSim?” obtained after the first year in which the simulator was used

Simplify the installation on Microsoft Windows.
The installation and the configuration should be more automatic.
Editor needs a little improvement, but, primarily, the installation process should be improved.
It was quite difficult at the beginning to install it, not only on Microsoft Windows, but on GNU/Linux. While some installation guides were later published, it should nevertheless be made easier.
A brief explanation below each instruction on step-by-step execution could benefit from some animation. In fact, in the next Computer Architecture course, they use a program that provides an animation of how the instructions are executed on a MIPS processor.

Finally, the laboratory teachers’ assessment of the simulator has been very positive and their impression has been that the students were more aware of what was happening as they were doing the simulator exercises, compared to in previous courses.

6 | CONCLUSIONS AND FUTURE WORK

In this article we have set out the latest versions of ARMSim and QtARMSim, which provide a simulated environment for the teaching of the ARM Thumb architecture, which is multiplatform, open, free, and easy to use. These applications have been developed with the aim of providing, in a simple way, comprehension of how a processor works, and it has been widely agreed that both students and teachers feel that this objective has been met.

In future work, we would like to provide new functions in the graphic interface in the form of an animation of what happens when each machine instruction is executed, a simulated console, and a contextual help function on machine instructions and data. In the simulation engine, apart from providing the functionality that the graphic interface requires to accomplish some of the aforementioned desired new functions, we would like to add full support of ARM Thumb 2, expand the memory model to support simulated input/output devices, and add a cache simulator module.

References

1. Clements A. The undergraduate curriculum in computer architecture. *IEEE Micro* 2000; 20(3): 13–22.
2. Clements A. Selecting a processor for teaching computer architecture. *Microprocessors and Microsystems* 1999; 23(5): 281–290.
3. Clements A. ARMs for the poor: Selecting a processor for teaching computer architecture. In: IEEE; 2010: T3E 1–6.
4. Harris S, Harris D. *Digital Design and Computer Architecture: ARM Edition*. Morgan Kaufmann . 2015.
5. Barrachina Mir S, Fabregat Llueca G, Martí Avilés JV. Utilizando Arduino DUE en la docencia de la entrada/salida. In: Canaleta X, Climent A, Vicent L., eds. *XXI Jornadas sobre la Enseñanza Universitaria de la Informática*; 2015: 58–65.
6. Barrachina Mir S, Castillo Catalán M, Claver Iborra JM, Fernández Fernández JC. *Prácticas de introducción a la arquitectura de computadores con el simulador SPIM*. Pearson Educación . 2013.
7. Nikolic B, Radivojevic Z, Djordjevic J, Milutinovic V. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *Education, IEEE Transactions on* 2009; 52(4): 449–458.
8. IEEE/ACM Joint Task Force on Computing Curricula. Computer Engineering. . Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. tech. rep., IEEE Computer Society Press and ACM Press; 2004.
9. Clements A. *Computer Organization and Architecture. Themes and Variations*. Cengage Learning . 2014.
10. Hohl W, Hinds C. *ARM Assembly Language: Fundamentals and Techniques*. CRC Press. 2 ed. 2014.
11. Horspool RN, Lyons D, Serra M. Armsim#—a customizable simulator for exploring the arm architecture.. In: ; 2009: 223–228.
12. Barrachina Mir S, Fabregat Llueca G, Fernández Fernández JC, León Navarro G. ARMSim y QtARMSim: simulador de ARM para docencia. In: Canaleta X, Climent A, Vicent L., eds. *XXI Jornadas de Enseñanza Universitaria de la Informática*; 2015: 2–9.
13. Barrachina Mir S, Fabregat Llueca G, Fernández Fernández JC, León Navarro G. Utilizando ARMSim y QtARMSim para la docencia de Arquitectura de Computadores. *ReVisión* 2015; 8(3): 17–30.
14. Ardito C, De Marsico M, Lanzilotti R, et al. Usability of e-learning tools. In: ACM. ; 2004: 80–84.
15. ARM Limited . *ARMv7-M Architecture Reference Manual*. 2010.

AUTHOR BIOGRAPHY

Sergio Barrachina Mir is an associate professor of the Computer Architecture and Technology Area at the Jaume I University (Spain). His research interests include high performance computing and architectures. He received his PhD in Computer Engineering from the Jaume I University.

Germán Fabregat Llueca is an associate professor of the Computer Architecture and Technology Area at the Jaume I University (Spain). His research interests include fault-tolerant computing, embedded systems and networks of sensors. He received his PhD in Physics from the University of Valencia.

Juan Carlos Fernández Fernández is an associate professor of the Computer Architecture and Technology Area at the at Jaume I University (Spain). His research interests include cloud systems and power-aware computing. He received his PhD in Computer Science from the Polytechnic University of Valencia.

Germán León Navarro is an associate professor of the Computer Architecture and Technology Area at the at Jaume I University (Spain). His research interests include embedded systems and FPGAs. He received his PhD from the Jaume I University.

How to cite this article: Barrachina S., G. Fabregat, J.C. Fernández, and G. León (2020), QtARMSim: an ARM simulator for teaching Computer Architecture, *Journal of Software: Practice and Experience*, *2020;xxxx.*