Nintendo DS: A Pedagogical Approach to Teach Computer Architecture

Maria J. Santofimia[1], **and Francisco Moya**[1]
[1]Computer Architecture and Networks Group. School of Computer Science,
University of Castilla-La Mancha, Ciudad Real,Spain

Abstract— *This work reveals the benefits obtained from an innovative pedagogical experience based on the use of a game console as the platform to teach Computer Architecture. This paper shows how the selection of an appropriate platform, not only motivates students, but also helps them to acquire the theoretical concepts by means of real applications. The Nintendo DS console turns out to be a great platform to explore the structure and organization of a fully equipped computer. In this regard, a brief description of the lab sessions is presented to highlight the great potentialities of this platform.*

Keywords: Computer Architecture education, Nintendo DS.

1. Introduction

Nowadays, the use of electronic devices is commonly extended to almost all facets of human life. The increasing capabilities along with their decreasing prices, provide an appealing field to be exploited for educational purposes.

Educational games or applications for computers and consoles have succeeded in improving knowledge acquisition and retention as well as motivation in short-age students. Video-game companies have targeted the educational market by means of devices that provide learning and teaching facilities to both teachers and students. Although universities are not unaware of the potentialities of such facilities, the high level knowledge taught or the self-motivation supposed to students do not provide the same appealing context for educational games, as the primary or secondary education. The pedagogical value of these new platforms and devices is granted by the running applications and not so much by the platform on which they run.

The majority of the first year students in Computer Science suddenly find themselves dealing with new concepts that require some level of abstraction. Courses such as Computer Architecture, taught on the second semester of the first year is aimed at providing students with the foundations to understand the internal organization of a computer, analyze the different functional units, as well as comprehend the role they play in the execution of instructions.

Hands-on experience turns out to be essential in showing how theory is applied. In this regard, the use of simple computer architecture emulators is the most extensive approach for this purpose. However, this approach fails on concreting theoretical concepts into real ones.

On the one hand, it is desired to count on an architecture simple enough to allow students to focus their attention on the course objectives, and not on understanding the complex aspects of a particular architecture. However, if the architecture is too simple, it might truncate student progress and motivation due to the limited possibilities to experiment with it. Therefore, the architecture used needs to be complex enough to allow students to experiment with all the theoretical knowledge acquired during lectures, but at the same time, simple enough to keep the focus on the structure and organization and not in complex aspects.

A few years ago, the School of Computer Science, at the University of Castilla-La Mancha, undertook the innovative teaching experience of using the PIC16F84 from Microchip[1] microcontroller for the lab sessions of the Computer Architecture course. Some of the most appealing features supporting this innovative experience were its 8 bit architecture, a simple RISC instruction set, Flash, EEPROM and RAM memory, the variety of I/O peripherals, and its low cost. However, when adopting the use of the PIC16F84 microcontroller, it soon becomes apparent that this approach fails to address most of student expectations. Advanced students found the platform insufficient to explore the advanced concepts learnt in lectures. On the contrary, beginner students tend to be overwhelmed by the requirements imposed by the platform, missing the point of the lab sessions. The use of a microcontrollers, such as the PIC16F84, might be suitable to address those aspects related to the embedded systems design, but it is not an optimal choice for the Computer Architecture course.

This paper presents the shortages identified in the teaching of the Computer Architecture course, as well as the proposed solution to help students to acquire the concepts and overcome the identified deficiencies.

The remainder of this paper is structured as follows. First section presents the reasons why the PIC16F84 approach failed in achieving the course objectives. The following section exposes the foundations of the Nintendo DS (NDS from now on) based proposal. Finally, the last section summarizes the conclusions obtained from both approaches.

2. Hands-on experience with the PIC16F84

Computer Architecture is considered as one of those grounding courses, for the Computer Science degree, where foundations for upper courses must be acquired. Therefore, a correct understanding of the contents reveals essential for a thriving degree achievement and the training future professionals.

In this regard, the first step to justify the use of the NDS console in the labs is to analyze the main shortages identified not only on first year students, but also in the following years.

One of the main objectives of providing hands-on experience as a complementary part of teaching is showing students real applications of the explained concepts, as well as reinforcing the comprehension of the same concepts. However, at some point, student tend to miss the connection that links the theory and the practise, what makes them fail on mapping the acquired concepts to real computers.

Something similar occurs with the concept of memory and how it is organized. The majority of the students do not find difficulties to understand the contents explained in lectures. Nevertheless, there are many evidences of a weak understanding. For instance, it is easy to find last year students struggling with debugging tasks, mainly due to the poor comprehension of concepts such as the stack frame, the stack pointer or the frame pointer. Probably, the meaning of these concepts was clear in the specific context where they were taught, but in a different context, they seem meaningless for many students.

Given these observations, it seems that the main problem lays on a deficient transfer of theory contents to practice. In other words, it is expected that by using the right platform, students will be able to apply the acquired knowledge to real world computers leading to a better understanding of the concepts presented in the lectures.

The underneath subsection presents the approach followed in the School of Computer Science at the University of Castilla-La Mancha, in finding the right platform to overcome the aforementioned shortages. This experience lead to the adoption of the current platform, the NDS, as an optimal solution to provide hands-on experience with real architectures.

2.1 The PIC16F84 approach, strenghts and weaknesses

One of the main arguments supporting the use of simulation tools for hands-on experience finds its roots at the little resource requirements, since simply a computer suffices to run the application. Moreover, using software instead of hardware brings to students the opportunity to practise at home. Some reference books[2][3] for Computer Architecture and architecture provide their own virtual architecture for pedagogical purposes, along with the simulation tool. This approach normally consists of a simple architecture that focuses on the elements composing a computer and their functionality rather than on how to generate specific applications.

Despite the strengths of an approach based on simulation tools, the experimental results reveal that the experiences carried out with simulation tools seem futile for approaching the Computer Architecture foundations to the real computers. The vision gathered by the students when working with a software tool is not much different than the one perceived from the lectures. In this sense, many students still remain unaware of the multiple applications of their acquired concepts, not only for debugging purposes, as mentioned before, but for writing efficient code that optimizes the architecture where the application will be running on, for accomplishing an effective memory management, or even for making decisions about what architecture is the more suitable for a specific needs.

Drawing these points together, the need for a real platform to provide hands-on experience is more than evident. However, the wide range of available architectures, with very different features and targeted to a very heterogeneous audience, makes hard to decide what processor is the more convenient for pedagogical purposes.

Microcontrollers turns out to be a proper solution since they are complete computers but at the same time of a limited complexity. Among the wide spectrum of microcontrollers, the PIC16F84 has great acceptance, mainly due to its low cost in comparison to its potential. It is a small integrated circuit, of just eighteen, powerful enough to run microcontroller-like applications.

Providing hands-on experience with a microcontroller such as the PIC16F84 provides students with the opportunity to get familiar with a complete ISA by learning its assembler language. The arising question is to what extent does this knowledge contribute on the education of Computer Science students? However, in spite of the strengths that supported this choice, the obtained results bring into light their multiple weaknesses.

The stack concept and the role it plays is emphasized during the lectures, although hard to explain with the PIC16F84, since it simply counts on a hardware stack and a limited RAM memory, made of registers, that do not support a feasible way for implementing it. Furthermore, the RAM memory is divided in two banks of 128 bytes, although just the 80 first of each bank are physically implemented. A direct consequence of such memory organization is that students do not have the opportunity to meet the differences between register and the external RAM.

The indirect addressing is also hard to explain on the PIC16F84 since it is achieved by means of the FSR register, that instead of helping students on the understanding of the addressing mode, make it more complex. However, the direct

addressing does not seem to be easier. Memory is organized as pages of just 1K of addressing memory that involve a page swapping whenever further positions need to be acceded.

However, this is not to say that the use of the PIC16F84 brings no benefits. On the contrary, the acquired contents were improved in comparison to the use of simulation tools, although there was still some place for improvements, what motivated the searching for a new platform capable of overcoming the handicaps identified in the PIC16F84 approach.

3. The Nintendo DS approach

Apart from being a very successful game console, the Nintendo DS provides an excellent platform for hands-on experience, as well as being a great element to motivate the student enthusiasm, and attract their interest and attention. The access to a well documented architecture as well as the active homebrew forums, provide the perfect grounding for helping students on applying learnt contents.

The following subsections go through the motivations and the technical details that turns a game console into a very attractive and efficient learning platform.

3.1 The Nintendo DS hardware, its greatest asset

On the contrary to the expected complexity, the programming interface with the console is simple enough to prevent first year students from finding themselves overloaded by aspects that are extrinsic to the course. In this regard, the labs are guided to explore those aspects of the console capable of helping them to understand the contents already presented.

The NDS contains two processors integrated in one, an ARM946E-S and ARM7TDMI, 4MB of main RAM, and a long list of peripheral devices such as a pair of 2D and one 3D graphic engines, touch screen, speakers and microphone among some of the most relevant.

From a pedagogical perspective, the use of the two ARM processors provides the foundation for introducing certain concepts such as the Application Binary Interface (ABI) and the role it plays, how it is connected with the ISA, as well as its influence in the compiler, making the most of the opportunity to point out some aspects of the compilation stages, and giving special emphasis to the cross-compiling character of the process. The use of CPU registers is also well addressed by these processors. Furthermore, by means of the debugging tool it is possible to present to the students the tasks performed by registers in a program execution.

The ARM memory resources lay the foundations for accomplishing an exploration of several aspects presented in lectures. It serves to present to students different sort of memory technologies. Furthermore, students are prompted to work with these memories, emphasizing all along the process, the different memory technologies that are being used and their particularities.

The two screens and the printing process poses students with an appealing context for exploring the interruption concept, understanding the importance role they play, and how to manage them.

Yet another issue is the use of graphic modes due to the many theory concepts that are involved in its use. A comprehensive view of the platform is required when dealing with tiled and frame-buffer graphics, where not only memories but also processors, and interruptions are involved.

To summarize, it can be concluded that the combination of the aforementioned elements makes feasible to get students involved in the application of theory contents to something as real and exciting as developing an applications for a game console. The hands-on experience is enhanced by the added value of working with motivated students.

3.2 The programming language

Traditionally, the students of Computer Architecture are prompted to get familiar with the most common assembler instructions as well as to understand the meaning of the instructions given by the ISA. However, the overload imposed by the programming in assembler often distracts students from accomplishing the learning outcomes of this course.

Nevertheless, this brings the opportunity to introduce students into the use of the C programming language, from a low level perspective. Although programming theory is out of the scope of the Computer Architecture course, students simply need basic notions of those language aspects that help them on exploring the architecture, with a low algorithmic complexity. Moreover, C is the most employed language for embedded applications. Special attention is paid to the concept of pointers, since these are to be the tools to manage and explore memory.

Some debuggers, such as *kdbg*[4], list the assembler code instructions generated for each of the C statements in the source file. In this regard, students can explore the assembler code generated for a procedure call, loops, or assignments, for instance, and get a clear notion of how assembler code is related to the C code.

4. The session planning

The following subsections describe the lab session planning according to the targeted outcomes. These sessions propose real applications of the contents covered in the lectures. As described underneath, each of the proposed sessions intends to provide students with the appropriate context to test their understanding of the basic concepts. One of the key aspects to keep student attention is the fact that each session introduce them to new aspects of the NDS programming. In this sense, from the student perspective, they are learning the foundations of the homebrew for the NDS, while at the same time, from the teaching perspective, each session is intentionally designed to explore theoretical contents.

4.1 Introduction to the required tools

The proposed lab sessions involve, at different levels, the use of a small set of tools, that help students to focus their attention on new concepts rather than in small technicalities that remain out of the scope of the course.

During the first session, students are introduced to the different tools composing the *devkitPro*[8] toolchain, that supports the code generation for the NDS. Among some of the toolchain resources, there is a set of templates that prevent students from having to start coding from scratch. These templates include a Makefile that automates the generation of the executable files. Therefore, students are just prompted to get familiar with the content of the Makefile, and how to use the *make* tool.

Debugging tools are also introduced in this first session. In its simplest case, by means of the executable files generated from the example templates and the debugger, students are able to explore some basic aspects of the architecture. The debugging tool that comes along with the devkitPro is the *arm-eabi-gdb*. This is a command-line tool, that can be used on its own or endowed with one of the available front-ends, such as *kdbg*[4], *ddd*[5], and *emacs*[6].

Among these front-ends, kdbg stands out for bearing a twofold aim: from the one side, students can explore memory and CPU register contents during the program execution, while at the same time it is possible to explore the associated assembler code for each of the C statements.

Finally, emulator tools are of a great help for testing and debugging purposes, such as the case of *DeSmuME* that support both task, in contrast to others that do not provide support for debugging.

The use of real NDS consoles is highly encouraged, as a mean to make students aware of the real applications of the contents exposed in lectures. However, the emulator tools provides a decent substitute to undergo basic experiences, in a straightforward manner. Nevertheless, complex capabilities are not yet well addressed by emulator tools, for instance the wireless support.

4.2 Application Binary Interface

Advocating for real experiences following theoretical sessions find its roots in the benefits reported to the student comprehension of concepts such as the Application Binary Interface (ABI).

The ABI can be considered as a collection of standard documents intended to specify the interface among binary or object files. Hence, one of the hands-on experience sessions shows students the important role played by the ABI in specifying many aspects, that involve different issues such as the data size and alignment, procedure calls, or argument passing, just to name a few.

During this session, students are challenged to validate that the code generated by the ARM compiler, effectively meets the restrictions imposed by the ABI. In this sense, students write some programs that help them on exploring those features specified by the ABI, and by means of the debugging tool, explore aspects such as the generated assembler code, he values stored in registers and memory, and their evolution.

The procedure call is easily explained by means of a sample code, run step by step, by using a debugger. In addition to the rules stated by the ABI specifying how to perform a procedure call, it is also detailed the policy of argument passing, as well as the retrieval of return values. By means of code, it is showed to students how, as far as possible, arguments are stored in registers. Memory accesses are minimized, which in turn improves performance.

4.3 Framebuffer and tiled graphics

From a pedagogical point of view, one of the greatest assets of the NDS approach lays on the explicit memory management expected from students when displaying graphics on the console screens. Deep understanding of the memory organization is revealed essential, since it involves storage and retrieval of the data to be displayed, directly from the video memory.

Among the different ways of displaying graphics on the NDS screens, these lab sessions simply focus on the framebuffer and the tiled graphics. The understanding of these two methods involve knowing two different forms of dealing with the memory for displaying purposes.

The framebuffer mode map screens to a specific region of memory. Hence, modifications performed over this memory region has an effect over the screen appearance. On the other side, tiled graphics, count on a matrix of tiles, where each tile represent a bitmap of 8x8 pixels size. The use of this graphic mode is interesting because tiled backgrounds are composed of a set of tiles and a tile map. Both parts of the background are stored in the video memory, in blocks known as *map base* and *tile base*, where each tile base share the memory region with eight map bases. This is a great challenge for students that need an overall view of the video memory for displaying graphics.

4.4 Timers and scrolls

Timers and scrolls are two features of the NDS platform that provide an appropriate context for introducing students to the interrupt concept. Both, timers and scrolls, are employing interrupts, managed by the processor.

This lab session pursues the student familiarization with the basis of interrupt handling. To this end, the use of timers and scrolls presents two effective ways to get students engage in handling interrupts. Therefore, implementation of counters can be accomplished by using timers. Moreover, scrolling graphics along the screens require a full understanding of how interrupts work.

Screens are redrawn in a line by line sequence. The period of time spent in between drawing the end of one line and

the beginning of the next one is known as horizontal blank. Furthermore, the vertical blank is the period of time that takes place between drawing the bottom line of the screen and the first one at the top. These two periods of time, vertical and horizontal blank, represent the right time to modify the data being display, since in any other case, displayed data might be a mix of old and new data.

These periods of time that can be used to redraw the screen can configured to generate interrupts, informing of their occurrence. Therefore, effective handling of these interrupts reveals as the proper way to display new data in the screens.

Graphic scrolling depend on the handling of such interrupts in order to generate the effect of a graphic moving along the screen. Students are asked to show a moving background by means of interrupts.

5. Conclusions

Lab sessions plays an essential role in the education of Computer Science students, mainly due to its eminently practical character. However, it is neither a straightforward nor simple the task of selecting the platform for supporting a hands-on experience.

This article compile the most relevant conclusions obtained from the two different experiences carried out for the Computer Architecture course.

Several handicaps arose when the PIC16F84 was selected as the preferred platform for the labs, give rise to a need for a new architecture capable of filling all the identified gaps. In this endeavors an innovative idea came around, why not use a well known platform, such as the Nintendo DS console, as the tool for the labs?

The idea was proposed to students as a pedagogical experiment, and they were given the chance to decide the platform they preferred. Over thirty percent of the students chose the NDS platform, what provided an acceptable feedback for comparing results regarding the acquired knowledge. Currently undertaking the second year of the experience, it cannot be said that adopting NDS represents a silver bullet for supporting students on its education. However, the gathered results prove an important improvement in terms of the contents understood and retained by students.

References

[1] Microchip Technology Inc. *Microchip home page* [online]. http://www.microchip.com. Retrieved on February 25th, 2009.

[2] David A. Patterson, John L. Hennessy, Peter J. Ashenden, James R. Larus, *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2004, ISBN 1558606041.

[3] Javier Garcia, Jose M. Angulo, and Ignacio Angulo, *Fundamentos Y Estructura De Los Computadores*. Paraninfo, 2003, ISBN 8497321804

[4] Johannes Sixt. *KDBG. A Graphical Debugger Interface* [online]. http://www.kdbg.org/. Retrieved on February 25th, 2009.

[5] Free Software Foundation, Inc., *DDD. Data Display Debugger* [online]. http://www.gnu.org/software/ddd. Retrieved on February 25th, 2009.

[6] Free Software Foundation, Inc., *GNU. Emacs* [online]. http://www.gnu.org/software/emacs/. Retrieved on February 25th, 2009.

[7] Free Software Foundation, Inc., *GDB: The GNU Project Debugger* [online]. http://www.gnu.org/software/gdb/. Retrieved on February 25th, 2009.

[8] DevkitPro, *DevkitPro home page* [online]. http://www.devkitpro.org/. Retrieved on February 25th, 2009.