Undergraduate Assembly Language Instruction Sweetened with the Raspberry Pi

Jalal Kawash
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
+1 403 220 6619
jkawash@ucalgary.ca

Andrew Kuipers
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
+1 403 220 6015
amkuiper@ucalgary.ca

Leonard Manzara
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
+1 403 220 3518
manzara@cpsc.ucalgary.ca

Robert Collier
School of Computer Science
Carleton University
Ottawa, Ontario, Canada
+1 613 520 4333
robert.collier@scs.carleton.ca

ABSTRACT
It is widely recognized that motivating students in an undergraduate assembly language course is a tremendous challenge, principally because of the perception that the subject matter is both difficult and tedious. The Raspberry Pi is a small and inexpensive single-board computer that was created for educational purposes, and in this paper we describe how we successfully incorporated this device into the curriculum of an undergraduate assembly language course. We describe, in detail, the objectives for this course and the dedicated lab that uses the Raspberry Pi as an embedded device, and then evaluate the effectiveness of our approach. Our findings (obtained by exploring changes in student performance and examining the results of an engagement/enjoyment survey) strongly indicate that the introduction of the Raspberry Pi was well received by the students and contributed positively to their learning outcomes.

Keywords
Computer Science Education, Raspberry Pi, Assembly Language Programming, Hardware/Software Interface

1. INTRODUCTION
In recent educational practice, computer science students are typically introduced to programming by learning a high-level computer language. Although the importance of being able to program in a high-level language is undeniable, it would be difficult to argue that learning assembly language is not crucial for understanding the underlying computer architecture. Unfortunately, although assembly language remains a fundamental topic in computer science, novice assembly language programmers often find the process of programming in it tiresome and frustrating. This is not surprising, since programming in assembly is typically a tedious process, and is often described as "dry" [1], "difficult to learn", and "confusing" [2]. Naturally, this impression can be exaggerated if the course is concerned with difficult advanced assembly language programming techniques, despite the fact that these techniques are considered essential in many computer science, information technology, and engineering programs. As a result, post-secondary educators are strongly motivated to make the process of learning assembly language more engaging and rewarding for students.

The Raspberry Pi, released in 2012 as an educational tool for computer science, is an affordable, relatively small device that contains an ARM11 [6] processor and supports a wide range of peripherals. Its inclusion of a General Purpose Input/Output (GPIO) interface [7] makes it especially suitable for teaching a wide variety of I/O protocols. Our decision to use the Raspberry Pi in an assembly language programming course was motivated by the desire to make the process of learning assembly language more enjoyable for the students, while offering them exposure to a contemporary architecture (similar to what they may encounter in today's ubiquitous mobile devices). It is also our belief that both seeing and closely working with an actual physical device (as opposed to an invisible and remote server) will help students break down the conceptual barrier between software and hardware. Hence, the main research question this paper intends to answer is whether the adoption of the Raspberry Pi can improve both the learning experience and outcomes of students.

The Raspberry Pi is usually equipped with a Linux operating system, and educational institutions that employ the Pi typically use it as a stand-alone computer (running Linux). Our approach, in contrast, is to use the Pi as an embedded device. Programming for direct hardware control is not possible when there is an overprotective operating system guarding hardware resources, so we use a minimal OS (employing only a first-stage bootloader) on the Pi, which allows students to program the hardware directly.

Although the Raspberry Pi has been adopted as an educational tool by many colleges (and, in [8], as a motivator for high school students), the novelty of the device entails that there are only a few studies that analyse these experiences in depth. For example, Brock et al. discuss how the Pi is being used across a range of secondary-school computer courses [5], while Foltz discusses how it is used to teach network administration at the undergraduate level [9]. Although Hooper discusses (at an introductory level) how to program the Raspberry Pi at a low level using C++ and ARM assembly in a workshop [10], to the best of our knowledge our use of the Raspberry Pi to teach low-level programming in an undergraduate computer science course is unique.

There are alternatives to the Raspberry Pi for teaching the hardware/software interface (for example, embedded devices such as the Arduino or FPGAs have been used for this purpose), but these devices often do not feature a comparable set of built-in peripheral interfaces. Also, it is possible to use an architecture that supports an operating system with a "real-mode" (allowing direct access to hardware) and the course described in this paper evolved

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA
© 2016 ACM. ISBN 978-1-4503-3685-7/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2839509.2844552

from a course that used Windows 98 running in real-mode on the x86 architecture [3]. Nevertheless, the results of our investigation indicate that the Raspberry Pi is a viable alternative platform that improves both student engagement and performance.

The remainder of this paper is organized into seven sections. Section 2 and Section 3 are used to describe the course and the typical hardware/software setup of a workstation in our lab. The course objectives are discussed in Section 4 and Section 5 gives a brief description of the format and topics of the previous version of the course (for comparison). Section 6 gives a comparative (and quantitative) assessment of student performance in each version of the course, and detailed student feedback on the use of the Raspberry Pi is discussed and analyzed in Section 7, with our final discussions presented in Section 8 to conclude the paper.

2. THE COURSE

The subject of this paper is the *Computing Machinery II* (CPSC 359) course at the University of Calgary. It is a required course for students pursuing an undergraduate degree in computer science at the University of Calgary, and is a prerequisite course for the third-year courses *Computer Structures*, *Computer Networks*, and *Principles of Operating Systems*. As suggested by the course title, this is the second of a pair of early (second-year) undergraduate courses, both of which are mandatory for a Bachelor's degree in Computer Science. In the prerequisite course, students are exposed to assembly language programming and general computer architecture and organization concepts, while in CPSC 359 the emphasis is on the hardware/software interface. Students enrolled in this course are exposed to digital logic design, microarchitecture design, interrupts and interrupt handling, and hardware I/O. It should be noted that, out of all these topics, this paper is concerned primarily with hardware I/O.

The enrollment in the course for the Fall 2013, Winter 2014, and Winter 2015 semesters were 61, 112, and 120 students, respectively, with students divided into groups of at most 22 students per lab tutorial. A typical week consisted of 150 minutes of lecture time complemented by 100 minutes of lab time. The lab sessions, designed to parallel the lectures by providing a hands-on experience with the Raspberry Pi in a specialized lab (Section 3), were conducted by graduate teaching assistants.

The main student deliverable in the Raspberry Pi component of the course is an interactive video game developed entirely in ARM assembly language, using a Super Nintendo™ Entertainment System (SNES) controller [12] as an input device, and an HDMI monitor for output. The game is required to run on "bare metal" (on the ARM processor without an operating system), in a manner that is typical for many embedded systems. Although the game itself would vary between semesters, a representative example would be a game that required the player to traverse a maze. In addition to the actual graphical representation of the maze itself, the students would also need to draw a title screen and menu elements. The player would only be permitted a finite number of actions and would need to collect items in order to proceed, all of which comprises game data that the program must track and report to the user. The game would receive input from the SNES controller (opening doors and moving a character), and present the player with success and failure messages as required. It should also be noted that students were evaluated on the quality of their code, and that the students were given the option to work in groups of up to three members.

3. LAB SETUP

Writing bare-metal assembly code for the Raspberry Pi introduces a set of challenges not encountered when developing programs directly on a Pi equipped with a full-featured operating system such as Linux. The principal difficulty is that the Raspberry Pi will be running a minimal operating system that does not provide a compiler and system-level I/O. Consequently, compilation and debugging cannot take place on the device itself, but instead must be performed on a host computer. Basic user input also poses a problem. While the Raspberry Pi is equipped with a USB interface, it is unreasonable to expect second-year undergraduate students to write a USB keyboard or mouse driver in assembly, and providing a library or pre-made driver to handle user input does not help them understand how input is handled at the lowest software level. To solve these challenges, a dedicated lab was created with 30 student workstations, each of which includes a host computer and a suite of supporting hardware that interfaces with the Raspberry Pi. The details of these workstations, from both the hardware and software perspectives, are described below.

3.1 Hardware Setup

In order to facilitate students writing bare-metal assembly code for the Raspberry Pi, the workstations need to provide support for cross compiling a program targeted to the ARM architecture, transferring compiled programs to the Raspberry Pi, debugging programs running on the Raspberry Pi, and facilitating user I/O. The additional hardware connected to the Raspberry Pi (not including the host computer) is shown in Figure 1, and includes:

a) Raspberry Pi
b) Custom Breakout Board
c) Segger™ JLink EDU Debugger [11]
d) Super Nintendo™ Entertainment System (SNES) Controller
e) HDMI Interface
f) USB to TTL Serial Interface

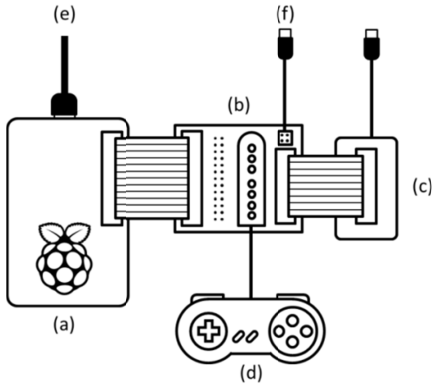

Figure 1. Workstation Hardware Suite.

Much of the hardware used in the workstations needs to communicate over the Raspberry Pi's General Purpose Input / Output (GPIO) interface, and so a custom breakout board (b) was created to connect the Raspberry Pi's 26-pin GPIO header to the various devices used in the workstation. Programs compiled on the host computer are transferred to the Raspberry Pi and remotely debugged using the JLink EDU debugging unit (c), which connects to the host computer's USB interface and to the Raspberry Pi across the GPIO interface. The USB to TTL serial interface (f) also connects the host computer to the Raspberry Pi across the GPIO interface, and provides a means of performing text-based serial input and output. The SNES controller (d) is also connected to the GPIO interface, and provides an additional

method of user input. Finally, the Raspberry Pi's native HDMI interface (e) is connected to a monitor, which provides a means of graphical output from the Raspberry Pi.

3.2 Software Setup

Writing bare-metal assembly for the Raspberry Pi involves a different software setup than would be used to write and run programs directly on the device itself. Given that the Raspberry Pi will not be using a full-featured operating system (that is, instead of Linux we use a simple operating system that does not provide facilities such as text editors, compilers, or debuggers), students must edit and compile their programs on a different computer system, herein referred to as the *host*. The compiler on the host must be able to cross-compile source code to produce executable binaries that run on the Raspberry Pi. Debugging provides an additional challenge, since the host debugger is interacting with a program running on a remote system.

The software setup used on the workstations involves a distribution of the Linux operating system. A version of the GNU Compiler Collection (gcc), built to target the Raspberry Pi's ARM11 architecture, was installed on the workstations, providing a cross-platform compiler and debugger. Additionally, the GNU Debugger (gdb) Server provided by Segger™ (JLinkGDBServer) was installed on the host computers to allow communications between the gdb debugging client on the host and the JLink EDU remote debugging unit connected to the Raspberry Pi.

In order to establish a connection between the JLink EDU remote debugging unit and the Raspberry Pi, the GPIO lines used for JTAG communication must be set to the correct mode using a sequence of software commands. To accomplish this, a "minimal" kernel image with just enough code to enable the JTAG GPIO lines is compiled by the students and copied to the SD card from the host computer, and the Raspberry Pi is powered on with this SD card inserted. One difficulty here is that, at the time that this paper was authored, the details of the first-stage bootloading process were not currently publicly available, so students were not able to recreate this stage of the bootloading process themselves. To solve this problem, an SD card is first written with the original Raspbian distribution (available on the Raspberry Pi website, and which includes the first-stage bootloader), and then modified by overwriting the kernel image file used in the second-stage of the bootloading process with the "minimal" kernel image.

Once the Raspberry Pi is powered on with an SD card containing the "minimal" kernel image, the JLinkGDBServer can be started on the host computer. After a successful connection has been made and the debugger server is running, the students can start the ARM-targeted gdb client using a kernel image they have compiled with the ARM-targeted compiler, and establish a remote debugging connection to the JLinkGDBServer instance. At this point, the student executes a command in the gdb client to upload the kernel image to the Raspberry Pi's memory, sets the Raspberry Pi's program counter register to the beginning of the kernel image, and continues execution on the device. If successful, the student's new kernel image will now be running on the device.

3.3 Additional Considerations

While the overall experience of working with the Raspberry Pi was generally advantageous to students, it should be noted that the specialized lab we created came with a price tag of more than $30,000 (CAD) for about 34 workstations (including 4 workstations for instructional staff). Although this cost might be considered prohibitive for some institutions, it is worth noting that the lab was constructed with completely new equipment, including new top-of-the-line host machines and dual monitors (one for the host and a HDMI monitor for the Raspberry Pi). Consequently, it should be noted that this price tag could be reduced substantially by simply equipping an existing lab of host machines; the add-on price for converting an existing lab to make it Raspberry Pi-ready can be very minimal, and would include only the price for HDMI monitors (if necessary), JLink devices, and a few inexpensive accessories. Furthermore, there are cheaper display options than the HDMI monitors used in our lab that can be used with the Raspberry Pi's RCA video output.

The cost of the Raspberry Pi itself and its accessories is very low; the price for an assembled kit (which includes a power supply adapter, SIM card, cables, and protective case) was about $85 (CAD) in Winter 2015. Note that all students were required to purchase the Pi and accessories when registering for the course.

It should also be noted that, with the enrollment in the winter semester exceeding 100 students, the students were struggling to find lab time and space during peak times close to assignment due dates. Not being able to work from home (due to the specialized hardware required) was a clear disadvantage to students, but duplicating a lab workstation at home incurs a relatively low cost, and can be encouraged in future offerings. Students would need only an extra HDMI screen, a JLink EDU device, and our locally-manufactured breakout board.

4. COURSE OBJECTIVES

As this paper describes the use of the Raspberry Pi for assembly language instruction, it is worth noting that several Raspberry Pi topics are considered course objectives, including (but not limited to) the following:

Advanced ARM Assembly Programming: Students are exposed to stacks, the concept of subroutines (as implemented in assembly language), two-dimensional arrays, and interrupt service routines.

General-Purpose Input/Output: GPIO is a well-known mechanism used to provide flexible I/O for microcontrollers. GPIO allows the students to connect a wide range of I/O devices to the Raspberry Pi. However, a substantial amount of programming is required to write device interface functions (i.e., GPIO is not plug-and-play). Students are expected to become proficient in the use of GPIO registers, including those that set the function for a GPIO line (e.g., input, output, or a special function) and those that read or set the values of the GPIO lines [7].

Universal Asynchronous Receiver/Transmitter Protocol: UART is a low speed serial protocol used ubiquitously for simple communication between devices. The Raspberry Pi provides support for UART (and a reduced implementation called mini-UART) through its GPIO interface. The protocol is relatively easy to present to students, and students are required to implement a driver for a generic UART device from scratch.

Video Programming: The Raspberry Pi provides support for video programming via a frame-buffer architecture accessed through memory-mapped I/O (that is, the frame-buffer is accessed through memory load and store instructions). On the Raspberry Pi, the buffer is initialized through a mailbox interface, and is used to establish communication between the ARM core and the video core (or GPU). A suite of registers in addition to RAM is used for this communication, and the use of these registers, along with the protocol for initializing the frame-buffer [7,13], are both areas in which the students must become proficient.

Interrupts and Exceptions: The ARM architecture has a number of different execution modes, and software or hardware

exceptions (i.e., interrupts) change the execution mode of the processor. Writing and registering exception handlers requires the students to understand the vector table architecture, and how to use the Current Program Status Register (CPSR) and the Saved Program Status Register (SPSR) on the ARM CPU [6]. Students learn to program an interrupt service routine to handle a hardware event such as a rising or falling edge on a GPIO line.

5. PREVIOUS VERSION

It is worth noting that the previous version of this course covered similar topics using a different platform. The lab for the predecessor course was based on an Intel x86 platform running Windows 98. This setup allowed students to work in real address mode, which enabled direct access to hardware. In order to develop an interactive video game (which, in the current iteration of this course, remains a significant course deliverable), the students had to write specialized device drivers without relying on privileged operating system calls. Unfortunately, the more recent versions of the Windows operating system no longer provide this option, and other operating systems (e.g., Linux) are too overprotective and do not allow developers to write programs that access the hardware directly.

The nature of the video game deliverable did not change significantly when this course was adapted to the Raspberry Pi. In the predecessor course, students needed to use a low-level video library (the VESA BIOS Extension (VBE) [15]), necessitating that they work with both port-mapped and memory-mapped I/O. It was also necessary for students to work with XGA monitors (with a resolution of 1024 × 768 with 256 colors), and develop specialized PS/2 keyboard and/or mouse drivers (requiring that they program and register their own interrupt handlers).

As noted previously, versions of Windows subsequent to Windows 98 do not permit students to work in real address mode and, consequently, the previous version of the course also required a specialized lab (i.e., x86 machines with Windows 98). Although this was suitable to fullfil the objectives of the course, there were two significant issues with this lab setup. First, in spite of a substantial explanation that course concepts were universal and still applicable (regardless of the computing environment), many students reported that they were not convinced that the course was relevant to current computing practices. Feedback received from the students indicated that a significant number of them were discouraged when working with a computing environment that they literally characterized as "ancient".

The second issue that arose was related to the sustainability of a lab that consisted of x86 machines running Windows 98. Unsurprisingly, it was becoming progressively more difficult to repair and find spare parts for any failing machines. To address this, we resorted to DOSBox [17] in subsequent offerings of the course [3], but we were ultimately dissatisfied with the quality of experience the students were receiving while using an emulator rather than working directly with the actual hardware.

6. COMPARATIVE ASSESSMENT

According to the undergraduate grading system at the University of Calgary, letter grades between F and A+ are mapped to grade point values between 0.0 and 4.0, inclusively (with A's and A+'s worth 4.0, A- worth 3.7, B+ worth 3.3, etc.) Consequently, if the letter grades assigned to each student for their respective performance on the exam that followed the x86 / ARM course module are mapped to grade point values according to this scheme, it is possible to calculate mean grade point values for different categories of students. Since the Raspberry Pi was first adopted for this course in the Fall semester of 2013, the body of students that have completed this course in the last three years can be divided into two categories: those that completed the *Computing Machinery II* while the x86 architecture was in use, and those that completed the course since the Raspberry Pi was adopted. We should note that in both course versions, the ARM and x86 material was tested in an exam separate from other course topics. The marks presented here are for this exam. It should be highlighted that current and previous versions were taught by the same instructor. In both formats, no textbooks were required, but we recommended in each case a book that covered the basic parts of speech of each assembly language (x86 and ARM). Hence, the platform and the lab were the only major changes.

In the available data, there were 138 students in the former category (i.e., those that took the course with the x86 between September 2012 and April 2013) and 319 students in the latter category (i.e., those that took the course with the Raspberry Pi, after September 2013). After mapping the letter grades to numeric values, we observe a mean grade point value of 2.542 (0.846 standard deviation) for the students in the former category, and a mean grade point value of 2.986 (0.730 standard deviation) for the students in the latter. These results correspond to a mean letter grade of between C+ and B- for the students that completed the course with the x86, and a mean letter grade of B for those that completed the course with the Raspberry Pi. The distribution of the marks for each population of students is plotted in Figure 2.

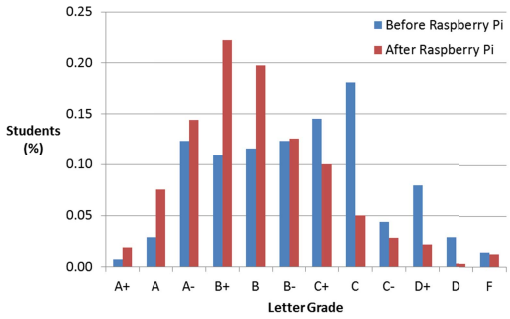

Figure 2. Distribution of the ARM/x86 Exam Grades.

It was necessary to test the normality of these data sets before testing whether the difference between the mean grades was statistically significant. As evidenced by the quantile-quantile plots in Figure 3, the distribution of grades across the cohort of students that completed the course with the x86 was not quite normal (n.b., an R^2 of 0.9085 for the quantile-quantile plot trend line) and, thus, required an unpaired, nonparametric student t-test to demonstrate statistical significance.

This test showed that the difference was extremely statistically significant (p-value < 0.0001), and so it is not unreasonable to interpret it as evidence that the use of the Raspberry Pi has had a very positive impact on student performance. This result is consonant with the feedback provided by the students, as discussed in the following section.

7. SURVEY RESULTS

Students registered under the new format of the course were asked to participate in a voluntary, anonymous survey that assessed their enjoyment, learning experience, motivation, and independent learning. We have received a total of 198 valid instruments from students in two consecutive semesters, from a total enrollment of 337 students (i.e., an overall response rate of 58%).

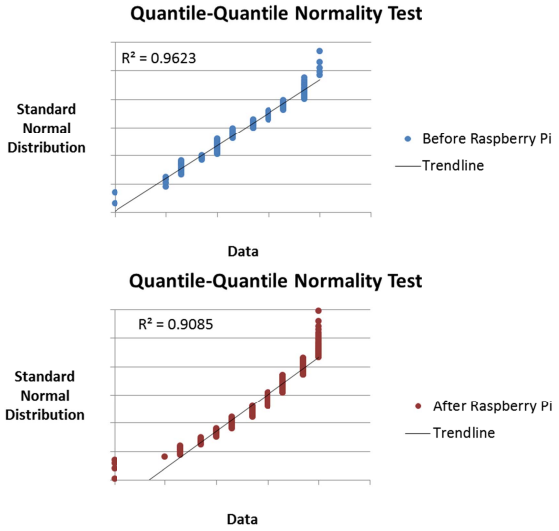

Figure 3. Normality Testing with Quantile-Quantile Plots.

7.1 Quantitative Summary

Tables 1 and 2 show the basic frequency statistics for the survey. To clarify, these tables show the fraction of survey participant responses associated with each of the five different values from the Likert scale used on the instrument (i.e., Strongly Agree, Agree, Neutral, Disagree, and Strongly Disagree).

Table 1 shows the proportion of the survey participants that responded with each of the different responses to items about working with the Raspberry Pi. It is clear from these results that the students felt very positively about the impact of the Raspberry Pi, with a substantial 82.32% majority reporting agreement that their learning was enhanced. The other responses expressed similarly positive reactions concerning enjoyment and motivation.

	Item (a) I enjoyed working on the Raspberry Pi.		Item (b) The use of the RPi enhanced my learning.		Item (c) The use of the RPi motivated me...	
Strongly Agree	22.73%	**66.16%**	29.29%	**82.32%**	23.74%	**67.17%**
Agree	43.43%		53.03%		43.43%	
Neutral	20.71%		15.66%		24.75%	
Disagree	7.58%	**13.13%**	1.01%	**2.02%**	5.05%	**8.08%**
Strongly Disagree	5.56%		1.01%		3.03%	

Table 1. Survey Responses Concerning the Raspberry Pi.

Table 2 presents similar data for items about the feeling students reported concerning pursuing future studies. The majority of the participants responded with agreement for each of the items (i.e., low-level programming, game programming, and independent studies). In all cases, the majority of the survey participants have reacted positively to the use of the Raspberry Pi.

To facilitate further analyses, study participants were grouped into two classes: the "Enjoyers", who agreed or strongly agreed to item (a), and the "Non-Enjoyers", who were neutral, disagreed, or strongly disagreed with the same item. By translating the Likert scale responses on the other items into numerical values (between 1 and 5 inclusive, with 1 and 5 assigned to responses of "Strongly Agree" and "Strongly Disagree", respectively), the results could be subjected to additional statistical analyses.

	Item (d) I would like to learn more about low-level programming.		Item (e) I would like to learn more about game programming.		Item (f) I would like to pursue independent learning with similar topics.	
Strongly Agree	16.67%	**52.53%**	38.89%	**71.21%**	24.75%	**60.61%**
Agree	35.86%		32.32%		35.86%	
Neutral	18.69%		14.65%		19.70%	
Disagree	18.69%	**28.79%**	7.07%	**14.14%**	10.61%	**19.70%**
Strongly Disagree	10.10%		7.07%		9.09%	

Table 2. Survey Responses Concerning the Future Studies.

When comparing the responses of the Enjoyers with the Non-Enjoyers, item (b) was used to assess whether the participants felt that the use of the Raspberry Pi enhanced their individual learning experiences. The mean translated response from the Enjoyers (4.29) corresponds to a response between agreement and strong agreement, while the value from the Non-Enjoyers (3.69) corresponds to a response between neutrality and agreement. Although the mean responses would characterize both groups as feeling "positively" about their learning with the Raspberry Pi, the difference between the mean translated responses were extremely statistically significant (p-value < 0.0001).

Similarly, in item (c) (assessing whether the students felt that the Raspberry Pi contributed positively towards their individual motivation), the mean translated responses were 4.08 and 3.24, for the Enjoyers and the Non-Enjoyers respectively. This indicates, once again, that the mean response from the Enjoyers corresponds to agreement / strong agreement, in contrast to the mean response from the Non-Enjoyers, corresponding to a response between neutrality and agreement. The difference between the mean translated responses was, once again, extremely statistically significant (p-value < 0.0001) and for both this item and the previously explored item (b), the survey results indicate that those who enjoyed using the Raspberry Pi thought that the experience contributed to both enhanced learning and improved motivation.

7.2 Qualitative Summary

The survey participants were also given the opportunity to explain their choices in a free-form format, and these were subjected to a qualitative analysis. This analysis, restricted to those instruments on which the students reported "Neutral", "Disagree", or "Strongly Disagree" with item (a) (i.e., "I enjoyed working on the Raspberry Pi") indicated the presence of two underlying themes.

The first theme saw the majority of these students express dissatisfaction about the "fragility" and "awkwardness" of the working environment. One student (with a neutral rating) stated this clearly: "*The fact that we're working on relevant hardware is nice; however, the experience was terribly tarnished by system instability, over-complication, and J-Link being absolutely awful.*" This student goes on to say: "*The relevance of the Raspberry Pi to modern day technology is interesting.*" Another student who strongly disagreed with the question under discussion stated: *"ARM is a current technology and the Raspberry Pi is super novel, but painful due to hic-ups in the system.* S/he further explains: "*The method for communication/work with/on the Raspberry Pi was over complicated by J-Link."*

The second theme was centered on the workload of the course itself and the scarcity of lab space and time, or the inability to work from home. A representative student states: "*Although I*

enjoyed the idea of the RPi, the projects consumed a lot of my time, more than any other course by far. […] It was interesting to work below the OS […], but the projects required a lot of troubleshooting and we could only work on the RPi on campus."

8. CONCLUSION

Teaching assembly language programming at the undergraduate level is often challenging, and this can often be attributed to the perceptions (of the students) that the topic is dull and tedious. Earlier attempts to offset this bias (in the predecessor *Computing Machinery II* course discussed herein) included the incorporation of video game development to introduce a "fun factor" into the student experience. While we have retained this element in our present course design, we looked at incorporating a contemporary platform that students would find more relevant to the current state of the computer industry. The use of a credit card-sized computer with an ARM processor served this purpose, largely because of prevalence of mobile devices in the students' daily lives. In this paper, we show that a significant majority of the students were more motivated to learn about the hardware/software interface and assembly language due to the use of the Raspberry Pi as a learning platform. Another majority also indicated that the Raspberry Pi improved their learning experience and enjoyment of the topic.

These results, compiled from the "self-assessments" provided by the students on the survey instruments, closely align with our own quantitative analysis of student performance. We found that there was a significant increase in mean grade point value for the exam associated with this module of the Computing Machinery II course when the course switched to the Raspberry Pi from the x86 platform. Furthermore, a reduced standard deviation (in the context of a greater mean grade value and in conjunction with a visual inspection of the distributions from Figure 2) indicates that fewer students are being "left behind" in this challenging subject.

On the task of improving the experience further, it should be noted that the Raspberry Pi seemed to convince only about 30% of the surveyed students to pursue future independent learning about low-level programming. Although this is certainly a respectable result, the task of increasing this value further remains daunting. This result raises the following question: If a student enjoys programming with assembly on the Raspberry Pi and indicates that this was a memorable learning experience, why would they remain unconvinced to learn more about the subject independently, and what can be done to address this?

Finally, while the experience was generally advantageous to a large majority of students, qualitative analysis also indicated two areas for improvement: the "unfriendly" development environment, and the accessibility to the lab. There is not much that can be done to address the perception of the development environment as "unfriendly", since cross-compilation and the use of the JLink debugging device is, at the moment, a necessary complication. Improving lab access is also difficult, since increasing the size of our lab facilities requires finding new space when space is scarce, and expending more money. One possible solution is to provide help and resources for students who wish to duplicate the development environment on a home computer. The additional cost to students is relatively minor (and, of course, entirely optional if they would rather work on campus).

There will always be trade-offs when choosing particular platforms to achieve general pedagogic aims. As we have shown, the use of a specific platform to teach general concepts by exposing them to real-world experiences can be very beneficial to students, but, special care must be taken to ensure that "platform-specific details do not swamp pedagogic objectives." [14]

9. ACKNOWLEDGMENTS

Our thanks go to the students of CPSC 359 who took part in this study. We also thank the anonymous reviewers.

10. REFERENCES

[1] C. Zilles. 2005. SPIMbot: an engaging, problem-based approach to teaching assembly language programming. In *Proc. of the 2005 workshop on Computer architecture education: in conj. with the 32nd Int'l Symp. on Computer Architecture* (WCAE '05). ACM, New York, NY, USA.

[2] D. Crookes. 1983. Teaching Assembly-Language Programming: A High-Level Approach. *Software Microsystems*, vol. 2, no. 2, pp. 40-43.

[3] J. Kawash and R. Collier. 2012. Using Video Game Development to Engage Students of Assembly Language Programming. In *Proc. of the 14th annual ACM SIGITE conf. on Information technology education* (SIGITE '13). ACM, New York, NY, USA, 71-76.

[4] www.raspberrypi.org

[5] J. D. Brock, R. F. Bruce, and M. E. Cameron. 2013. Changing the world with a Raspberry Pi. *J. Comput. Sci. Coll.* 29, 2 (December 2013), 151-153.

[6] ARM. 2010-2011. *ARM Compiler Toolchain, Using the Assembler* (pages 3 to 21). http://infocenter.arm.com/help/topic/com.arm.doc.dui0473c/DUI0473C_using_the_arm_assembler.pdf

[7] Broadcom Corporation.2012. *BCM 2835 ARM Peripherals*.

[8] J. Black, J. Brodie, P. Curzon, C. Myketiak, P. W. McOwan, and L. R. Meagher. 2013. Making computing interesting to school students: teachers' perspectives. In *Proc. of the 18th ACM conf. on Innovation and technology in computer science education* (ITiCSE '13). ACM, New York, NY, USA, 255-260.

[9] C. Bryan Foltz. 2014. Network administration with the Raspberry Pi. *J. Comput. Sci. Coll.* 29, 5 (May 2014), 66-67.

[10] W. H. Hooper. 2013. Easy as Raspberry Pi: an inexpensive platform for machine language instruction. *J. Comput. Sci. Coll.* 29, 1 (October 2013), 102-103.

[11] www.segger.com/jlink-debug-probes.html

[12] www.gamefaqs.com/snes/916396-snes/faqs/5395

[13] Broadcom Corporation. *VideoCore® IV 3D Architecture Reference Guide*.2013.

[14] Joint Task Force on Computing Curricula, ACM and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA.

[15] VESA. Retrieved 2013/05/20 from: www.vesa.org

[16] DOS Protected Mode Interface (DPMI) Specification: Version 0.9 Printed 1990/07/26. Retrieved 2013/05/20 from: http://homer.rice.edu/~sandmann/cwsdpmi/dpmispec.txt

[17] DOSBox, an x86 emulator with DOS. Retrieved 2013/05/20 from www.dosbox.com