USING THE GAME BOY ADVANCE TO TEACH COMPUTER SYSTEMS AND ARCHITECTURE*

Ian Finlayson
Assistant Professor of Computer Science
University of Mary Washington
Fredericksburg, Virginia

ABSTRACT

This paper presents an approach to teaching computer systems and architecture using Nintendo's Game Boy Advance handheld game console. In this approach, students learn to write programs in C and assembly for this system. The system is also used to illustrate concepts such as memory systems, memory-mapped I/O, direct memory access and bitwise operations, all of which are needed to effectively program the console. Intended benefits of this approach are to motivate interest by using a platform that many students know and own, and also to get students "closer to the metal" by writing code for a device where you must interact with the hardware more directly than most other systems.

1 - INTRODUCTION

The Game Boy Advance (GBA) is a handheld video game console created by Nintendo, and first released in 2001 [3]. The system was a commercial success selling over 33 million consoles in the Unites States [1]. Developing programs for the console requires writing for the hardware in a way that is unusual on most modern computer systems. The GBA does not have an operating system, so programs interact with the hardware directly, and have full control over system memory. This paper presents a computer systems and architecture course based around the Game Boy Advance. In this course, students learn to write programs for the GBA in C and assembly and learn how the hardware of the GBA works, including typical topics such as memories and processors, but also including special-purpose graphics hardware. There have been

similar approaches using embedded systems in teaching computer architecture such as the Arduino[5] and PIC [7] systems.

The rest of this paper is organized as follows: In Section 2, we discuss the hardware features of the GBA. In Section 3, we provide a brief overview of the programming mode of the GBA which makes it interesting for teaching computer systems. In Section 4, we discuss the design of a course incorporating GBA programming. In Section 5, we provide an evaluation of this course including student survey feedback. Finally, in Section 6 we draw conclusions.

2 - GAME BOY ADVANCE HARDWARE

The Game Boy Advance features a 3-stage ARM processor running at 16.78 MHz [3]. Despite the fact that the system is somewhat old, the fact that it runs the ARM instruction set makes learning about it relevant for students. The ARM instruction set is used in nearly all tablets, cell phones, and in embedded computers such as the Raspberry Pi. Along with x86, it is one of the two major instruction sets in wide use [6]. The fact that the processor is so slow by today's standards means that one always has to be cognizant of performance to program it effectively.

The GBA has 384 kilobytes of memory, not counting the memory of the game cartridges [3]. 288 kilobytes of this memory is general-purpose and is used for global variables and the stack. There is an additional 96 kilobytes of memory dedicated to video RAM (VRAM). The usage of VRAM depends on which graphics mode the system is in (as described in the next section). In the simplest form, VRAM is simply an array of pixel data which is mapped onto the screen. There is also 1 kilobyte of memory-mapped control registers, such as for controlling the graphics system, setting scroll amounts, reading button states, and so on.

The memory system supports direct memory access (DMA) to copy from one section of memory to another. Because the GBA has no operating system, the programmer is able to manually turn on DMA, which allows us to quickly copy data from one section of memory to another - such as copying color data into VRAM. The GBA provides a more hands on platform for talking about DMA than most systems, where the operating system does not let you trigger DMA directly.

The GBA has a 240x160 pixel screen which supports 15-bit color (32,768 different colors). There are 5 bits each for the red, green, and blue color components. The console also has 10 hardware buttons.

3 - PROGRAMMING THE GAME BOY ADVANCE

This section provides a brief overview of how one writes programs for the Game Boy Advance. The goal of this section is, of course, not to provide a comprehensive tutorial on programming for the console, but merely to give an overview of the programming model that the system presents.

Programs for the GBA are written in C, or ARM assembly (or some combination of both). There is a port of the GCC compiler tool chain available under the name “DevKit

Advance" [2]. While the resulting programs can be run directly on GBA hardware (using a special link cable or special-purpose cartridge), it is easier to run them under an emulator. Such emulators are freely available for Linux, Windows, OSX as well as Android, iOS and Chrome.

As stated above, the GBA does not have an operating system. It also does not have any sort of API or library of functions built in for controlling graphics, sound and input. Instead, we interact with the hardware directly through memory-mapped registers. For instance, the GBA has a "button state register" which is updated by the hardware to reflect the status of the GBA's 10 buttons [8]. The register is 16 bits large, and is accessed through memory location 0x04000130. The lower ten bits of this value reflect whether each button is pressed (0) or unpressed (1). So to check if the first of these (the 'A' button) is pressed, we could use code like the following:

```
volatile unsigned short* button_state = 0x04000130;
if ((*button_state & 0x1) == 0) {
    // Button A is pressed!
}
```

There are also memory-mapped registers for controlling various aspects of the hardware. For instance, the "display control register" stores various settings related to the graphics including which mode the hardware is in (described below), which of the various background layers are enabled, whether sprites should be drawn, and so on. Because there are multiple settings stored in this one register, we must again use the bitwise operators to combine them. Students programming the GBA must quickly become comfortable doing bitwise operations on integer values.

The way the graphics on the GBA work depend on which of the six graphics mode the system is in [8]. Modes 0-2 are tile-based modes, and modes 3-5 are bitmap modes. With the bitmap modes, VRAM is simply a 2D array of pixel data. The programmer puts graphics on the screen by writing colors into this array. In modes 4 and 5, there are actually two such arrays, which allows for smoother graphics with double-buffering, though each of these modes has a downside: mode 4 requires that colors are stored in a palette, and mode 5 has a reduced screen resolution.

The bitmap modes are very simple, and very flexible, but the downside is that all graphics must be done in software. We place pixels on the screen manually, by storing them into VRAM. The issue with this is performance. If we want to update the whole screen, such as when scrolling it to the left or right, that will mean writing each of the 240x160 = 38,400 pixels. Because the clock speed of the CPU is just 16.78 MHz, we only have 437 clock cycles to spend on each pixel, each second. If we want our program to run at 60 frames per second (the native refresh rate of the GBA screen), then that's only 7 clock cycles per pixel, which is not nearly enough to do anything useful.

For this reason, most games on the GBA actually use one of the tile modes [8]. With the tile modes, the programmer specifies a number of tiled backgrounds including what images to use for each 8x8 tile, and how they should be displayed on the screen. The GBA has special hardware for pushing the pixel data from the tiles onto the screen. The difference between the three tile modes just lies in the number, size and capabilities of the tiled backgrounds. To scroll the tiled background either vertically or horizontally, we just write the number of pixels to scroll into a memory-mapped hardware register.

This really drives home to students the concept of doing something “in hardware” vs. doing it “in software”. Scrolling the screen in software (in a bitmap mode) is impossibly slow, but doing it in hardware (in a tile mode) is basically instantaneous.

The GBA also has a hardware sprite system [8]. A sprite is a 2D object (such as a character, enemy, or projectile) which can be moved about the screen. To use hardware sprites, the programmer specifies the positions, and pixel data for any of 128 available sprites. They can be moved, or flipped vertically or horizontally, by modifying their attributes in a special section of memory for storing sprites. This sprite attribute memory section (starting at address 0x7000000) stores this data for all of the 128 sprites.

Because of the lack of an operating system, there are no system calls available, and also no C library functions that rely on system calls, such as printf, scanf, time, etc. Apart from the special areas of memory we interact with, and the lack of operating system support, programming the GBA is similar to writing any other C or assembly program.

4 - COURSE DESIGN

The learning objectives for our Computer Systems and Architecture course include programming in the C programming language and some assembly language, as well as computer organization, and digital design topics. This section describes how the Game Boy Advance was incorporated into this course so as to fulfill these learning objectives.

Likely the biggest course objective that using the GBA addresses is C programming. Students had two sizable programming projects using the C programming language. The nature of the GBA also really emphasizes programming with pointer variables. In a typical computer system, pointers are somewhat abstract, because we never really know what the value of a pointer will be until we run the program. Furthermore, the pointer values we get at runtime are in fact virtual memory addresses, not real ones. These layers of abstraction, though necessary under a modern operating system, can get in the way of understanding. With the GBA, we know exactly which address certain values are at. For instance, VRAM always starts at the real address 0x06000000.

The GBA as a platform also facilitates teaching assembly programming. It uses an ARM processor which is one of the two major instruction sets in use. This means that there are a plethora of learning materials for ARM assembly programming, including a new ARM version of Patterson and Hennessy's classic Computer Organization and Design text[4]. ARM is a relatively simple RISC style architecture. It is not as simple as architectures like MIPS, DLX or LC3, but unlike those architectures, ARM is widely used, and students can run their programs on actual hardware.

As mentioned above, core concepts like memory-mapped I/O, DMA, and having hardware support for certain operations all have hands-on examples when working with a computer without an OS, such as the GBA. The ARM CPU of the GBA also utilizes pipelining with a simple 3-stage pipeline that is easy relatively to understand. It does not utilize more advanced techniques such as superscalar or out of order execution.

The first full programming assignment given in this course was to implement the game “Pong” using the simple graphics mode where the screen is simply an array of pixel data. This assignment utilizes pointers, array manipulation, and bitwise operators. In

order to have their Pong games run at an acceptable speed, students also had to avoid redrawing the entire screen. They did this by keeping track of which areas of the screen (those near the ball and paddles) had changed since the last frame - this is known as the "dirty rectangles" optimization.

For the second full programming assignment, students were able to write any sort of game they wanted using a tiled background and sprites. Amongst the games students chose were partial clones of games such as Super Mario World, Zelda, Flappy Bird, Battleship and Space Invaders. The level of time students put into this assignment was somewhat uneven, with some spending far more time than anticipated, and making quite sophisticated games, while others did the minimum to fulfill the requirements. This assignment also required students to write some of their game in ARM assembly (two functions of at least eight lines each).

5 - EVALUATION

In order to attempt to evaluate how this approach to the course went with students, they were given a short, ungraded, anonymous survey. 34 of the 54 students in the course responded (62.96% response rate).

The first question asked students if they had ever owned a GBA system. 91% of respondents answered yes to this question. Furthermore, 55% of respondents reported that they still own a GBA. The system was and is still very popular, and writing games for a console that they had used was a good motivator for many students.

Next, students were asked whether they preferred writing programs for the GBA more than a typical computer system, as they have done in other classes. This question used a Likert scale where a 5 is "strongly preferred the GBA" and a 1 is "strongly preferred a typical computer system". The average response to this question was 3.42 with a standard deviation of 1.23. Thus, there was a slight preference to programming for the GBA, which we found encouraging given that the platform is objectively more difficult to program for.

The next question asked students whether they liked learning about the graphics system of the GBA and felt it added to the course. This also used a Likert scale where 5 is "strongly agree that it added to the course" and 1 "strongly disagree that it added to the course". Here, the average was 4.36 with a standard deviation of 0.65.

One concern we had was whether there would be a difference in preferences of the GBA based on gender. Broadening participation in computer science courses is an important goal, and the concern was that perhaps the game system might appeal more to males than to females. In order to test this, we also asked respondents for their gender and used the results to run a t-test on the results of the previous two questions. We found that there was no statistically significant difference between the male and female preference for programming the GBA compared to typical computer systems, $t(28) = 0.06$, $p = 0.48$. We did the same thing for the question about enjoying learning about how the graphics systems and also found there was no statistically significant difference between the preferences of males and females, $t(28) = 0.47$, $p = 0.32$.

We also included a free-form response question on the survey. Many students expressed enthusiasm for working with the system, and also expressed that the assignments were challenging. The biggest complaint students had was that they did not have enough time for the assignments which was partially due to a large number of snow days.

6 - CONCLUSIONS

This paper presented an approach to teaching computer systems and architecture using Nintendo's Game Boy Advance handheld game console. Our goals for doing this included a desire to appeal to the students who had grown up using these systems, and also to more effectively present course material including C and assembly programming, pointers, memory-mapped I/O, DMA, and general computer architecture and design concepts.

Overall, it went well enough for us to consider using this approach again. Students did enjoy learning about the system, and also mostly enjoyed writing programs for it, despite the added difficulty of doing so relative to most computer systems.

Downsides of this approach are that it is rather difficult to write programs for the device, so C and assembly programming ends up being rather a large portion of the course. In our curriculum, this is intended, but there may not be enough room in other courses to incorporate all of the C programming concepts needed. There are also not many resources for writing GBA programs. Most of the materials are online tutorials, the best of which is [8]. On the flip-side one of our survey respondents pointed out that "there's not much material for it on the internet so it forces people to actually do it instead of just copying from stack overflow" which is perhaps an unintended side benefit.

REFERENCES

[1] Behrens, M., Nintendo Sales Through End of November Revealed, http://www.n-sider.com/contentview.php?contentid=2984, December 1, 2006. Retrieved April 29, 2016.

[2] DevKit Advance, http://devkitadv.sourceforge.net/, June 23, 2003. Retrieved April 29, 2016.

[3] Granett, D., Game Boy Advance: It's Finally Unveiled, http://www.ign.com/articles/2000/08/24/game-boy-advance-its-finally-unveiled, August 23, 2000. Retrieved April 29, 2016.

[4] Patterson, D., Hennessy, J., Computer Organization and Design: *The Hardware Software Interface: ARM Edition*, Burlington, MA: Morgan Kauffman, 2016.

[5] Sprague, Nathan., *Arduino as a Platform for a Computer Organization Course*. Journal of Computing Sciences in Colleges 28.3 (2013): 53-60.

[6] Turley, J., Intel vs. ARM: Two Titans' Tangled Fate, http://www.infoworld.com/article/2610369/processors/intel-vs--arm--two-titans--tangled-fate.html, February 27, 2014, Retrieved April 29, 2016.

[7] Valentine, D., *Using PIC Processors in Computer Organization*, Journal of Computing Sciences in Colleges, 24, (1), 116-122, October 2008.

[8] Vijn, J., Tonc v1.4.2 : Table of Contents, http://www.coranac.com/tonc/text/, March 24, 2013. Retrieved April 29, 2016.