Using the Game Boy to Teach Digital Systems and Computer Architecture

Uso de la Game Boy para enseñar técnicas digitales y arquitectura de la computadora

Guillaume Hoffmann[0009−0001−8196−8819]

CONICET - Universidad Nacional de Córdoba, Argentina
Guangdong Technion-Israel Institute of Technology, China
guillaume.hoffmann@conicet.gov.ar

Abstract. This article presents the use of the Nintendo Game Boy, originally released in 1989, as a pedagogical tool for teaching computer systems and architecture in an undergraduate course given in the 2020s. The Game Boy is a well-documented platform, and proves useful for teaching fundamental concepts such as CPU architecture and assembly programming. Unlike many programming platforms described in classic computer architecture textbooks, the Game Boy has built-in support for graphics rendering, which allows for producing interactive software written in assembly in a reasonable amount of time. The course design includes a programming project in which students develop turn-based puzzles and simple real-time games. Using the vibrant modern Game Boy ecosystem, the course aims to provide students with a practical understanding of digital systems and computer architecture.

Keywords: Digital Systems, Computer Architecture, Assembly Programming, Retro Gaming

1 Introduction

1.1 Course and Program

The course Digital Systems and Computer Architecture, or just Digital Systems from now on, is typically part of undergraduate programs such as electrical and computer engineering, telecommunications engineering, and computer science. Digital Systems usually takes place in the second year of the program and has prerequisites of courses of introduction to programming and computer science.

The Digital Systems course of this article is a second-year undergraduate course that is part of a recently created bachelor's degree in Mathematics and Applied Mathematics in the People's Republic of China. The course is the third computer science course of the program, coming after two computer science courses that involve programming in high-level languages such as C and Java.

This course has been taught at least once a year since 2021. In 2021, it linearly followed the syllabus and textbook. After this first experience, the course was reorganized in order to include an assembly programming lab and a final project, after which the final exam takes place. Since 2022, the course has been taught 5 times in this format.

The weekly load of the course is 4 hours of lecture and 2 hours of tutorial. Since 2022, 2 hours of lecture have been specially dedicated to the assembly programming lab, which works as a parallel track with the main syllabus, except for a few crossovers at the end of the semester. Most importantly, near the end of the course and before covering micro-architecture, the architectures and instruction encoding of MIPS32 (a RISC approach) and Game Boy CPU (a CISC approach) are compared.

The rest of the course mostly follows the textbook and syllabus in a no-fluff approach. The result is a digital systems course with a project in assembly programming that aims at providing hands-on experience of software development. Thus, it contains many connections with the other computer science courses of the program.

1.2 Choosing the Game Boy

The Nintendo Game Boy, originally released in 1989, has sold about 120 millions units worldwide. It is thus one of the most successful gaming platforms ever. As of 2025, the company still sells emulated games of the original Game Boy on newer platforms such as the Switch.

Currently, the Game Boy has a vibrant homebrew community that still releases new games and maintains programming tools such as emulators, assemblers and assets editors. Also, source code of disassembled commercial games is available online, such as Tetris or Pokemon Red and Blue, which can thus be studied. It also still attracts attention on YouTube channels dedicated to retro game mechanics.

Optionally, it is possible to get a real Game Boy hardware and run self-created programs on it by use of a microSD adapter cartridge. Original Game Boy and Game Boy Advance are available for sale in the second hand market, but emulators are the way to go in this course as emulation is virtually perfect on any modern computer.

The Game Boy CPU architecture is a classic CISC close to the Intel 8080 and Z80. In particular, function calls are stack-based. As the course aims to show to students, CISC architectures are programmer-friendly while RISC architectures are compiler-friendly.

As important as its features, its lack of features make the Game Boy CPU interesting for a Digital Systems course. Unlike the 68K CPU, from the 1980s, it lacks any supervisor features (Clements, 1999), so it does not support protection for multitasking or operating system. It also lacks any loop or repeat instructions, thus the implementation of C-like control structures does not have any high-level instructions as with the 68K or Z80 CPUs. Its memory addressing is pointer-based and the CPU does not handle any built-in address computation for array or structure field access. Its stack management is barebone, there is no concept of "stack frame" handled by the CPU. Also, memory access timings are constant because of lack of cache. It is possible to compute by hand the cycles of a code snippet even with memory accesses. All these topics are covered in later computer science courses of the program, namely Computer Organization and Operating Systems, both using x86-64 as a platform. Thus, it makes sense to use a more primitive platform for the Digital Systems course.

Note that the Game Boy hardware also supports Direct Memory Access (DMA) for faster memory copy, but the topic is not covered in the course due to its complexity of execution and is even forbidden in the project. Hardware interrupts are also omitted, preferring a more linear execution of programs.

Thus, the Game Boy is not strictly used in its intended form or in its most performant setting, but all code created as part of the course can run on real hardware and emulators.

2 The Game Boy Hardware and Programming Environment

From the point of view of students of the course, the Game Boy platform is relatively complex, but gracefully degrades from a pedagogical point of view. Thus, it is possible to cover only a subset of its features to develop interesting projects.

2.1 CPU

The CPU model in the Game Boy is Sharp SM83. It is also commonly called GBz80 due to its similarity with the Zilog Z80 CPU. It actually has fewer features than the Z80, in particular, it lacks any repeat instructions such as LDIR or DJNZ. This makes it similar to both the Z80 and the Intel 8080, released in the 1970s. Unlike the Intel 8080, it offers relative jump instructions.

The SM83 has 8-bit general-purpose registers and its memory is addressable with 16-bit addresses. It is an accumulator-based architecture, hence one of the register, A, is both the first operand and the destination register of the supported 8-bit arithmetic and logic instructions: add, sub (both without and with carry-in), cp (a subtraction instruction that only updates the flags), and, or, xor. The other 8-bit registers are B, C, D, E, H and L, and can also be used as 16-bit pairs of registers BC, DE and HL for memory addressing and simple arithmetic (however, only adding to the HL pair of register is supported as a single instruction).

In the course, we only use two flags of the CPU, that is zero and carry, the other two flags being for packed binary-coded decimal numbers, a feature we omit in the course. In the first half of the course, zero is essentially the only flag used.

The SM83 has stack instructions and a stack pointer, a feature introduced in the Intel 8080 CPU, as opposed to previous Intel CPUs. It provides the push, pop, call and ret instructions, plus a few instructions specific to the stack pointer that we do not cover in the course.

2.2 Memory Map

The Game Boy has a 16-bit addressable memory (64 KBytes), with addresses ranging from `$0000` to `$FFFF`. The first 32 KBytes map to the ROM, where the game's instructions and data are stored. As a consequence of a built-in commercial protection scheme, execution actually starts at address `$0100` instead of `$0000`.

Then, two ranges of addresses are particularly relevant. The 8 KBytes Work RAM (or WRAM) for general-purpose storage is permanently available to the CPU. Another 8 KBytes of RAM, the Video RAM or VRAM, is used to store graphic data. This data needs to be copied from the game ROM. As we will see later, most of the time, the VRAM is inaccesible to the CPU, because it is accessed by the graphic rendering chip.

Another relevant place is the OAM or Object Attribute Memory, an array of 160 bytes that stores the coordinates and attributes of on-screen moving objects (also called sprites). Finally, a few memory-mapped I/O registers are available.

Developing a real project on the Game Boy usually requires bank switching to allow for bigger ROMs. Most commercial games were indeed 64 KBytes to 512 KBytes big. However, since in this course, programming projects run much below the 32 KBytes limit, this aspect is omitted, hence the memory map remains static.

2.3 Graphic Rendering

Apart from the CPU, the Game Boy contains another essential chip called the PPU for Pixel (or Picture) Processing Unit. The PPU reads data from the VRAM and OAM and renders the LCD screen at nearly 60 fps. Thus, the Game Boy does not use a bitmap for drawing the screen but a tile-based background and sprites system, as many retro systems such as the Sega Master System, the Nec PC Engine or the Nintendo Famicom/NES (Altice, 2015).

The screen itself is 160x144 pixels big and can show any part of a 256x256 pixels scene made out of a background of 32x32 tiles, each tile being 8x8 pixels. On top of it, the PPU can draw up to 40 sprites (or objects in the Game Boy vocable) on screen, also 8x8 pixels. The VRAM can actually hold two background maps, and the PPU can quickly switch between both.

Now, the VRAM and OAM are critical resources shared between both the CPU and the PPU. In particular, when the PPU is on, VRAM and OAM can only be modified at very specific moments by the CPU. The CPU needs to wait for the Vertical Blank (VBlank), a short moment after each frame is rendered by the PPU, to update, for instance, the position of the objects on the screen (the OAM). The CPU can wait for the PPU to be in VBlank by polling an I/O register that reports the state of the PPU.

The CPU can also control several parameters of the PPU such as the viewpoint of the screen among the wider background, to generate a scrolling effect.

Unlike newer platforms such as the Game Boy Advance (Finlayson, 2017), the original Game Boy PPU does not support any bitmap mode. Thus, all game graphics must fit into this tile-based philosophy.

2.4 More Omitted Parts

Direct memory access (DMA) is a hardware feature that allows to copy from a Shadow OAM to the real OAM, in fewer cycles than doing it with CPU memory accesses. It helps making the most of VBlank time. However, it requires a tricky setup, by copying and executing instructions in High RAM (another uncovered feature) then waiting a certain number of cycles until the copy is done. DMA is omitted in the course, since it is still doable to write games handle VBlank time properly without it.

Not all CPU features and instructions are covered. For instance, Binary Coded Decimal support is left out. The CPU supports interrupts, but we omit them.

The Window layer of the PPU is also skipped, and palettes are only mentioned and used for black and white graphics. Indeed, each pixel of the screen can be displayed in 4 colors or levels of gray, but in this course, we only use black and white (1pp) graphics, for simplicity's sake and to focus more on game logic than graphics development.

The PPU supports two modes of indexing graphic tiles, the default one allowing for the background layer and the objects to use different (but intersecting) tiles set; an alternative one makes both background and objects take their graphic tiles from the same set. In the course, we use the second mode only, which unfortunately requires some explanations about these alternatives.

Finally, the audio output, the link cable data transfer and any specific feature of the Game Boy Color (a popular backward-compatible update of the Game Boy hardware) are completely ignored.

2.5 Programming Tools

Rednex Game Boy Development System (RGBDS in short) is a free and open-source toolchain to convert a source assembly file into a usable ROM image. It provides an assembler (RGBASM), a linker (RGBLINK), and a program to patch the ROM header to allow it to run on real hardware (RGBFIX). These tools are actively maintained as of 2025 and can be downloaded as binary or source code for all major OSes.

For the executing and debugging part, several emulators are available free of charge. The ones recommended in the course are BGB and Emulicious, since they both contain a disassembler and debugger, and a VRAM viewer. BGB is only available as a binary for Windows, thus requires Wine to run on other plaform, while Emulicious runs on all major platforms and requires Java. Both are closed source; an open-source alternative is SameBoy, which currently lacks a few features to help debug programs.

All these programming tools are actively maintained. Namely RGBDS had four major updates, and BGB, Emulicious and SameBoy had at least two major releases each since 2022, when we started including the assembly programming lab in the Digital Systems course.

3 Course Design

The duration of the course semester is 13 teaching weeks, followed by a period for final exams.

3.1 Weekly Syllabus

Here is the weekly syllabus of the assembly programming lab. Weeks 1-4 cover the CPU and memory, weeks 5-9 the rest of the Game Boy hardware and Weeks 10-12 are for advanced programming techniques. Week 13 is for projects presentations.

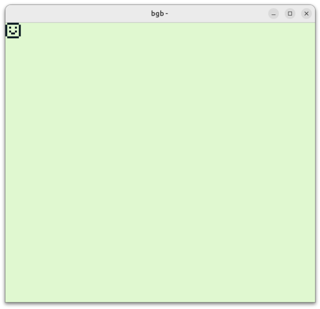
1. CPU 8-bit registers, arithmetic and logic instructions.
2. Stored program. Flags. Control flow.
3. Memory access. Variables and arrays.
4. Stack. Functions.
5. Programming toolchain (from ASM to ROM). Instructions size and cycles.
6. Graphic rendering, OAM, objects, VBlank
7. Entities and metaobjects
8. Keypad input. Background layer.
9. Jump tables. Finite state machines.
10. Lookup tables. Development tools.
11. Displaying text and integer values. Instructions optimizations.
12. Labels and macros of RGBASM. Development tools.
13. Projects presentations.

In the first phase of the course, exercises are about writing functions on paper or solving Parson's problems in the form of Moodle's gapfill exercises. The classes also involve MCQ quizzes to check understanding of the covered topics in real time.

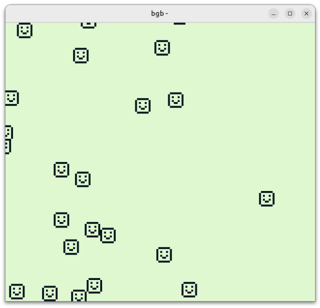
Once we switch to the development of executable programs on emulators (from week 6 on), the exercises are based on existing assembly programs that need to be modified.

3.2 Sample Program and Exercises

The first complete program is `first.asm`, which contains the minimum to show a single static object on the screen, without background. That would be conceptual equivalent to a "Hello world" program on the Game Boy:



`walk.asm` or Random Walk, is the first program that shows a non-trivial behavior and displays the full use of the OAM:



The program implement a classic separation between a Shadow OAM array stored in Work RAM, on which all processing is done by the CPU while the PPU updates the screen, and then during VBlank, Shadow OAM is copied to the real OAM in Video RAM. The outline of the program is as follows:

```
Initialization:

  1. Wait for VBlank
  2. Turn off PPU
  3. Set objects palette
  4. Copy tile from ROM to VRAM
  5. Initialize ShadowOAM
  6. turn on PPU (only object layer, no background layer)

Main loop:
  7. Update ShadowOAM
  8. Wait for VBlank
  9. Copy ShadowOAM to OAM
```

Its implementation fits in about 150 lines of code in RGBASM syntax. A few comments about the outline:

– the OAM is an array of 40 entries of 4 bytes each: (y,x,tile ID, flags)

- screen is 160 (h) x 144 (v) pixels big, objects coordinates range in (0,255) and screen's top left corner is has coordinates (y=16,x=8)
- 2. must be done during VBlank
- 5. setting object coordinate to 0 makes it invisible, effectively disabling it,
- setting the tile ID to zero means selecting the first tile in VRAM
- 7. is the actual random walk part, in which each object position is updated one step into one random direction among the 4 possible directions
- 8. as soon as PPU is on, the screen refreshes at 60 Hz
- 9. must be done within the time of VBlank ; this is why step 7 is done before VBlank on a "shadow" OAM

From this basic program, many exercises can be implemented, such as:

- tweak the random number generator function for better visual result
- at each frame, randomly choose one object to move instead of all of them
- implement 16x16 sprites made of 4 objects each
- instead of a random walk, make objects move in diagonal movement and implement bouncing on the edges of the screen
- implement "reset" and "pause" features on the press of some key

Almost all of these modifications require declaring new variables. In particular for 16x16 sprites (or metaobjects), special attention is paid to choosing adequate data structures for the entities in the program. This involves deciding which entity kind is a first-class citizen of the program. Only when this is done correctly, it is possible to combine bouncing objects with metaobjects in a reasonable amount of time.

3.3 Project

The programming project consists of re-implementing a known game for the Game Boy. It is announced after the first 9 or 10 classes, which leaves about 3 weeks for students to complete it. The project is done in groups of 2 to 4 students, or 2 to 3 in smaller cohorts. Beyond mastering assembly programming, the project is designed to help students understanding how code interacts with hardware components such as memory, graphics, and input, and to practice modular design by organizing their program into well-defined functions. It also involves planning and deciding the most adequate variables, data structures and algorithms to implement the mechanics of the game.

Group Size and Project Choice In the first iterations of the assembly lab, the projects were chosen freely, but this has proven too chaotic. Now, only one project topic, or a small list of alternatives is given to avoid delaying the actual start of the project. Also, it ensures that the level of difficulty is adequate.

In the case of a list of project topics being provided, a project fair is organized, in which students groups can register, choose a group name and give their preference among the available projects (an option ”no preference” is also available). Then projects are assigned such that the number of groups per project do not differ by more than one. Priority is given to the groups with fewer students first, thus nudging students into forming smaller groups.

Projects Topics

Puzzles for 1 player turn-based puzzles, such as 15-slide puzzle, 2048, Sokoban, Klotski

Real-time games for 1 player such as the Chrome Dino, 1D Pacman, Pong, Slime Volley

Outcome of projects The Winter 2024 semester was the 4th iteration of the assembly programming lab. Out of 47 enrolled students, 15 groups were formed, spread into 4 groups of 2, 5 groups of 3 and 6 groups of 4 students. However, one group of 2 students later dropped from the course and did not submit any project.

The available projects were Sokoban, 1D Pacman and Klotski, whose got assigned each to 5 groups. Only two groups were assigned a project that was not their first choice, both of 4 students. Klotski was the project of major algorithmic and data structure complexity, followed by Sokoban and finally 1D Pacman, so the grading was adjusted according to which game was implemented.

All the submitted projects were at least in a playable state and all seemed to be of the sole authorship of the groups that submitted them, with high confidence. We interpret that the level of difficulty of the project was adequate.

In the Spring 2025 semester, with 23 enrolled students, only one project was proposed: Sokoban Deluxe, in which the ”Deluxe” part came from for the implementation of two special features: a step counter and an undo feature. The number of students per group was limited to at most 3. Only 18 students submitted the project (thus 5 dropped from the course), 8 groups were finally formed, with 4 groups of 3 students, 2 groups of 2 students and 2 individual submissions. Unfortunately, one of the submitted project was plagiarized from the previous semester, by a group of 3 students.

The Spring 2025 semester introduced the idea of project resubmission. A grade is given at the first presentation of the project, then within a week, the group must fulfill a list of improvements to their projects to improve their grade by up to 10%. This system was used by 6 of the 8 groups to improve their grades.

Reflecting on these last two iterations of the projects make us think of a few good practices to better guide students. One is to set restrictions on the graphical aspect of the submitted programs, by only allowing 1bpp graphics and limiting the amount of different tiles in the program (e.g., 64 tiles max). This avoids having some group member(s) who is only responsible for data or graphics. In the case of a project that uses a list of levels, such as Sokoban or Klotski, providing the level data file can be a good idea so that the groups focus on the code. Finally, these restrictions, and others such as: no interrupts, no DMA, no sound etc. help prevent the reuse of existing code from the internet.

4 Conclusions

Since its creation in 2022 and over 5 iterations, the assembly programming lab has evolved into a solid part of the Digital Systems course. It was updated after each semester according to the difficulties and feedback of students and the observations of the teacher. It became a modern assembly programming lab using a retro gaming platform whose programming tools are actively maintained to this day. The detailed syllabus includes progressive exercises that evolve into more challenging programming task as the semester progresses. Not only it claims to teach its course contents of computer architecture, but it also tries to connect with the surrounding course's higher level perspective about software engineering.

It culminates with a group programming project whose objective is to bring a hand-on approach to both computer architecture and software engineering. By choosing among a list of projects of moderate difficulty, almost all students groups were able to submit projects at least in a working state, a sign that the project is motivating (Groeneveld & Aerts, 2020). There is still work to do about situations where students consider the project to be too difficult and choose either to quit, to free-ride in a group (Benning, 2024) or to submit a plagiarized project (Vahid et al., 2023).

In the future, the lab will evolve into incorporating more exercises and quizzes to improve feedback and self-assessment of students. Different kinds of projects can be explored too, for instance coming back to the first semester's idea, with the implementation of a non-interactive demo, or the implementation of an assembly emulator of a subset of the Game Boy platform in C in the vein of PokeGB (Smith, 2021).

References

Altice, N. (2015). *I am error: The Nintendo family computer/entertainment system platform*. MIT Press.

Benning, T. M. (2024). Reducing free-riding in group projects in line with students' preferences: Does it matter if there is more at stake? *Active Learning in Higher Education*, *25*(2), 242–257.

Clements, A. (1999). Selecting a processor for teaching computer architecture. *Microprocessors and Microsystems*, *23*(5), 281–290.

Finlayson, I. (2017). Using the Game Boy Advance to teach computer systems and architecture. *Journal of Computing Sciences in Colleges*, *32*(3), 78–84.

Groeneveld, W., & Aerts, K. (2020). Sparking creativity with the Game Boy Advance. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 1326–1326.

Smith, B. (2021). POKEGB: A gameboy emulator that only plays pokémon blue. https://binji.github.io/posts/pokegb/

Vahid, F., Downey, K., Pang, A., & Gordon, C. (2023). Impact of several low-effort cheating-reduction methods in a CS1 class. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 486–492.