Street Fighter II - MIPS Version

Iago Lobo Ribeiro de Moraes[*] Cristiano Krug Brust[†] José Marcos da Silva [‡] Yan Victor dos Santos[§]
André Luiz de Moura Ramos Bittencourt[¶]

University of Brasília, Department of Computer Science, Brazil

Figure 1: Street Fighter II Artwork

Abstract

Street Fighter is a popular franchise of fighting games in which the players select combatants from around the world, each with their own special moves, against one another. Capcom released the second game in the series in 1991 worldwide. Since videogames were programmed in low-level languages and had to deal with microprocessors back then, our course of Computer organization and architecture's final project is a playable version of the classic Super Nintendo classic, utilizing and applying the knowledge acquired throughout the semester.

Keywords: Videogames, Computer organization and architecture, Street Fighter, MIPS.

1 Introduction

In 1991, Street Fighter II was released on many platforms, becoming one of the most played games of all time. With 8 diverse playable characters and 4 bosses, every character has special moves acessable by doing specific button sequences. At the time, the Super Nintendo port was considered innovate for keeping good graphics from the arcade and having decent game speed. Seeing this amazing accomplishment, professor Marcus Vinícius Lamar, University of Brasília, decided to challenge his students.

Games back then were made with specific microprocessors in mind and low-level languages were the only option. These two situations match perfecty with our course, therefore our final project is to make a new version of Street Fighter II on microprocessor MIPS, utilizing the development kit DE2-70. Our objective is to learn about the microprocessor while developing the game. The requirements of the project are: functional menu and presentation, 12 characters each with a special move, music, sound effects, SD card for storage, keyboard response and two game modes, arcade and versus.

[*]e-mail: iago.wolf.moraes@gmail.com
[†]e-mail: cristianobrust123@gmail.com
[‡]e-mail: leite.josemarcos@gmail.com
[§]e-mail: yanvictor_ds@hotmail.com
[¶]e-mail: bittencourtandre@hotmail.com

2 Theoretical and Technical Basis

2.1 Hardware requirements

First of all, the microprocessor MIPS that we used has been in development for years. Each semester, the class contribuits implementing one or more new features. Alongside the development kit DE2-70, a monitor and VGA cable were used to show the graphics. To control the game in general, a USB Keyboard worked fine. Also the SRAM. SRAM, or Static random-access memory, is a type of semiconductor memory that uses bistable latching circuitry, also known as flip-flops, to store each bit. SRAM exhibits data remanence, however it's volatile in the conventional sense that data is eventually lost when the memory is not powered.

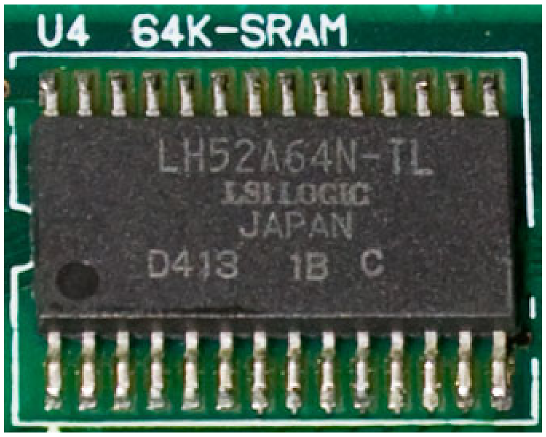
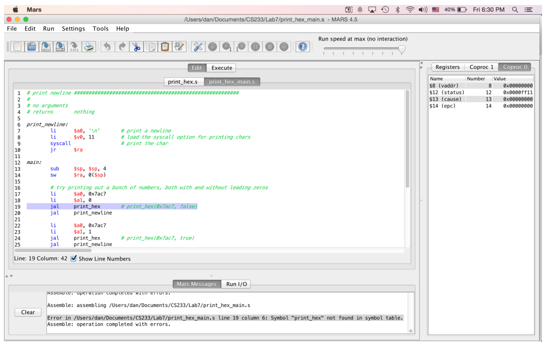


Figure 2: Super Nintendo's SRAM

2.2 Software requirements

To simulate the microprocessor, MARS was used. Created with Java, it runs in every Operating System and can be customized. The low-level language of choice is Assembly MIPS, teached in our course. With MARS, it's easy to locate errors, addresses, count instructions, ajust image output, etc, helping on the development of the game.



/Users/dan/Documents/CS233/Lab7/print_hex_main.s - MARS 4.5

```
# print newline ###############################################################
#
# no arguments
# returns       nothing

print_newline:
        li      $a0, '\n'       # print a newline
        li      $v0, 11         # load the syscall option for printing chars
        syscall                 # print the char
        jr      $ra

main:
        sub     $sp, $sp, 4
        sw      $ra, 0($sp)

        # try printing out a bunch of numbers, both with and without leading zeros
        li      $a0, 0x7ac7
        li      $a1, 0
        jal     print_hex       # print_hex(0x7ac7, false)
        jal     print_newline

        li      $a0, 0x7ac7
        li      $a1, 1
        jal     print_hex       # print_hex(0x7ac7, true)
        jal     print_newline
```

Line: 19 Column: 42 ✓ Show Line Numbers

```
Assemble: operation completed with errors.

Assemble: assembling /Users/dan/Documents/CS233/Lab7/print_hex_main.s

Error in /Users/dan/Documents/CS233/Lab7/print_hex_main.s line 19 column 6: Symbol "print_hex" not found in symbol table.
Assemble: operation completed with errors.
```

Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

Figure 3: Example of Assembly MIPS code in MARS

3 Methodology

In order to follow the project planning and to execute it with the maximum of mastery possible, it was necessary to understand the main concepts of the processors that work with pipeline, unicycle and multicycle. For our work, we selected the pipeline processor for its high performance during assembly code execution.

Through the activities proposed in class for the laboratories, we observe the performance difference of each processor. We concluded that, as described in the reports related to these activities, the pipeline processor would be the most feasible in practice given the complexity of the game implementation that could affect performance during execution.

We used the MARS 4.7 Custom 7 tool to simulate code execution in the Street Fighter game assembly. In order to build the game's well-defined logic, we consider the limitations of the DE2-70 board so that no unforeseen events occur during execution. Among the known limitations, we are faced with memory limitation, audio synthesizer to play the sound of the game and difficulty of using some instructions that were not implemented previously on the card.

In order to run the project in the processor proposed by the discipline, in pipeline, we encode in Assembly, with the help of the MARS tool, and export the files to a format recognized by the processor. We use an SD card to store data that will be processed during the execution of the game. To play, the user needs to use a keyboard and view the output of the processing on a monitor. To listen to the music you play during the game, you need to insert headphones.

3.1 Developing the Game

Towards the goal, we produce the game with modularized code to facilitate access to necessary components elsewhere. We have defined a number of sprites that would be used in the game, converting them to the *.bin* format that can be read during execution. The mechanics of the game is based mainly on replacements of images, image on image, to simulate movement, exchange of options in the Menu screen, among others.

For the Menu screen, we allow you to choose from three options: Arcade mode, versus mode and quit to quit the game. The game also allows the choice of two players, if the Vs. mode is selected.

3.1.1 The SD Card

For the storage of data, we made use of the SD card that keeps all the images (sprites) necessary for the successful execution of the game. Reading the SD card was the biggest problem we encountered that resulted in the difficulty of completing the project. Their limited ability has shown us the need to find a viable solution for storing data. For this, we define a unique size for all sprites, facilitating access to the data. We take into account the size of the largest image to standardize the size of the images inside the SD card. However, for the larger image, it is not possible to load all the images into a SSRAM memory on the card. Therefore, we ignore the heavier image and use the size of the second larger image, which allowed us to use many sprites given the memory limitation. Only when the greatest image was needed would it be "called." For a given sequence of movements, images that were previously on an SD card are passed to the SSRAM memory of the DE2-70 card to increase the speed with which the image is passed to the monitor screen.

Figure 4: SD Card used in Street Fighter II - MIPS Version

3.1.2 Mapping of images

All sprites are mapped to an SD card. For each action required during user interaction with the game, if it is necessary to show a multi-frame animation, these images that were previously passed from the card to the DE2-70 SSRAM will fetch the image to be displayed on the monitor by the VGA. An animation is represented by a set of images shown in sequence.

3.1.3 The Keyboard

During the fight, it is always verified if any key on the keyboard has been pressed, taking the decision corresponding to which the key was defined. The movement of the players depends on the keyboard input. However, while the player does not press any key, sprites set for when there are no keystrokes show steady movement in the game, revealing to players a fighting stance.

We map the keyboard with the 'w', 's', 'd' and 'a' keys for the first player to make basic moves, and the 'e' and 'q' keys for attacks. For the second player, we map in the same way, but with the 'i', 'j', 'k' and 'l' keys. The 'u' and 'o' keys for attack. For special attacks, we map the 'z' key to the first player and 'm' to the second player.

3.1.4 Basic Movements of Players

For the mechanics of the game, players can perform basic tasks such as: walking forward, walking backward, jumping, kicking, striking and executing a special blow on their opponent. Two bars that indicate the amount of life of each player decides to close the match. For each character in the game, there is a scenario corresponding to it.

As a consequence of the fighting movements between the players, each player has a life bar, which decreases as the player receives attacks from the opponent. This bar indicates when a player will lose the match. Much of the structure implemented was based on the original Street Fighter game.

3.1.5 The Audio Module

Before commenting our procedures, it's essential to explain what MIDI pattern is. MIDI, Musical Instrument Digital Interface, is a technical standard that describes a communications protocol, digital interface and electrical connectors and allows a wide variety of electronic musical instruments, computers and other related songs and audio devices to connect and communicate with one another.

For the audio module, the synthesizer presented problems that made it difficult to play and output sounds. The fact that you have multiple tasks waiting to run on a processor, you need to consider that performance will not necessarily be greater. In a code that follows a linearity, it is necessary to be constantly checked so that it is possible to play a next note until a melody is identified during the game.

All issues in this subsection have been resolved through code implementation. In the case of music, the theme song of famous character Guile, we insert notes that play whenever no command is requested to play and the note is time to play. The conflict of linearity, without parallelism, generated a delay in music.

Adapting the music to use the Altera DE2-70's synthesizer with syscall 31 was also one of our works. Utilizing pattern Midi, a C++ language program converted the notes to a text file.

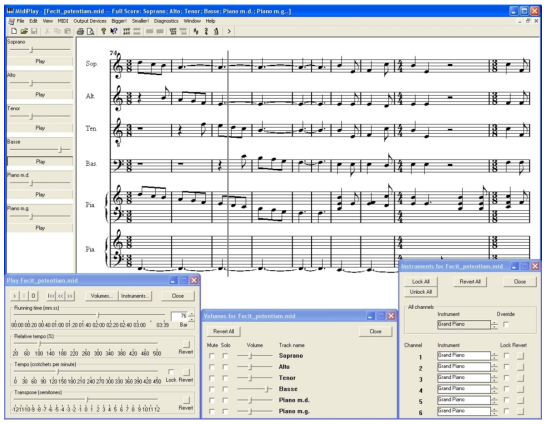


Figure 5: MIDI file on MidiPlay, a MIDI editor

4 Results

Like in any project, our group had some difficulties during the development. Programming the collider of the characters took some time, as well as jumping and special moves. Our limited knowledge on Assembly also gave us some trouble, since this was our first project in a low-level language. Also, running our program on actual hardware required a lot of workarounds and adjustments.

After the implementation of our final project, we managed to complete most of the requirements. First, functional menu and select character screen, using the ASWD keys to navegate.

Second, all twelve characters and their respective stages were implemented, as well as two game modes, arcade and versus, working life bars and music.

Third, regular punchs and kicks, both standing and jumping, and special moves for all twelve characters were implemented, as well as blocking your opponent's attacks.

4.1 Learning About the Proposed Work

To create the game, it was necessary to know the capacity of the proposed processor to execute the code built. Therefore, many problems and limitations were found preventing the implementation of many game improvement features.

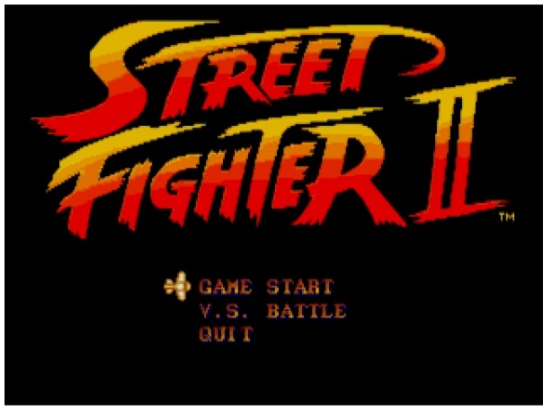


Figure 6: Menu Screen of Street Fighter II - MIPS Version

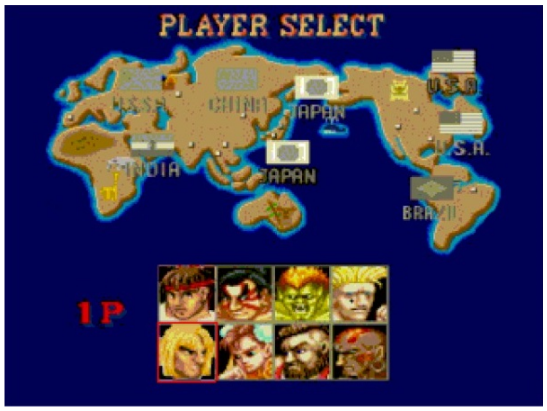


Figure 7: Select Character Screen of Street Fighter II - MIPS Version

For mandatory functionalities proposed for the work of the discipline, it was necessary to overcome the problems found, such as the low capacity of the SSRAM memory. In order to solve them, it was necessary to thoroughly understand their operation and exchange experience with other teams to share information.

The practice at work contributed to fix the theory passed in the classroom, by the teacher, and the 5 laboratories that were proposed to perform. The execution of the code in DE2-70 allowed us to visualize how the code is handled when exceptions, system calls, successful calls, execution flow and processing speed occur.

5 Conclusion

Combining hardware with software isn't always easy. In the 90's, these requirements were all limitations developers had to overcome by being creative. Programming this new version of Street Fighter II shows how difficult it were back then. Even so, in a couple of months, we managed to have a playable build, having fulfilled most of the requirements of the final project. With this task, we learned more about the development kit DE2-70, completing our objective.

6 Future Works

For future projects, we'll focus on implementing and using the floating-point unit in pipeling. It would help when running the program in general, especially when two or more heavy things need to be executed, like music or graphics, for example.

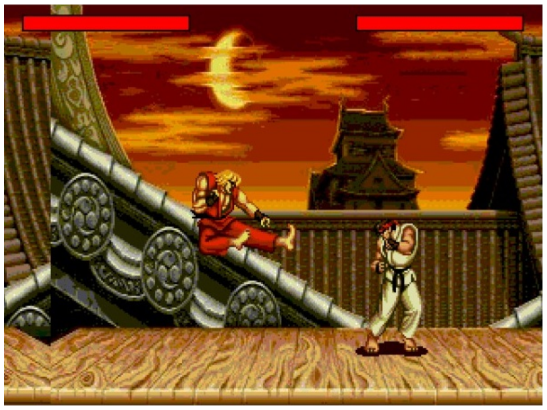


Figure 8: Ryu's Stage, Street Fighter II - MIPS Version

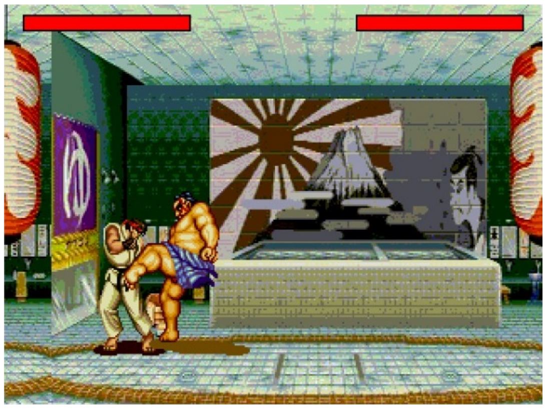


Figure 9: E.Honda's Stage, Street Fighter II - MIPS Version

7 REFERENCES

[1] David A. Patterson, John L. Hennessy. Organizao e Projeto de Computadores - a interface Hardware/Software. Editora LTC, 2000, 551 p.
[2] John L. Hennessy, David A. Patterson. Arquitetura de computadores: uma abordagem quantitativa. Editora Campus, 2003, 827 p.
[3] Chris Carle. "Street Fighter: The Complete History". Chronicle Books, 2010, 176 p.