IGS - Instructional Game Simulator

Table of contents

Installation

Welcome to the Python instruction set simulator project!
This project includes an assembler (ASM) for converting assembly code into machine code, a virtual machine (VM) for executing machine code, and a cache for storing frequently used data. The goal of this project is to allow users to write and execute their own machine code using a predefined set of instructions.

Usage

First, you will always need to put a path to a file for the simulator to work

Assembly

For the following examples we will take this simple assembly programs that puts the number 1 in the first register and prints its value on screen :

```
#data.txt
add r0, 1, r1
sub r1, 1, r0
stop
```

In order to assemble a program into binary instructions you need to use the ASM command in python :
For Windows :

```
python ASM.py data.txt
```

For Linux :

```
python3 ASM.py data.txt
```

You will have this answer if you don't respect args :

```
Usage: python ASM.py <filename>
```

The content of the binary file should look like that :

```
0x0 0x8200021
0x1 0x10600020
0x2 0x98000000
```

Virtual Machine

Using the python command, you can execute an assembled file, here we take the last example:

```
python VM.py
```

Your terminal should display :

```
C:\Users\baptl\PycharmProjects\pythonProject\Microprocesseur>py VM.py
Etats des registres : [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Execution time : 0.0 seconds
```

Explanation of the code

In this part I will explain how my python scripts work :)

Assembly

The ASM.py script is divided into three steps that will allow to get a binary file.

The first step is to format the basic file, the unprocessed file goes through the "file_layout()" function below:

With this function, we remove unwanted elements from the file, such as comments identified with "#", or punctuation...
The "file_layout()" function returns the list of instructions in str() form.

This list is then called in the "decode_instr_bin()" function, which will transform the str() format into int(), by interpreting the content of each instruction.

Finally, the "instr_dec()" function takes care of the 32-bit shifting of the instructions, respecting the structures seen in class:

The script returns a .txt with each instruction translated in hexadecimal, and their associated address.

Virtual Machine

The VM.py script is broken down into three steps to allow the execution of the asm code.
First, in a similar way to the Assembly script, the hexadecimal instructions file is arranged, to separate the instructions from their address, thanks to the "file_layout()" function.

Then the instructions in hexadecimal are decoded in the "instr_decode()" function. According to the operating code, each element of the instruction is correctly shifted.

Finally, the "instr_operation()" function performs the operations for each instruction until the end of the program.

Cache Memory

The CacheMemory.py script allows to simulate in a rather simplistic way the cache memory.
As seen in class, the cache memory has a very precise structure, a number of sets S, lines E, blocks B, a number of address bits m.

The cache memory simulator is composed of the function "get_data()" to read in the memory, and "write_data()" to write in it. The memory is presented as a text file "memory.txt" with an address column and a column for the associated data.

Some Applications

Now I will present you some applications to test the simulator.

Mean

The file "mean.txt" is a small assembly code allowing to test some arithmetic operations, but especially the use of SCALL (System Call).
The program invites us to fill in a value twice, which will be stored in the first register, before being copied to another. The purpose of the program is to calculate the average between several register values.

Syraccuse

The file "syraccuse.txt" is an example of code provided in class, it allows to show the use of arithmetic operations, branz, braz etc...

Test ALL

The file "test_all.txt" allows to test all operations, adding in each register the expected results. Additional explanations are provided in the comments.

Guessing Game

The file "guessing_game.txt" is a copy of the well-known game of "fair price". The goal is to guess the mystery number. A nice little application of the simulator :)

Conclusion

First, I met several difficulties, the VM performance in particular with a rather weak MIPS. This is explained by my programming choices (too many comparisons and recursive loop for some operations). This will be a point to improve in the future.
I also have an error management in my programs that could be much improved.

This project allowed me to familiarize myself with the architecture of a microprocessor, I was able to understand each step, from the compilation (another course) to the ISS. The fact that I worked alone was beneficial, I gave my best to go forward.