Technical University of Denmark

02135 - Introduction to Cyber Systems

Implementation of an instruction set architecture (ISA) simulator in Python

Designed by:
Daniela Bahneanu- s184366
Guillaume Ratier- s184466

Contents

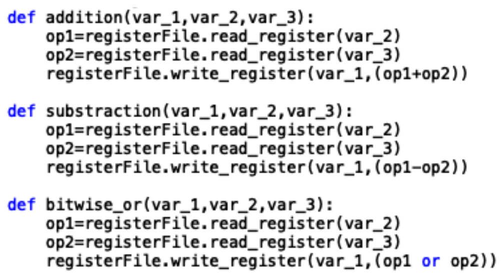
1 Introduction

This assignment aims to simulate an instruction set architecture. The idea behind it is to write a code that performs a basic set of arithmetic operations and writes the data in a specified register. The datapath consists of the ALU (a digital electronic circuit that performs bitwise operations on integer binary numbers) and the registers where the data is stored. The processor simulated in this assignment contains 16 registers (R0 - R15), each of them having a size of 8-bits (each register having an unsigned value that can go up to 255).

2 Testing our code

2.1 Methods

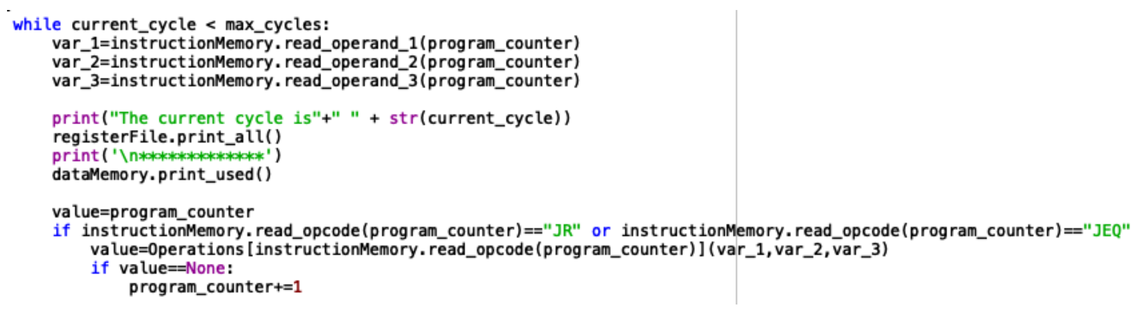
We started our code by creating a dictionary in order to access the opcodes (since it was a txt file, the operands weren't predefined). Probably, the most difficult part was to write the functions for each opcode as it was needed to analyze the arithmetic operations defined by the opcode, perform the operation on operand_2 and operand_3 and save the data into operand_1.



```python
def addition(var_1,var_2,var_3):
    op1=registerFile.read_register(var_2)
    op2=registerFile.read_register(var_3)
    registerFile.write_register(var_1,(op1+op2))

def substraction(var_1,var_2,var_3):
    op1=registerFile.read_register(var_2)
    op2=registerFile.read_register(var_3)
    registerFile.write_register(var_1,(op1-op2))

def bitwise_or(var_1,var_2,var_3):
    op1=registerFile.read_register(var_2)
    op2=registerFile.read_register(var_3)
    registerFile.write_register(var_1,(op1 or op2))
```

Figure 1: example of function used to define the opcodes

The variable used in the functions are defined later on in the while loop and they represent the name of the registers on which the arithmetic operation is performed (op1 and op2 represent the content of these registers).
The program counter is a variable that keeps track of the program execution. It contains the address of the instruction that has to be executed. It was mostly used in the while loop (the core of our program). The while loop analyzes whether the program counter should be incremented by 1 with each cycle.

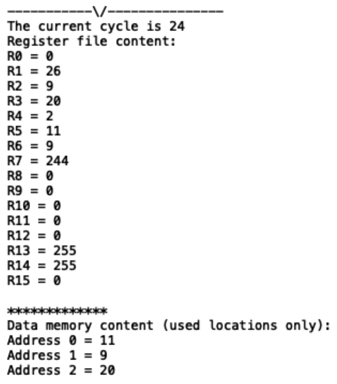


```python
while current_cycle < max_cycles:
    var_1=instructionMemory.read_operand_1(program_counter)
    var_2=instructionMemory.read_operand_2(program_counter)
    var_3=instructionMemory.read_operand_3(program_counter)

    print("The current cycle is"+" " + str(current_cycle))
    registerFile.print_all()
    print('\n*************')
    dataMemory.print_used()

    value=program_counter
    if instructionMemory.read_opcode(program_counter)=="JR" or instructionMemory.read_opcode(program_counter)=="JEQ"
        value=Operations[instructionMemory.read_opcode(program_counter)](var_1,var_2,var_3)
        if value==None:
            program_counter+=1
```

Figure 2: The while loop used (the use of the program_counter variable)

2.2 Test 1 - Methods and approach

The first test is straight forward as it requires 24 cycles to be executed. All instructions are used and each of them consists of an opcode and a list of operants. What is different about this assignment is that the 3 given classes (The RegisterFile class, The DataMemory class and The InstructionMemory class) are used to access and rewrite the data. The RegisterFile class includes a set of functions that are designed to read, write or print the data in the registers. On the same note, it is important to emphasize the fact that the data in the first register (R0) can only be read. The second class: DataMemory class is used to access the memory content of the second argument of the simulator. It is used to read, write or print the content of the datamemory at a specified address. The last given class is The InstructionMemory class, used to read or print the instructions (opcode and operands).
The output for this test was the expected one, proving that the code was able to identify the opcode, perform the arithmetic operation and rewrite the expected data into one of the 16 registers

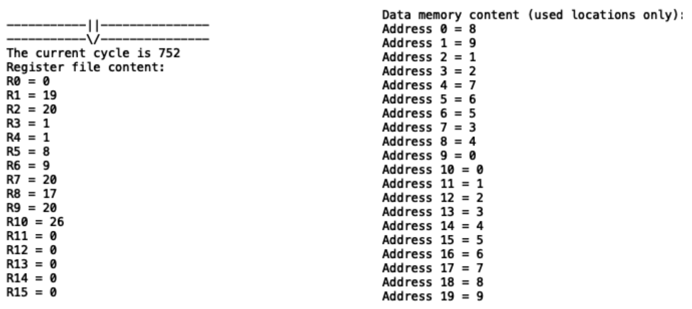


```
-----------\/---------------
The current cycle is 24
Register file content:
R0 = 0
R1 = 26
R2 = 9
R3 = 20
R4 = 2
R5 = 11
R6 = 9
R7 = 244
R8 = 0
R9 = 0
R10 = 0
R11 = 0
R12 = 0
R13 = 255
R14 = 255
R15 = 0

*************
Data memory content (used locations only):
Address 0 = 11
Address 1 = 9
Address 2 = 20
```

Figure 3: Expected output for test 1

2.3 Test 2

The second test didn't require the addition of any special lines of code as it was similar with test one except it has a greater input of instructions. The program provides an array of length 10 filled with values stored in the DataMemory. The simulation it is performed in 752 cycles and the dataMemory content has 19 used addresses.



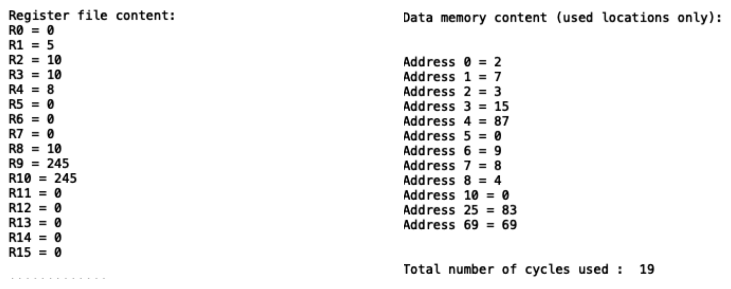
```
-----------||--------------
-----------\/--------------
The current cycle is 752
Register file content:
R0 = 0
R1 = 19
R2 = 20
R3 = 1
R4 = 1
R5 = 8
R6 = 9
R7 = 20
R8 = 17
R9 = 20
R10 = 26
R11 = 0
R12 = 0
R13 = 0
R14 = 0
R15 = 0
```

```
Data memory content (used locations only):
Address 0 = 8
Address 1 = 9
Address 2 = 1
Address 3 = 2
Address 4 = 7
Address 5 = 6
Address 6 = 5
Address 7 = 3
Address 8 = 4
Address 9 = 0
Address 10 = 0
Address 11 = 1
Address 12 = 2
Address 13 = 3
Address 14 = 4
Address 15 = 5
Address 16 = 6
Address 17 = 7
Address 18 = 8
Address 19 = 9
```

Figure 4: The expected output for Test 2

2.4 Test 3

This test is just a mix of all the different methods that were introduced during this assignment, no precise goal or real life example is in this test. The code works with this test, as predicted which means that our code will work with any instruction and data memory files provided, which fulfills the purpose of the assignment. This test working marks the end of the coding part of the assignment.



```
Register file content:
R0 = 0
R1 = 5
R2 = 10
R3 = 10
R4 = 8
R5 = 0
R6 = 0
R7 = 0
R8 = 10
R9 = 245
R10 = 245
R11 = 0
R12 = 0
R13 = 0
R14 = 0
R15 = 0
```

```
Data memory content (used locations only):


Address 0 = 2
Address 1 = 7
Address 2 = 3
Address 3 = 15
Address 4 = 87
Address 5 = 0
Address 6 = 9
Address 7 = 8
Address 8 = 4
Address 10 = 0
Address 25 = 83
Address 69 = 69


Total number of cycles used :  19
```

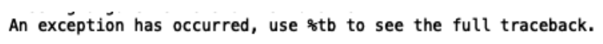
Figure 5: The expected output for Test 3

3 What was different in this project

Compared to the first assignment where we had to build a fsmd, when writing the code for the isa, we were first introduced to object oriented programming. This type of programming made accessing the functions easier making the code more efficient

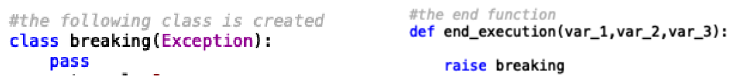
3.1 Special lines of code used

We notices that in both, assignment 1 and assignment 2, when using the sys.exit() method to end the simulation, an error message would show up in the terminal :



```
An exception has occurred, use %tb to see the full traceback.
```

In order to exit the program without having this message in the output, we created a new class called breaking and defined a function called end_execution that will end the simulation without givin the %tb error.



```
#the following class is created
class breaking(Exception):
    pass
```

```
#the end function
def end_execution(var_1,var_2,var_3):

    raise breaking
```

Figure 6: New class to end the simulation

4 Conclusion

Even though in the beginning it was hard to choose the right approach, after realizing that if statement aren't very efficient we build a dictionary which made the code more compact and user-friendly. Performing all three test showed us that the code works and is able to analyze and perform operation on the registers. Hence reaching the expected level for this assignment.