Introduction to Cyber Systems
Assignment 2 - Implementation of an ISA simulator in Python

Benjy Jepsen
João Ramiro

September 2018

1 Introduction

The task was to create a simulator that could simulate a single cycle processor (no pipeline), by decoding the instructions and executing them. The simulator takes in .txt files, one of them describes the program to be executed which is in assembly code and the other one contains the initial values of the registers and of the memory. Two test cases to simulate Assembly programs were provided: test_1 and test_2. In addition, we created our own test case, test_3.

2 Simulator

Code to parse the .txt documents into python variables was provided. The code that was written to simulate the ISA is relatively simple as it mostly relies on this helper code.

The main loop iterates over a predetermined amount of cycles set by the user to prevent the code from running forever if the endstate of the program isn't reached.

Functions representing the permissable operations of the ISA were written in python. A dictionary of these functions, with their associated operation codes, was created to be able to "search" for a function by its operation code. The functions all take in 3 registers as variables, even though they may not be used in the function, to facilitate the execution of the functions once they are "found" in the dictionary of functions.

The loop starts at program counter zero and takes in the operation code associated with program counter zero from the program.txt file. The corresponding function to this operation is executed by searching the dictionary of functions using the opcode, and executing the function assoceiated with the operation code. The program counter is either incremented by 1 if no jumps were executed, or jumps to the program counter determined by the jump. The code runs until the operation code "END" is reached, at which point the program quits.

When the code searches in the dictionary of functions, a break is executed after the function is found so as to prevent the entire dictionary being searched.

3 Test 1 and Test 2

Test 1 and Test 2 were provided as tests to determine if the Assembly simulator runs correctly.

Test 1 is a code that has no purpose. It is only used to check that the simulator is running correctly. After comparing the end results of the simulator after running Test 1 with the `expected_1.txt` file, the simulator was deemed to be working properly.

Test 2 implements the widely known bubble sort algorithm. The values to sort are initially in Memory and are as follows: `[8,9,1,2,7,6,5,3,4,0]`. After running program_2 in the simulator, the values were sorted correctly, in ascending order, as follows: `[0,1,2,3,4,5,6,7,8,9]`.

4 Test 3

We had to provide the the third Assembly program to test in the simulator. We decided to implement a simple program that return statistics about an array introduced in memory. The returned stats are the following

- Sum in address 249;
- Maximum in address 250;
- Minimum in address 251;
- Average (Integer) in address 252;

The current program returns this statistics for the following array: `[2,4,10,1,20]`, which correspond to:

- Sum: 37;
- Maximum: 20;
- Minimum: 1;
- Average: 7;

The average is `n`, where `n` is the amount of times that we can subtract `#elements` from the `sum`, before getting a negative value.

Bellow it is shown a pseudocode that shows how the assembly code is structured:

```
array=[2,4,10,1,20]
max=array[0]
min=array[0]
accumulator = array[0]

#min max loop
for (i=1; i<array.length; i++)
    if(array[i]>max)
        max=array[i]

    if(array[i]<min)
        min=array[i]

    accumulator += array[i]

#average
avg=0
aux=accumulator
while(array.length>aux)
    aux -= array.length
    avg++

exit
```

It is important to note that it is possible to return statistics for another array, with up to 256 elements, to do it the `data_mem_3.txt` should be changed to have that array where the first element of it will be on adress 0 (0x0h). Also, on the `program_3.txt` in the line ”`6: LI R10, 5;  # Number of elements in array`” the value `5` should be changed to the new array length.

Almost all the available registers were used in the implementation of our assembly code. The function of certain registers changed throughout multiple parts of our assembly code. The Registers were used as follows:

Register	Function
R0	0
R1	Points to instruction memory address for storing the average
R2	Holds current maximum value
R3	1
R4	counter variable used for the min/max loop and finding the average
R5	holds value to be compared with the current minimum and maximum
R6	Unused
R7	Points to the instruction memory address for the setting the maximum
R8	Points to the instruction memory address of min/max loop
R9	Points to the instruction memory address for storing the max, min and sum
R10	Number of elements (for this example is 5)
R11	Holds the minimum value
R12	Points to the instruction memory address of the set minimum function
R13	Points to a instruction memory address inside the max/min loop
R14	First it will store the sum of the values in the data memory, then is used to calculate the average afterwards, and is the remainder in the end
R15	Auxiliar variable used to store the statistics calculated

Table 1: Registers and their assigned functions

5 Conclusions

This assignment was a good introduction to understanding ISAs. Perhaps the most challenging part of the assignment was being able to create a program in Assembly that would run as desired. Being able to create our own loops in Assembly and being able to figure out how to interesting and rewarding