Salim Kemal Tirit
Bengisu Kübra Takkin

PROJECT DOCUMENTATION

General Process

Project consists of parts. First part takes assembly code as an input, converts it into binary machine code in hexadecimal format and the second part takes these hexadecimal input and then executes it.

First Part (assembler.py):

This part of the code takes assembly code and by going over the code two times, converts the file into hexadecimal format. In the first tour, it first checks whether the given input is a label or not and if it is a label, It marks the location by putting the label into a dictionary with the number of the instruction.

In the second tour, It first detects opcode by using an array, detects addressing mode by looking at operands and via using them it generates 3 byte instruction. Then send it into the queue.

Note that we designed the assembly.py in such a way that It can detect syntax errors and invalid inputs. In those cases, the code will not generate a .bin file.

Second Part (cmpe230exec.py):

This part of the code takes hexadecimal 3 bytes instructions as an input and by mimicking a CPU, it executes these instructions and writes output to a .txt file, if there is any.

Some Important Elements:

Registers : a dictionary which stores the name of the register with their pointers. It is assumed that each register has 2 byte size.

Memory : an array with capacity 2^{16} which stores instructions and datas at the low addresses. At the high addresses, It stores datas of the stack. We implemented it in such a way that each index illustrates 1 byte cell of the memory.

Some Important Features:

Taking Input from Console :

By implementation, when the program encounters with the read command, it asks from the user a single character. In case that there are consecutive read commands, the program accepts characters again one by one, note that the user has to press enter before writing another character.

Ups and Downs:

In the first part of the project, even though there are 28 instructions, we avoided writing each of them one by one. Instead we used an array and put commands into the array positions according to their decimal opcode value.

Another powerful way of the project is that it can determine invalid instructions given in the asm files.

How did we approach the problem?

We started by trying to understand the problem. We started a little bit late because we knew there were things in the description that were going to change. The description was not telling us much about the project so we read piazza from top to bottom. Then we implemented the first part using our imagination. It was pretty easy indeed but we were not sure if we were doing the right thing.

Salim Kemal Tirit
Bengisu Kübra Takkin

Then we started the second part. It was easier to understand at first but we were -again- not sure. We started by implementing the flags, memory and registers. Then we started implementing the functions. We shared functions and controlled each other's implementation afterwards. Then we debugged the project together and changed lots of things. When we were finished we tried some inputs that we created. We were confident about our project until everything changed. There was not much problem with the test cases; we handled our code's problems quickly. Then we learned that we needed to detect if there were mistakes in the input given. We changed our code that way, spending a couple of hours. We were sure it was finished for the second time. Then all of a sudden we learned this was not the case and it was unnecessary for us to spend those hours instead of working for our finals. At the end we are now confident with our code. This was a good project for us actually. We learned a lot about python and gained the intuition of how the cpu works by implementing it. We used regex in some places and it was a good practice to use it. Even if there were times that we were stressed, we were happy during our implementation process. Thank you for this project.

What Could be the Improvements?

Multiplication and division operations could be added.