Hasan Bingölbali – 2019 400 276

Osman Fehmi Albayrak – 2018 400 198

We are asked to implement an assembler and the execution simulator of an imaginary cpu namely cpu230. Cpu230 consists of SF, CF, ZF flags and PC, A, B, C, D, E, S registers. Firstly, we must implement an assembler of the given program formatted as an asm file. Then we are free to move onto the next stage and implement execution simulation of this hypothetical cpu.

In this cpu each instruction has fixed length of 3 bytes. Instructions are given in the following format: 6 bits opcode – 2 bits addressing mode – 16 bits operand. At the very beginning of our implementation we created a function, instructions(), that takes 3 inputs (opcode, addressing mode, operand) and returns 3 bytes instructions in hex format. Then we created two dictionaries: dict_forInstructions, dict_forRegister. The first one is used for matching instructions with its hex code, and the second one is used for matching register with their bit patterns. After these preparations we were ready to assemble the given program. We took input file, cleaned unnecessary blank spaces. We kept track of program counter and determined immediate value of uninitialized variables. Finally, by using instructions function and those 2 dictionaries we assembled asm file and outputted a bin file.

Before dealing with a bin file, we first implemented functions for most of the given instructions. In each function we processed operands and checked them to set effected flags from the process. We implemented adding function for ADD and INC, substraction function for SUB and DEC, xorFunc for XOR, andFunc for AND, orFunc for OR, notFunc2 for NOT, shiftLeft function for SHL, shiftRight function for SHR instructions. Then again we created a function for instructions, instructions(), but this time we extracted opcode, addressing mode and operand instead of combine them together. We created 2 dictionaries: dict_Register and Memory. In dict_Register we matched register bit patterns with the current values of it, in the Memory we kept memory addresses and their current values. We created a deque structure from collections library and used it as the stack of our cpu. We created an output file formatted as a txt file. We initialized ZF, SF and CF with False value and started to execute given bin file. For each instruction we created an if block and process them according to their proper addressing modes. At this point processing was easy because we already implemented most of the instructions as functions, so we used them.

We could improve our code by using classes for some common structures. There were many common functions, and dictionaries in cpu230assemble.py and cpu230exec.py files. Declaring them in a common source could save us some time and make our code cleaner. We implemented our project in python 3.8.10 and faced some compatibility issues with 3.6.9. It took some time to solve this problem. We should have used the same version at the beginning.

In our opinions, this project granted us a detailed understanding about how cpu works, what does an assembler do, how does an execution process work. Also, collaborating with a friend gained us a good experience of teamwork.