Spring 2024 - CSCI 6461 Team 5 Project

Design Notes

- By Swarup Totloor, Yashita Pobbareddy, Ravi Gavade

Overview

We approached this Part 0 of the project by building the assembler as a class. This meant that whenever an assembler object is created, it would take the input filename in the constructor. We then define all necessary and helper functions as methods of this class.

The Opcodes are loaded from a predefined file named InstructionList.txt, into a HashMap for lookup. `OpCode`, `InputInstructions`, `ListingFileContent` and `LoadingFileContent` are private variables unique to the Assembler object.

We realised it would be better to not process instructions based on the type of instruction, but rather the number of operands in the provided instruction. With the availability of bit level control, it becomes easier to handle each case depending on the number of operands.

Some of the key functions that needed forethought are mentioned below -

`Assembler(String inputFilename)`

- This is the constructor for the Assembler class. It creates the `Opcode` HashMap used to lookup the Op Codes for a given instruction, as well as reads and stores the input instructions.

`void Assembler.Do()`

- This function handles the entire processing of all instructions and creating of the listing and loading files content. It is relatively straightforward, and simply iterates through each line, incrementing or changing the program counter.
- The output of every instruction fully processed is then added to the HashMaps of the listing file and loading file respectively.

`String Assembler.ProcessInstruction(String[] inst)`

- Since we have bit level manipulation available, and many different types of instructions have similar bit layout, it becomes convenient to write all the cases to be handled.
- We used a simple `switch` case to handle all the instructions, where all instructions having the same number of operands are allowed to fall through and be handled at once. If there is are additional steps to be done, they are handled separately.
- This function is also set to private, as it makes no sense for an outside object to make this call, other than the assembler object itself.

`String FormatNumberString(String s, int n)`

- A convenient method to format numbers to meet a specified length by trailing 0s. Since this formatting is needed many times between conversions, we decided to make it easier to read and implement.

`ArrayList<String[]> ReadInput(String filename)`

- This method is used to read a given input file and split it into lines and words for easier processing in the future.
- This method is static and private.

`void CreateListingAndLoadingFiles()`

- As the name suggests, we generate the `listing.txt` and `loading.txt` files using the HashMaps used to store the output earlier.