Computer Architecture: Spring 2023

Phase 2: Assembler

Team C-2

Team members

Name	ID
Ahmed Ameen	1190071
Ali Hassan	2200011
Hazem Montasser	2200003
Yousef Mahmoud Gilany	4200342

Assembler for the RISC like ISA

This Python code provides an assembler for our RISC-like ISA. The assembler takes an input file containing assembly code and generates binary code that can be loaded into a memory module in ModelSim. The output can be in either binary or hexadecimal format.

How to use

The assembler can be run from the command line using the following command:

```
python assembler.py input_file output_file [-b] [-x] [-m memory_size]
```

Example input

```
foo@bar:~$ python assembler.py code.asm output.mem -x -m 512
```

- **input_file**: The name of the input file containing assembly code.
- **output_file**: The name of the output file where the binary code will be written.
- **-b or --binary**: Optional flag to output in binary format.
- **-x or --hexadecimal**: Optional flag to output in hexadecimal format.
- **-m or --memory**: Optional flag to specify the size of memory in words (1 word = 16 bits). The default value is **512**.

Code Structure

The code is structured as follows

- Assembler class: The main class that contains the methods for assembling instructions and generating binary code.
    - register_indices: A dictionary that maps register names to their corresponding binary values.
    - ALU_funct: A dictionary that maps ALU operations to their corresponding binary values.
    - opcodes: A dictionary that maps opcodes to their corresponding binary values.
    - getOpcode method: This method takes an instruction as input and returns its corresponding opcode in binary format.
    - assemble method: This method reads the input file, assembles the instructions, and writes the binary code to the output file.

How it works

The assembler reads the input file and processes each line of assembly code one by one. It uses the getOpcode method to determine the opcode for each instruction and generates the corresponding binary code. The binary code for each instruction is stored in a dictionary with the memory address as the key. Finally, the binary code is written to the output file in the specified format.

The getOpcode method first splits the instruction into its constituent parts and removes any commas. It then uses the opcodes dictionary to determine the opcode for the instruction. If the instruction is a register-based instruction, it uses the register_indices dictionary to determine the binary values for the registers. If the instruction is a memory-based instruction, it converts the address or data value to binary format. Finally, it returns the complete opcode for the instruction in binary format.