Assignment

Thoughts and Ideas on the Big Picture:

The fundamental purpose of a MIPS assembler is to transform human-readable MIPS assembly code into machine code that can be executed on a MIPS architecture. Here are the fundamental concepts and processes at work:

Tokenization and reading:

1. Line by line, read the supplied MIPS assembly code file.
2. To extract specific components like labels, opcodes, registers, and immediate values, tokenize each line.

Identification of Segments:

1. Identify and separate the.data and.text code parts.
2. Concentrate on the.text portion for instructions.

Label Production:

1. Identify and store labels with their respective addresses in the first pass.
2. To store these labels and their addresses, create a data structure (e.g., a symbol table or dictionary).

4. Directions Recognition:

1. Recognize the kind of each instruction (R, I, or J) on the second pass.
2. To identify instructions, look for certain keywords or patterns in the assembly code.

Instructions for Assembling:

1. Convert assembly instructions into machine code using MIPS instruction formats, depending on the instruction type.
2. Handle R-type instructions (e.g., registers, shamt, funct), I-type instructions (e.g., op, rs, rt, immediate), and J-type instructions (e.g., op, address).

6. Handling Errors:

1. Put in place error detection and reporting tools to capture syntax mistakes and invalid commands.

7. Machine Code Output:

1. Save the resulting machine code to a file for later use.

8. Verification and testing:

1. Create test cases with the supplied assembly code files (testfile.asm) and compare the assembler's output (output.txt) to the expected results.
2. To automate the testing process, implement a testing module.

Documentation: 9

1. Create detailed documentation that explains how the assembler works.
2. High-level design principles, data structures, and detailed implementation should all be documented.

10. (Optional) User Interface:

1. Create an easy-to-use command-line interface for user engagement, allowing users to enter the assembly file and receive the machine code file.

Workflow and Corresponding Assistant Interpretation:

Input for reading:

1. To comprehend its components, the assembler analyses the input assembly code line by line and tokenizes each line.
2. It is possible to identify labels, opcodes, registers, and immediate values.

Label Processing (First Pass):

1. The assembler recognizes labels and addresses in the first run. A symbol table is used to store them.

Identification of Segments:

1. The assembler recognizes and divides.data and.text segments.

Recognition of Instructions (Pass 2):

1. It recognizes the kind of instruction (R, I, J) in the second pass.
2. Instructions are classified using certain phrases or patterns.

Instructions for Assembling:

1. The assembler creates machine code in MIPS instruction formats (R, I, J) based on the instruction type.

Handling Errors:

1. Syntax problems and invalid instructions are recognized, and the user receives error messages.

Verification and testing:

1. With input files, the assembler is tested, and the resulting machine code is compared to the predicted results.

Machine Code Output:

1. The resulting machine code is saved to a file.

Documentation:

1. To explain the assembler's architecture, data structures, and implementation, extensive documentation is provided.

User Interface (Optional):

1. A user interface module, which is optional, allows users to interact with the assembler by providing input and retrieving output.

Workflow:

Preparation (preprocessing):

1. Each line is cleaned by removing comments and trimming whitespace.

Process Labels (findLabelAddress):

1. In the first run, it identifies labels (that terminate in a colon).
2. The labels and their addresses are saved in the labels array.

Parsing Instructions (assembleR/I/JInstruction):

1. Keywords distinguish instruction kinds (R, I, J).
2. Lines are tokenized in order to extract important components such as registers, immediate values, and labels.
3. Initializes the machine code and then assembles it.

Instructions for Assembling R (assembleRInstruction):

1. Encodes opcode, registers, and function code to handle R-type instructions (add, sub).
2. Produces machine code in the desired format.

Instructions for Assembling I (assembleIInstruction):

1. Formats opcode, registers, and immediate values for I-type instructions (lw, sw).
2. Creates machine code in the format specified.

Instructions for Assembling J (assembleJInstruction):

1. Encodes opcode and addresses to manage jump instructions (j, jal).
2. Using the label table, resolves label names into their appropriate addresses.

Handling Errors:

1. Unknown or unsupported commands are detected and reported.
2. When problems occur, error messages are displayed.

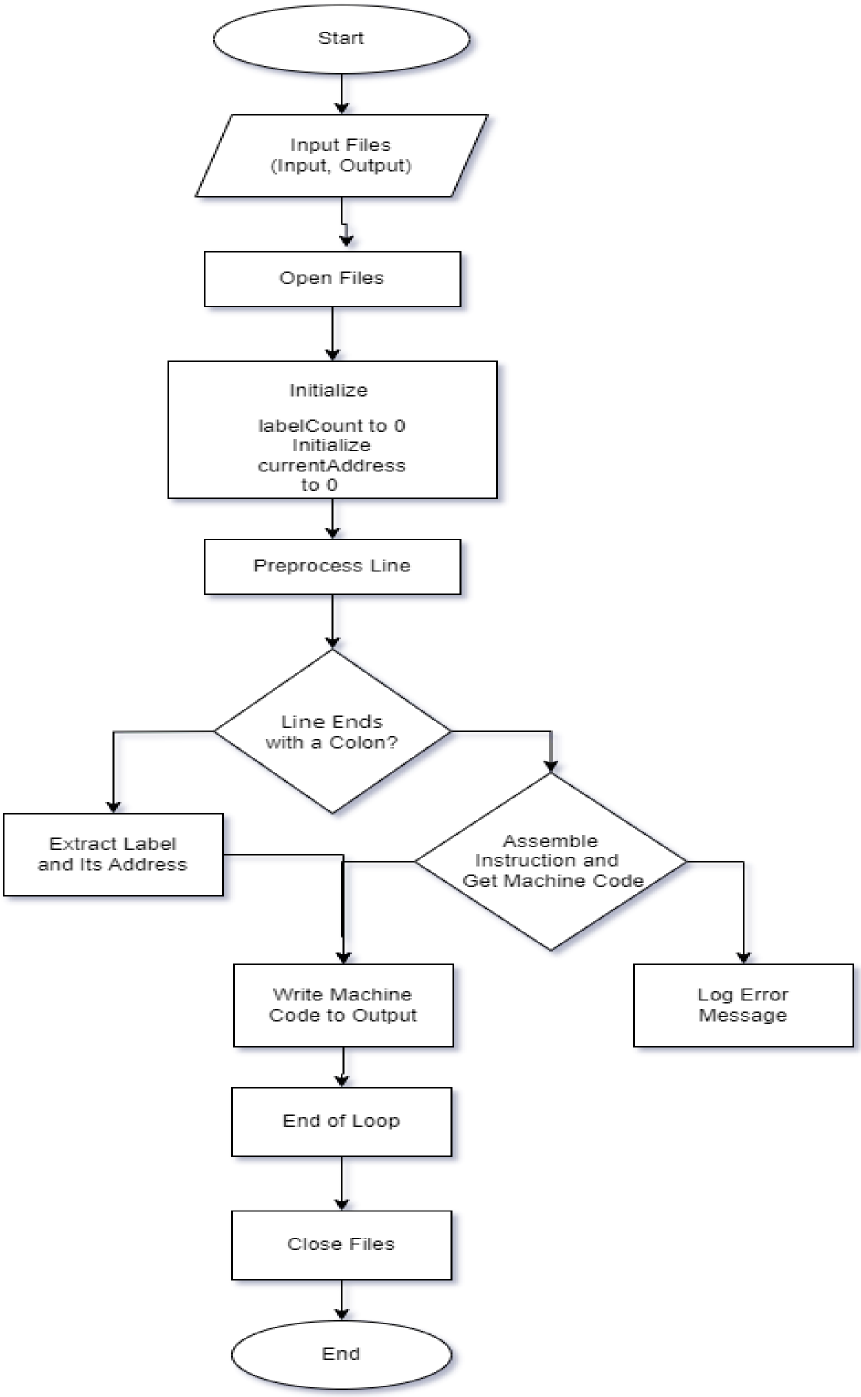
I/O to files (main function):

1. The command-line options are used to open the input and output files.
2. Lines from the input file are processed.
3. The resulting machine code is written to the output file.
4. While processing, it keeps track of the current address.

File Closure and Post-Processing:

1. Handles any post-processing tasks, such as addressing in the.text section and reporting errors.
2. Ensures that input and output files are properly closed.

Flow Chart:


Start
Input Files (Input, Output)
Open Files
Initialize
labelCount to 0
Initialize
currentAddress
to 0
Preprocess Line
Line Ends with a Colon?
Extract Label and Its Address
Assemble Instruction and Get Machine Code
Write Machine Code to Output
Log Error Message
End of Loop
Close Files
End