Contribution Report: Rashmitha

I designed the blueprint for the new linker program and modified the header of the structure file.

1. **File Created: Linker_design.md**
Total Lines Written: 171

Line Numbers: 8-179

This document is the foundational plan for the entire linker.

Lines 8-15 (High-Level Overview): This section defines the linker's core purpose: to perform Symbol Resolution, Relocation, and Section Merging, directly referencing the goals outlined in the main specification document.

Lines 19-35 (Input: .o Format): Specified the exact binary layout of the relocatable object file. This is the contract between the assembler and the linker. It defines the STAO magic number and the 20-byte header containing the sizes of the four subsequent sections. This detailed specification is what my teammates used to implement the assembler's emitter and the linker's parser.

Lines 37-48 (Output: .vm Format): This section defines the final output format, including the STAK magic number and a 16-byte header that the Virtual Machine team will use for loading the program.

Lines 52-90 (Core Linking Algorithm): Broke down the complex process of linking into a clear, five-step algorithm:

Parse Arguments: Read command-line inputs.

Read Object Files: Parse all .o files into memory.

Build Global Symbol Table: This crucial step details how to merge symbols from all files, calculate their final memory addresses, and detect "duplicate symbol definition" errors.

Perform Relocation: This step explains the "patching" process, where placeholder addresses in the code are overwritten with the final addresses calculated in the previous step.

Write Executable: This final step describes how to assemble the final .vm file from the merged and patched sections.

Lines 126-179 (Linker Data Structures): This section contains C++ code defining the primary data structures for the linker.

ObjectFileSymbol (Lines 138-158): Defines how a symbol from a .o file will be represented in memory, including its name, type (TEXT/DATA), binding (LOCAL/GLOBAL), and its address relative to its section.

ParsedObjectFile (Lines 164-179): This class acts as a container for all the information parsed from a single .o file. It includes a static from_file method, which sets the stage for the parsing logic implemented by other team members.

Contribution Report: Nishchith

Role: Object File Parser (Code & Header)
My work spanned both the assembler and the new linker. I upgraded the assembler's parser to understand modern directives and wrote the main entry point and high-level parsing logic for the new linker.

1. File Modified: CSD_MIPS_Assembler/src/parser.cpp
Total Lines of Code: 141
Lines Added/Modified: 25-141 (significant rewrite)

Code Explanation
Overhauled the parser to support sections and directives.

CurrentSection enum (Line 28): A state variable was introduced to track whether the parser is currently inside the .text or .data section.
Pass 1 Logic (Lines 36-80): The parser was rewritten into a two-pass system. This first pass identifies all symbol definitions (: for labels, .static for data) and builds a preliminary symbol table.
Directive Handling (Lines 43-62): The parser now checks for lines starting with .. It handles .text, .data, and .static to correctly assign addresses. It also collects all .global symbols into a list (globals_to_process) to be dealt with after all symbols are found.
Global Symbol Processing (Lines 83-88): After the first pass, the parser iterates through the collected global symbol names, finds them in the symbol table, and flags their binding as GLOBAL.
Pass 2 Logic (Lines 91-140): The second pass reads the file again. This time, it parses the actual instructions (now that all symbols are known) and the values for .static data entries, populating the AssemblyUnit structure.

2. File Created: CSD_MIPS_Linker/src/linker.cpp
Total Lines Written: 39
Line Numbers: 1-39
Code Explanation
This file is the main entry point for the new linker program.

Argument Parsing (Lines 10-18): The code parses the command line to separate the output file name from the list of input .o files.
Parsing Loop (Lines 22-28): This is the high-level control flow. The code iterates through each input file path provided by the user.
ParsedObjectFile::from_file() (Line 26): For each path, it calls the static method that triggers the complete parsing of one .o file. This call orchestrates the parsing logic implemented by Kowshik.
Module 3 Goal (Lines 31-32): The code successfully completes the goal of Module 3, which is to parse the files. It prints a message indicating that the next steps will happen in future modules.

‘

Contribution Report: Kowshik

Role: Object File Parser (Data Section)

My work also spanned both projects. He defined the new data structures and rewrote the assembler's back-end to produce the .o format. He then implemented the low-level binary parsing logic for the new linker.

1. File Modified: CSD_MIPS_Assembler/src/emitter.cpp
Total Lines of Code: 149

Lines Added/Modified: This file was almost a complete rewrite.

Code Explanation
I transformed the emitter from a simple bytecode generator into a sophisticated object file constructor.

emit() Logic for Jmp and Invoke (Lines 80-116): The emit methods for instructions that use labels were made much smarter.

They now look up the symbol in the AssemblyUnit's symbol table.

If the symbol is LOCAL, its known address is written directly into the bytecode.

If the symbol is GLOBAL, the emitter writes a placeholder address (0) and creates a RelocationEntry, signaling to the linker that this spot needs to be patched.

emit_object_file() (Lines 120-179): This is the main function that constructs the .o file.

Section Generation (Lines 123-161): It generates each of the four main sections (code, data, symbol table, relocation table) in memory as separate std::vector<uint8_t>. The symbol table and relocation table are serialized according to the format Rashmitha designed.

Header and Stitching (Lines 164-179): It creates the 20-byte header, writing the STAO magic number and the calculated sizes of each section. Finally, it concatenates the header and the four section vectors into one final byte vector, which is the complete .o file.

2. File Created: CSD_MIPS_Linker/src/object_parser.cpp
Total Lines Written: 83

Line Numbers: 1-83

Code Explanation
This file contains the low-level logic for reading the binary .o file format.

Binary Helper Functions (Lines 11-28):

read_int32(): Reads exactly 4 bytes from the file and interprets them as a little-endian integer.

read_string(): Implements a length-prefixed string reading protocol. It first reads a 4-byte integer for the length, then reads that many bytes for the string content. This matches the output format produced by the assembler's emitter.

ParsedObjectFile::from_file() (Lines 33-82): This function implements the parsing for all sections of a .o file.

Header Parsing (Lines 42-48): It reads the 20-byte header, validates the magic number, and stores the section sizes.

Section Parsing (Lines 51-60): It reads the code and data sections as raw blocks of bytes into their respective vectors.

Symbol Table Parsing (Lines 63-73): It reads the number of symbols, then enters a loop. In each iteration, it reads a symbol's name, type, binding, and address, constructing an ObjectFileSymbol object.

Relocation Table Parsing (Lines 76-82): It reads the number of relocation entries, then loops to read each entry's offset and target symbol name.