Text

Description automatically generated

**Project-2 Report on**

**System Programming**

**(BTECCE22504)**

**Submitted to Vishwakarma University, Pune**

**By**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roll No.** | **SRN No.** | **Name** | **Div** |
| 20 | 31232438 | Pratik Manoj Dharam | H |

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2024-2025**

**Design and Implementation of Assembler/Macro Processor**

**Project Statement :**

Develop a program to build the linker data structures necessary for the assembly program in Project 1. The program should accept a link origin from the user and create RELOCTAB (Relocation Table) and LINKTAB (Link Table).

**Project Objective :**

 **Understanding Assembly Language**:

* To gain a comprehensive understanding of assembly language syntax, semantics, and how it translates into machine code.

 **Implementing a Two-Pass Assembler**:

* To design and implement a two-pass assembler that can read assembly language source code and generate the corresponding machine code along with relevant tables (symbol, relocation, and link tables).

 **Table Generation**:

* To create a symbol table that maps labels to memory addresses, facilitating easy reference during code execution.
* To develop a relocation table that specifies how addresses in the generated machine code should be modified when the program is loaded into memory.
* To produce a link table indicating the types of variables used (e.g., packed decimal) and their respective addresses.

 **Memory Management**:

* To explore and implement memory management techniques for defining storage locations for variables and instructions.

 **Error Handling**:

* To implement error detection and handling mechanisms for invalid syntax or undefined labels in the assembly code.

 **Simulation of Execution**:

* To simulate the execution of the generated machine code and verify its correctness against expected outcomes.

**Project Outcome:**

 **Successful Assembly Code Compilation**:

* The project will demonstrate the ability to convert assembly language code into machine code accurately, reflecting a thorough understanding of assembly language concepts.

 **Generation of Comprehensive Tables**:

* A complete symbol table will be produced, mapping labels to their respective addresses, enabling efficient reference and identification of variables and instructions in the code.
* A relocation table will be generated, detailing how addresses in the machine code should be adjusted based on the program's memory load address, ensuring the correct execution of instructions.
* A link table will be created, listing global variables and their types (e.g., packed decimal), which provides information on how the data should be processed during execution.

 **Error Detection and Handling**:

* Implementation of error handling mechanisms will lead to improved robustness, allowing the assembler to gracefully report syntax errors and undefined labels, enhancing the overall user experience.

 **Improved Memory Management**:

* The project will showcase effective memory management strategies, demonstrating how to allocate and track memory addresses for variables and instructions dynamically.

 **Understanding of Assembly Language Execution**:

* The project will provide insights into the execution process of assembly language, helping to bridge the gap between high-level programming and low-level machine code execution.

 **Documentation of the Development Process**:

* Comprehensive documentation will be created, detailing the design, implementation, and testing phases, along with explanations of the generated tables and code structures, serving as a reference for future projects.

 **Enhanced Skills in C Programming**:

* The project will result in improved proficiency in C programming, particularly in the areas of file handling, data structures, and algorithm implementation.

**Methodology/Theory:**

 **Understanding Assembly Language**:

* Begin with a comprehensive review of assembly language concepts, focusing on the syntax, structure, and instructions used in the assembly code. Understanding how assembly language maps to machine code is critical for the development of the assembler.

 **Designing the Two-Pass Assembler**:

* **Pass 1**: The primary goal is to analyze the source code to identify labels, symbols, and literals while generating a symbol table and determining the memory locations for each instruction.
  + **Label Collection**: Traverse the assembly code to collect all labels and their corresponding addresses.
  + **Location Counter**: Maintain a location counter to assign addresses to instructions and data. Increment the counter appropriately based on the type of instructions encountered (e.g., DC, DS, or regular instructions).
  + **Symbol Table Generation**: Construct a symbol table that maps each label to its address for later reference.
* **Pass 2**: The second pass converts the assembly instructions into machine code, utilizing the information gathered in Pass 1.
  + **Machine Code Generation**: Replace labels in the assembly code with their corresponding addresses from the symbol table, generating the final machine code.
  + **Relocation and Link Tables**: Identify which addresses need adjustment and create relocation and link tables to document these changes, particularly for global variables and externally defined labels.

 **File Handling**:

* Implement file I/O operations to read the assembly source code and write the output to various files, including the machine code, symbol table, relocation table, and link table. Ensure robust error handling for file operations.

 **Data Structures**:

* Utilize appropriate data structures (e.g., arrays, structs) to store instructions, labels, and addresses efficiently. This organization is essential for quick lookups and modifications during the assembly process.

 **Testing and Validation**:

* Develop a series of test cases with varying assembly codes to validate the functionality of the assembler. Check the output for accuracy in terms of machine code generation, symbol table completeness, and correct entries in the relocation and link tables.
* Conduct thorough debugging to ensure that any errors in the assembly code are properly handled and reported.

 **Documentation**:

* Document the entire development process, including design choices, code explanations, and the rationale behind the implementation decisions. This documentation will serve as a reference for future projects and help in understanding the workings of the assembler.

 **Final Review and Presentation**:

* Prepare a comprehensive report summarizing the project objectives, methodology, outcomes, and conclusions. Include diagrams or flowcharts where necessary to illustrate the assembler's workflow and logic.

**Implementation:**

#include <stdio.h>

#include <string.h>

typedef struct {

char label[10];

int address;

} Symbol;

void generateRelocationAndLinkTables(Symbol symbols[], int symbolCount) {

FILE \*relocationTableOutput = fopen("relocation\_table.txt", "w");

FILE \*linkTableOutput = fopen("link\_table.txt", "w");

if (!relocationTableOutput || !linkTableOutput) {

perror("Error creating output files");

return;

}

// Write Relocation Table Header

fprintf(relocationTableOutput, "Label\tNew Address\n");

// Update addresses and write to the relocation table

for (int i = 0; i < symbolCount; i++) {

int newAddress = symbols[i].address + 100; // Example relocation offset of 100

if (strcmp(symbols[i].label, "LOOP") == 0 ||

strcmp(symbols[i].label, "INNER") == 0 ||

strcmp(symbols[i].label, "SKIP") == 0 ||

strcmp(symbols[i].label, "ENDLOOP") == 0) {

fprintf(relocationTableOutput, "%s\t%d\n", symbols[i].label, newAddress);

}

}

// Write Link Table Header (Only include ARR as the globally defined variable)

fprintf(linkTableOutput, "Label\tType\tNew Address\n");

for (int i = 0; i < symbolCount; i++) {

if (strcmp(symbols[i].label, "ARR") == 0) {

fprintf(linkTableOutput, "%s\tPD\t%d\n", symbols[i].label, symbols[i].address + 100);

}

}

fclose(relocationTableOutput);

fclose(linkTableOutput);

}

int main() {

Symbol symbols[] = {

{"LOOP", 104},

{"INNER", 110},

{"SKIP", 118},

{"ENDLOOP", 128},

{"ONE", 200},

{"ZERO", 201},

{"N", 202},

{"COUNT", 203},

{"ARR", 204}, // ARR is the global variable to be added to the link table

};

int symbolCount = sizeof(symbols) / sizeof(symbols[0]);

generateRelocationAndLinkTables(symbols, symbolCount);

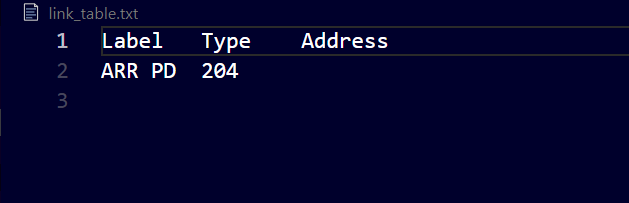
printf("Relocation table and link table generated.\n");

return 0;

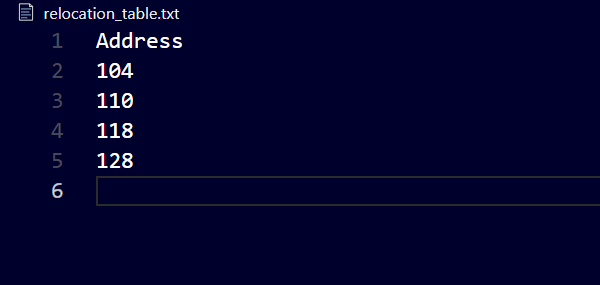
}

**Output:**

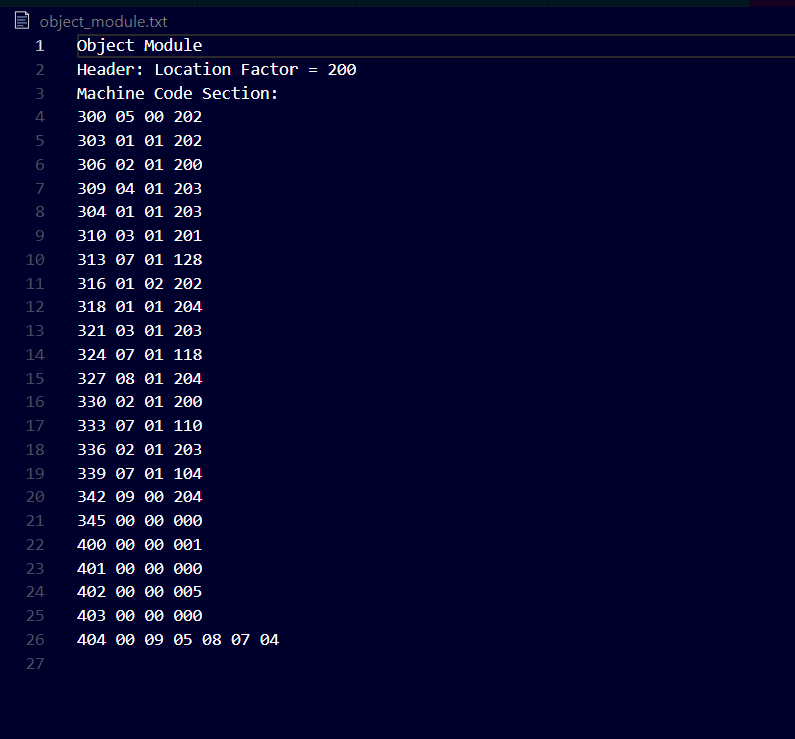
Linktab Table



Relocation Table



**Object Module**



**Conclusion:**

1. **Understanding Assembly Language**: Enhanced knowledge of assembly language and its relationship with machine code, crucial for systems programming.
2. **Programming Skills**: Improved C programming skills through the design and implementation of the assembler, showcasing effective data management and file handling.
3. **Table Generation**: Highlighted the importance of symbol, relocation, and link tables in the assembly process, aiding memory management during execution.
4. **Debugging and Validation**: Validated the assembler's robustness through testing, emphasizing the significance of debugging in software development.
5. **Documentation**: Reinforced the value of thorough documentation for future reference and effective learning.

In summary, the project achieved its technical goals and provided a rich learning experience in low-level programming and software development practices.