資訊三丁D1072931 蔡明偉

**SIC/XE Assembler Algorithm Process**

1. Initial Setup

* Initializes the operation table, symbol table and register table with data from external files.
* Reads source code from input files.

1. Pass 1 (Symbol Collection)

* Parses each line of source code, separating labels, opcodes, and operands.
* Calculates and updates (Location Counter) based on instruction or directive type (such as RESW, RESB, WORD, BYTE).
* Constructs SYMTAB by storing labels and associated memory locations.
* Stores instructions and related data in an “intermediate” (variable in code) structure.

1. Pass 2 (Object Code Processing)

* Process each instruction in “intermediate”, converting the instruction into object code.
* Uses format2, format3, and format4 functions to process instructions according to their format in SIC/XE.
* Handles special cases such as BASE, BYTE instructions, and operand handling.
* Stores the object code associated with instructions in “intermediate”.

1. Output

* Create intermediate files, symbol files, and output files.
* Intermediate files contain source code representations with addresses and object codes.
* The output file contains headers, text, and modification notes according to standard SIC/XE format.

1. Memory Management and Registration

* Manage memory allocation via LOCCTR.
* Manage registration and use of labels.
* Process format 2, 3, and 4 instructions by paying attention to flags and register usage.

1. Processing Special Instructions

* Processing format 4 instructions by considering the use of extended format and relocation markers.

1. Management of BYTE and WORD Instructions

* Convert BYTE and WORD data to binary or hexadecimal representation according to specifications.

**Reflection**

1. Deep Understanding of SIC/XE Architecture

Building this assembler required a broad understanding of how SIC/XE machines work, including handling different instruction formats, address calculations, and register usage. This process emphasizes the importance of deep architectural knowledge in low-level software development.

2. Recognition of Coding Complexity

This project opened my eyes to the complexity of creating software that interacts directly with hardware. Each function, from format2 to format4, must be carefully designed to ensure that the resulting object code complies with the SIC/XE specification.

3. Experience in Memory Management

Managing LOCCTR and symbol tables provides valuable experience in memory management. This illustrates how efficient memory allocation and references are key in creating system software.

4. Expertise in Data Parsing and Interpretation

Reading and parsing input from a file, then translating it into executable instructions, is a complex process. This shows how important skills are in interpreting and manipulating data in various formats.

5. Value Detail and Accuracy

The slightest error in the assembler can cause errors in the entire program. This project increased my appreciation for the need for attention to detail and accuracy in programming.

6. Development of Problem Solving Abilities

Encountering and solving these challenges in assembler construction strengthened my problem-solving skills, especially in the context of low-level programming and systems development.

7. Awareness of the Importance of Documentation and Structured Code

This project highlights how important it is to have good documentation and well-structured code, which not only makes debugging easier but also allows others to understand and maintain the code more easily.