SIC / XE Assembler

Name: AYUSH RANJAN
Enrolment Number: 22114018
Email: ayush_r@cs.iitr.ac.in
Computer science and engineering
Bachelor of Technology, 2nd year
IIT Roorkee

SIC/XE Assembler User Guide

The assembler supports literals, expressions and program blocks assembly along with normal assembly of SIC/XE instructions.

This guide provides instructions on how to use the SIC/XE assembler program.

Before you begin:

1. Ensure you have the following folders in the current directory:
   I. data: it contains the following files:
      - input.txt
      - intermediate.txt
      - listing_file.txt
      - error.txt
      - object_program.txt
      - SYMTAB.txt
      - BLOCKTAB.txt

   II. src: it contains the following files:
      - assembler.cpp
      - pass1.h
      - pass2.h
      - functions.h
2. Open the "input.txt" file present in the "data" folder and write your SIC-XE assembly program in it. Ensure that program is written in all upper case and there should not be any leading spaces in each line.

Running the Assembler:

1. Open a terminal in the "src" directory containing the "assembler.cpp" file.
2. Compile the "assembler.cpp" file using a command like `g++ assembler.cpp -o assembler` (replace `g++` with your C++ compiler if needed).
3. Execute the generated executable file produced (e.g., `./assembler.exe`).

4. The “assembler.cpp” file will get executed, the status of assembly will be printed in the command line of the terminal and the output files will be filled according to the result of the program.

Output Files:

The program will generate three output files and some intermediate files:

Output files:

- `object_program.txt`: stores the final object program of the assembly.
- `listing_file.txt`: displays the original assembly code with additional information like line number, addresses (Location counter), program block numbers, instruction, operand and object code for each line.
- `SYMTAB.txt`: contains the symbol table i.e. the symbols generated during the first pass of the assembler.
- `BLOCKTAB.txt:` contains the program block table information generated during the first pass of the assembler.

Intermediate files:

- `error.txt:` contains all the errors occurred during pass1 and pass2 of the assembler along with the line number and reason for the error.
- `intermediate.txt`: contains the data which is read by pass2 of the assembler, it is generated during pass1.

Internal Data Structures

The assembler utilizes several data structures to manage symbols, opcodes, and other information. These include:

- **LITTAB (Literal Table):** stores information about literals (constant values) in the program.
- **BLOCKTABLE (Block Table):** Tracks program blocks and their information which includes block_name, block number, block length, starting address, and block location counter (which stores current value of location counter of each block in pass 1).
- **SYMTAB (Symbol Table):** Maintains a record of all symbols defined in the program contains label, block number, value, information regarding relative or absolute label.
- **OPTAB (Opcode Table):** provides definitions for supported opcodes (instructions) with opcode value and their formats.

- **assembler_directives (a set of Assembler Directives):** contains recognized assembler directives.
- **TEXT_RECORDS:** Array of text records used during object program generation in the pass2 of assembler with each text record having fields initial, start address, length, object_code and object_code_length.

Global Variables:

program_name: store the name of the program.

program_length: store the total program length.

Pass1:

- **pass1_input:** input file
- **pass1_outpit:** intermediate file
- **pass1_error:** error file
- **line**: each line read from the input file is stored in this string for processing.
- **label, opcode, operand:** contains label, instruction/assembler directive, operand name in each line.
- **LOCCTR:** stores the current location counter for current block.
- **ERROR_FLAG_PASS1:** this error flag will be true if any error is encountered in pass1
- **current_block_no:** stores the current block number in which instructions are being assembled.
- **current_block_name:** stores the current block name in which instructions are being assembled.
- **total_blocks:** contains the total number of different blocks encountered.
- **line_no:** stores the user program currently assembled line number.
- **star_count:** stores the number of places where * is used in the operand.

Pass2:

- **pass2_input:** intermediate file
- **pass2_list:** listing file
- **pass2_object:** object_program file
- **pass2_error:** error file
- **ERROR_FLAG_PASS2:** this flag will set true if any error encountered in pass2
- **line:** contains each line as read from intermediate file
- **line_no, label, opcode, operand:** stores the current line number, labels, instruction or assembler directive and operand as each line is processesed.

- **locctr, program_block_no:** stores the location counter and program block number as each line is processesed.
- **BASE_REGISTER_VALUE:** stores the base register value to be used in Base relative addressing.
- **BASE_RELATIVE_ADDRESSING:** Boolean flag to check if Base relative addressing is enabled or not.
- **current_text_record:** stores the current text record.
- **current_object_code:** contains the object code to be added in text record.
- **prev_block_no:** stores the program block number of the previous instruction.
- **MODIFICATION_RECORDS:** stores all modification records.

Assembler Functionality:

Main Functions:

- `main(): this function in “assembler.cpp” calls function pass1() present in file "pass1.h” and pass2() present in file “pass2.h” Also calls function write_tables() in present in “functions.h” file which fills the symbol and block tables.`
- `pass1()`:
  - Processes the program file line by line.
  - Generates a symbol table and Block table containing information about all the symbols and blocks.
  - Creates an intermediate file to store data for the second pass.
  - Lists all the errors encountered during the assembly process in the “error.txt” file.
- `pass2()`:
  - Reads the intermediate file generated in pass1.
  - Generates a listing file containing the original assembly code with additional information like addresses, block numbers, and object code.
  - Creates an object program file containing machine code instructions in various record formats (H, T, M, E).
  - Lists all the errors encountered during the assembly process in the “error.txt” file.

Source Code Files:

- `functions.h`: Implements the data structures mentioned earlier and also contains various utility functions for pass1(), pass2() and main() functions.
- `assembler.cpp`: calls the main program logic functions, including `pass1()` and `pass2()` functions.