# Made by:

seif El-deen Ahmed Elsaied 202200495

Omar Samir 202201677

hesham elsaid 202200323

Mahmoud ashry 201901241

# SIC Assembler Development Project Report

## Objective

Develop a two-pass assembler for the Simplified Instructional Computer (SIC) that translates assembly language programs into object code. This project combines theoretical concepts with practical implementation, enabling a deeper understanding of assembly language programming.

## Project Description

The assembler has been implemented using Java and follows a two-pass approach:

### Pass 1:

1. Reading SIC Code: Parses the assembly code from the input file.  
2. Address Assignment: Uses a Location Counter (LOCCTR) to assign addresses to labels and instructions.  
3. Intermediate File Creation: Outputs a file containing parsed code and assigned addresses.  
4. Symbol Table Creation: Generates a symbol table mapping labels to their addresses.  
5. Op Table Loading: Loads operation codes from a file into a map for reference.

### Pass 2:

1. Reading Intermediate File: Processes the output of Pass 1.  
2. Object Code Generation: Converts instructions into object code using the symbol and operation tables.  
3. Handling Literals and Variables: Correctly processes data like literals, WORD, RESW, and BYTE.  
4. Output File Generation: Produces the final object program and related files.

## Code Implementation

The assembler was implemented in Java. Key highlights include:  
  
- File Paths: All input and output file paths are parameterized for flexibility.  
- OPTAB and SYMTAB: Maps are used for the operation table (OPTAB) and symbol table (SYMTAB).  
- Error Handling: Handles invalid labels, duplicate symbols, and unsupported instructions gracefully.  
- Output:  
 - Intermediate file.  
 - Symbol table.  
 - Object program.

## Pseudo Code

### Main Program

Start Program:  
 Load the operation table (OPTAB)  
 Perform Pass 1:  
 - Generate intermediate file  
 - Create symbol table (SYMTAB)  
 Perform Pass 2:  
 - Generate object code  
 - Write output files  
 Print "Execution completed successfully!"  
End Program

### Load Operation Table

Method loadOpTable:  
 Open OPTAB file  
 For each line in the OPTAB file:  
 Split line into instruction and opcode  
 Add to OPTAB map  
 Close file  
End Method

### Pass 1: Generate Intermediate File and Symbol Table

Method pass1:  
 Open input SIC code file  
 Open intermediate file for writing  
 Open symbol table file for writing  
  
 Initialize Location Counter (LOCCTR) to 0  
  
 Read first line from SIC file  
 If it contains START directive:  
 Set LOCCTR to the starting address  
 Write the line with address to intermediate file  
 Read next line  
  
 While not end of SIC file:  
 If line is empty or a comment:  
 Skip to next line  
 Else:  
 Split line into LABEL, OPCODE, and OPERAND  
 If LABEL is present:  
 If LABEL already in SYMTAB:  
 Throw error for duplicate symbol  
 Else:  
 Add LABEL with LOCCTR to SYMTAB  
 Write line with address to intermediate file  
 Update LOCCTR based on instruction size  
 Read next line  
  
 Write SYMTAB to symbol table file  
 Close all files  
End Method

### Pass 2: Generate Object Program

Method pass2:  
 Open intermediate file for reading  
 Open output file for writing  
 Open used operation table file for writing  
  
 Initialize Header Record and Text Record  
 Initialize first executable address  
  
 Write column headers to output file  
  
 While not end of intermediate file:  
 Read intermediate line  
 Split line into address, label, opcode, and operand  
 If opcode is START:  
 Set Header Record  
 Set first executable address  
 Else:  
 Generate object code for opcode and operand  
 Add object code to Text Record  
 Write formatted line to output file  
  
 Write Text Record to output file  
 Write End Record with first executable address  
 Write used opcodes to OPTAB file  
 Close all files  
End Method

### Generate Object Code

Method generateObjectCode(opcode, operand):  
 If opcode exists in OPTAB:  
 Get opcode from OPTAB  
 If operand exists in SYMTAB:  
 Append address of operand from SYMTAB to opcode  
 Else:  
 Append "0000" to opcode  
 Return full object code  
 Else If opcode is BYTE:  
 If operand starts with 'C', convert characters to hex  
 If operand starts with 'X', return hex value  
 Else If opcode is WORD:  
 Convert operand to 6-digit hexadecimal  
 Return empty string for unsupported opcode  
End Method

## Deliverables

1. Source Code: Fully implemented in Java.  
2. Documentation: Includes this report and detailed comments in the source code.  
3. Output Files:  
 - Intermediate file with parsed SIC code.  
 - Symbol table.  
 - Object program file.  
4. Test Cases: SIC programs used to validate the assembler.

## Evaluation Criteria

- Correctness: Produces accurate object code.  
- Code Quality: Clean, modular, and well-documented code.  
- Completeness: Generates all specified output files.  
- Error Handling: Identifies and reports invalid inputs.

## Conclusion

The project successfully implemented a two-pass SIC assembler in Java. The assembler demonstrates a comprehensive understanding of assembly language processing and file-based I/O operations.