CSE112 Computer Organisation | Winter Semester 20**20**
DOCUMENTATION

ASSEMBLER PROJECT
2019346- YATHARTH TANEJA.
2019376- PRAKHAR PRASAD.

General Overview

Assembly language is any low-level programming language in which there is a very strong correspondence between the instructions in the language and the architecture's machine code instructions. Because assembly depends on the machine code instructions, every assembler has its own assembly language which is designed for exactly one specific computer architecture.

Assembly code is converted into executable machine code by a utility program referred to as an assembler.[1] The conversion process is referred to as assembly, as in assembling the source code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a certain type of processor.

We have designed a two pass assembler, which also addresses the problem of forward referencing.

Project Overview

To design an assembler that takes a text file as an input containing assembly code. And translates the assembly code to machine code. The machine code is stored in a separate text file.

Assumptions and limitations:

1) Declaration of variables has been omitted.
2) The default value of START is **0** if not provided.
3) Directives **“START”** and **“ END”** must be specified.
4) All literal must be alpha-numeric and be of the format:’<whole_numbers>’.
5) The length of each symbol and literal has been assumed as one word length which is **12 bits.**
6) The format of label declaration and definition is: <label_name>, which must be followed.
7) All the operations, symbols, literals and addresses must be space-separated.
8) The instruction length is **12 bits**, and the address space is **8 bits.**

9) All operands are declared at the end of instructions, sequentially and in order of their appearance in the code
10) Only single-line comments are supported and it is assumed that those lines will start with **(“//”).**
11) Macros, subroutines, and procedures **have not been dealt with.**
12) There should be **no** statements after end (not even blank statements).

About the project:
The project has 3 files labelled 2019346_2019376_code.py, input.txt, output.txt.
The main program is python3 based which reads the input file line by line which contains your assembly code. And after a successful compilation, it stores the machine code in output.txt.

Working :
The code initializes the program counter and location counter globally as zero and creates empty symbol, literal and opcode table.

The **first pass** is **executed.**

The file is read line by line and determines what type of statement is given.

There are 3 types of statements:-
Imperative statements: The program segregates between a literal, symbol (variable), label and inserts literals and literal table and symbols and labels in symbol table but doesn’t allocate memory.
Declarative statements(Declaration of labels): During label declaration the assembler allocates memory address to label declared with respect to the value of ILC .
Assembler Directives: The program searches for multiple and missing START and End statements and throws an error for the same.

Then intermediate code is generated i.e the symbol table the literal table and opcode table are generated.

Next **second pass** is **executed** which converts the intermediate code to machine code.

Errors Handled :

START missing.
When the assembly directive START is not present in the code.

Parameter Error.

Too many parameters supplied.

Too few parameters supplied.

Multiple END.
When the END directive is present multiple times in the code.

Missing END.
When the directive END is not present in the code.

Multiple START directives.
When the START directive is present multiple times in the code.

Instructions after END directive.
When the END statement is further followed by assembly instructions.

The label has already been declared.

Label declaration error.

Label given but not defined.

Opcodes cannot be used as an Identifier.

The variable identifier has been declared as a Label.
Since both, label names and variable names can be alpha-numeric, SUB L1 and BRN L1, both can be valid, to avoid this, an error is thrown if the variable has already been declared as a label.

Label Identifier has been declared as a Variable.
Since both, label names and variable names can be alpha-numeric, SUB L1 and BRN L1, both can be valid, to avoid this, an error is thrown if the label has already been declared as a variable.

Literals must be numeric.

Address overflow.

The maximum memory size is 256 words, which can be stored using 8 bit addresses. If the program size is greater than this, the assembler will throw an error and terminate.

NOTE :

1. THE INPUT SHOULD FOLLOW SIMILAR FORMAT AND MUST HAVE START AND END.
2. Since we have assumed that the address is 8 bits and 4 bits are of opcode address overflow will take place once ILC gets greater than 255 (11111111).
3. All three files should be in the same location.
4. If you want to see the intermediate code uncomment the print statements

[1] - Wikipedia