DOCUMENTATION

ASSEMBLER

Project By :
Shruti Jha (2019274)
Akshat Wadhwa (2019231)

This document contains information about the Assembler. We have implemented a two-pass assembler.

1) About the Assembler:

This project is the implementation of a two-pass assembler. A basic two pass assembler involves the conversion of a given assembly code to its corresponding machine code. This is a 12 - bit assembler, hence the output machine code will have 12 bits.

1.1) *Key Features:*

- This assembler handles labels and their referencing in the provided assembly code.
- It also allocates virtual addresses to the provided labels and variables/symbols used in the assembly code.

1.2) *Machine Requirements:*

- 12 bit accumulator architecture.
- Should have at least 2^8 bit memory.

1.3) *Workflow:*

- First Pass:
    a) We initiate a variable named location counter to traverse throughout the raw assembly code line by line. It increments its value by one after each iteration.

b) We scan the lines from a text file using a buffered reader, and read the next line in each provided iteration.
c) Firstly, we look out for the type of instruction and the number of parameters passed with it. We also throw required errors in just the first scanning of the question.
d) Then, we look out for labels in the given instruction. We have defined a particular format of labels(mentioned later in this documentation).
e) We seperate the label and store it in the label table if it does not have an error.
f) Next, we check for the opcode in the line. If it is a valid opcode, the argument passed with the opcode is stored in the symbol table. Necessary errors are thrown if present.
g) When the label is used in the branch statement, it is stored in the label table with the value of -1. If it is defined in the code anywhere, then the value of the label in the label table is replaced with the value of the location counter of the line where the definition occurs.
Similarly, the symbols are stored in the symbol table with the value of -2. If it is defined later in the code (*assumption*: *a symbol is considered defined when it is used with INP opcode later anywhere in the input file*), then the value of the symbol in the symbol table is updated with the value of the location counter where the definition occurs.
h) If after pass 1, any symbol in the symbol table or any label in the label table has a value of -1 or -2 respectively stored against it, an error is thrown that the symbol or label is used but not defined.
i) If any error is reported, the compiling is stopped and the user is asked to remove the error and try again
j) Hence, output file is created only if there are no errors in pass 1.
k) If there is no error in the code, another table is made for designing an input to pass 2.

- <u>Second Pass:</u>
    a) This pass is the final step of the process, in which the conversion of the assembly code to machine code takes place.
    b) The table which has the input for pass 2 is read line by line and machine code generated.
    c) The label calls are replaced by the value of location counter at label definitions and variables are replaced by their storage addresses (which is the value of the location counter at the INP statement for the given symbol), both provided by the location counter during the first pass.

2) Instruction Set:

2.1) *OPcode Table*:

Assembly OPcode	Machine OPcode	Function	No. of operands
CLA	0000	CLEAR THE ACCUMULATOR	0
LAC	0001	LOAD TO ACCUMULATOR	1
SAC	0010	STORE VALUE FROM ACCUMULATOR TO ADDRESS	1
ADD	0011	ADD ADDRESS CONTENTS TO ACCUMULATOR	1
SUB	0100	SUBTRACT ADDRESS CONTENTS FROM ACCUMULATOR	1
BRZ	0101	BRANCH TO LABEL IF ACC = 0	1
BRN	0110	BRANCH TO LABEL IF ACC < 0	1
BRP	0111	BRANCH TO LABEL IF ACC > 0	1
INP	1000	TAKE INPUT FROM TERMINAL	1
DSP	1001	DISPLAY VALUE OF ADDRESS ON TERMINAL	1
MUL	1010	MULTIPLY ACCUMULATOR AND ADDRESS CONTENTS	1
DIV	1011	DIVIDE ACCUMULATOR CONTENTS BY ADDRESS CONTENT	1
STP	1100	STOP EXECUTION OF THE COMMAND	0

2.2) Assembler Directives:

a) START:

First line of the Program
Specifies a specific memory address to load the program at (StartAddress)
Generates error if used in any other line other than line 1
Syntax: START StartAddress.

b) STP:

Last line of the Program
Stops translation of the program beyond this point
Syntax: STP

3) General Instruction Format:

3.1) Syntax:

<Label Name> OPcode < Operand(s)> <Comment>

3.2) Label name:

Optional in an instruction.
Cannot be an opcode
Cannot contain spaces
Label name should be followed by a hash "#"

3.3) OPcode:

Should be exactly the same as given in the opcode table
Number of operands should be equal to the number specified in the table

3.4) Operands:

3.4.1) Variable Names:

Should begin with a letter only (cannot start with digits)
Cannot consist of spaces
Cannot be Opcodes
Cannot be labels

3.4.2) Memory addresses:

Value can range from 0 to 255

3.5) Comments:

Optional in an instruction
Only single line comments supported
Comment should begin with “/*”

4) Providing the input file

A text file consisting of Assembly language program saved in the same directory as the assembler program with the name given below:
File Name: “input.txt”

5) Output file format

A text file containing the Machine Language program corresponding to the input assembly program is created in the same directory as the Assembler program.
File Name: “output.txt”
The first 4 bits of each instruction constitute the opcode and the rest 8 bits define the address of the operand.

6) Errors Handled

Errors abort the execution of the program after the first pass is complete. The user is required to rectify the error in order to translate their program successfully.

The assembler can handle the following errors:

1) Argument passed with START is not an integer.
2) Arguments passed are either more or not sufficient to complete the operation.
3) Label not defined, but used in a branch statement.
4) Label defined more than once.
5) Symbol does not start with a letter.
6) Symbol used but not defined in the code.

7) Warnings

Warnings are displayed at points where the assembler detects a possible error in the user's logic like a missing STP instruction in the input code. It should not be seen as an error as the output code file is successfully created.

The assembler can throw following warnings:

1) STP statement not present in code.
2) Symbol defined but not used in code. Thus memory space is wasted.