**ASSEMBLER PROJECT**

Pratik Pathak-2017255

Yash Kalyani-2017273

**FILES USED**

The main program file is 2017255\_2017273\_Project1.py

The files used in the program execution will be:

* **input.txt**

Contains the assembly code for execution.

* **tables.py**

Stores the opcode required for the accumulator architecture and variable declaration

* **errors.txt**

Stores all the errors encountered during the first pass and second pass of the assembler operation

* **symbol\_table\_file.txt**

Stores the symbol table created which includes all the operands and their corresponding memory address

* **literal\_table\_file.txt**

Stores the literal table created which includes all the distinct literals used in the program along with their values.

* **label\_table\_file.txt**

Store the label table created which includes all labels and variables declared in the assembly code along with their memory address

* **external\_reference\_table.txt**

Stores the extern table created in the program

* **entry\_point\_table.txt**

Stores the entry point table created in the program

* **value\_table\_file.txt**

Stores the values of the variables declared using the DC opcode

and the relocationfile.txt which s

* **relocationfile.txt**

Stores the offset value of the relocation counter

* **machinecode.txt**

Stores the machine code generated by the assembler using the assembly code provided

ASSEMBLER CODE WORKING

The offset value from the relocation dictionary is taken and the instruction location counter is initialised by that value. After this the instructions are read serially.

First, it is checked that the instruction is a comment or not. If the entire line is a comment, then that instruction is skipped, and the next instruction is read, else the comment part of the instruction is ignored. Comments in the assembly code for this program should start with '//', and only single line comments are allowed in the program. Multiline comments are not allowed. If comments are added to the instructions of the assembly code, then should be at the end of the assembly code.

After this the program checks for the valid opcode in the instruction. If an opcode is present in the instruction, the program checks for suitable operands to the given opcode. If insufficient operands are provided, then it reports an error and stores it in the file. After this the program checks for STP opcode. If it is not present in the input file, it throws an STP error. After this, the program reads the remaining instruction and checks for labels, literals or symbols. If any label is present in the instruction, it stores it in a label\_table along with the memory address of the label. Similarly, it stores operand is present in the instruction, it stores it in the symbol\_table.

It also checks for literals in the program. If a literal is found, it stores the value of the literal in a memory address and stores the address in the literal\_table. Note that in this program, literals are defined using the ‘=1’ syntax. Functions are defined to check for each of them in the instruction. Apart from these, if a variable is declared using the DC opcode, it stores the variable name along with its value in the value\_table.

If a valid opcode is not present in the line, the line is checked whether it is “Global Main”, “EXTERN“ instruction or “Public” declaration. For using EXTERN and Public, only one operand should be provided. Upon encountering the EXTERN command, we insert the operand in the external\_reference\_table. When the Public command is encountered, the operand is inserted in the entry\_point\_table. If the instruction does not contain any suitable operation, it throws an invalid opcode exception and stores it in the error file.

The program also checks for multiple declarations of a symbol and reports the same if encountered. During the entire processing of the instructions, it increments the value of the location counter accordingly.

After executing the instructions, it checks if a symbol is undefined in the label\_table. If yes, then it throws an error and stores the symbol name in the error file and insert it in the label\_table. The program also checks whether any label or symbol has the same name as that of any opcode, in which case, it prints an error. It is still acknowledged as a variable and stored as a symbol/label. Finally, it stores the values of the tables in the corresponding table file.

One important thing to note is that the program handles certain errors in the assembly code input and fix them. These errors include multiple declaration of the label, no declaration of a used symbol, wrong label name etc. But it cannot handle errors such as insufficient or too many operands for a particular opcode, memory limit exceeded which is restricted to 256 bits etc. In this case the machine code is not generated and an error line is displayed. If no such errors occur, the machine level code for the given input is displayed and also stored in the machinecode.txt file. The machine code output consists of 2 to 3 columns in which the first column is the address of that particular instruction, second column is the opcode number and third is the address of the operand if necessary. The first column of the output is not a part of the machine code file. The instructions which were used for declaring a word are not shown in the code but are rather assigned a memory space.

ERROR HANDLING

The following errors are handled in the assembler code:

* **Memory limit exceeded**
* **Invalid instruction with length greater than 3**
* **Insufficient or too may operands for an opcode**
* **Multiple declarations of a symbol**
* **Invalid EXTERN command**
* **EXTERN command used without defining Global Main**
* **Invalid PUBLIC command**
* **No opcode exists**
* **Symbol is not defined but used in the code**
* **Invalid label name**
* **No STP opcode**