**DOCUMENTATION**

The main program file is 2017241\_2017245\_Code.py

The files used in the program execution are named aptly and functionality of the same are explained in concise below:

For storage of different tables files namely symbol,label,literal,external reference,entry point,errors are used.

**ASSUMPTIONS,LIMITATIONS and CAVEAT**

Following are the assumptions that are made and are crucial to the execution to the code

Comments Assumptions

1. NO MULTI LINE comments are allowed.

2.If comments are added to an instructions they should be at the end of the line.

3.If entire line is comment ie starts with a comment then it is skipped.

User should not expect //comment1 ADD J to work.

**Flow of Execution Assumptions**

LITERAL

1.Literals are defined using the ‘=1’ syntax

2.If a literal is found, it stores the value of the literal in a memory address and stores the address in the literal\_table.

LABEL

1.If any label is present in the instruction, it stores in a label\_table along with the memory location.

SPECIAL SYMBOLS

DC,DS et. al

1. If a variable is declared using the DC opcode, it stores the variable name along with its value in the value\_table.Along with it, it regularly increases the value of the location counter after executing the instruction.

**LIMITATIONS**

1.Global, Public ,Main and Extern

Note to User: If a programmer/user want to treat these Keywords then the only possibility in the flow of execution where they might reside is the first keyword they might encounter therefore after checkifopcodeexists check , they could be handled ie temp[0]==”Global”/”Public” and temp[1]==”Main”

2.After executing the instructions, it checks if a symbol is undefined in the label\_table.

IF YES,

1.Throws an error and stores the symbol name in the error file.

2.We still insert it in the respective label table for output purpose.

For name clashes reporting with OPCODE reporting is done but storage is not affected as they are analogous to warnings. It is well understood by the programmers of this code that such clashes are often dangerous as they directly affect the concept of reserved keywords.

**CAVEAT**

1.Serializing of object has been done in the code so as to provide future functionality(large system should provide a last visit to things up until things were working). Users are advised to deserialize the files for their purpose. \_pickle library is used for the same/  
2.Tables to be used in program are supplied in the code. For modularity purpose users can create an inputtables.py file and then uncomment the first line.

**FLOW OF EXECUTION**

The instructions are read line by line and executed as follows:

First, it is verified that the instruction is a comment or not. Assumptions 1,2 and 3 of COMMENTS are followed and else part is read if there is no occurence of comment.

After this the program checks for the valid opcode in the instruction.

IF NO ERROR THEN

Program checks for suitable operands to the given opcode.

IF INSUFFICIENT OPERANDS

It reports an error and stores it in the file.

IF ERROR

Error is stored in the error.txt. file.

After this the program checks for STP opcode.

IF NOT PRESENT IN INPUT FILE

Throws an STP error.

At this point Labels, Literals and Symbols are checked and stored in respective tables

Functions are defined to check for each of them in the instruction.

Apart from these,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.

We now print the machine level code for the given input.

**OUTPUT FORMAT**

The first column of the output denotes the address assigned to the instruction, which isn’t a part of the machine code file.

The second column denotes the opcode.

The last column includes the operand’s address.

The instructions which were used for declaring a word are not shown in the code but are rather assigned a memory space.