***DOCUMENTATION:ASSEMBLER***

Team Members-AYUSH MAHANT(2019353)

N SUDEEP REDDY(2019313)

Group No.- 9(B. TECH FIRST YEAR IIITD)

**INTRODUCTION:**

We have been given the project to make an assembler(converting the assembly language code to its corresponding Machine Code).We work on memory address mode.

**LANGUAGE USED:**

We have used Python 3.7.4 for developing the assembler because of the following reasons-

1. Python does not have a complex syntax.It is well handled.
2. We needed to store the data in dictionaries and lists which were easy to be used in python.
3. Python is useful in String manipulation which was important in our assembler.

**ASSUMPTIONS:**

1.The commentsstart with ‘#’.

2.The label is defined before ‘:’ and the opcode is in the same instruction as the label.

3.’STP’ is only considered at the end of the instructions.

If any stp statement is present before the last one, we are not considering it.

4.We are placing the literals, symbols, and the labels, altogether in the symbol table.

5.We are assuming that the length of instructions won’t be more than 255 bits.

6. We are assuming there is no default in the declarative statements and the address is not greater than 12 bits.

**ERRORS HANDLED:**

1. No Opcode given in the instruction line.
2. Multiple opcodes given in a line.(SAC LAC A)
3. Label is incomplete.(L1: )
4. The symbol is already declared.
5. The operand can’t come before the opcodes.(A LAC)
6. More than one label or symbol provided to the opcode.(SUB A,B)
7. Error in the format-No instruction passed.
8. Same name of the symbol and opcode.(LAC LAC)
9. The label is already declared.
10. Cannot branch to a symbol.(BRZ A)
11. Cannot branch to an opcode.(BRZ SAC)
12. Cannot input an opcode.(INP LAC)
13. Cannot input a label.
14. Cannot store to accumulator another opcode.(SAC INP)
15. Cannot store to accumulator, a label.

**WORKING OF THE ASSEMBLER:**

Our assembler first reads the text file, and does the following-

The First Pass:

firstpass()-RETURN TYPE=>Dictionary

This function iterates over the text file "test.txt". Through iterating, it creates a suitable format for checking each line and then reports the errors as soon as they are caught.

While checking for errors it updates the location counter , keeps a marker on each line and updates the dictionary "symbol" for storing symbols as key and their address as value.

At the end it adds a new key into the dictionary,symbol,named "error" for counting the number of errors.We have used the “split” function of the python language and stored it in the list.We check for the comments by ‘#’.If the instruction line is not a comment, then, we check for the label and the errors occuring.

Even though it is to be assumed that the declarative statements would be at the end, it could work otherwise.

We have made the symbol table which consists of the symbols, labels, and the literals. We could not find the need of making separate tables for all. We do not have a different file for the intermediate code.Only when the symbol is declared, it is given an address otherwise it is ‘-1’.We are not checking if forward referencing has taken place because it is given that the declarative statements are given at last.

We have used three other functions for checking the opcode name,number of opcodes used in the single instruction line and the machine code of the opcode.

The storing of the address is all set with a location counter which always starts with 0.

The second pass:

second pass(symbol)-->RETURN TYPE-void

This function takes in a dictionary,assigns the binary address and makes the "output.txt" file which contains the required output as per the file "test.txt''.

It also creates "Symbol.txt" to display the Symbol table.

**TYPES OF STATEMENTS USED:**

* 1. Imperative statements- indicates an action to be performed during the execution of  
  the assembled program. Each imperative statement typically translates into one  
  machine instruction.  
  2. Declaration statements- the syntax of declaration statements is :  
  DS [Label] <constant>  
  DS [Label] <value>  
  The DS (short) for declare storage)statement reserves areas of memory and associates names with them.e.g.  
   DS A 1  
  The above statement reserves a memory area of 1 word and associates the name A with it.  
  The DC (short for declare constant) statement declares memory words containing constants.

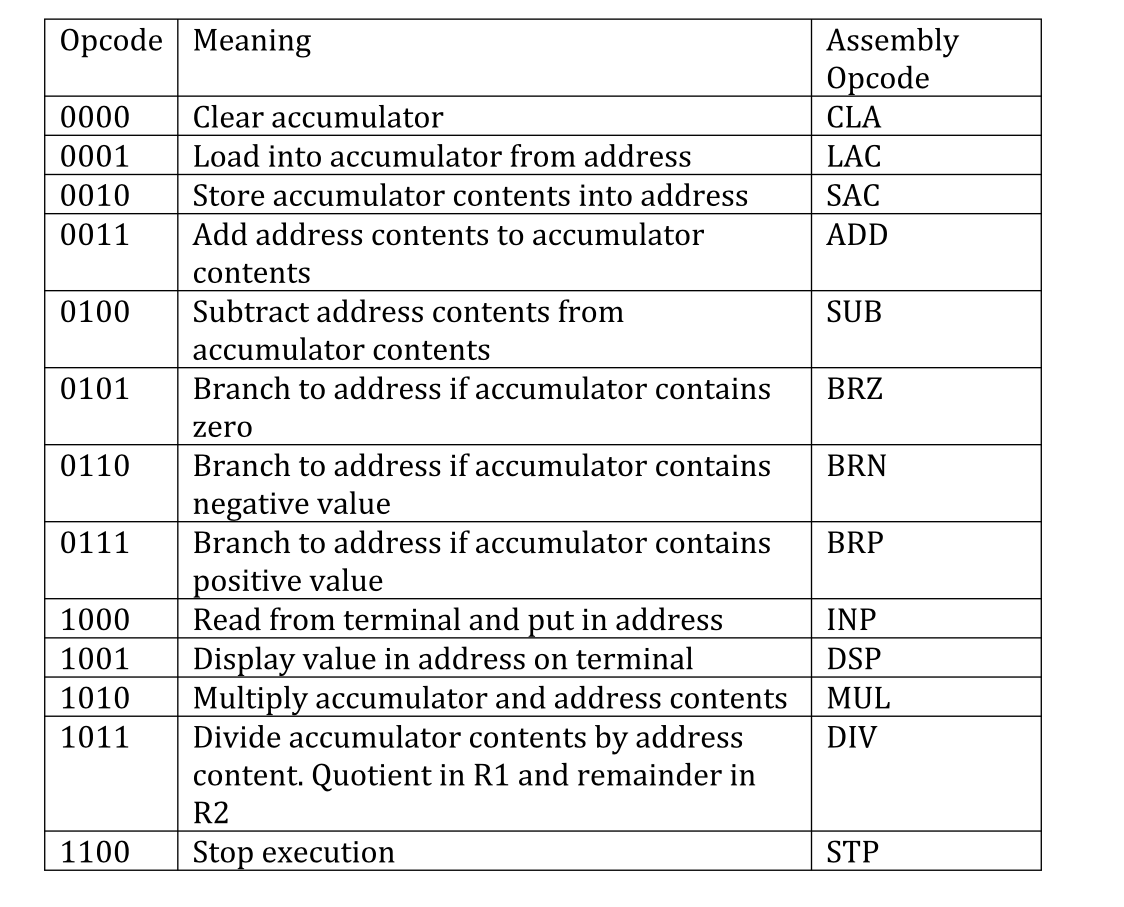
3. Assembler directives

Assembler directives are Pseudo-Instructions They provide instructions to the assembler itself.They are not translated into machine operation codes Basic assembler directives

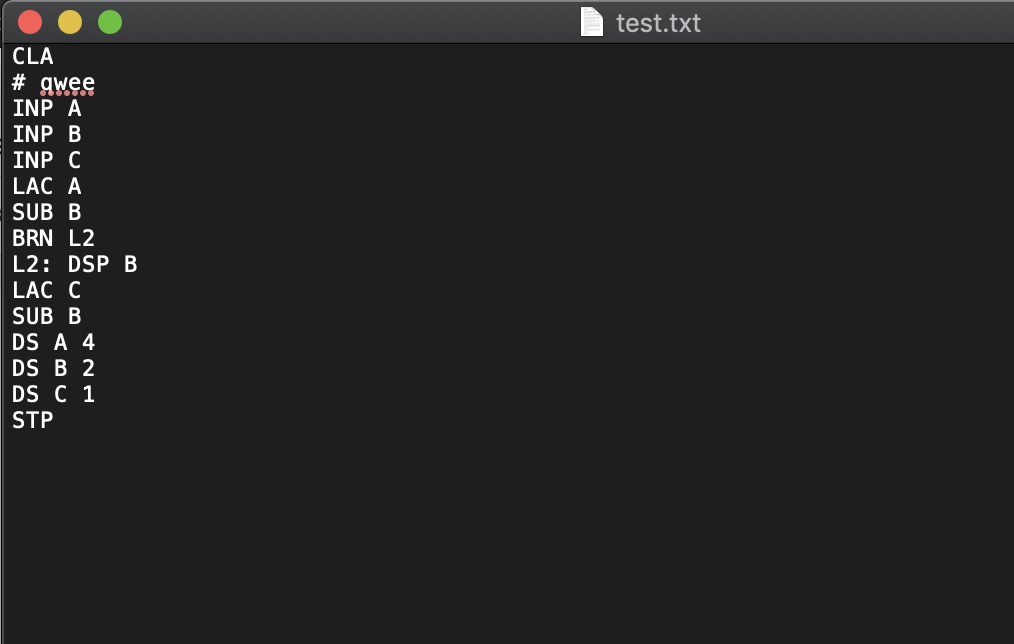
START: specify name & starting address.

END : end of source program, specify the first execution instruction

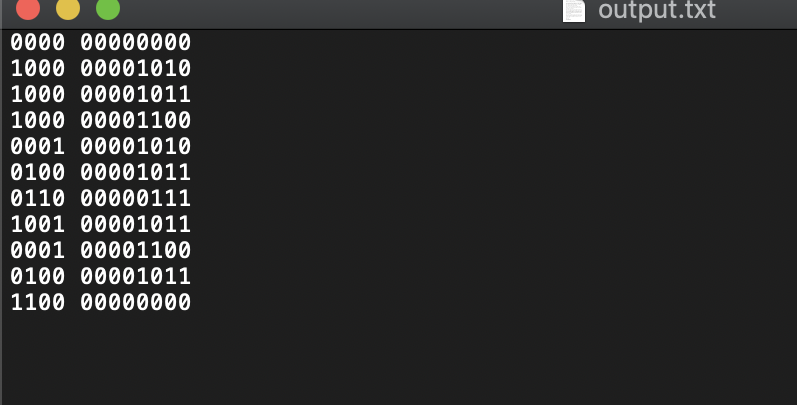
**OPCODES USED AND THEIR CORRESPONDING MACHINE CODE-**

****

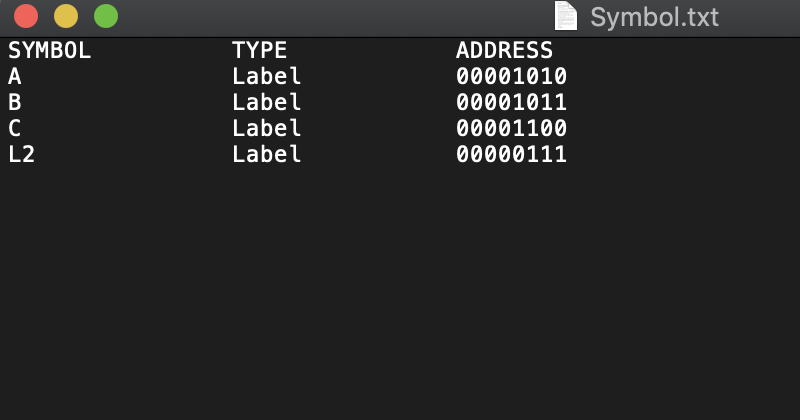
**TEST FILE:**

****

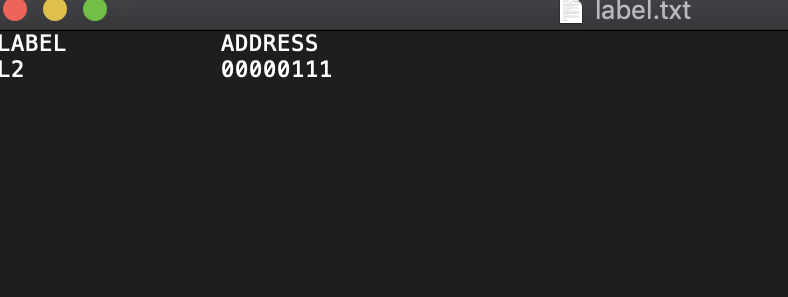
**CORRESPONDING OUTPUT:**

****

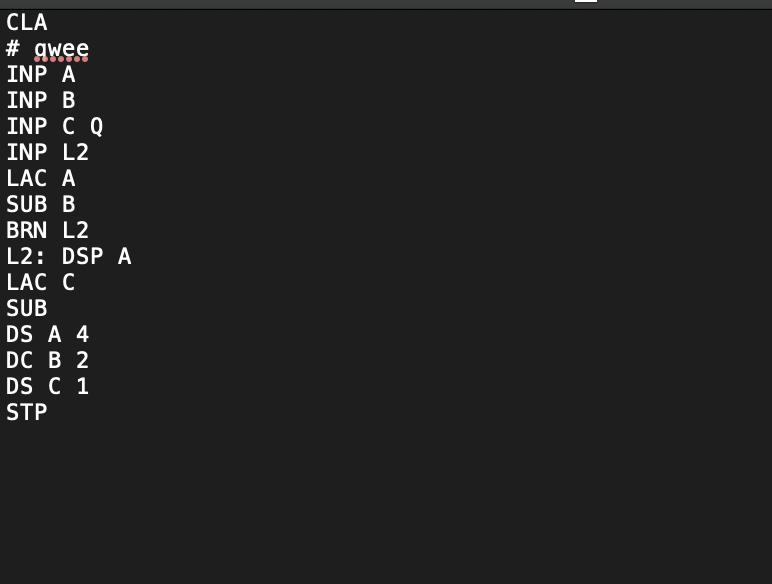
**SYMBOL TABLE:**

****

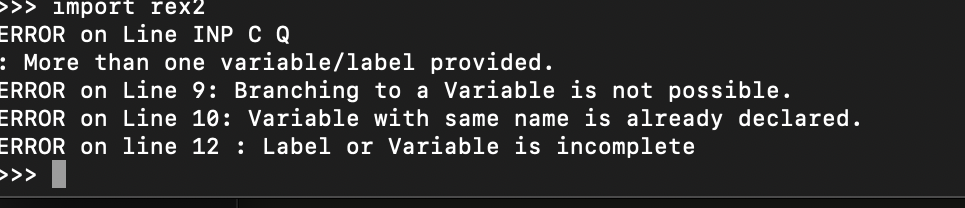
**LABEL TABLE:**

****

**TEST FILE WITH ERRORS-**

****

**OUTPUT-**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THANKYOU**