**DOCUMENTATION**

**COMPUTER ORGANISATION**

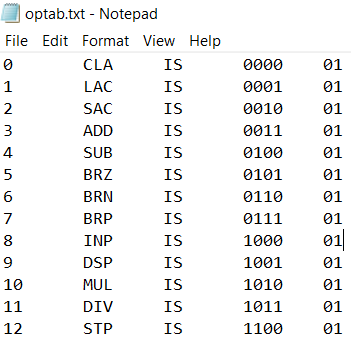
**PROJECT 1**

**(Rishita chauhan(2019383);Vasu kashyap(2019343))**

* **OPERATIONS CONSIDERED:**

FORMAT OF OPTAB:

S No. Operation Type of statement Opcode Instruction length

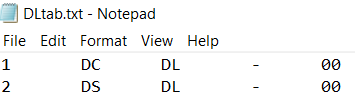


Declarative Statements used are :

* DC(Declare constant)
* DS(Declare storage)

FORMAT OF DLtab:

S No. Operation Type of statement Opcode Instruction length

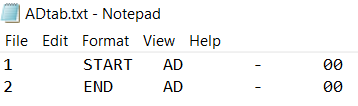


Assembly Directive Statements used are :

* START
* END

FORMAT OF ADtab:

S No. Operation Type of statement Opcode Instruction length



**REGISTERS:**

We have considered only 4 registers in our assembler

|  |  |
| --- | --- |
| NAME | ADDRESS(8 BITS) |
| R1 | 11111100 |
| R2 | 11111101 |
| R3 | 11111110 |
| R4 | 11111111 |

* We can’t assign the value of LC as any of the address value of any of the registers.
* We have taken an assumption that value of LC cannot be more than or equal to 252 as 252,253,254,255 are assigned as the address of registers and for value more than 255, address cannot be represented using 8 bits.(otherwise an error will occur)

**INSTRUCTION FORMAT:**

|  |  |
| --- | --- |
| OPCODE(4 Bits) | OPERAND ADDRESS(8 Bits) |

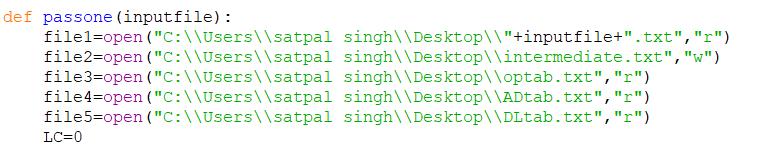
* **CODE:**

**(PASS ONE)**

****

The above screenshot is of all the classes I have made to make my code work.

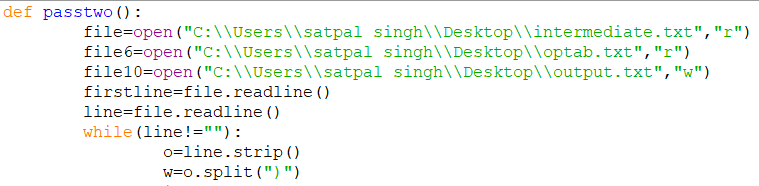
* **Symtab**: This class is used to make the symbol table .
* **Littab**: This class is used to make the literal table.
* **Registers**: This class is used to store the list of registers used in the assembler along with their addresses.
* **Legalopcodes**: class used to store the list of all the opcodes that are valid and is being used in the assembler. Used in asserting an error if the opcode used in the assembly code is illegal.
* **Operandopcodes**: class used to store the list of all the opcodes that requires atleast one operand to operate on. Used in asserting error if the opcode is not supplied with enough operands.



***\*To make this assembler work on some other system you need to change the location of the file in all the files in the code.***

1. First start reading the input file line by line.
2. Split each line into the list of string separated by spaces or tabs.
3. Set location counter(LC) to 0.
4. Checks whether there is “START” in the list. If no then raises an error.
5. Depending upon the length of the list, segregate label ,symbol,literal and opcodes and convert them into intermediate code, and keep incrementing the value of LC by 1.
6. Whenever find a label , variable, literal add them into respective tables.
7. If there is no end statement at the end then raises an error.
8. At last if there are any variables in the symbol table and literals in the literal table which are not assigned any memory location yet, then they are assigned the values after the location counter value accordingly.

**(PASS TWO)**



1. In pass two , we start reading the intermediate code file line by line.
2. Intermediate code for START statement is skipped.
3. Then we split the string from “)” to form a list of strings , and scanned each element of the list.
4. If the length of the list is 3, then that means no operand is used and if it is 4 , then it means that operand is present.



1. If the length is 3, first scan the first element of the list(which is the value of the LC) and convert it into binary number with 8 bits and write it into the output file.(it acts as the memory addresses of the instructions)
2. Then scan the second element of the list and find the S.No. of the opcode which is being used by reading the opcode table and write the binary opcode corresponding to the S.No. in the output file.



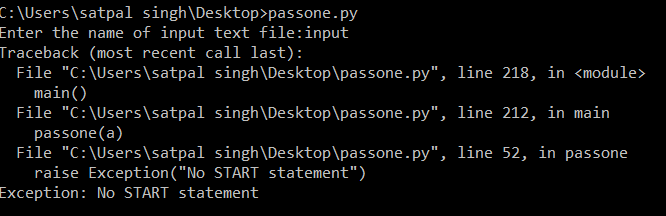
1. If there are 4 elements in the list , it means that there is an operand also.
2. So, in this case we have to scan the third element also.
3. In this element check whether the second character is “S”, “L” OR “R” as the first character will be “(“.
4. If the first character is “S” then scan the symbol table and find out which symbol is being used by the help of the fourth character and find out the memory address assigned to that symbol.
5. Then convert the address assigned to that symbol into binary number of 8 bits and write it in the output file.
6. Same happens if “L” is the first character of the string but instead of symbol table we have to traverse the literal table to find the address assigned to that specific literal.
7. And if the first character is “R” ,then it means that the operand is stored in the register.
8. So then we find which register is used by scanning the entire string and find the address assigned to that register with the help of the register class we have made and write it in the output file.

* **ERROR REPORTING:**

Errors that we have considered while making this assembler are:

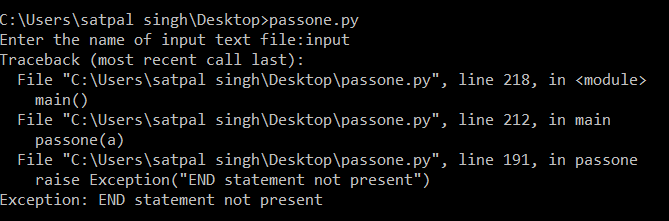
* **If START Statement is not present. The program will terminate.**

If the first assembler statement is not “START”, then the program will terminate.



* **If END statement is not present. After first pass , it will show the error.**

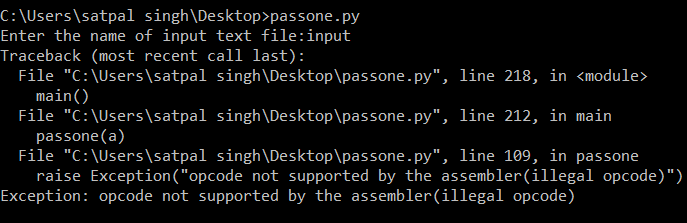
If the “END” statement is missing at the end, the assembler will throw error as follows:



* **If name in the opcode field is not a legal opcode, the program will terminate at that point.**

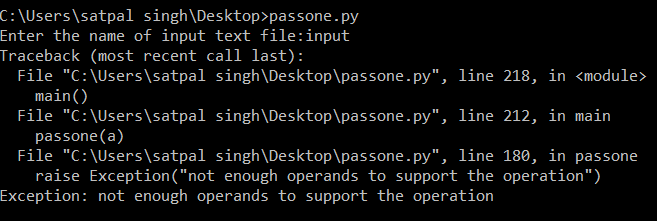
If we write some wrong opcode to perform a given operation, then the program will terminate.

Like to store accumulators content in an address , the legal opcode to be used is “SAC” but instead of that we used “STA” then the program will throw error.



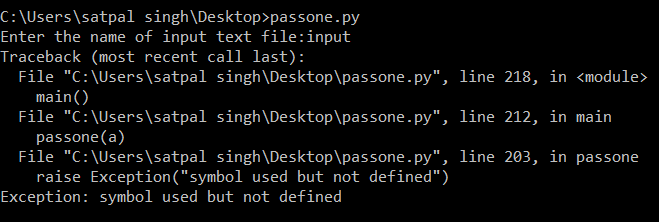
* **If opcode is not supplied with enough operand, program will throw an error.**

If the opcode is not provided by the operands , then the program will end at that very point.

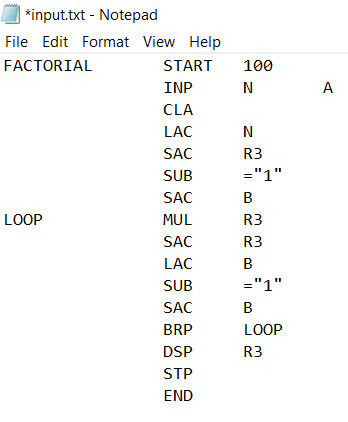


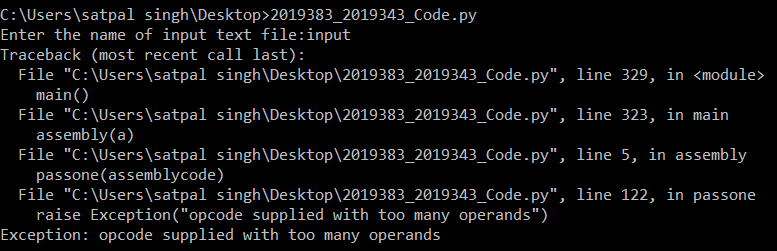
* **If a symbol(label) is used but not defined, program will throw an error.**

If suppose we are using branching opcode and jumping to a label which is not defined , then the program will throw an error.



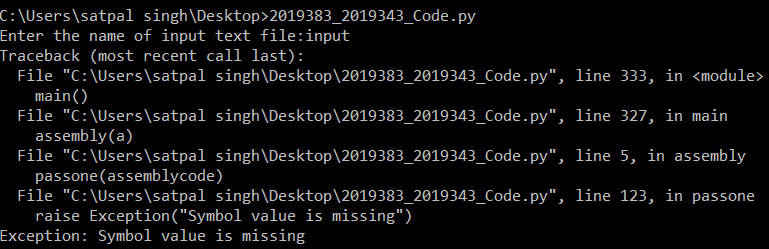
* **If operand is supplied with too many operands then the program will throw an error.**
* Suppose there is an instruction with the opcode CLA, which do not require any operand but still an operand is provided , then the program will terminate after showing an error message.
* Or another case could be that if an opcode(which require only one operand) is supplied with 2 or more operands(I have assumed that operands are separated by tab).





* **If a variable or a constant is declared using DC and DS as assembly directive but their value is not specified, then the program will raise an exception.**

Suppose there is no value written after DS or DC statement then the program will through an error stating that the “symbol value is missing”.

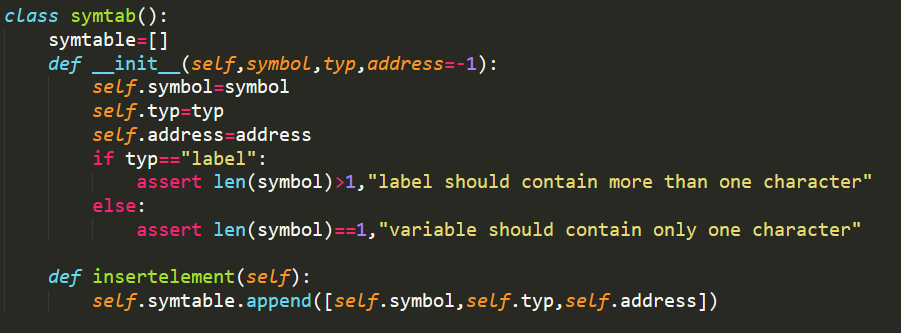


* ***We have made an assumption that variable can only be of one character like A, B, c , d, etc. Error will occur if this assumption is violated***
* ***Another assumption is that label should contain more than one character like AB, aB, Loop, etc. Error will occur if this assumption is violated.***
* **SYMBOL TABLE:**

We have used a separate class named symtab to form symbol table in the form of two dimensional array in which the elements are of the form:

[symbol name, type of symbol (label or variable) , memory address of symbol]

* (-1 is the default address assigned to all symbols)
* We have made an assumption that variable can only be of one character like A, B, c , d, etc. Error will occur if this assumption is violated
* Another assumption is that label should contain more than one character like AB, aB, Loop, etc. Error will occur if this assumption is violated.



Format of the symbol table (for the input code taken below) is:

|  |  |  |  |
| --- | --- | --- | --- |
| INDEX | SYMBOL | TYPE | ADDRESS(LC) |
| 0 | N | “variable” | 115 |
| 1 | R | “variable” | 116 |
| 2 | A | “variable” | 114 |
| 3 | B | “variable” | 117 |
| 4 | LOOP | “label” | 106 |

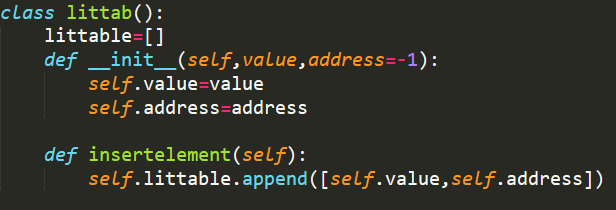
In our assembler, if there is a declarative statement for a given variable, the value of the location counter is assigned as an address to the variable and the value of the location counter is then incremented by one. As we have initially assumed that all the operands are declared after the end of instructions, when the addresses are assigned to all declared function at the end if the address of some variables are not assigned then , the value of the location counter is assigned to the variables (and LC is incremented) as addresses one by one in the symbol table in order but if the address of any symbol of type “label” is not assigned any value(that means label is used but not defined) the assembler will throw an error.

* **LITERAL TABLE:**

Like Symbol table, we have made literal table using a separate class named littab which forms a two dimensional array with the elements of the form:

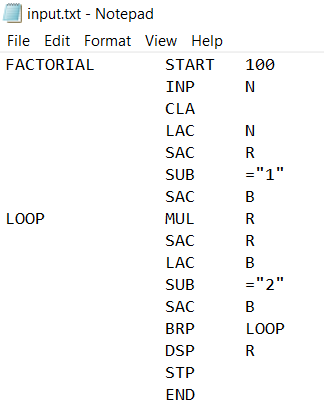
[literal value, memory address in which literal is stored]

* (-1 is the default address assigned to all literals initially)



Literals in the assembly code will be in the form =”3”, =”5”, =”6” or any other number.

Take an example of a code with literals:



Literal table for the above input code will be of format:

|  |  |  |
| --- | --- | --- |
| INDEX | LITERAL | ADDRESS |
| 0 | =”1” | 117 |
| 1 | =”2” | 118 |

* **If the value of two literals are same, still they will be assigned different memory location.**
* **INPUT FORMAT:**

Example of the assembly code for finding the factorial of number is as follows:

FACTORIAL START 100

INP N

CLA

LAC N

SAC R

SUB A

SAC B

LOOP MUL R

SAC R

LAC B

SUB A

SAC B

BRP LOOP

DSP R

STP

A DC “1”

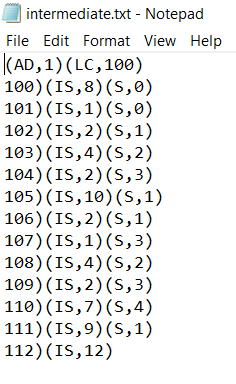
END

* **INTERMEDIATE TABLE:**
* The first line is corresponding to the start statement in the input code file, which is skipped while writing the machine code.
* The format for rest of the files are

LC)(Type of statement like IS for imperative, S.No. for that opcode in opcode table)(“S” for symbol table “L” for literal table, S.No. for the literal and symbol used as an operand)

* If register is used as an operand then after opcode ,(Name of register ) is present.
* If there is no operand used the that field is left blank both in intermediate code and output machine code.
* “END” statement is not converted into intermediate table.
* Declarative statement is not converted into intermediate table.

Corresponding intermediate table formed is as shown below in the screenshot:



* **OUTPUT FORMAT:**

The format of the output machine code of the assembler for the above example will be in the form:

01100100 1000 01110011

01100101 0000

01100110 0001 01110011

01100111 0010 01110100

01101000 0100 01110010

01101001 0010 01110101

01101010 1010 01110100

01101011 0010 01110100

01101100 0001 01110101

01101101 0100 01110010

01101110 0010 01110101

01101111 0111 01101010

01110000 1001 01110100

01110001 1100