Experiment 4

Program:

```
#code = open("code.txt", "r")
#macro = open("macro.txt","r")

code = open("/home/computer/Documents/ABC/code.txt", "r")
macro = open("/home/computer/Documents/ABC/macro.txt","r")
macrol = []
codel = []
macrocount = 0

for x in macro:
    macrol.append(x.split(' \n')[0].upper())

macroname = macrol[0+1]
macrol.remove(macroname)
macrol.pop(0)
macrol.pop()

for x in code:
    codel.append(x.split('\n')[0])

codelen = len(codel)

for i in range(len(codel)):
    if codel[i] == macroname:
        codel[i:i+len(macrol)-2] = tuple(macrol)
        macrocount += 1

for i in codel:
    print(i)

print("\n\nStatistical Output")
print("Number of instructions in input source code (excluding Macro calls) = {}".format(codelen
- macrocount))
print("Number of Macro calls = {}".format(macrocount))
print("Number of instructions defined in the Macro call = {}".format(len(macrol)))
print("Total number of instructions in the expanded source code = {}".format(len(codel)))
```

macro.close()
code.close()

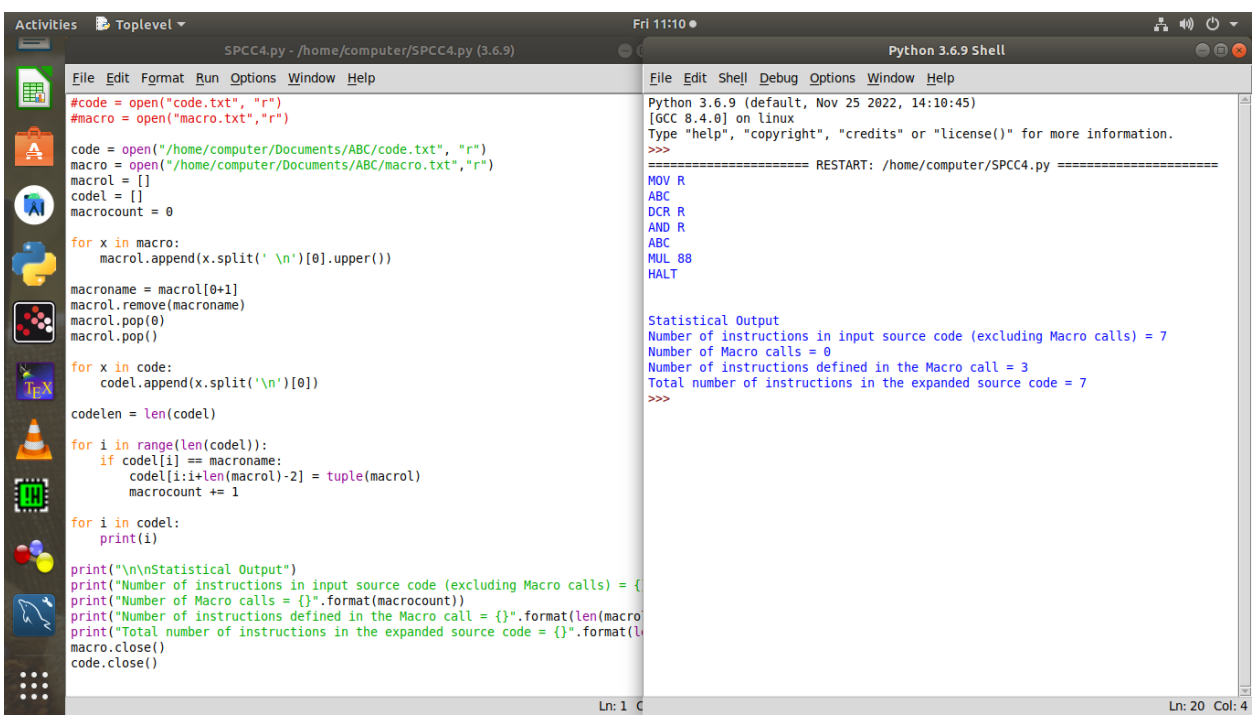
Code.txt file:

MOV R
ABC
DCR R
AND R
ABC
MUL 88
HALT

Macro.txt file:

MACRO
SURAJ
ADD 30
SUB 25
OR R
MEND

Output:



PROGRAM :

```
from sys import exit

motOpCode = ["MOV", "ADD", "SUB", "MUL", "DIV", "AND", "OR","LOAD", "STORE", "DCR",
"INC", "JMP", "JNZ", "HALT"]

keywords = ["MACRO", "CONST", "DOUBLE", "INT", "FLOAT", "SHORT", "LONG", "STRUCT",
"IF", "ELSE", "FOR", "SWITCH","CASE", "CHAR", "RETURN", "PRINTF", "SCANF", "AX", "BX",
"CX", "DX", "AH", "BH", "CH", "DH", "AL", "BL","CL", "DL"]

sourceCode = []

macroNames = []

macroDefinition = []

outputSourceCode = []

noOfInstructionSC = 0

noOfMacroCall = 0

noOfInstructionMC = 0

expandedCode = 0

totalArgs = []

x = 0

mapping = {}

mc = int(input("Enter the number of Macro Definition code line : "))

for i in range(mc):

    instruction = input(

        "Enter Macro code instruction {} :".format(i + 1)).upper()

    macroDefinition.append(instruction)

if macroDefinition[0] == "MACRO" and macroDefinition[-1] == "MEND":

    temp = str(macroDefinition[1])

    macroName, *argName = temp.split()

    temp = argName

    for i in range(len(temp)):

        if ',' in temp[i]:

            argName[i] = argName[i][0:-1]
```

```
    if macroName not in keywords and macroName not in motOpCode:
        macroNames.append(macroName)
else:
    print("Invalid Macro Definition.")
    exit(0)
sc = int(input("Enter the number of Source code lines : "))
for i in range(sc):
    instruction = input(
        "Enter Source code instruction {} : ".format(i + 1)).upper()
    sourceCode.append(instruction)
for i in range(sc):
    if macroName in sourceCode[i]:
        noOfMacroCall = noOfMacroCall + 1
    else:
        noOfInstructionSC = noOfInstructionSC + 1
for i in range(sc):
    if macroName in sourceCode[i]:
        x = x + 1
        noOfInstructionMC = 0
        temp = str(sourceCode[i])
        macroName, *argValue = temp.split()
        totalArgs.append(argValue)
        temp = argValue
        for j in range(len(temp)):
            if ',' in temp[j]:
                argValue[j] = argValue[j][0:-1]
# Create Dictionary for mapping
        for j in range(len(argName)):
            name, value = argName[j], argValue[j]
```

```
            mapping[name + str(x)] = value


        for j in range(2, mc - 1):
            for k in range(len(argName)):
                if argName[k] in macroDefinition[j]:
                    temp = macroDefinition[j]
                    opCode, value = temp.split()
                    tempValue = mapping.get(value + str(x))
                    temp = opCode + ' ' + str(tempValue)
            outputSourceCode.append(temp)
            noOfInstructionMC = noOfInstructionMC + 1
    else:
        temp = sourceCode[i]
        outputSourceCode.append(temp)
print("Expanded Source Code is : ")
for i in outputSourceCode:
    print(i)
    expandedCode = expandedCode + 1
print()
print("No of instructions in input source code : {}".format(noOfInstructionSC))
print("No of macro call : {}".format(noOfMacroCall))
print("No of instructions defined in macro call : {}".format(noOfInstructionMC))
for i in range(len(totalArgs)):
    print("Actual arguement during {} Macro call 'RAHUL' = {}".format(i + 1, ', '.join(totalArgs[i])))
    print("Total number of instructions in expanded code : {}".format(expandedCode))
```

OUTPUT :

```
Enter the number of Macro Definition code line : 6
Enter Macro code instruction 1 :MACRO
Enter Macro code instruction 2 :RAHUL &ARG
Enter Macro code instruction 3 :ADD &ARG
Enter Macro code instruction 4 :SUB &ARG
Enter Macro code instruction 5 :OR &ARG
Enter Macro code instruction 6 :MEND
Enter the number of Source code lines : 7
Enter Source code instruction 1 : MOV R
Enter Source code instruction 2 : RAHUL 30
Enter Source code instruction 3 : DCR R
Enter Source code instruction 4 : AND R
Enter Source code instruction 5 : RAHUL 55
Enter Source code instruction 6 : MUL 88
Enter Source code instruction 7 : HALT
Expanded Source Code is :
MOV R
ADD 30
SUB 30
OR 30
DCR R
AND R
ADD 55
SUB 55
OR 55
MUL 88
HALT

No of instructions in input source code : 5
No of macro call : 2
No of instructions defined in macro call : 3
Actual arguement during 1 Macro call 'RAHUL' = 30
Total number of instructions in expanded code : 11
Actual arguement during 2 Macro call 'RAHUL' = 55
Total number of instructions in expanded code : 11
```

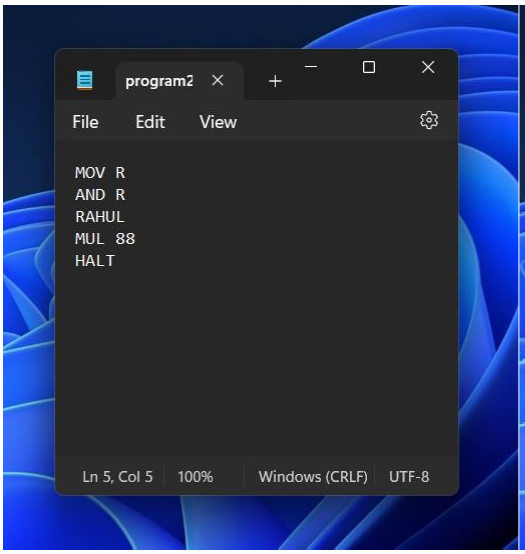
Experiment 8

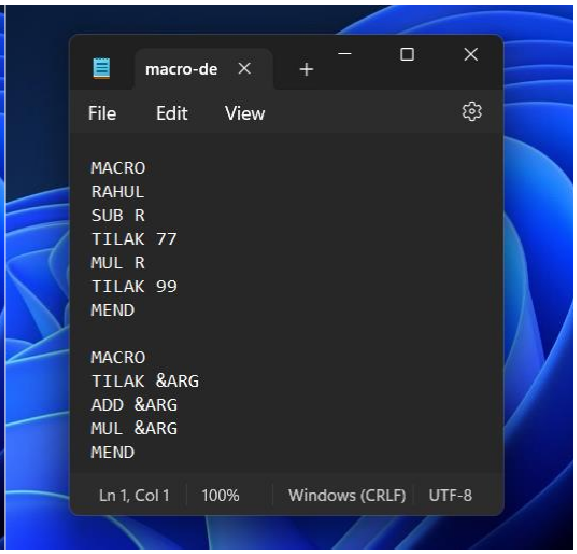
Program:

```
import sys
import sys
macro_file = sys.argv[1]
program_file = sys.argv[2]
macro_cache = {}
with open(macro_file) as f:
  data = [i.strip() for i in f.readlines()]
macro_state = False
for i in range(len(data)):


   if data[i] == 'MACRO':
      i = i + 1
      if data[i].split(" ")[0].startswith("&"):
         label = data[i].split(" ")[0]
         mname = data[i].split(" ")[1]
         pholders = ".join(data[i].split(" ")[2:]).split(',')
      else:
         label = None
         mname = data[i].split(" ")[0]
         pholders = ".join(data[i].split(" ")[1:]).split(',')



      pholder = {}
      count = 0
      for j in pholders:
         pholder[j] = "{" + f"{count}" + "}"
         count += 1
```

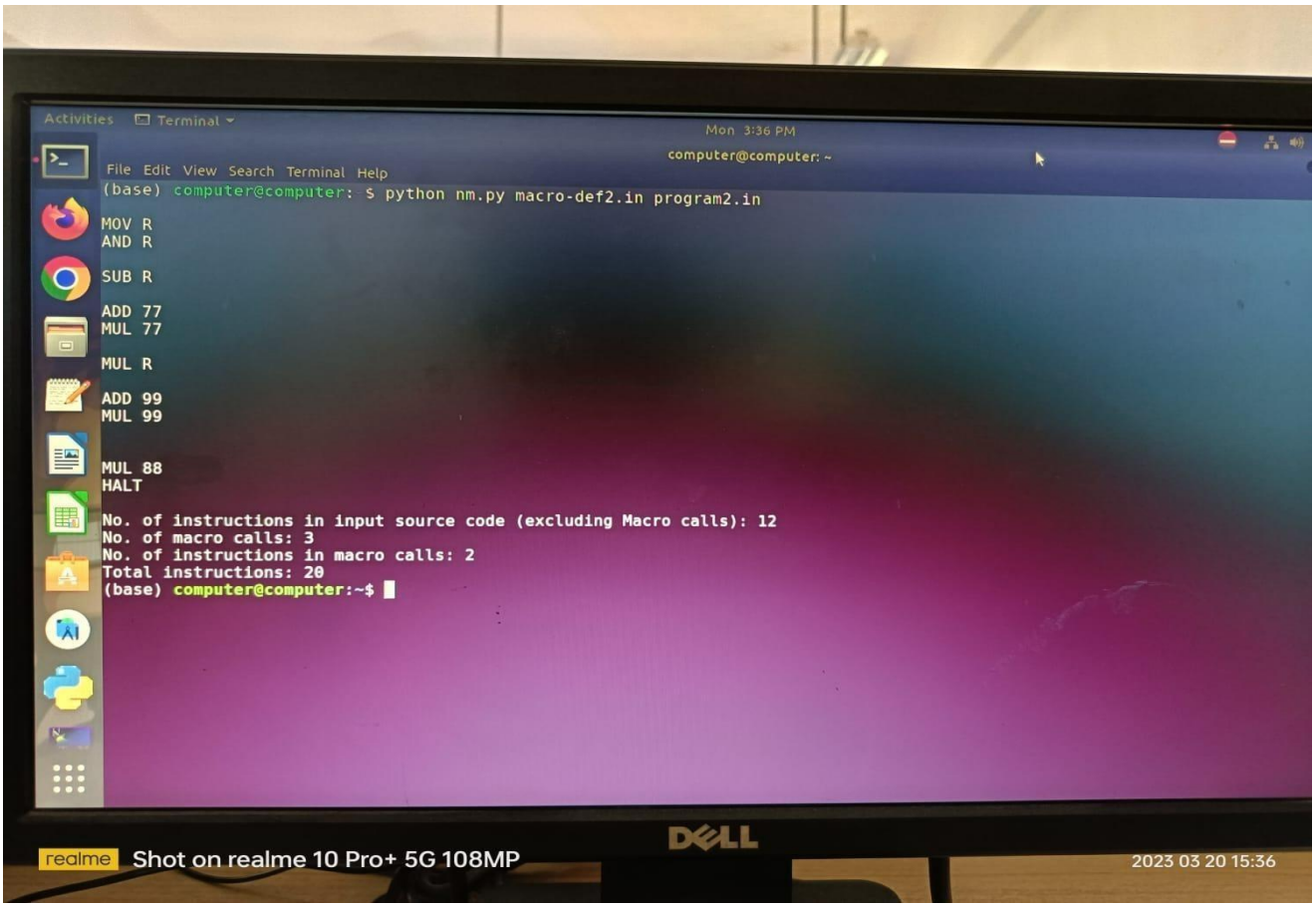
```
        # print(pholders)
        macro_cache[mname] = []
        i += 1
        while data[i] != 'MEND':
            for j in pholders:
                data[i] = data[i].replace(j, pholder[j], -1)
            if label != None:
                data[i] = data[i].replace(label, "{"+f"{count}"+"}")
                # print(j, data[i])
            macro_cache[mname].append(data[i])
            i += 1
        i += 1
macro_calls = 0
src_inst = 0
macro_calls_inst = 0
total = 0
# print(macro_cache)
print()
with open(program_file) as f:
    data = [i.strip() for i in f.readlines()]

    for qwe in range(2):
        output = []
        for i in data:
            if len(i.split(" ")) > 1 and i.split(" ")[1] in macro_cache:
                macro_calls += 1
                macro_calls_inst = len(macro_cache[i.split(" ")[1]])
                output.append("")
                for j in macro_cache[i.split(" ")[1]]:
                    output.append(j.format(*".join(i.split(" ")[2:]).split(","), i
```

```
.split(" ")[0]))
                total += 1
            output.append("")
        elif i.split(" ")[0] in macro_cache:
            macro_calls += 1
            macro_calls_inst = len(macro_cache[i.split(" ")[0]])
            output.append("")
            for j in macro_cache[i.split(" ")[0]]:
                output.append(j.format(*".join(i.split(" ")[1:]).split(",")))
                total += 1
            output.append("")
        else:
            src_inst += 1
            output.append(i)
            total += 1
    data = output
  for i in data:
    print(i)
print()
print(f"No. of instructions in input source code (excluding Macro calls): {src_inst}")
print(f"No. of macro calls: {macro_calls}")
print(f"No. of instructions in macro calls: {macro_calls_inst}")
print(f"Total instructions: {total}")
```


program2
File
Edit
View
MOV R
AND R
RAHUL
MUL 88
HALT
Ln 5, Col 5
100%
Windows (CRLF)
UTF-8


macro-de
File
Edit
View
MACRO
RAHUL
SUB R
TILAK 77
MUL R
TILAK 99
MEND
MACRO
TILAK &ARG
ADD &ARG
MUL &ARG
MEND
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

Output:


Activities
Terminal
Mon 3:36 PM
computer@computer: ~
File Edit View Search Terminal Help
(base) computer@computer: $ python nm.py macro-def2.in program2.in
MOV R
AND R
SUB R
ADD 77
MUL 77
MUL R
ADD 99
MUL 99
MUL 88
HALT
No. of instructions in input source code (excluding Macro calls): 12
No. of macro calls: 3
No. of instructions in macro calls: 2
Total instructions: 20
(base) computer@computer:~$
DELL
realme Shot on realme 10 Pro+ 5G 108MP
2023 03 20 15:36

Program:

Code:

```
mnemonics_arr = ["MOV R", "ADD R", "SUB R", "MUL R", "DIV R", "AND R", "OR R",
"ADD", "SUB", "MUL", "DIV", "AND", "OR", "LOAD", "STORE", "DCRR", "INCR", "JMP",
"JNZ", "HALT"]
size_arr = [1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 1, 1, 3, 3, 1]
opcode_arr = ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11',
'12', '13', '14', '15', '16', '17', '18', '19', '20']
source_code = []
alp_instruct = []
obj_code = []
absolute_addr_arr = []
absolute_addr = 1000
obj_index=0
print('Enter source code: ')
cmd = input()
source_code.append(cmd)

while source_code[-1] != 'HALT':
    cmd = input()
    source_code.append(cmd)

for code in source_code:
    mnemonic = 0
    code_arr = code.split(' ')
    if (code_arr[0] == 'ADD' or code_arr[0] == 'SUB' or code_arr[0] == 'MUL' or
code_arr[0] == 'DIV' or code_arr[0] == 'AND' or code_arr[0] == 'OR') and
code_arr[1] != 'R':
        for mnemonic in range(len(mnemonics_arr)):
            if mnemonics_arr[mnemonic] == code_arr[0]:
                obj_code.append(opcode_arr[mnemonic])
                absolute_addr +=1
                absolute_addr_arr.append(absolute_addr)
                alp_instruct.append(code)

                absolute_addr+=1
                obj_code.append(code_arr[1])
                absolute_addr_arr.append(absolute_addr)
                alp_instruct.append('       ')

    elif (code_arr[0] == 'STORE' or code_arr[0] == 'LOAD' or code_arr[0] == 'JMP'
or code_arr[0] == 'JNZ'):
        addr1 = ''
        addr2 = ''
        for k in range(0, len(code_arr[1])):
            if(k == 0 or k == 1):
                addr1 = addr1 + code_arr[1][k]
            if(k == 2 or k == 3):
                addr2 = addr2 + code_arr[1][k]
        for mnemonic in range(len(mnemonics_arr)):
```

```
            if mnemonics_arr[mnemonic] == code_arr[0]:
                obj_code.append(opcode_arr[mnemonic])
                absolute_addr+=1
                absolute_addr_arr.append(absolute_addr)
                alp_instruct.append(code)

                absolute_addr+=1
                obj_code.append(addr1)
                absolute_addr_arr.append(absolute_addr)
                alp_instruct.append('        ')

                absolute_addr+=1
                obj_code.append(addr2)
                absolute_addr_arr.append(absolute_addr)
                alp_instruct.append('        ')

    else:
        for mnemonic in range(len(mnemonics_arr)):
            if mnemonics_arr[mnemonic] == code:
                obj_code.append(opcode_arr[mnemonic])
                absolute_addr+=1
                alp_instruct.append(code)
                absolute_addr_arr.append(absolute_addr)


print("Absolute Address                        Object
Code                            ALP Instruction ")
for table in range(len(obj_code)):
    print(absolute_addr_arr[table], "                                        ",
obj_code[table],"                                        ",alp_instruct[table])
```

Output:

```
PS D:\sem 6\SPCC> python .\prac9.py
Enter source code:
ADD 20
MOV R
OR 55
MUL R
STORE 2000
HALT
Absolute Address                        Object Code                         ALP Instruction
1001                                        08                                  ADD 20
1002                                        20
1003                                        01                                  MOV R
1004                                        13                                  OR 55
1005                                        55
1006                                        04                                  MUL R
1007                                        15                                  STORE 2000
1008                                        20
1009                                        00
1010                                        20                                  HALT
PS D:\sem 6\SPCC>
```