Name: Adwait Hegde
Roll No: 2019130019
TE Comp (Batch-A)

EXPERIMENT 8

Aim:

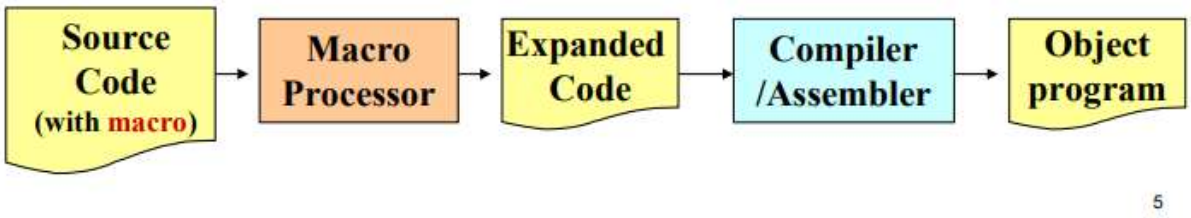
To implement one pass Macroprocessor SIC

Theory:

A Macro instruction is the notational convenience for the programmer. For every occurrence of macro the whole macro body or macro block of statements gets expanded in the main source code. Thus Macro instructions make writing code more convenient.

Salient features of Macro Processor:

- **Macro** represents a group of commonly used statements in the source programming language.
- Macro Processor replaces each macro instruction with the corresponding group of source language statements. This is known as the expansion of macros.
- Using Macro instructions programmer can leave the mechanical details to be handled by the macro processor.
- Macro Processor designs are not directly related to the computer architecture on which it runs.
- Macro Processor involves definition, invocation, and expansion.



DEFTAB

- A definition table used to store macro definition including
- macro prototype
- macro body
- Comment lines are omitted.
- Positional notation has been used for the parameters for efficiency in substituting arguments.

NAMTAB

- A name table used to store the macro names
- Serves as an index to DEFTAB
- Pointers to the beginning and the end of the macro definition

ARGTAB

- A argument table used to store the arguments used in the expansion of macro invocation
- As the macro is expanded, arguments are substituted for the corresponding parameters in the macro body.

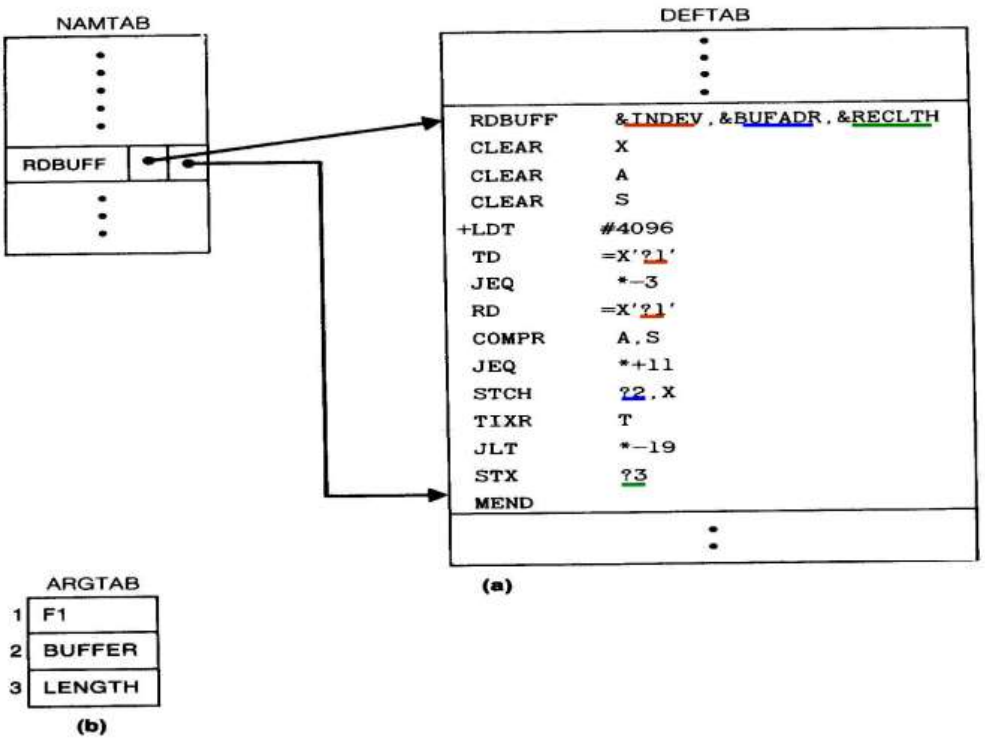
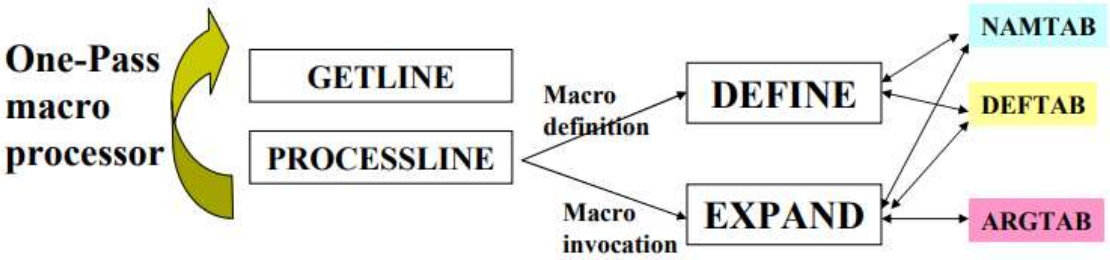


Figure 4.4 Contents of macro processor tables for the program in Fig. 4.1: (a) entries in NAMTAB and DEFTAB defining macro RDBUFF, (b) entries in ARGTAB for invocation of RDBUFF on line 190.

- Procedures
  - Macro definition: DEFINE
  - Macro invocation: EXPAND



Code:

```
inp = []
for i in open("i8.txt", "r"):
    if i.strip()=='':
        continue
    inp.append( i.strip().split() )

def getline(_ind=-1):
    global expanding
    res = ''
    if expanding:
        global deftab
        global ind_exp
        if _ind!=-1:
            res = deftab[_ind] #next line from deftab
        else:
            res = deftab[ind_exp]
    else:
        global inp
        global ind
        res = inp[ind] #next line from inp
        ind+=1
    return res

def expand(_line):
    mname = _line[0]
    global expanding
    expanding = True
    start, end = nametab[mname] #get the start and end from deftable
    op.append(['. '+' '.join(deftab[start])])
    global ind_exp
    ind_exp = start
    while ind_exp<end-1:
        ind_exp += 1
        line = getline()
        processline(line)
    expanding = False

def define(_line):
    name = _line[0]
    start_ptr = len(deftab)
    cur_ptr = start_ptr
    nametab[name] = [start_ptr]
    deftab.append([name])
    level = 1
```

```
    global ind_exp, expanding
    uska_start = ind_exp
    while level>0:
        l=''
        ind_exp += 1
        if expanding:
            l = getline(ind_exp)
        else:
            l = getline()
        if l[0] != '.':
            deftab.append(l)
            if len(l)>1 and l[1] == 'MACRO':
                level += 1
            elif len(l)>0 and l[0] == 'MEND':
                level -= 1
            cur_ptr+=1
            if cur_ptr>20:
                break
    nametab[name].append(start_ptr + ind_exp - uska_start)

def processline(inp):
    if len(inp)>0 and inp[0] in nametab:
        expand(inp) #add args
    elif len(inp)>1 and inp[1] == 'MACRO':
        define(inp) #add args
    else:
        op.append(inp)

op = []

ind_exp = 0
# deftab holds the instructions of the defined MACROs
deftab = []
# nametab stores the MACROs that are defined
nametab = dict()
# argtable holds the arguments for the metro called
argtab = []


ind = 0
expanding = False

while inp[ind][0] != 'END':
    line = getline()
    processline(line)
```

```
with open('op8.txt', 'w') as f:
    f.write('----------------------EXPANDED CODE----------------------\n')
    for i in op:
        f.write(' '.join(i))
        f.write('\n')
    f.write('\n-----------------------NAME TABLE-----------------------\n')
    for i in nametab:
        f.write(str(i)+' : '+str(nametab[i][0])+' '+str(nametab[i][1]))
        f.write('\n')
    f.write('\n--------------------DEFINITION TABLE-------------------\n')
    for i in deftab:
        f.write(' '.join(i))
        f.write('\n')
    f.write('\n---------------------ARGUMENTS TABLE-------------------\n')
    for i in argtab:
        f.write(' '.join(i))
```

Result:

Input:

i8.txt

```
1    MACROA MACRO
2    ADD
3    SUB
4    MACROB MACRO
5    MUL
6    MEND
7    MEND
8    START 1000
9    DIV
10   MACROA
11   DIV
12   MACROB
13   END
```

Output:

op8.txt

```
1   ----------------------EXPANDED CODE----------------------
2   START 1000
3   DIV
4   . MACROA
5   ADD
6   SUB
7   DIV
8   . MACROB
9   MUL
10
11  -----------------------NAME TABLE-----------------------
12  MACROA : 0 6
13  MACROB : 7 9
14
15  --------------------DEFINITION TABLE--------------------
16  MACROA
17  ADD
18  SUB
19  MACROB MACRO
20  MUL
21  MEND
22  MEND
23  MACROB
24  MUL
25  MEND
26
27  ---------------------ARGUMENTS TABLE--------------------
28
```

Conclusion:

In this Experiment, I learned about the concept of one pass Macroprocessor SIC and implement it programmatically using python.