Experiment No-5 : Defining a macro with multiple arguments, expansion of macro calls & generating expanded source code.

Aim:

The aim of this experiment is to understand the concept of defining macros with multiple arguments, expanding macro calls, and generating expanded source code in assembly language programming.

Introduction:

In assembly language programming, macros serve as powerful tools for code reusability and simplification. They allow programmers to define custom code blocks that can be used multiple times with different arguments. Macros with multiple arguments enhance flexibility and enable parameterized code blocks, contributing to efficient and maintainable code development.

Procedure:

- Write Assembly Language Program (ALP):
- Develop an assembly language program comprising a few instructions and macro calls with multiple arguments.
- Define Macro with Arguments:
- Define a macro with multiple arguments, specifying the macro name and its parameters.
- Write the instructions to be executed within the macro body.
- Write Expansion Program:
- Develop a program that reads the input assembly language code with macro calls and replaces each macro call with its expanded form.
- Ensure proper substitution of arguments in the expanded macro body.
- Generate Expanded Source Code:
- Execute the expansion program to generate the expanded source code.
- Output the expanded source code for further analysis and verification.

Output Statistics:

- Calculate and output statistical information such as:
- Number of instructions in the input source code (excluding macro calls)
- Number of macro calls
- Number of instructions defined in each macro call
- Actual arguments during each macro call
- Total number of instructions in the expanded source code.

Example:

- Consider an assembly language program with a macro called ADD with two arguments to perform addition:

```
MACRO ADD num1, num2
    MOV AX, num1
    ADD AX, num2
    MOV result, AX
MEND
ADD 10, 20
```

- The expansion of the ADD macro call would result in the following expanded source code:

```
MOV AX, 10
ADD AX, 20
MOV result, AX
```

Conclusion:

Through this experiment, we gain a deeper understanding of defining macros with multiple arguments, expanding macro calls, and generating expanded source code in assembly language programming. By leveraging macros effectively, programmers can enhance code modularity, readability, and maintainability, ultimately improving software development productivity and efficiency.

Code :

```
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":
```

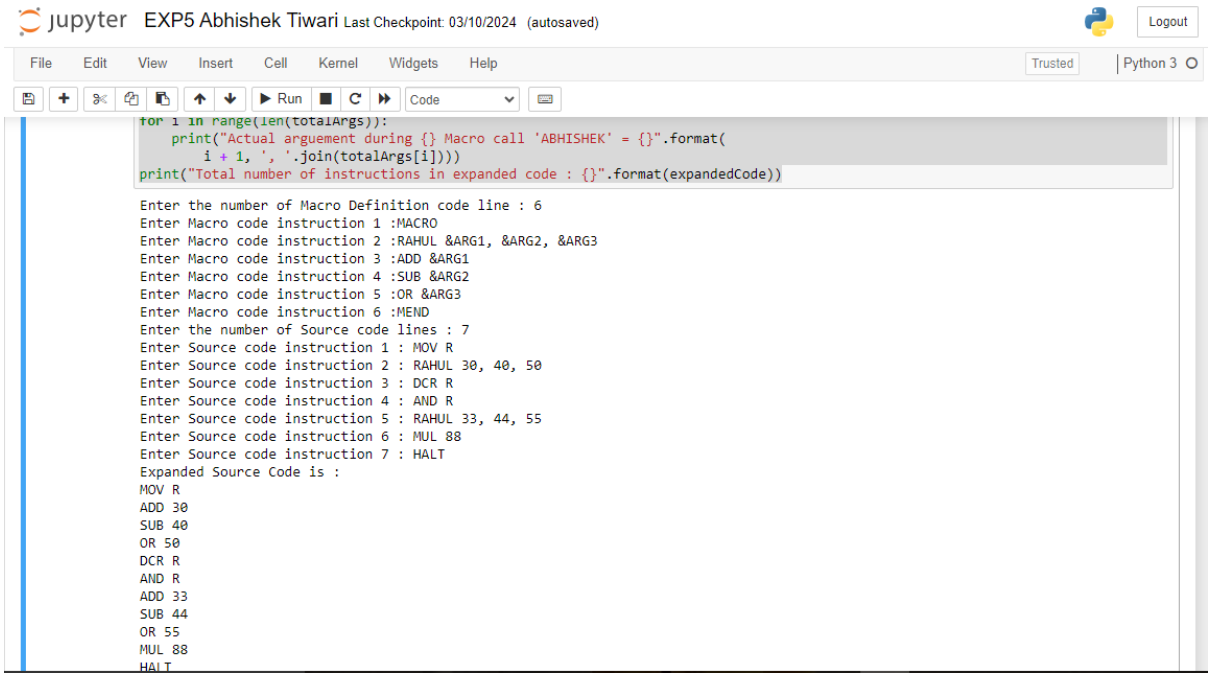
```
    temp = str(macroDefinition[1])
    macroName, *argName = temp.split()
    temp = argName
    for i in range(len(temp)):
        if ',' in temp[i]:
            argName[i] = argName[i][0:-1]
    if macroName not in keywords and macroName not in motOpCode:
        macroNames.append(macroName)
else:
    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()
```

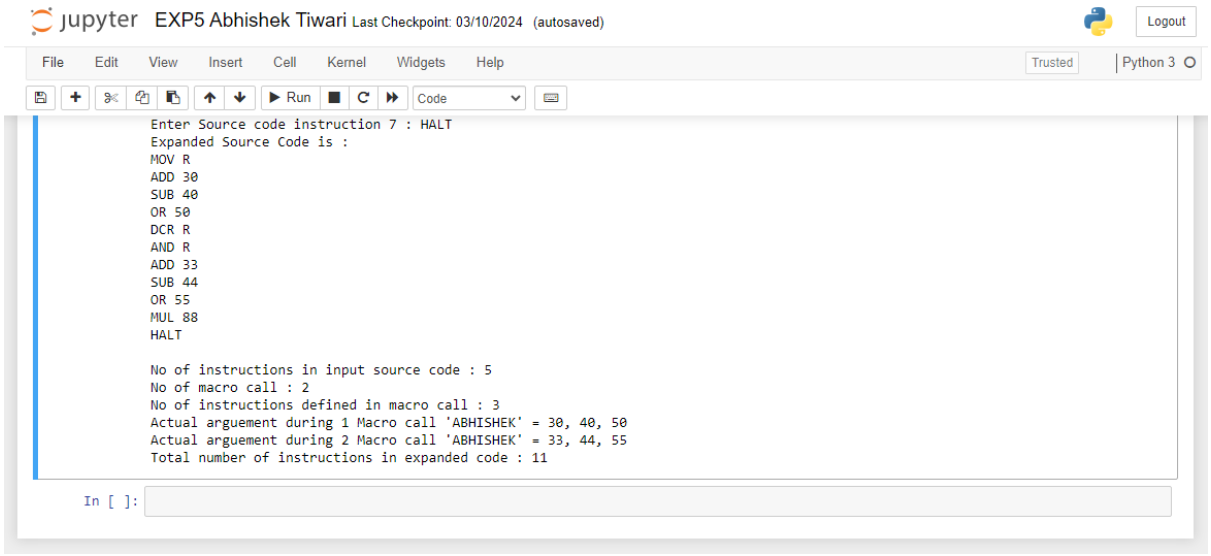
```
        totalArgs.append(argValue)
        temp = argValue
        for j in range(len(temp)):
            if ',' in temp[j]:
                argValue[j] = argValue[j][0:-1]

        # 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 'ABHISHEK' = {}".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 &ARG1, &ARG2, &ARG3
Enter Macro code instruction 3 :ADD &ARG1
Enter Macro code instruction 4 :SUB &ARG2
Enter Macro code instruction 5 :OR &ARG3
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, 40, 50
Enter Source code instruction 3 : DCR R
Enter Source code instruction 4 : AND R
Enter Source code instruction 5 : RAHUL 33, 44, 55
Enter Source code instruction 6 : MUL 88
Enter Source code instruction 7 : HALT
Expanded Source Code is :
MOV R
ADD 30
SUB 40
OR 50
DCR R
AND R
ADD 33
SUB 44
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 'ABHISHEK' = 30, 40, 50
Actual arguement during 2 Macro call 'ABHISHEK' = 33, 44, 55
Total number of instructions in expanded code : 11
```


jupyter EXP5 Abhishek Tiwari Last Checkpoint: 03/10/2024 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
for i in range(len(totalArgs)):
print("Actual arguement during {} Macro call 'ABHISHEK' = {}".format(
i + 1, ', '.join(totalArgs[i])))
print("Total number of instructions in expanded code : {}".format(expandedCode))
Enter the number of Macro Definition code line : 6
Enter Macro code instruction 1 :MACRO
Enter Macro code instruction 2 :RAHUL &ARG1, &ARG2, &ARG3
Enter Macro code instruction 3 :ADD &ARG1
Enter Macro code instruction 4 :SUB &ARG2
Enter Macro code instruction 5 :OR &ARG3
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, 40, 50
Enter Source code instruction 3 : DCR R
Enter Source code instruction 4 : AND R
Enter Source code instruction 5 : RAHUL 33, 44, 55
Enter Source code instruction 6 : MUL 88
Enter Source code instruction 7 : HALT
Expanded Source Code is :
MOV R
ADD 30
SUB 40
OR 50
DCR R
AND R
ADD 33
SUB 44
OR 55
MUL 88
HALT


jupyter EXP5 Abhishek Tiwari Last Checkpoint: 03/10/2024 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Enter Source code instruction 7 : HALT
Expanded Source Code is :
MOV R
ADD 30
SUB 40
OR 50
DCR R
AND R
ADD 33
SUB 44
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 'ABHISHEK' = 30, 40, 50
Actual arguement during 2 Macro call 'ABHISHEK' = 33, 44, 55
Total number of instructions in expanded code : 11
In [ ]: