Experiment No. 2

Aim : Implementation of Two passes Macro Processor.

Theory :

- **In Pass-I** the macro definitions are searched and stored in the macro definition table and the entry is made in macro name table
- **In Pass-II** the macro calls are identified and the arguments are placed in the appropriate place and the macro calls are replaced by macro definitions.

Pass 1 :-

- The input macro source program.
- The output macro source program to be used by Pass2.
- Macro-Definition Table (MDT), to store the body of macro-def.
- Macro-Definition Table Counter (MDTC), to mark the next available entry MDT.
- Macro- Name Table (MNT) - store names of macros.
- Macro Name Table counter (MNTC) - indicate the next available entry in MNT.
- Argument List Array (ALA) - substitute index markers for dummy arguments before storing a macro-def

Pass 2 :-

- The input is from Pass1.
- The output is an expanded source to be given to the assembler. ● MDT and MNT are created by Pass1.
- Macro-Definition Table Pointer (MDTP), used to indicate the next line of text to be used during macro-expansion.
- Argument List Array (ALA), used to substitute macro-call arguments for the index markers in the stored macro-def

Pass I Algorithm :

- Pass1 of the macro processor makes a line-by-line scan over its input.
- Set MDTC = 1 as well as MNTC = 1.
- Read the next line from the input program.
- If it is a MACRO pseudo-op, the entire macro definition except this (MACRO) line is stored in MDT.
- The name is entered into Macro Name Table along with a pointer to the first location of MDT entry of the definition
- When the END pseudo-op is encountered all the macro-def have been processed, so control is transferred to pass2

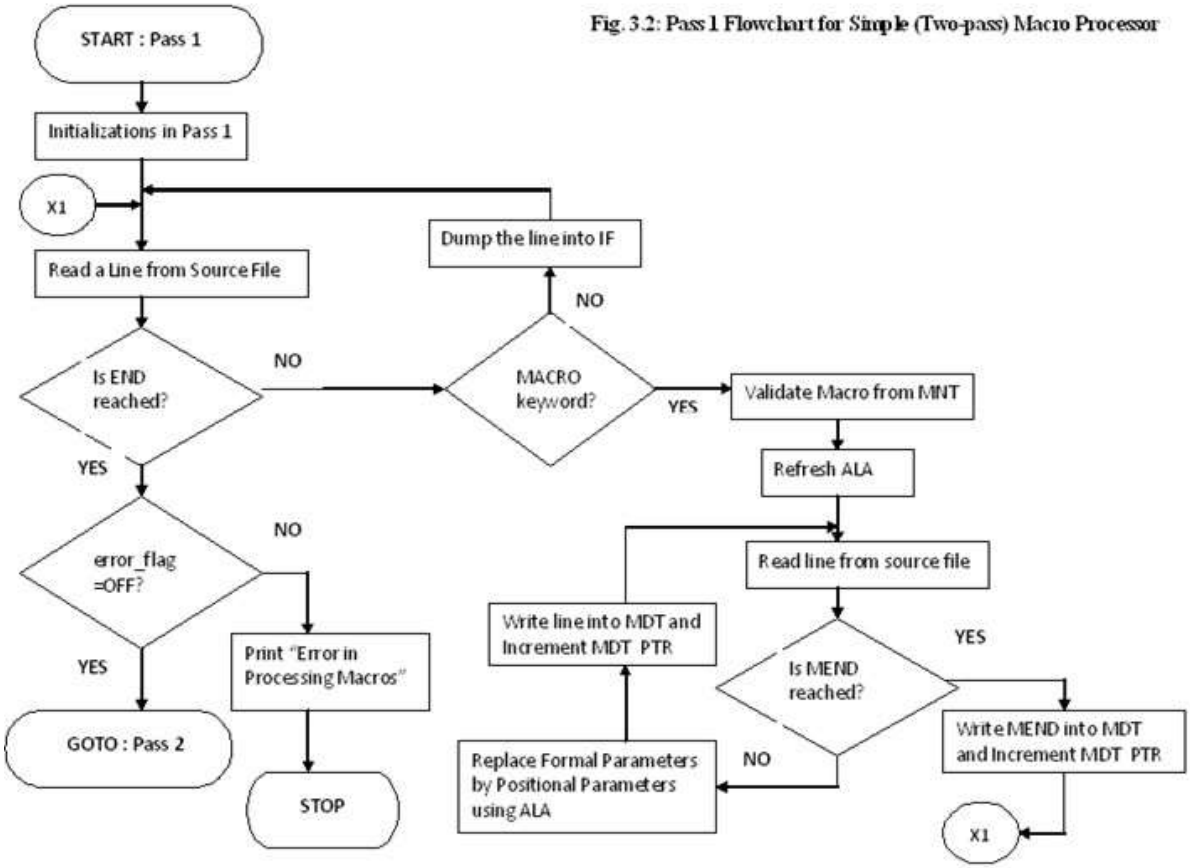

Fig. 3.2: Pass 1 Flowchart for Simple (Two-pass) Macro Processor

Pass II Algorithm :

● This algorithm reads one line of i/p prog. at a time.

● For each Line it checks if the op-code of that line matches any of the MNT entry.

● When a match is found (i.e. when a is a pointer called MDTF to corresponding macro-def stored in MDT.

● The initial value of MDTP is obtained from the MDT index field of MNT entry. ● The macro expander prepares the ALA consisting of a table of dummy argument indices & corresponding arguments to the call.

● Reading proceeds from the MDT, as each successive line is read, The values form the argument list one substituted for dummy arguments indices in the macro-def

● Reading MEND line in MDT terminates expansion of macro & scanning continues from the input file.

● When END pseudo-op encountered , the expanded source program is given to the assembler

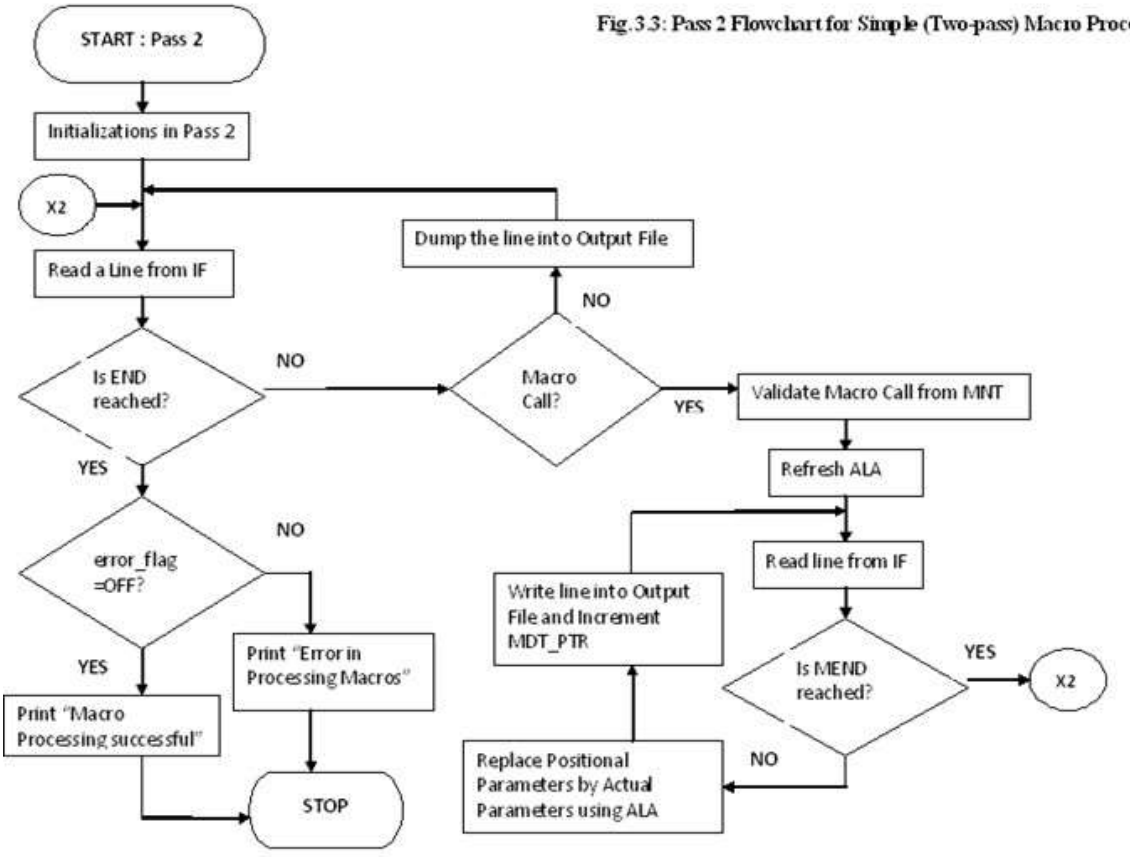

Fig. 3.3: Pass 2 Flowchart for Simple (Two-pass) Macro Processor

Program code:

```
fp = open('program3.txt','r')
program = fp.read().split('\n')
print('\nGiven assembly code \n')
for line in program:
    print(line)
fp.close()
fp = open('program3.txt','r')
program = fp.read().split('MEND\n')
fp.close()
mnt = []
mdt = {}
for line in program:
    line.strip()
    a = line.split('\n')
    if a[0] == 'MACRO':
        #print('Macro Name is: ', a[1])
        mnt.append(a[1])
        #print('Macro Instruction are:', a[2:])
        mdt[a[1]] = a[2:len(a)-1]
    else:
        prog = a
print('\nContent of MNT:\n')
for each_mn in mnt:
    print(each_mn)
print('\nContent of MDT:\n')
for k,v in mdt.items():
    for command in v:
        print(command)
print('\nAfter Macro Expansion\n')
for line in prog:
    identify_mc = line.split()
    for word in identify_mc:
        if word in mnt:
            value = mdt[word]
            for i in value:
                print(i)
        else:
            print(word,end=" ")
    else:
        print()
```

Input assembly program:

```
MACRO
INCR
MOVER AREG ONE
MOVER BREG TWO
ADD AREG BREG
MEND
MACRO
DNL
MOVER CREG ONE
MOVER DREG TWO
DIV CREG DREG
MEND
MACRO
INCR1
MOVER AREG ONE
MOVER BREG TWO
MULT AREG BREG
MEND
START 200
MOVER AREG FIRST
ADD AREG SECOND
MOVEM AREG RESULT
INCR
PRINT RESULT
RESULT DS 1
FIRST DC 5
INCR1
SECOND DC 7
INCR
DNL
END
```