Assignment No. 02

Problem Statement: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

Objectives:

1. To identify and design different data structure used in macro-processor implementation
2. To apply knowledge in implementation of two pass microprocessor.

Theory:

MACRO

Macro allows a sequence of source language code to be defined once & then referred to by name each time it is to be referred. Each time this name occurs is a program the sequence of codes is substituted at that point.

A macro consist of

1. Name of the macro
2. Set of parameters
3. Body of macro

Macros are typically defined at the start of program. Macro definition consists of

1. MACRO pseudo
2. MACRO name
3. Sequence of statements
4. MEND pseudo opcode terminating

A macro is called by writing the macro name with actual parameter is an assembly program. The macro call has following syntax <macro name>.

MACRO PROCESSOR

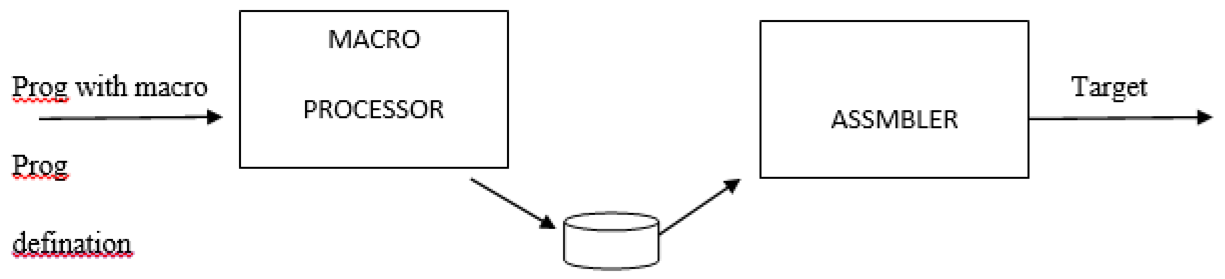


Fig 1. Assembly language program without macro

Macro processor takes a source program containing macro definition & macro calls and translates into an assembly language program without any macro definition or calls. This program can now be handled over to a conventional assembler to obtain the target language.

MACRO DEINITION

Macros are typically defined at the start of a program. A macro definition consists of

1. MACRO pseudo code
2. MACRO name
3. Sequence of statement
4. MEND pseudo opcode terminating macro definition

Structure of a macro Example

```
 MACRO
INCR & ARG
ADD AREG,& ARG
ADD BRA,& ARG
ADD CREG, & ARG
 MEND
```

MACRO Expansion:

During macro expansion each statement forming the body of the macro as picked up one by one sequentially.

a. Each statement inside macro may have as it is during expansion.

b. The name of a formal parameter which is preceded by the character ‘&’ during macro expansion an ordinary starting is retained without any modification. Formal parameters are replaced by actual parameters value.

When a call is found the call processor sets a pointer the macro definition table pointer to the corresponding macro definition started in MDT. The initial value of MDT is obtained from

MDT index.

8. Design of macro processor:

Pass I:

Generate Macro Name Table (MNT)

Generate Macro Definition Table (MDT)

Generate IC i.e. a copy of source code without macro definitions.

MNT:

Sr.No	Macro Name	MDT Index

MDT:

Sr. No	MACRO STATEMENT

ALA:

Index	Argument

Specification of Data Bases

Pass 1 database:

1. The input macro source desk.
2. The output macro source desk copy for use by passes 2.
3. The macro definition table (MDT) used to store the names of defined macros.
4. Macro name table (MDT) used to stare the name of defined macros.
5. The Macro definition table counter used to indicate the next available entry in **MNT.**
6. The macro name table counter (MNTC) used to indicate next available entry in MNT.
7. The arguments list array (ALA) used to substitute index markers for dummy arguments before starting a macro definition.

Pass 2 database:

1. The copy of the input source deck obtained from Pass- I

2. The output expanded source deck to be used as input to the assembler

3. The Macro Definition Table (MDT), created by pass 1

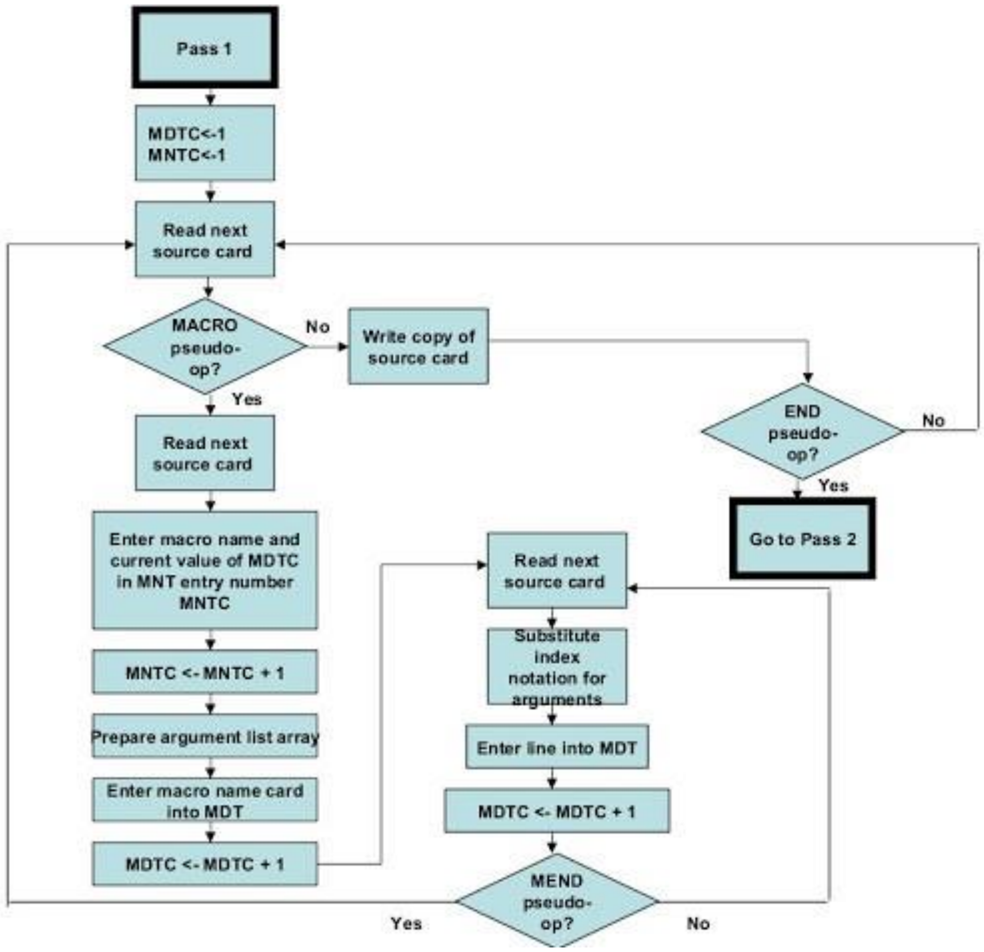
4. The Macro Name Table (MNT), created by pass 1

5. The Macro Definition Table Counter (MNTC), used to indicate the next line of text to be used during macro expansion
6. The Argument List Array (ALA), used to substitute macro call arguments for the index markers in stored macro definition

Pass II:
Replace every occurrence of macro call with macro definition. (Expanded Code)

There are four basic tasks that any macro instruction process must perform:
1. **Recognize macro definition:**
A macro instruction processor must recognize macro definition identified by the MACRO and MEND pseudo-ops. This tasks can be complicated when macro definition appears within macros. When MACROs and MENDs are nested, as the macro processor must recognize the nesting and correctly match the last or or outer MEND with first MACRO. All intervening text, including nested MACROs and MENDs defines a single macro instruction.
2. Save the definition:
The processor must store the macro instruction definition, which it will need for expanding macro calls.
3. **Recognize calls:**
The processor must recognize the macro calls that appear as operation mnemonics.
This suggests that macro names be handled as a type of op-code.
4. **Expand calls and substitute arguments:**
The processor must substitute for dummy or macro definition arguments the corresponding arguments from a macro call; the resulting symbolic text is then substitute for macro call. This text may contain additional macro definition or call.

Algorithm/Flowchart: Flow chart for pass 1 of two pass macro processor



Input: Input for pass 1 macro Processor

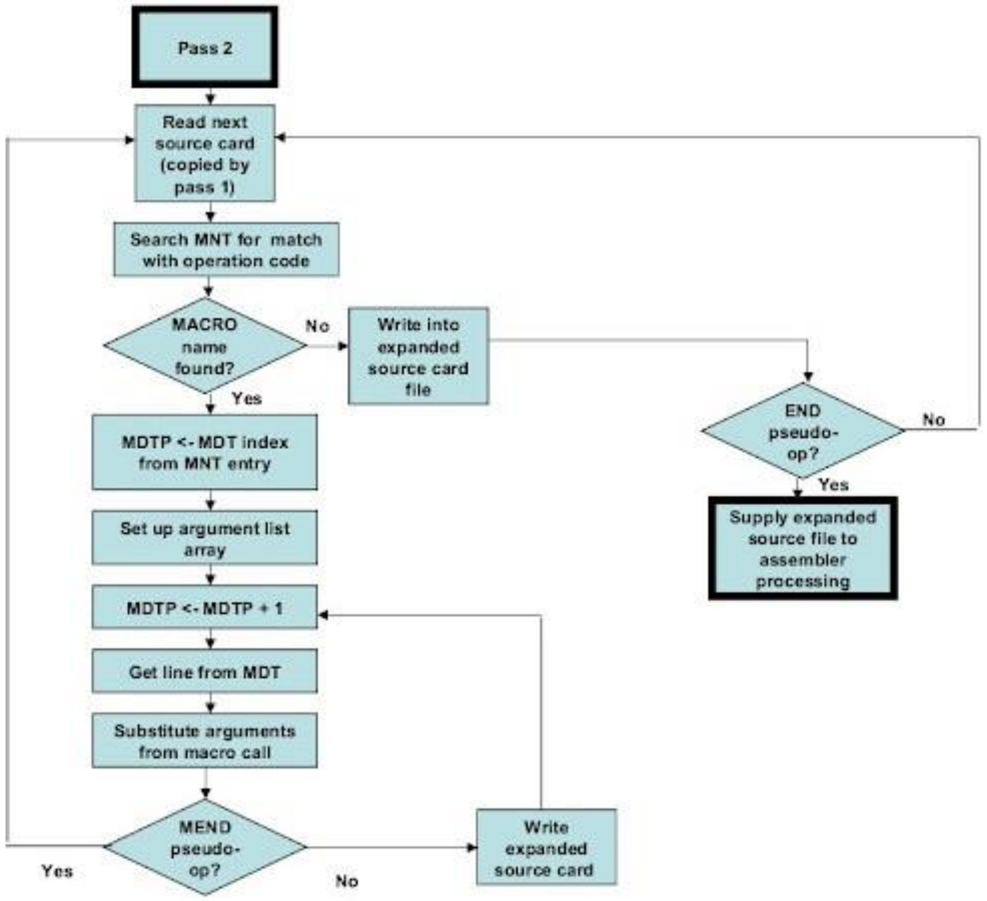
```
MACRO
M1   &X, &Y, &A=AREG, &B=
MOVER     &A, &X
ADD&A, ='1'
MOVER     &B, &Y
ADD&B, ='5'
MEND
MACRO
M2   &P, &Q, &U=CREG, &V=DREG
MOVER     &U, &P
MOVER     &V, &Q
ADD&U, ='15'
ADD&V, ='10'
MEND
START      100
M1   10, 20, &B=CREG
M2   100, 200, &V=AREG, &U=BREG
END
```

Output: Output for pass 1 macro Processor

Assembly language program without macro definition but with macro call and data structure as follows :

1. Intermediate code file (IC):
2. Macro Name Table (MNT):
3. Macro Definition Table (MDT):
4. Argument List Array (ALA):
   - 4.1 Parameter Name Table(PNTAB)
   - 4.2. Keyword Parameter Table(KPDT)

Algorithm/Flowchart: Flow chart for pass 1 of two pass macro processor


Pass 2
Read next source card (copied by pass 1)
Search MNT for match with operation code
MACRO name found?
No
Write into expanded source card file
Yes
MDTP <- MDT index from MNT entry
Set up argument list array
MDTP <- MDTP + 1
Get line from MDT
Substitute arguments from macro call
MEND pseudo-op?
Yes
No
Write expanded source card
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing

Input : Input for pass 2 macro processor

IC.text
MNT.text
MDT.Text
PNTab.Text
KPDT.Text

Output: **Output for pass 2 macro processor**
Assembly language program without macro definition and macro call.

Test Cases:

1. Check macro end not found.
2. Duplicate macro name found.
3. Check program output by changing macro name and parameter list.
4. Handle label in macro definition. Handle multiple macro definitions and calls.
5. Check macro definition not found.

Check program output by changing parameter list in macro call.

Software Requirement:

1. Fedora/Ubuntu
2. Eclipse
3. JDK

Hardware Requirement: N/A

Conclusion: We have successfully completed implementation of Pass-I and pass 2 of macro processor.