Assignment No. 02

Problem Statement: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macroprocessor. 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 pass1 and pass 2 of two pass microprocessor.

Software Requirement:
Operating System recommended :- 64-bit Open source Linux or itsderivative

Programming tools recommended: - Eclipse IDE

Hardware Requirement:I3 and I5 machines

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 Psuedo
2) MACRO Name
3) Sequence of statement
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>.

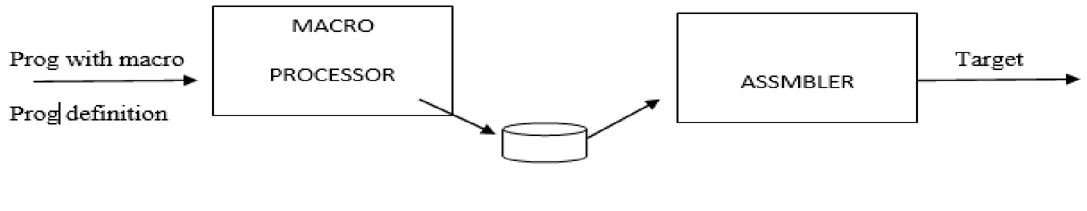
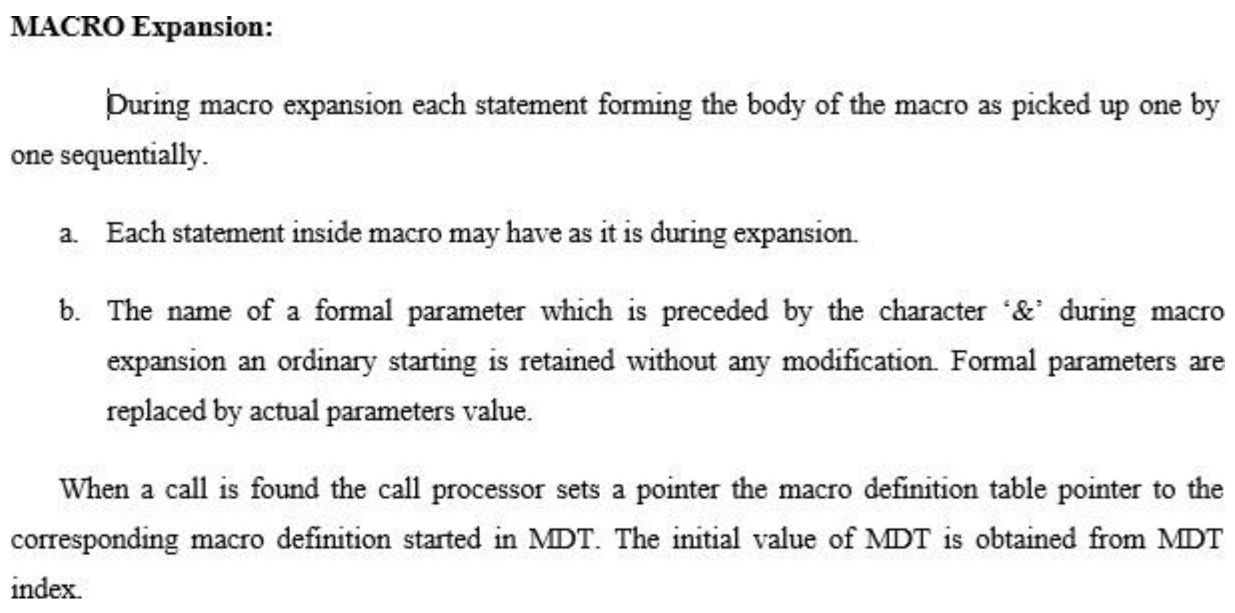


Fig 1. Assembly language program without macro

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 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.

Data structure for Macro pass I:

1) Generate Macro Name Table (MNT)
2) Generate Macro Definition Table (MDT)
3) Generate IC i.e a copy of source code without macro definition

MNT:

Sr.No	Macro Name	MDT Index

MDT:

Sr. No	MACRO STATEMENT

ALA:

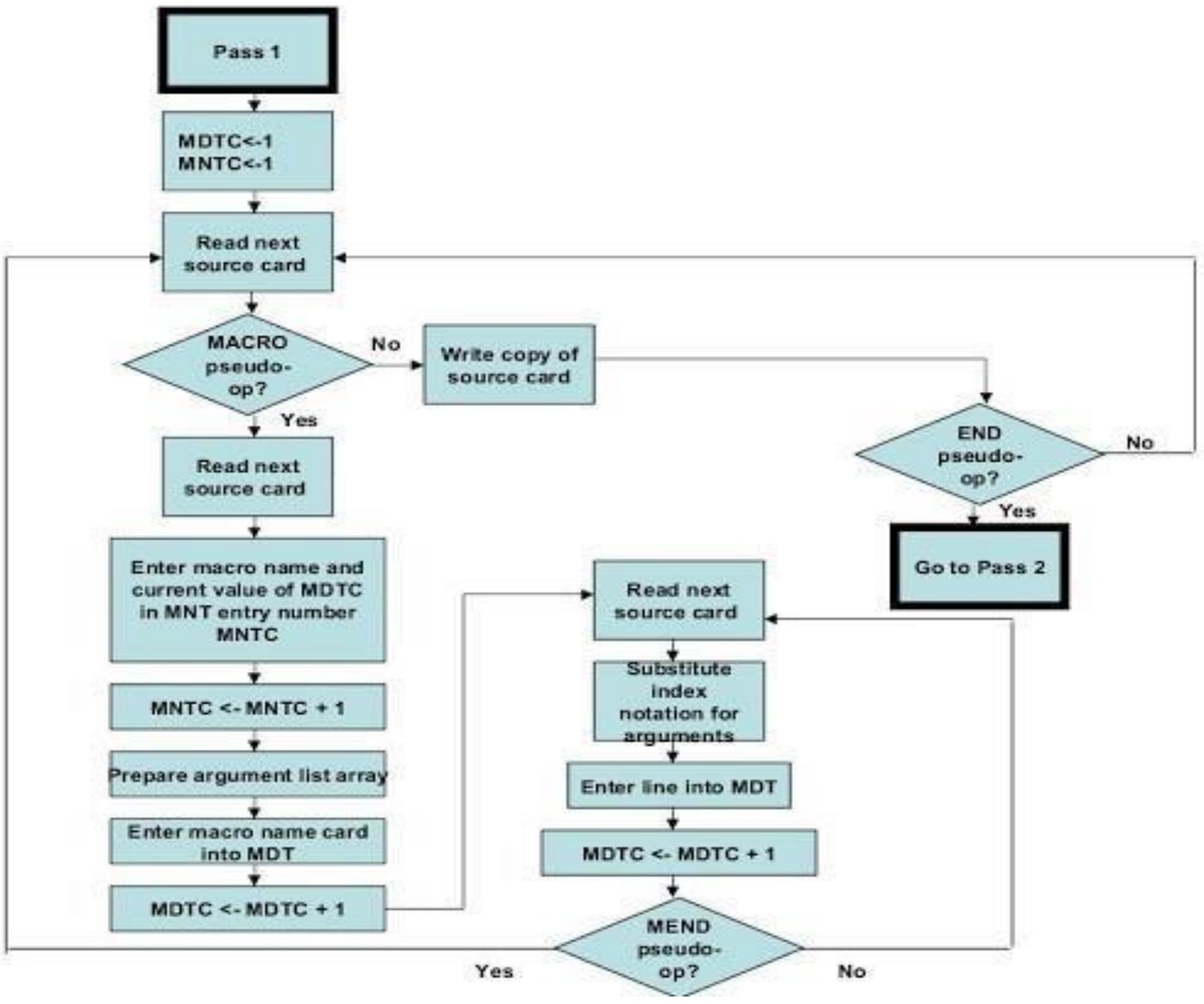
Index	Argument

Specification of Data Bases

Pass 1 data bases

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 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.

Algorithm/Flowchart:



Input:

Small assembly language program with macros written in file input.asm.

Output:

Assembly language program without macro definition but with macro call.
Note: Follow the following templates during implementation

Macro Name Table (MNT) :

Macro Definition Table (MDT) :
Argument List Array (ALA) :

Test Cases for Pass-I

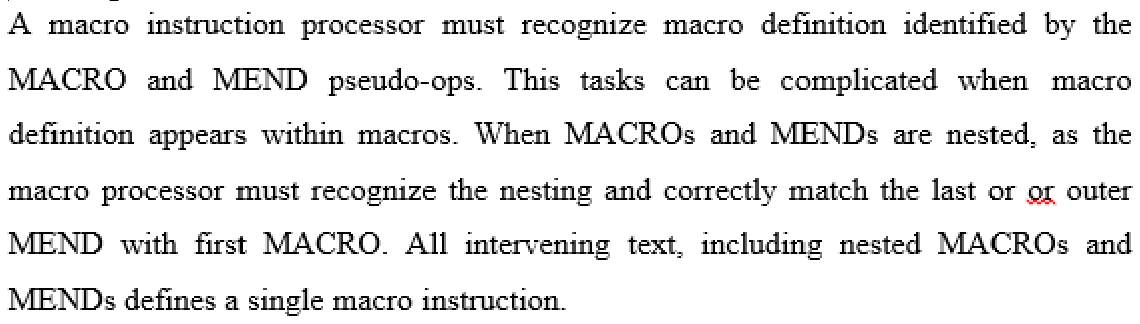
1. Check macro end not found.
2. Duplicate macro name found.
3. Check program output by changing macro name and parameter list.
4. Handle label in macro definition.
5. Handle multiple macro definitions and calls

MACRO 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:

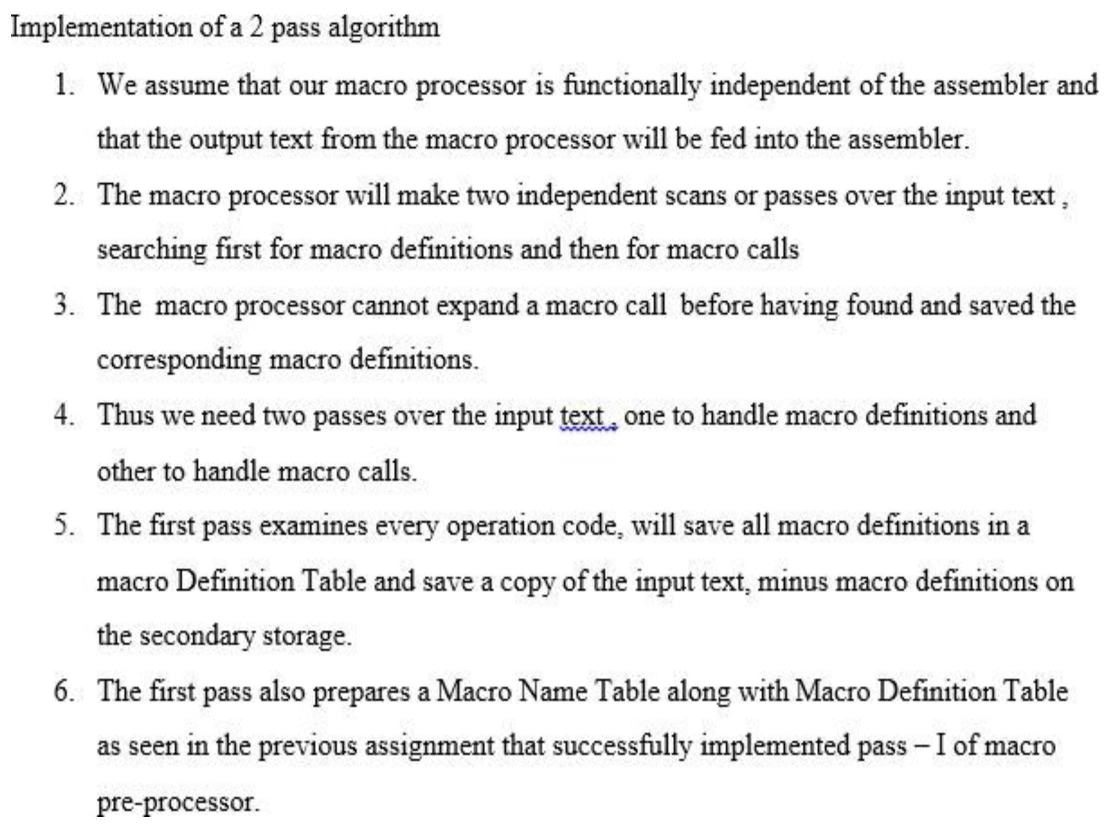
The processor must store the macro instruction definition, which it will need for expanding macro calls.

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:

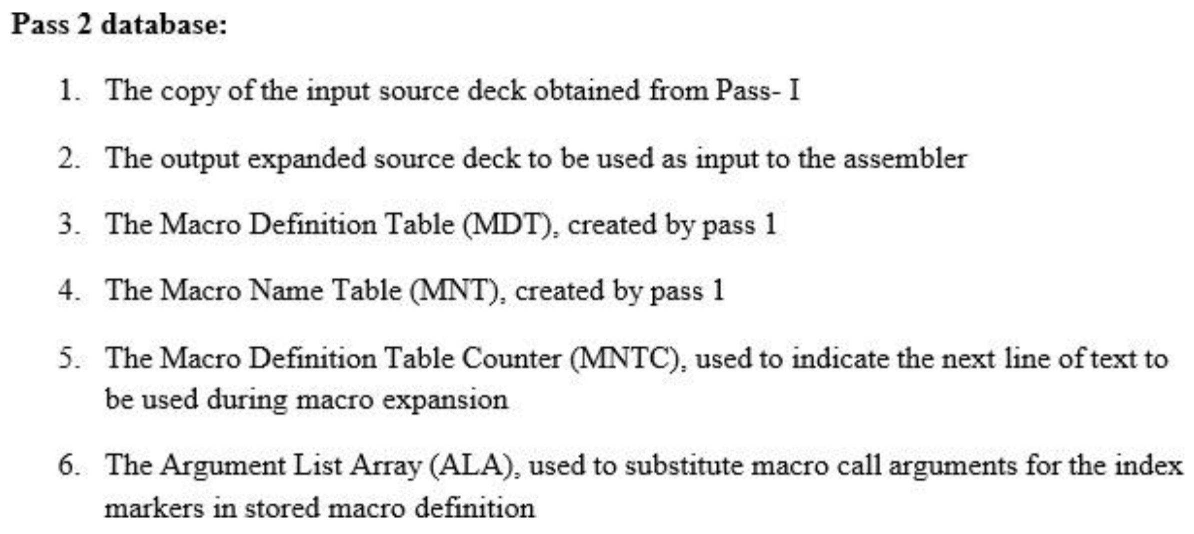
The processor must substitute for dummy or macro definition arguments the corresponding arguments from a macro call; the resulting symbolic text is then substitute for macro call. This text may contain additional macro definition or call.


Implementation of a 2 pass algorithm

1. We assume that our macro processor is functionally independent of the assembler and that the output text from the macro processor will be fed into the assembler.
2. The macro processor will make two independent scans or passes over the input text , searching first for macro definitions and then for macro calls
3. The macro processor cannot expand a macro call before having found and saved the corresponding macro definitions.
4. Thus we need two passes over the input text , one to handle macro definitions and other to handle macro calls.
5. The first pass examines every operation code, will save all macro definitions in a macro Definition Table and save a copy of the input text, minus macro definitions on the secondary storage.
6. The first pass also prepares a Macro Name Table along with Macro Definition Table as seen in the previous assignment that successfully implemented pass – I of macro pre-processor.

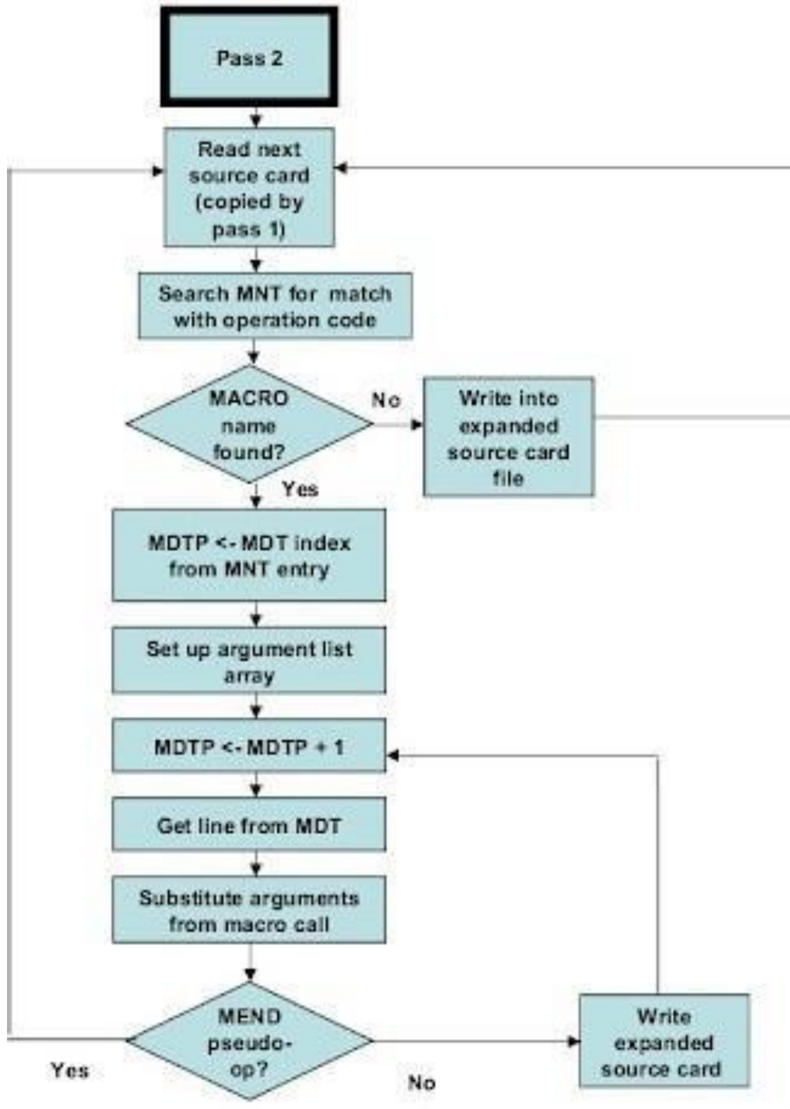
The second pass will now examine every operation mnemonic and replace each macroname with the appropriate text from the macro definitions.

Specifications of Database:


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

Algorithm/Flowchart of Pass- II



Pass 2
Read next source card (copied by pass 1)
Search MNT for match with operation code
MACRO name found?
No
Yes
Write into expanded source card file
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing
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

Input: Output of pass-1 (Intermediate File) given as a input to pass-2.

Output: Assembly language program without macro definition and macro call.

Test Cases:

1. Check macro definition not found.

Check program output by changing parameter list in macro call.

Conclusion: We have successfully completed implementation of Pass –I and Pass-II of macro.

Frequently Asked Questions:

1. Define macro?
2. Define purpose of pass-1 of two pass macro processor
3. List out types of macro arguments
4. What is the use of MDT-index field in MNT?
5. What is macro expansion?
6. Define purpose of pass-2 of two pass macro processor
7. What is positional arguments?
8. What is the use of MDT-index field in MNT?
9. What is the use of MNT table while processing macro call?