Module-3

Macros & Macroprocessor

3	**Macros and Macro Processor**	Introduction, Macro definition and call, Features of Macro facility: Simple, parameterized, conditional and nested. Design of single pass macro processor, data structures used.	**8**

Macro Instructions

- Macro instructions are single-line abbreviation for groups of instructions.
- In employing a macro, the programmer essentially defines a single “instruction” to represent a block of code.
- For every occurrence of this one-line macro instruction in his program, the macro processing assembler will substitute the entire block.

Macros

- The purpose of the macros is to either automate frequently used sequences or enable a more powerful abstraction

- Integral macro operations simplify debugging and program modification and they facilitate standardization.

Macro Instructions & Macro Processor

Macro instructions are usually considered an extension of the basic assembler language, and the macro processor is viewed as an extension of the basic assembler algorithm.

Example:

Consider the following program:

```
        .
        .
        .
A           1, DATA
A           2, DATA
A           3, DATA
        .
        .
        .
A           1, DATA
A           2, DATA
A           3, DATA
        .
        .
```

Macros

- We can invent a macro language that:
  - Allows us to specify the above as a macro definition and
  - Allows us to refer to the definition later

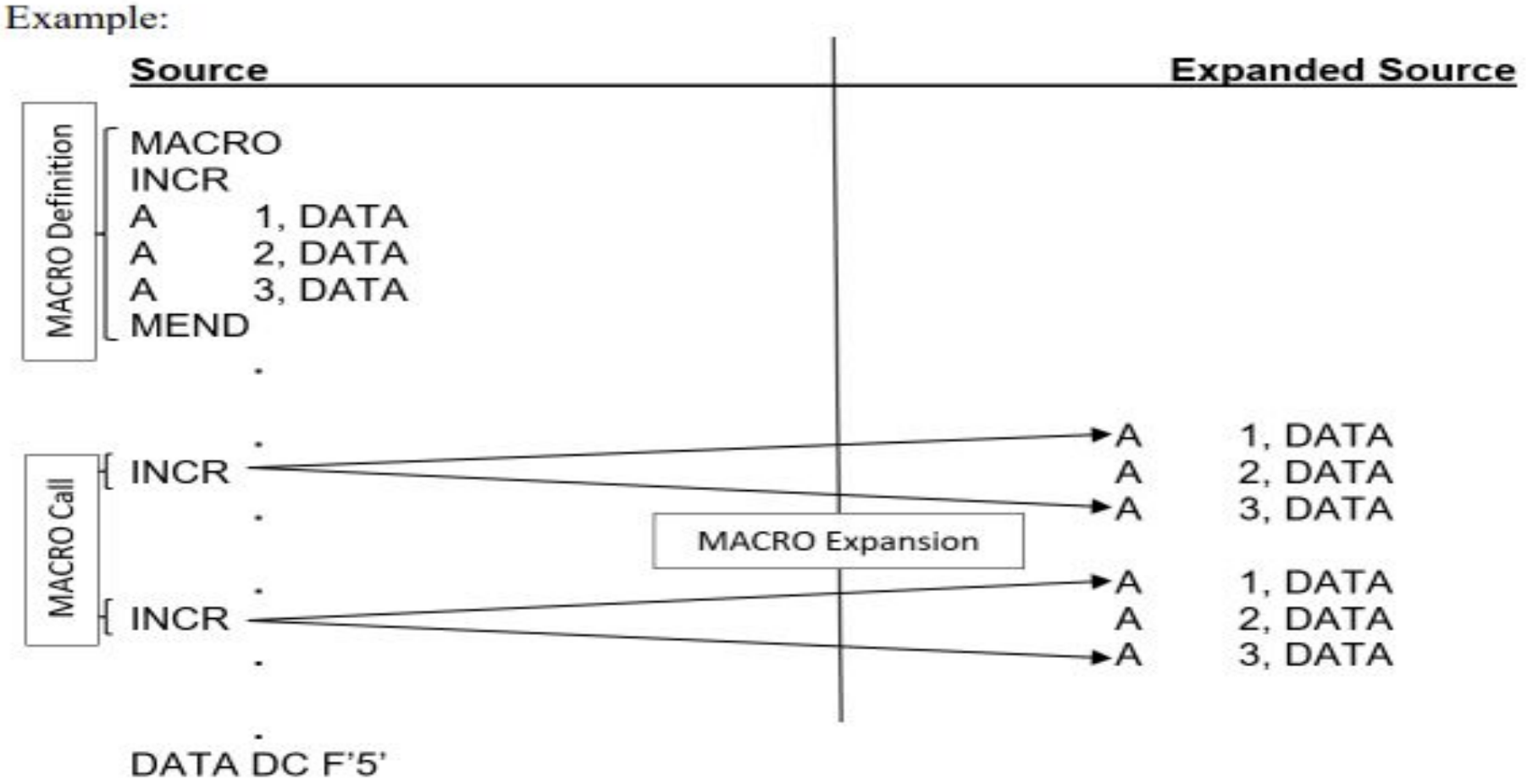
A macro processor effectively constitutes a separate language processor with its own language

General Format of Macro Definition:

General Format of Macro Definition:

Start of definition	→	**MACRO**
MACRO Name	→	| |
Sequence to be abbreviated	→	{ ____ ____ ____ }
End of definition	→	**MEND**

Example


Example:
Source
Expanded Source
MACRO Definition
MACRO
INCR
A 1, DATA
A 2, DATA
A 3, DATA
MEND
MACRO Call
INCR
INCR
MACRO Expansion
A 1, DATA
A 2, DATA
A 3, DATA
A 1, DATA
A 2, DATA
A 3, DATA
DATA DC F'5'

Macros

- The MACRO pseudo-op is the first line of the definition
- INCR is the macro instruction name.
- Following the name line is the sequence of instruction being abbreviated – instruction comprising the ‘macro’ instruction.
- The definition is terminated by a line with the MEND (macro end) pseudo-op.
- Once the macro has been defined, the use of macro name as an operation mnemonic in an assembly program is equivalent to the use of the corresponding sequence

Macro call

Macro Expansion

Macro vs Sub-Routine

Macro	Sub-Routine
MACRO name in the mnemonic field that leads to expansion only.	Subroutine name is a call statement in the program which leads to execution
MACROs are completely handled by the Assembler (Preprocessor) during assembly time.	Subroutine are completely handled by the hardware at runtime.
Macro definition and macro expansion are executed by the assembler. So, the assembler has to know all the features, options, and exceptions associated with them.	Hardware executes the subroutine call instruction. So, it has to know how to save the return address and how to branch to the subroutine.
The hardware knows nothing about macros.	The assembler knows nothing about subroutines
The macro processor generates a new copy of the macro and places it in the program.	The subroutine call instruction is assembled in the usual way and treated by the assembler as any other instruction

Macro vs Sub-Routine

Macro	Sub-Routine
Macro should be used for small tasks.	Functions should be used for the complex tasks.
Macro does not alter the flow of execution	Function does alters the flow of execution.
Macro calls a process at the translation time.	Function calls a process at the execution time.
Debugging in macro is difficult as compared to function as size of the code grows.	Debugging is comparatively easy.
Macros check the number of arguments; they do not check argument types.	Function checks both number of arguments and argument type

Benefits of Macros

- By defining the appropriate macro instructions, an assembly language programmer can create his own higher level facility in a convenient manner without any cost in control over the structure of his program
- The programmer can achieve the conciseness and ease in coding of high level languages without losing the basic advantage of assembly language programming.
- Integral macro operations simplify debugging and program modification and they facilitate standardization

Features of Macro Facility

Module-3

Macros & Macroprocessor

Features of Macro Facility/ Types of Macros

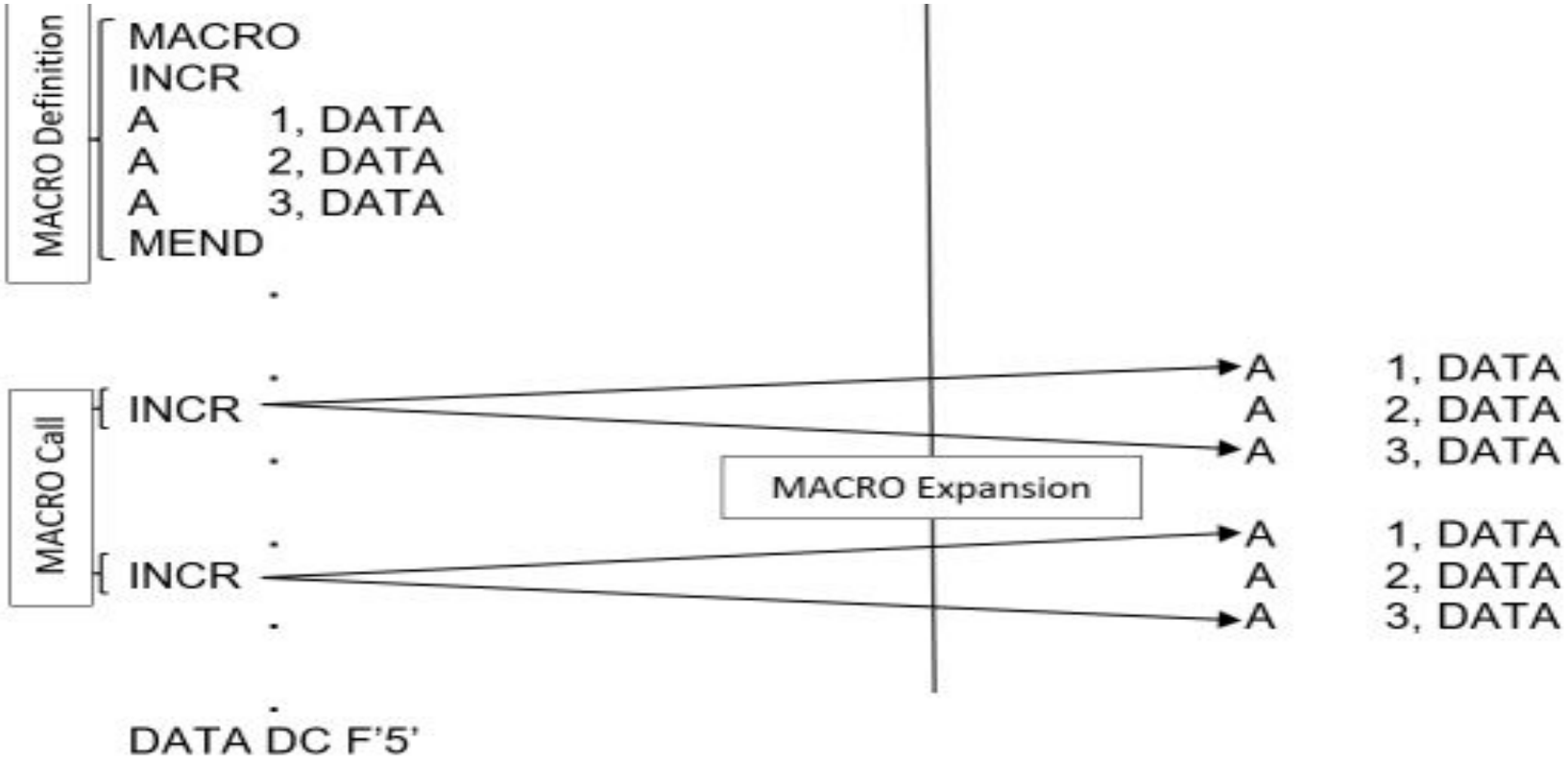
- Simple Macros
- Parameterized Macros
- Conditional Macros
- Nested Macros

Simple Macros

- The macro facility is capable of inserting blocks of instruction in place of macro calls.

- All of the calls to any given macro will be replaced by identical blocks.

Simple Macros



Simple Macros

- Disadvanatge:
- This macro facility lacks flexibility: there is no way for a specific macro call to modify the coding that replaces it.
- Solution:

Parameterized Macro

- It consists of the facility for providing for arguments, or parameters, in macro calls.
- Corresponding macro dummy arguments (preceded by ‘&’ symbol) will appear in macro definitions.

Parameterized Macro

Parameterized Macro

EXAMPLE 1

- Consider the sample program

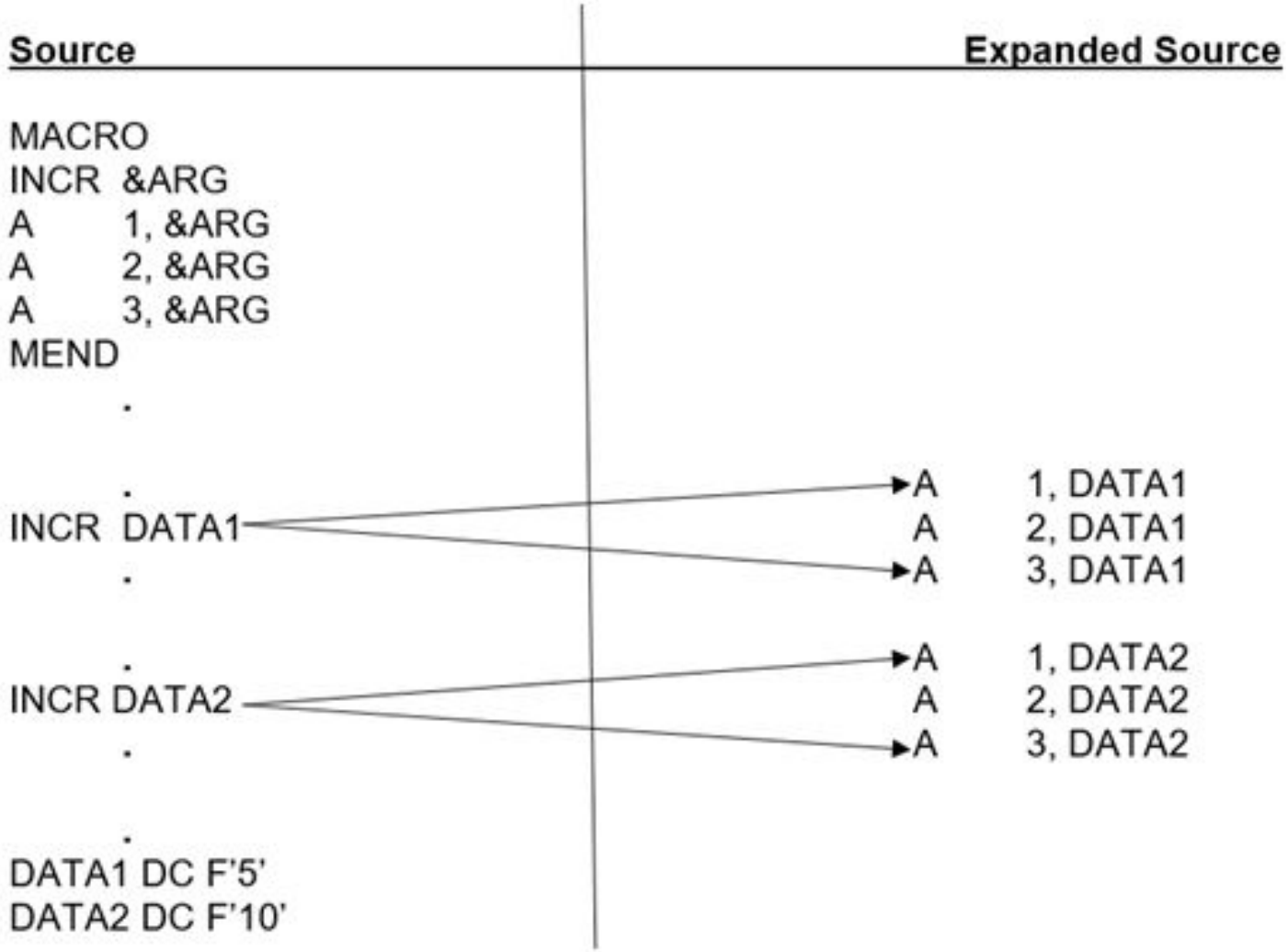
```
                .
        A       1, DATA1
        A       2, DATA1
        A       3, DATA1
                .
                .
                .
        A       1, DATA2
        A       2, DATA2
        A       3, DATA2
                .
                .
                .
DATA1   DC      F'5'
DATA2   DC      F'10'
```

Parameterized Macro

- In this case the instruction sequences are very similar but not identical.
- The first sequence performs an operation using DATA1 as operand; the second, using DATA2.
- They can be considered to perform the same operation with a variable, or argument.
- Such parameter is called as macro instruction argument or dummy argument.

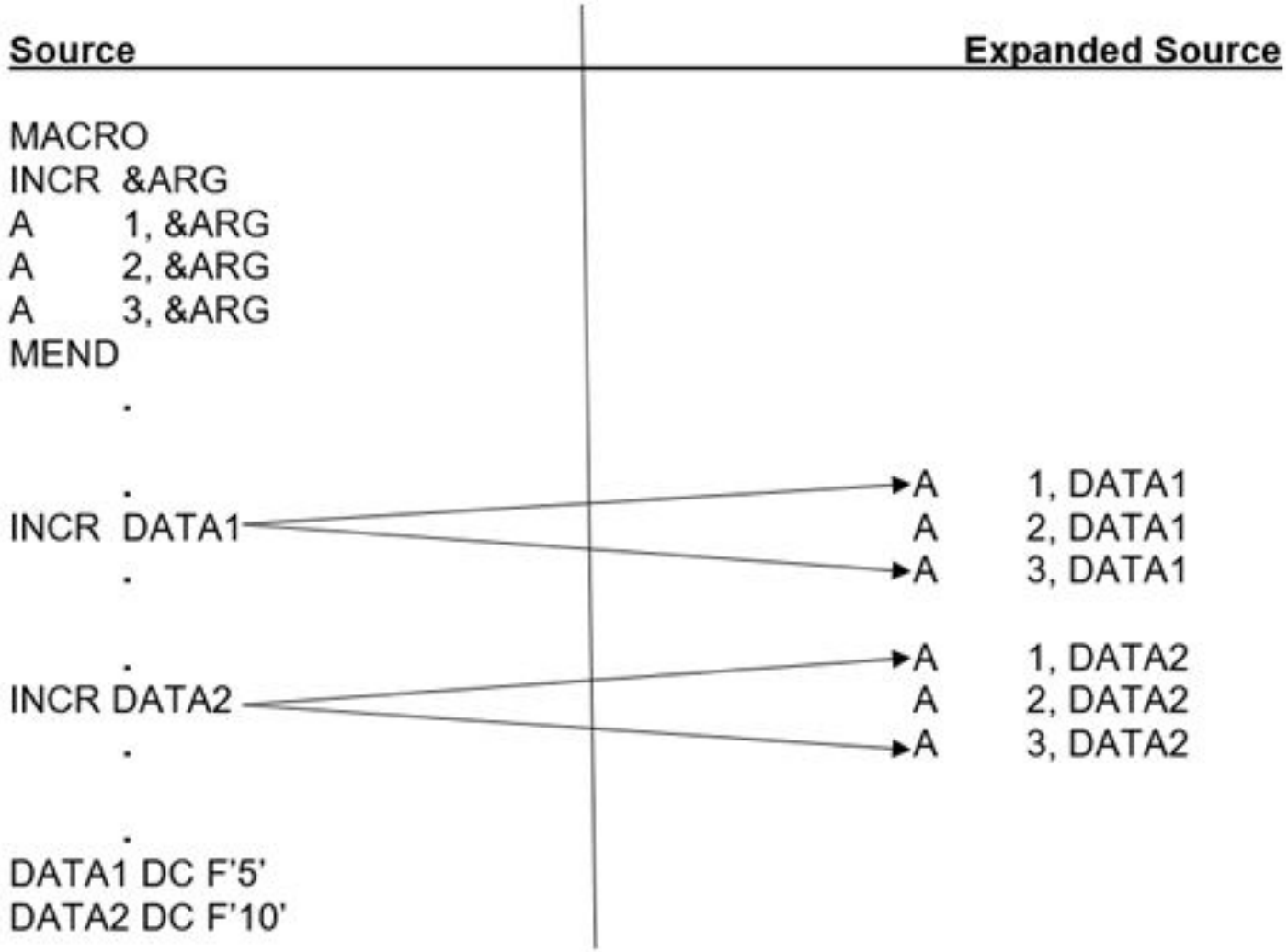
Parameterized Macro

Resultant Program



Parameterized Macro

Resultant Program



Parameterized Macro

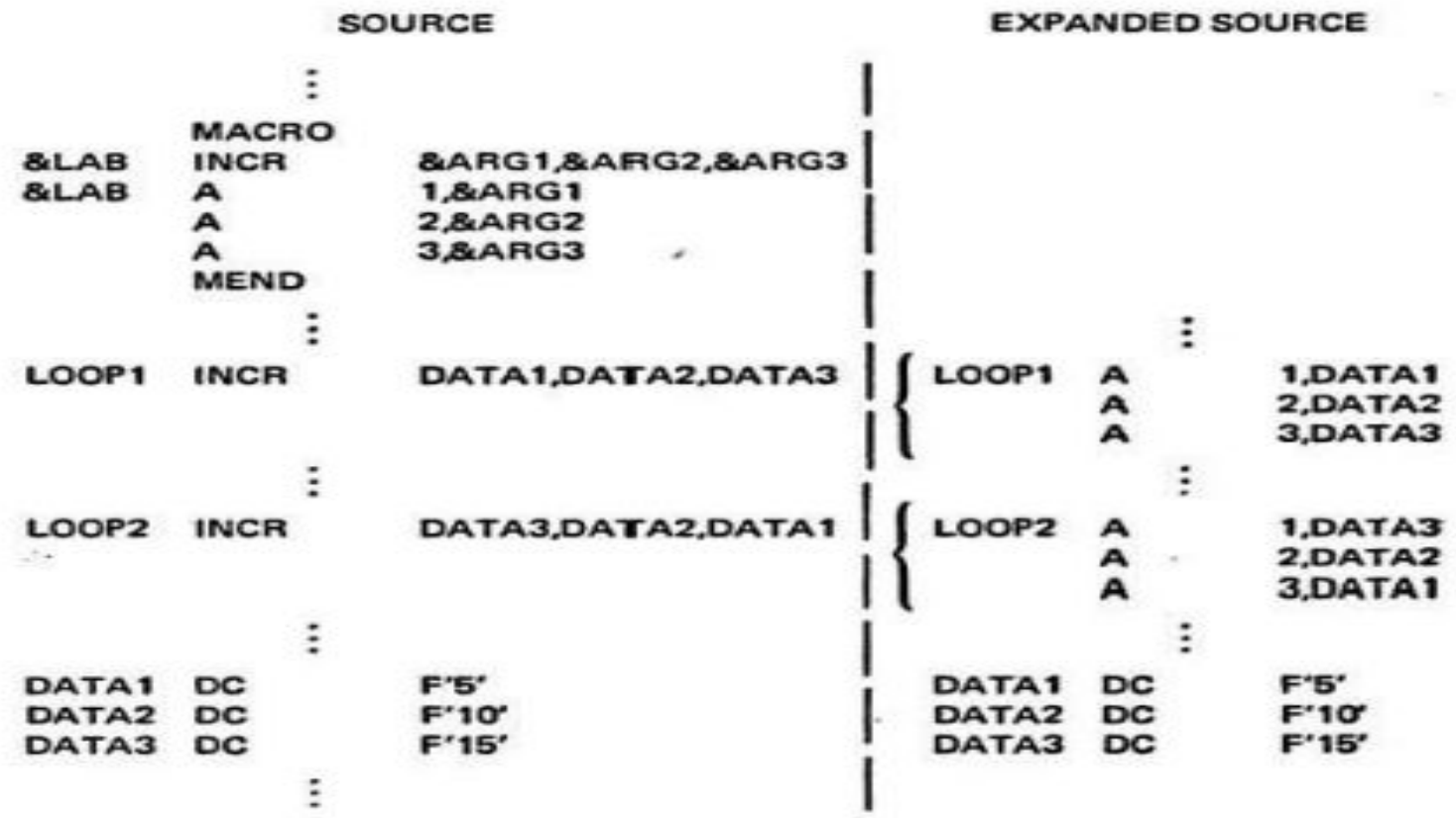
Parameterized Macro

- **EXAMPLE 2:**

```
          :
LOOP1     A       1, DATA1
          A       2, DATA2
          A       3, DATA3
          :
LOOP2     A       1, DATA3
          A       2, DATA2
          A       3, DATA1
          :
DATA1     DC      F'5'
DATA2     DC      F'10'
DATA3     DC      F'15'
```

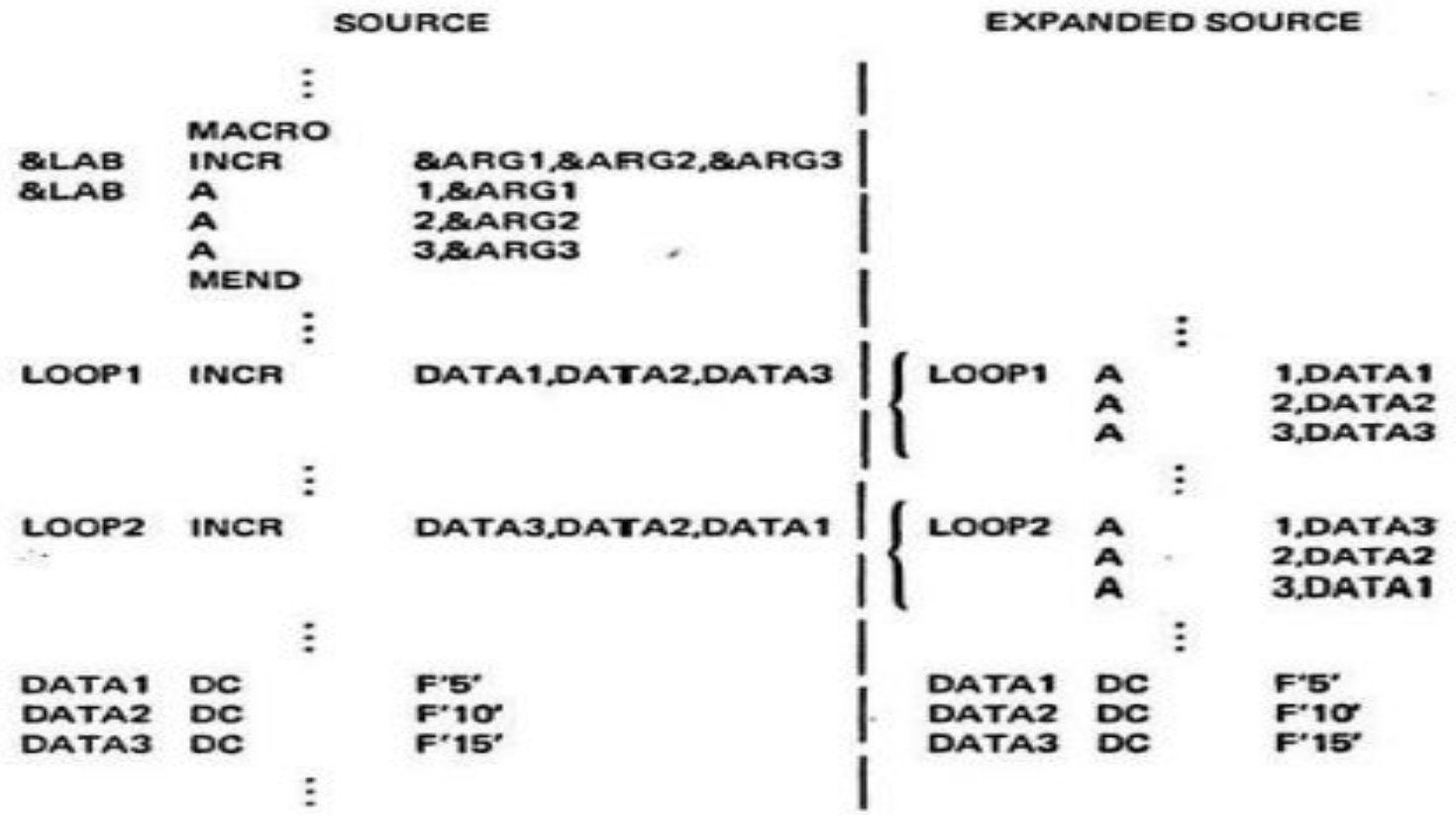
Parameterized Macro

- Resultant Program



Parameterized Macro

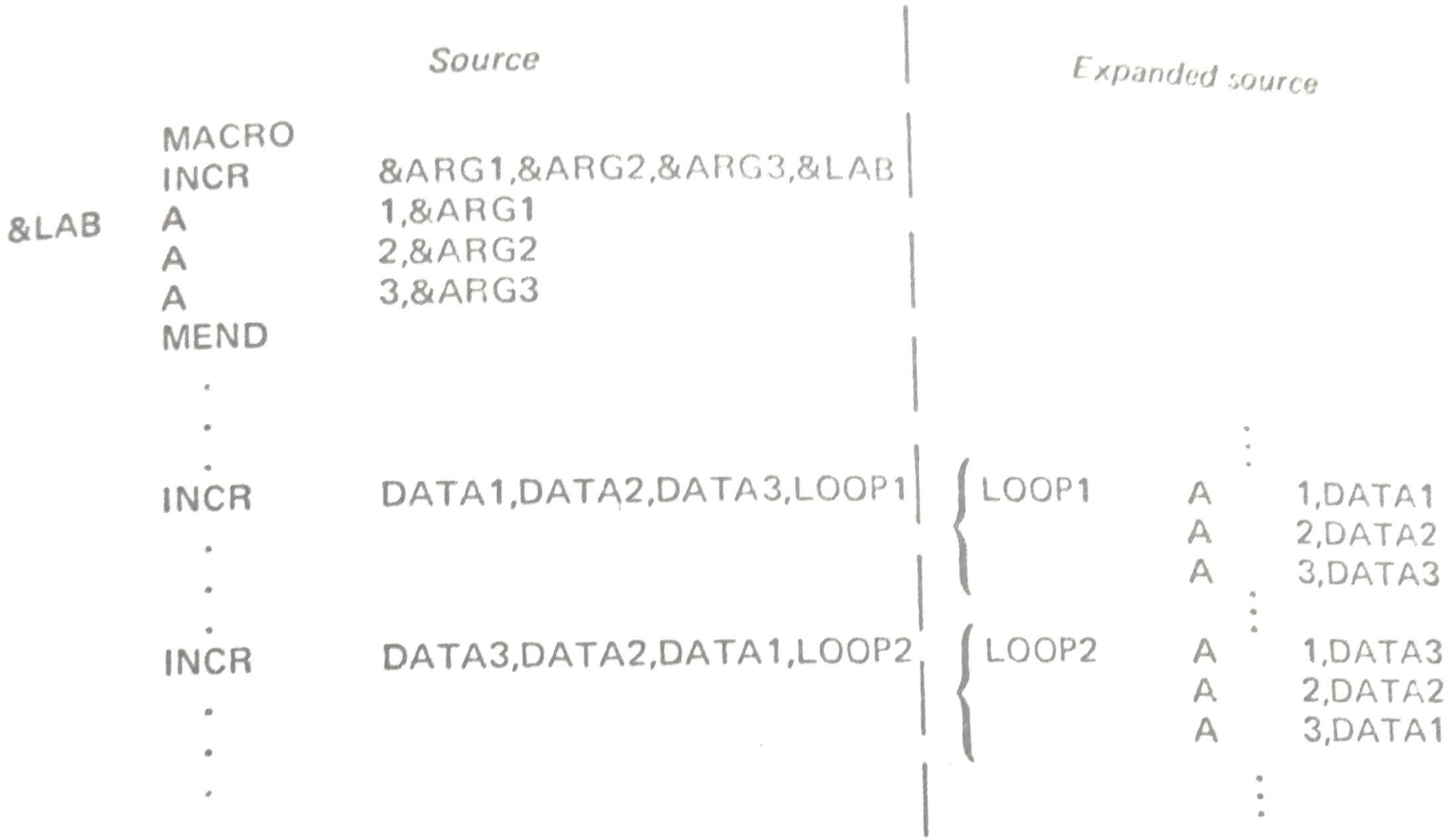
- Resultant Progr



```
                SOURCE                                  EXPANDED SOURCE
                  :                                     |
                MACRO                                   |
&LAB            INCR    &ARG1,&ARG2,&ARG3               |
&LAB            A       1,&ARG1                         |
                A       2,&ARG2                         |
                A       3,&ARG3                         |
                MEND                                    |
                  :                                     |
                  :                                     |
LOOP1           INCR    DATA1,DATA2,DATA3               |  LOOP1   A    1,DATA1
                                                        |          A    2,DATA2
                                                        |          A    3,DATA3
                  :                                     |
                  :                                     |
LOOP2           INCR    DATA3,DATA2,DATA1               |  LOOP2   A    1,DATA3
                                                        |          A    2,DATA2
                                                        |          A    3,DATA1
                  :                                     |
                  :                                     |
DATA1           DC      F'5'                            |  DATA1   DC   F'5'
DATA2           DC      F'10'                           |  DATA2   DC   F'10'
DATA3           DC      F'15'                           |  DATA3   DC   F'15'
                  :                                     |
```

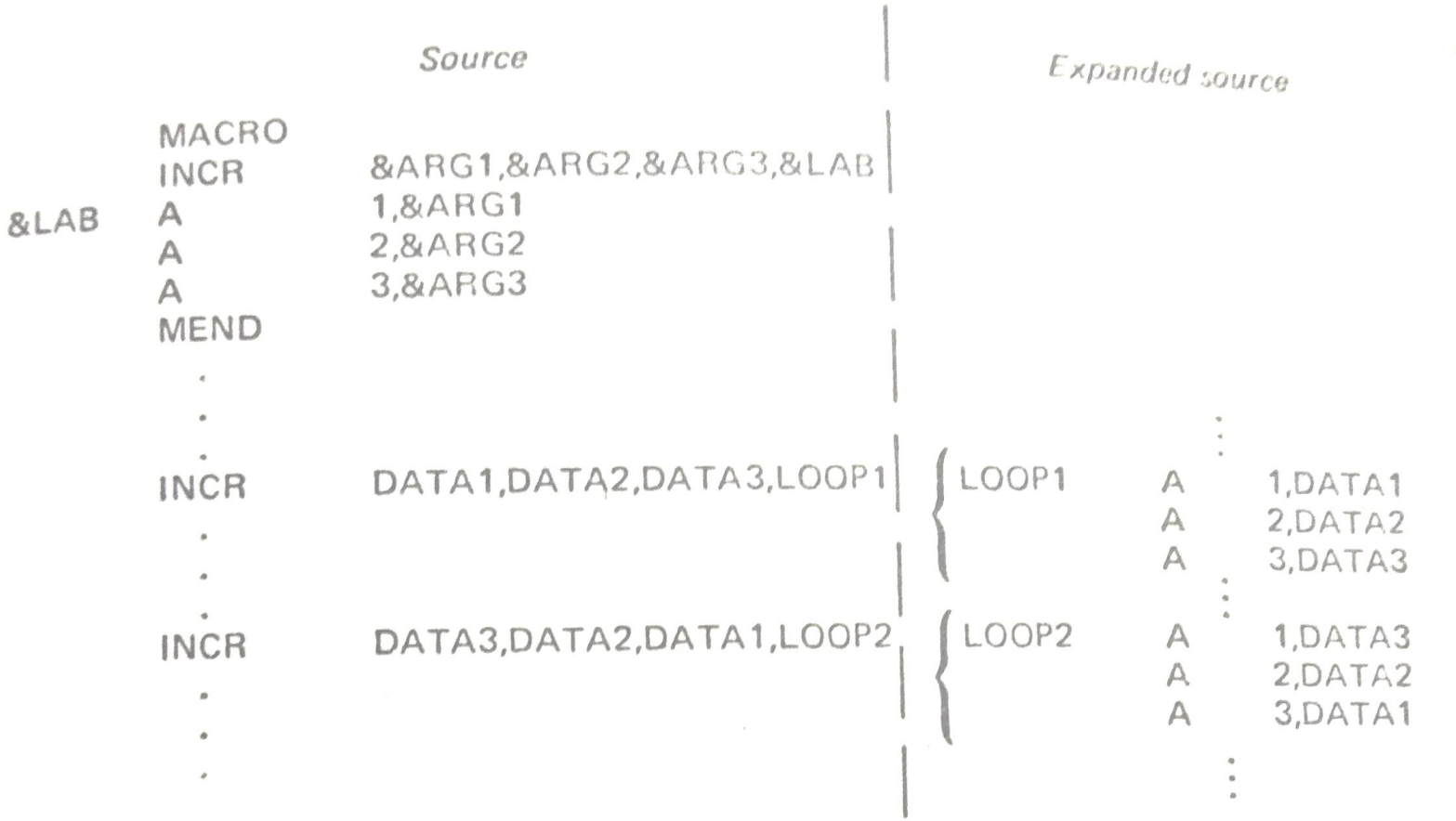
Parameterized Macro

Resultant Program



Parameterized Macro

Resultant Program



Parameterized Macro

- There are generally two ways of specifying arguments to a macro call:

- Positional arguments
- Keyword arguments

Parameterized Macro

- **Positional arguments:**
- In this arguments are matched with the dummy arguments according to the order in which they appear with the positional parameter
- E.g. INCR A, B, C

Parameterized Macro

- **Keyword arguments:**
- They allow reference to the dummy arguments by name as well as by the position.
- E.g.
- 'INCR &ARG1 = A, &ARG3 = C, &ARG2 = B' Or 'INCR &ARG1 = A, &ARG2 = &ARG3 = C'

Conditional Macros

- Two important macro processor pseudo-ops **AIF and AGO**, permit conditional reordering of the sequence of macro expansion.
- These allows conditional selection of the machine instruction that appears in expansions of macro call.

Conditional Macros

Conditional Macros

Conditional Macros

- Example:

```
        ⋮
LOOP1   A       1, DATA1
        A       2, DATA2
        A       3, DATA3
        ⋮
LOOP2   A       1, DATA3
        A       2, DATA2
        ⋮
LOOP3   A       1, DATA1
        ⋮
DATA1   DC      F'5'
DATA2   DC      F'10'
DATA3   DC      F'15'
        ⋮
```

Conditional Macros

- Resultant Program

```
                .
                .
                MACRO
&ARG0           VARY    &COUNT,&ARG1,&ARG2,&ARG3
&ARG0           A       1,&ARG1
                AIF     (&COUNT EQ 1).FINI          Test if & COUNT = 1
                A       2,&ARG2
                AIF     (&COUNT EQ 2).FINI          Test if & COUNT = 2
                A       3,&ARG3
.FINI           MEND
                .
                .
                .
LOOP1           VARY    3,DATA1,DATA2,DATA3
                .
                .
                .
LOOP2           VARY    2,DATA3,DATA2
                .
                .
                .
LOOP3           VARY    1,DATA1
                .
                .
                .
DATA1           DC      F'5'
DATA2           DC      F'10'
DATA3           DC      F'15'
```

Expanded source

```
                .
                .
                .
LOOP1           A       1,DATA1
                A       2,DATA2
                A       3,DATA3
                .
                .
LOOP2           A       1,DATA3
                A       2,DATA2
                .
                .
LOOP3           A       1,DATA1
                .
                .
```

Conditional Macros

Conditional Macros

Conditional Macros

Nested Macros

- In this feature of macro facility when one MACRO expansion is taking place we can encounter another macro call or during one macro call another macro call is invoked.

- In this macro, bodies contain macro calls and processes macro definition during expansion time

- It can be of two type:
  - Macro calls within Macros
  - Macro instructions defining Macros

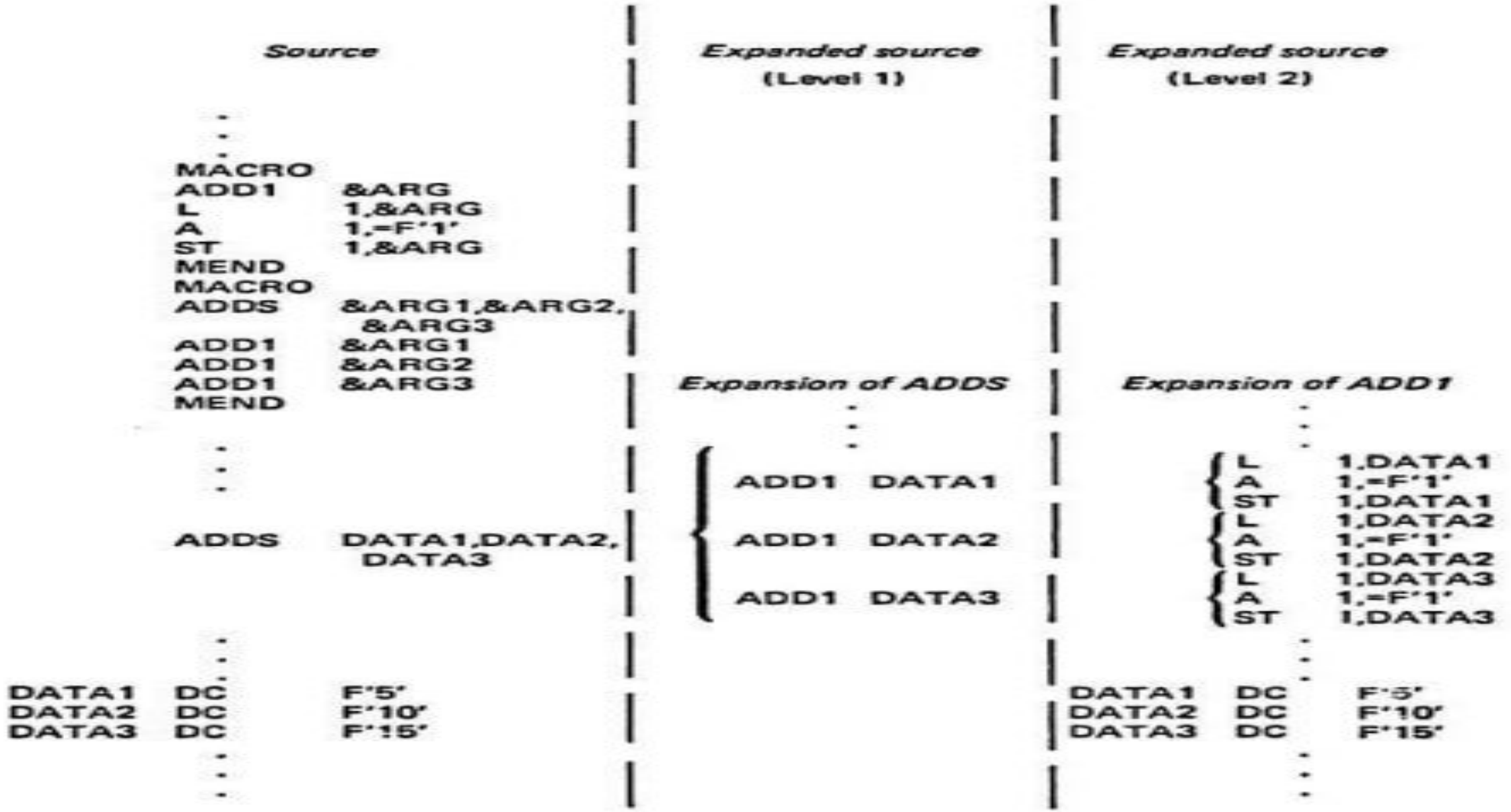
Nested Macros

Example

```
        MACRO
        ADD1    &ARG
        L       1,&ARG
        A       1,=F'1'
        ST      1,&ARG
        MEND
        MACRO
        ADDS    &ARG1,&ARG2,&ARG3
        ADD1    &ARG1
        ADD1    &ARG2
        ADD1    &ARG3
        MEND
```

Nested Macros

Resultant Program



Nested Macros

- A single macro instruction might be used to simplify the process of defining a group of similar macros.

- To call the inner macro it is necessary to define the outer macro first.

Nested Macros

Module-3

Macros & Macroprocessor

Tasks of a Macro Processor

Recognize Macro Definition	A macro instruction processor must recognize macro definitions identified by the MACRO and MEND pseudo-ops. This task can be complicated when macro definitions appear within macros.
Save the definitions	The processor must store the macro instruction definitions, which it will need for expanding macro calls.
Recognize Calls	The processor must recognize macro calls that appear as operation mnemonics. This suggests that macro names be handled as a type of op-code.
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 in then substituted for the macro call.

Design of a two pass Macroprocessor

Assumptions

Passes of Macroprocessor

Specification of Databases

Specification of Database Format

Algorithm

Assumptions:

1. Macro processor is functionally independent of the assembler and the output text from the macro processor will be fed to the assembler

2. Initially no macro calls or definitions within macros are permitted

3. In assembly language, lines are related to other lines by addressing

Example: A line can refer to other line by its address or name whichever is available to the assembler

Whereas in Maro language, lines are not so closely interrelated.

Macro definitions refer to nothing outside themselves and macro calls refer only to macro definitions

Working

Macro Processor Algorithm will take two passes over the input text:

1) Searching for MACRO definition
2) Searching for MACRO calls

Why two passes:

Macro processor cannot expand a macro call before having found and saved the corresponding macro definition.

So, two passes are required. One pass to handle definitions another to handle calls

Pass-1 and Pass 2

Pass-1

Pass-2

Specification of Databases

Pass 1 Databases:

1. The input macro source deck
2. The output macro source deck copy for use by pass 2
3. The Macro Definition Table (MDT), used to store the body of macro definitions
4. The Macro Name Table (MNT), used to store the names of defined macros
5. The Macro Definition Table Counter (MDTC), used to indicate the next available entry in the MDT
6. The Macro Name Table Counter (MNTC), used to indicate the next available entry in the MNT
7. The Argument List Array (ALA), used to substitute index markers for dummy arguments before storing a macro definition

Specification of Databases

Pass 2 Databases:

1. The copy of the input macro source deck
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 Pointer (MDTP), 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 the stored macro definition

Databases of Pass 1 and Pass 2

MODULE-3

Macros & Macroprocessor

Databases of Pass 1 and Pass 2 / Data structures

Pass - 1 : MNT, MDT

Pass - 2 : ALA

Specification of Database Format

- Argument List Array(ALA)
- Macro Definition Table (MDT)
- Macro Name Table (MNT)

Argument List Array(ALA)

- The Argument List Array (ALA) is used during both pass 1 and pass 2 but for somewhat reverse functions.
- During pass 1, in order to simplify later the argument replacement during macro expansion, dummy arguments in the macro definition are replaced with positional indicators when the definition is stored.
- During pass 2 it is necessary to substitute macro call arguments for the index markers stored in the macro definition

Argument List Array(ALA)

- During Pass 1: The ith dummy argument on the macro name card is represented in the body of the macro by the index marker symbol #

- #i is a symbol reserved for the use of the macro processor

- These symbols are used in conjunction with the argument list prepared before expansion of a macro call.

- The symbolic dummy arguments are retained on the macro name card to enable the macro processor to handle argument replacement by name rather than by position.

ARGUMENT LIST ARRAY(ALA) EXAMPLE

Sample + Program

```
LOOP1   A     1, DATA1
        A     2, DATA2
        A     3, DATA3
        .
        .
        .
LOOP2   A     1, DATA3
        A     2, DATA2
        A     3, DATA1
        .
        .
        .
DATA1   DC    F'5'
DATA2   DC    F'10'
DATA3   DC    F'15'
```

macro name

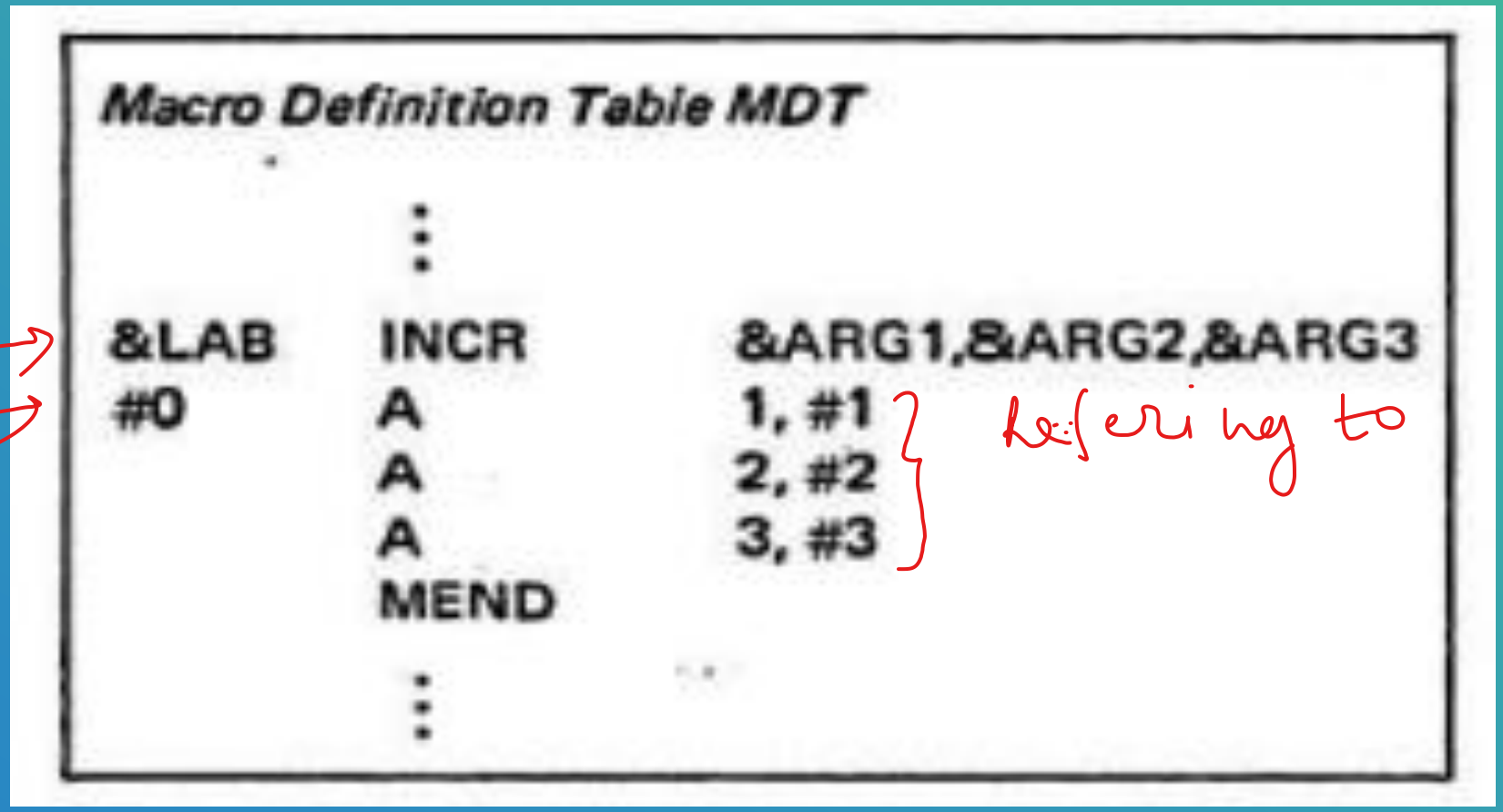
macro definition

Source	Expanded Source
MACRO	
INCR &ARG	
A 1, &ARG	
A 2, &ARG	
A 3, &ARG	
MEND	
.	
macro call	macro expansion
INCR DATA1	A 1, DATA1
	A 2, DATA1
	A 3, DATA1
.	
INCR DATA2	A 1, DATA2
	A 2, DATA2
	A 3, DATA2
.	
DATA1 DC F'5'	
DATA2 DC F'10'	

set of instructions

ARGUMENT LIST ARRAY(ALA) EXAMPLE

In Pass-1 all macro definitions will be saved in the MDT

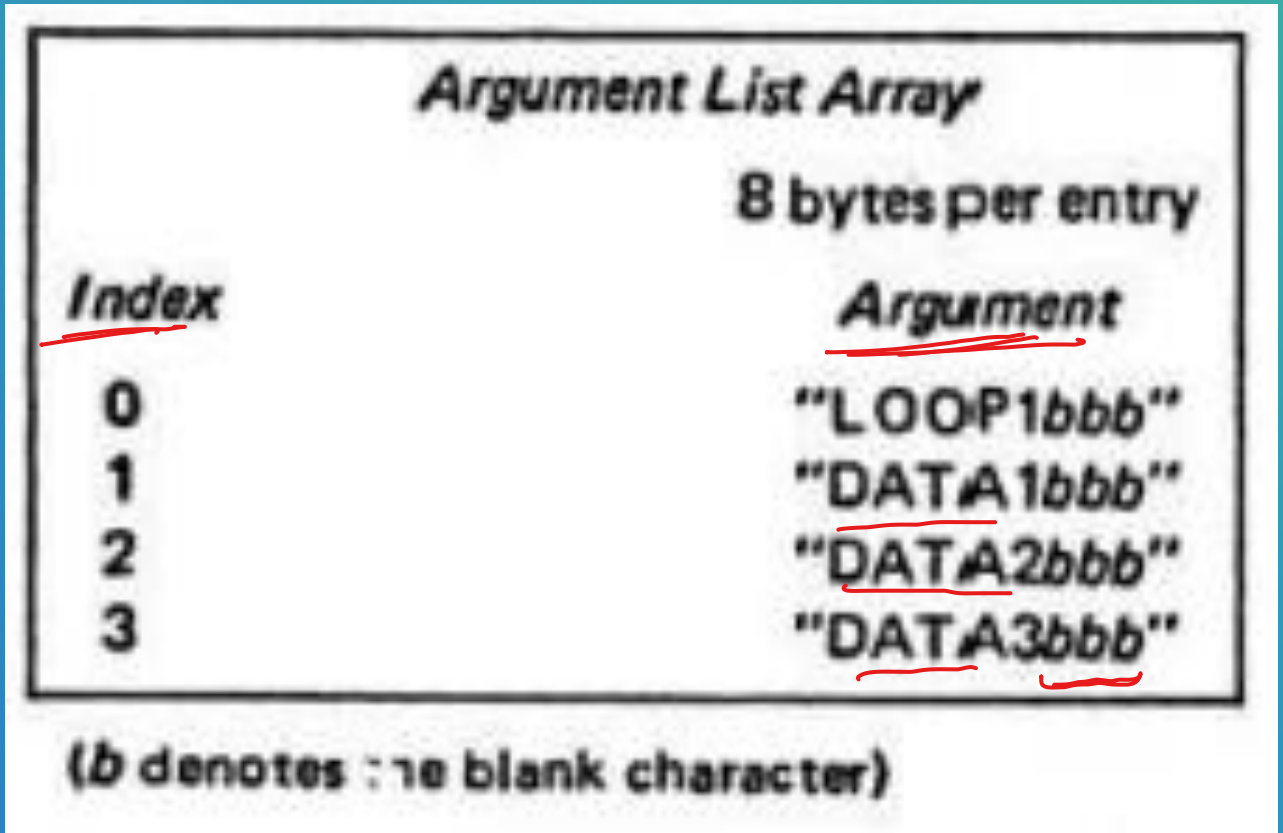


ARGUMENT LIST ARRAY(ALA) EXAMPLE

In Pass-2 upon encountering the call

LOOP INCR DATA1, DATA2, DATA3

The macro call expander would prepare an argument list array:



The list would be used only while expanding this particular call.

ARGUMENT LIST ARRAY(ALA) EXAMPLE

Suppose that a succeeding call is:

```
INCR     &ARG1=DATA3,&ARG2=DATA2,&ARG3=DATA1
```

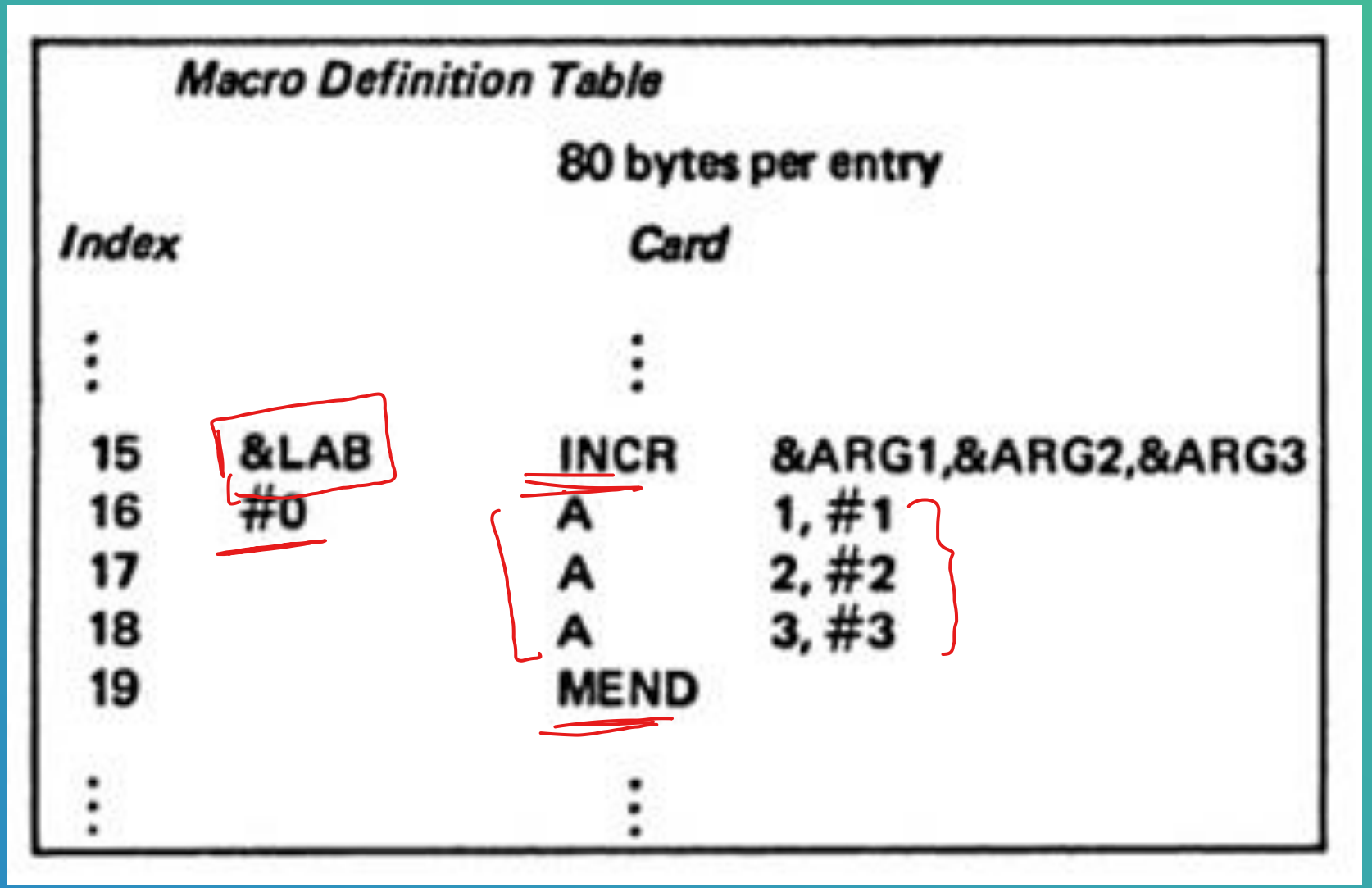
Argument List Array	
Index	**Argument**
0	"bbbbbbbb" (all blank)
1	"DATA3bbb"
2	"DATA2bbb"
3	"DATA1bbb"

Macro Definition Table (MDT)

- The Macro Definition Table (MDT) is a table of text lines;
- If input is from 80-column cards, the MDT can be a table with 80-byte string as entries.
- Every line of each macro definition, except the MACRO line, is stored in the MDT.
- The MEND is kept to indicate the end of the definition;
- The macro name line is retained to facilitate keyword argument replacement

MACRO DEFINITION TABLE (MDT) EXAMPLE

The INCR macro will be stored as follows:



Macro Name Table (MNT)

The Macro Name Table (MNT) serves a function very similar to that of the assembler's Machine-Op Table (MOT) and Pseudo-Op Table (POT).

Each MNT entry consists of a character string (the macro name) and a pointer (index) to the entry in the MDT that corresponds to the beginning of the macro definition

MACRO NAME TABLE (MNT) EXAMPLE

The MNT entries for INCR macro will be as follows:

3 columns

Index	8 bytes Name	4 bytes MDT index
⋮	⋮	⋮
3	"INCRbbbb"	15
⋮	⋮	⋮

Algorithm

PASS 1- MACRO DEFINITION:

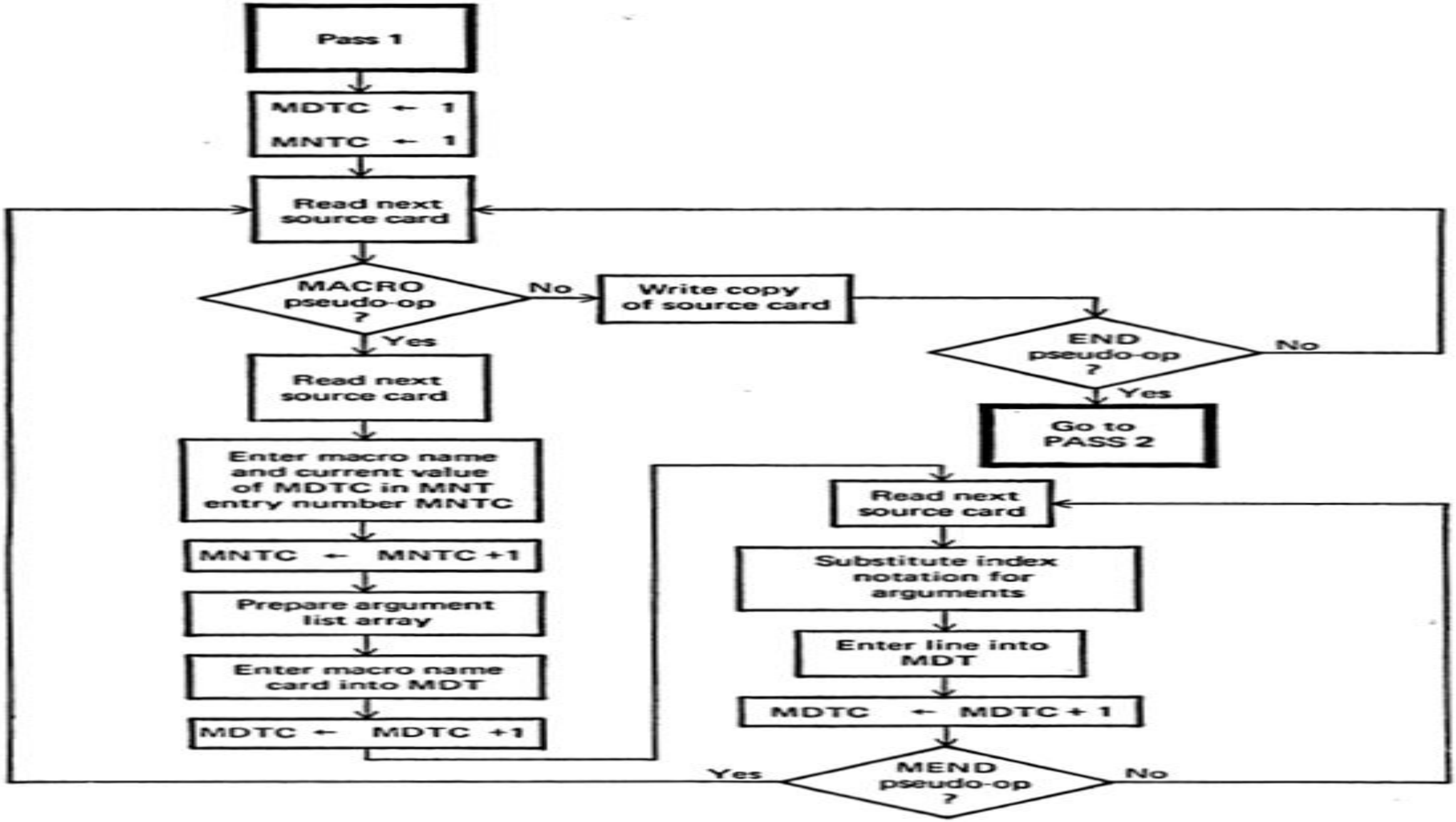
1. The algorithm for pass 1 tests each input line.

2. If it is a MACRO pseudo-op, the entire macro definition that follows is saved in the next available locations in the Macro Definition Table (MDT).

3. The first line of the definition is the macro name line. The name is entered into the Macro Name Table (MNT), along with a pointer to the first location of the MDT entry of the definition.

4. When the END pseudo-op is encountered, all of the macro definitions have been processed so control transfers to pass 2 in order to process macro calls.

Flow Chart for Pass 1 - Processing of Macro definitions



Algorithm

PASS 2- MACRO CALLS AND EXPANSION:

1. The algorithm for pass 2 tests the operation mnemonic of each input line to see if it is a name in the MNT.
2. When a call is found, the processor sets a pointer, the Macro Definition Table Pointer (MDTP), to the corresponding macro definition stored in the MDT. The initial value of the MDTP is obtained from the "MDT index" field of the MNT entry.
3. The macro expander prepares the Argument List Array (ALA) consisting of a table of dummy argument indices and corresponding arguments to the call.

This list is simply a succession of symbols ordered to match the dummy arguments on the name card (the first is the label argument, which is considered to have an index or zero).

Flow Chart for Pass 2 - Processing of Macro calls and expansion

Macros and Macro Processor

MODULE 3

Content

- Introduction
- Macro Definition and Macro Call
- Features of Macro Facility
- Design of Two Pass Macro Processor
- Data Structures used

Macros/Macro Instruction

- Single Line abbreviation for group of instructions in sequential order
- Single Instruction represent BLOCK OF CODE
- Every Occurrence of this One Line : Substitute Entire Block
- Macro Processor: Extension of Basic Assembler Algorithm

Macros/ Macro Instruction

Example 1

```
. . .
. . .
. . .
A   1,  DATA
A   2,  DATA
A   3,  DATA
. . .
. . .
. . .
A   1,  DATA
A   2,  DATA
A   3,  DATA
. . .
. . .
DATA  DC  F'5'
```

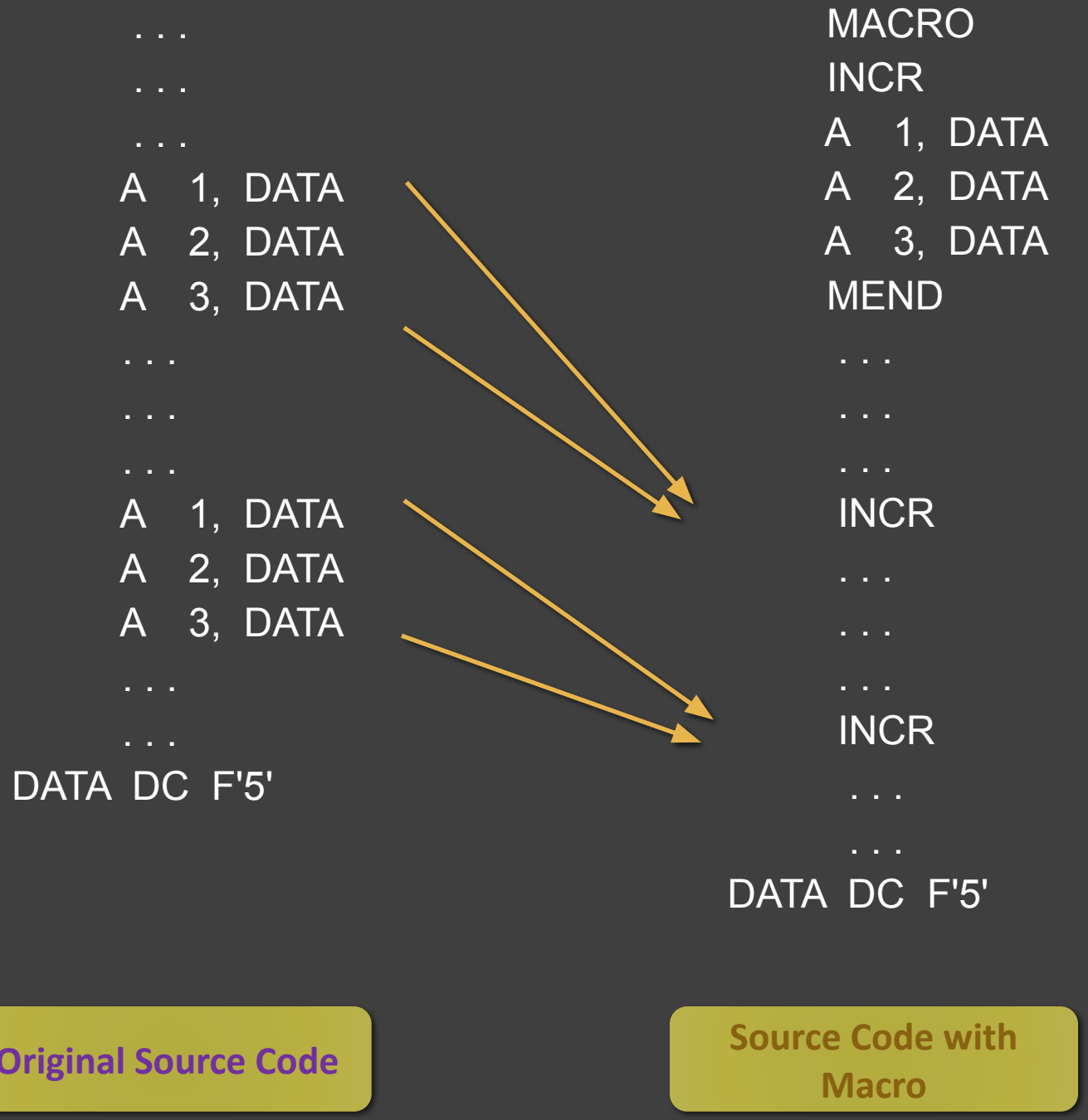
- Set of instructions occurred twice: Macro can be defined
- Macro Processor: A Separate Language Processor with its own language

Macro Instruction Definition

MACRO	(Start of Definition)
[. . .]	(Name of Macro)
. . .	
. . .	(Sequence to be Abbreviated)
. . .	
. . .	
MEND	(End of Definition)

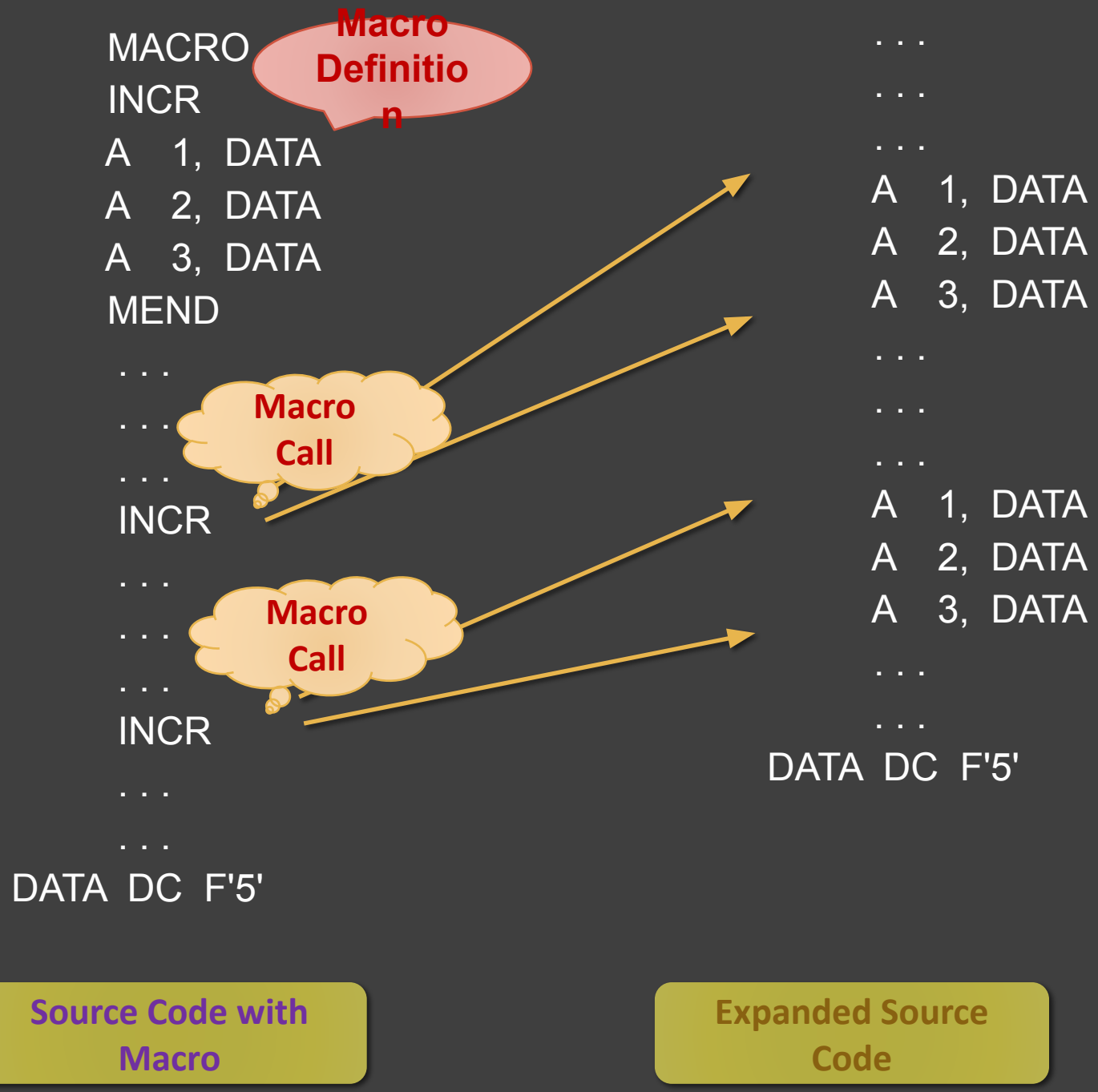
- Use of Macro Name: Operation Mnemonic in Assembly Program
- Equivalent to use of corresponding instruction sequence

Macro Instruction Definition



Macro Expansion

- Process of replacing Macro: MACRO EXPANSION
- Occurrence of Macro name in source program: MACRO CALL
- Macro Definition doesn't appear in the expanded source code



Features of Macro Facility

SIMPLE

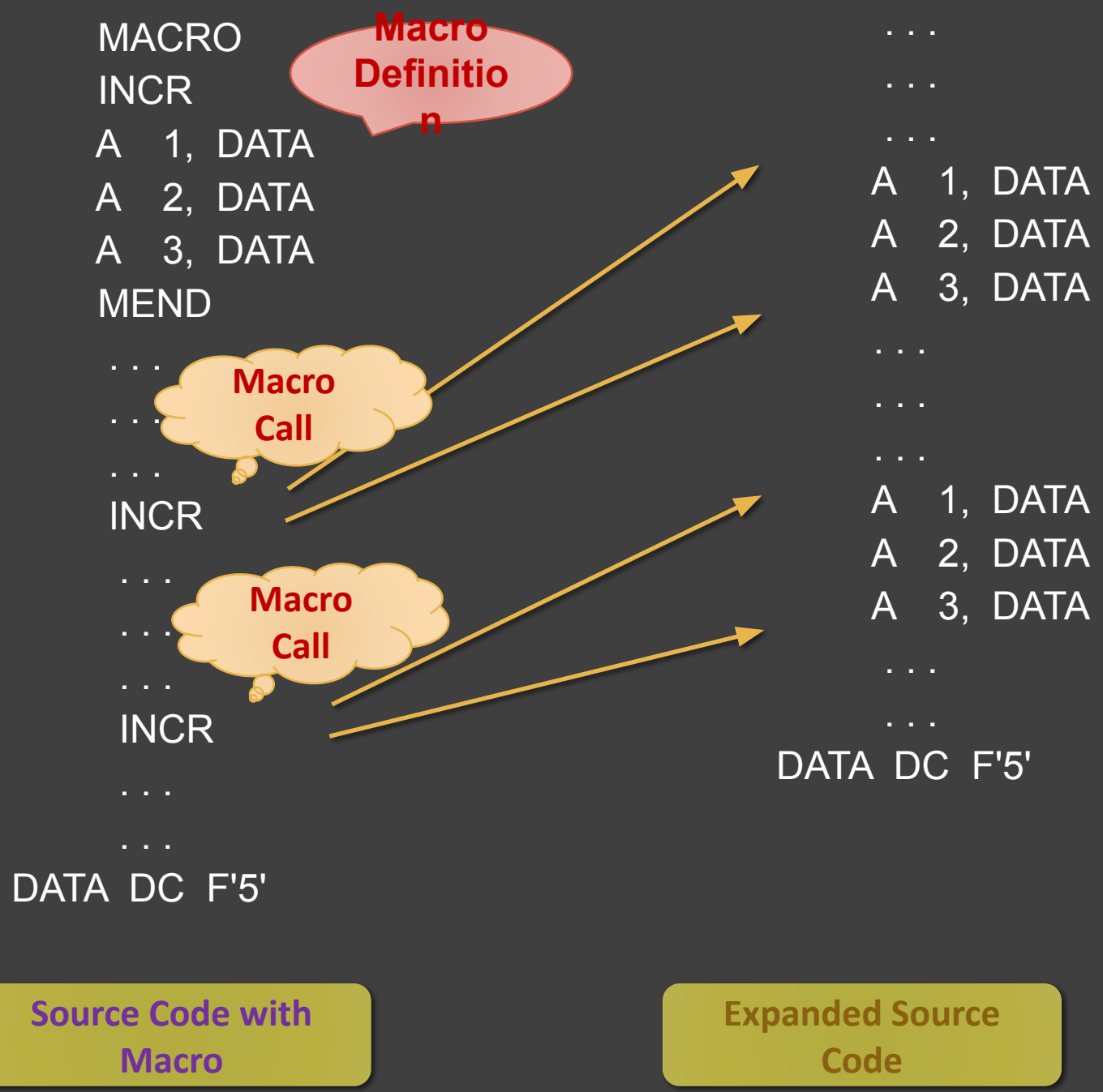
PARAMETERIZED

CONDITIONAL

NESTED

1. Simple Macro

- Process of replacing Macro Call with Macro Definition
- No Other processing or replacement is required



2. Parameterized Macro

(Macro Instruction Argument)

Example 2

```
        . . .
        . . .
        . . .
        A    1,  DATA1
        A    2,  DATA1
        A    3,  DATA1
         . . .
         . . .
         . . .
        A    1,  DATA2
        A    2,  DATA2
        A    3,  DATA2
        . . .
        . . .
DATA1   DC   F'5'
DATA2   DC   F'10'
```

- Instructions are very similar but NOT Identical
- Operations are same but with different parameter
- Such parameter is called MACRO INSTRUCTION ARGUMENT or DUMMY ARGUMENT

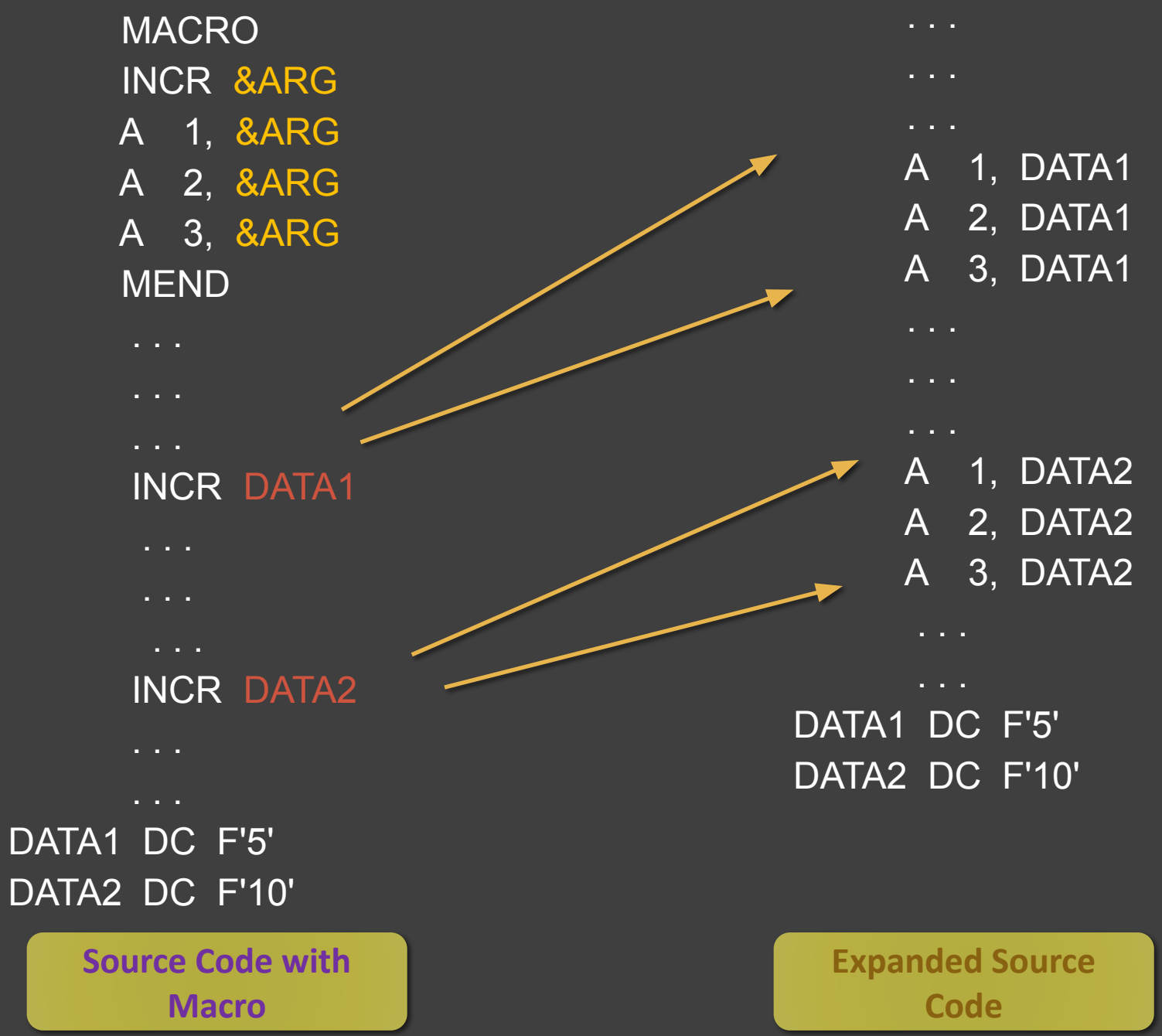
Dummy Arguments

✔ Specified on Macro Name Line

✔ Distinguished by '&' symbol which is always its first character

2. Parameterized Macro

- More than one argument can be supplied for macro call
- Each argument must correspond to definition Dummy Argument on Macro Name line of Macro Definition
- When a Macro Call is processed, the arguments are



2. Parameterized Macro

(Macro Instruction Argument)

Example 3

```
. . .
. . .
LOOP1   A   1, DATA1
        A   2, DATA2
        A   3, DATA3
. . .
. . .
. . .
LOOP2   A   1, DATA3
        A   2, DATA2
        A   3, DATA1
. . .
. . .
DATA1   DC  F'5'
DATA2   DC  F'10'
DATA3   DC  F'15'
```

- Program can be rewritten using MACRO with MORE THAN ONE dummy arguments
- Label arguments can be put in Label Location

2. Parameterized Macro

```
        MACRO
&LAB    INCR  &ARG1, &ARG2, &ARG3
&LAB    A   1,  &ARG1
        A   2,  &ARG2
        A   3,  &ARG3
        MEND
         . . .
         . . .
         . . .
LOOP1   INCR  DATA1, DATA2, DATA3
         . . .
         . . .
         . . .
LOOP2   INCR  DATA3, DATA2, DATA1
         . . .
         . . .
DATA1   DC  F'5'
DATA2   DC  F'10'
DATA3   DC  F'15'
```

Source Code with Macro

```
         . . .
         . . .
         . . .
LOOP1   A   1,  DATA1
        A   2,  DATA2
        A   3,  DATA3
         . . .
         . . .
         . . .
LOOP2   A   1,  DATA3
        A   2,  DATA2
        A   3,  DATA1
         . . .
         . . .
DATA1   DC  F'5'
DATA2   DC  F'10'
DATA3   DC  F'15'
```

Expanded Source Code

2. Parameterized Macro

- Label Arguments can be declared with other arguments in Macro Definition
- The way of Specifying argument with their position in Macro Definition is known as POSITIONAL

```
        MACRO
&LAB    INCR  &ARG1, &ARG2, &ARG3
&LAB    A    1,  &ARG1
        A    2,  &ARG2
        A    3,  &ARG3
        MEND
         . . .
         . . .
LOOP1   INCR  DATA1, DATA2, DATA3
         . . .
LOOP2   INCR  DATA3, DATA2, DATA1
```

Label Argument at its position

```
        MACRO
        INCR  &ARG1, &ARG2, &ARG3, &LAB
&LAB    A    1,  &ARG1
        A    2,  &ARG2
        A    3,  &ARG3
        MEND
         . . .
         . . .
        INCR  DATA1, DATA2, DATA3, LOOP1
        . . .
        INCR  DATA3, DATA2, DATA1, LOOP2
```

Label argument used as general argument

2. Parameterized Macro

KEYWORD ARGUMENTS:

- It allows reference to dummy arguments by NAME as well as POSITION
- Both the calls are valid and performing same expansion
- If any argument is not supplied, then it is presumed blank by

```
MACRO
INCR  &ARG1, &ARG2, &ARG3
A    1,  &ARG1
A    2,  &ARG2
A    3,  &ARG3
MEND
 . . .
 . . .
INCR  &ARG1 = A, &ARG2 = B, &ARG3 = C
 . . .
INCR  &ARG1 = A, &ARG3 = C, &ARG2 = B
```

3. Conditional Macro Expansion

There are two pseudo-ops to be used for defining conditional reordering of MACRO instructions

1. AIF –
   - It is conditional branch pseudo-op
   - It performs arithmetic test and branches only if tested condition is true
2. AGO –
   - It is an unconditional branch on pseudo-ops
   - on 'goto' statement

AIF and AGO control the sequence in which the macro processor expand the statement in MACRO Instruction

Labels starting with period (Example .FINI) are Macro Labels and do not appear in output of Macro Processor

3. Conditional Macro Expansion

Example 4

In this example, the operand, labels and number of instructions generated change in each sequence

```
        . . .
        . . .
        . . .
LOOP1   A    1,  DATA1
        A    2,  DATA2
        A    3,
DATA3           . . .
        . . .
        . . .
LOOP2   A    1,  DATA3
        A    2,  DATA2
        . . .
          . . .
          . . .

LOOP3   A  1, DATA1
          . . .
          . . .
DATA1   DC  F'5'
DATA2   DC  F'10'
DATA3   DC  F'15'
```

Original Source Code

```
          MACRO
&ARG0 VARY &COUNT, &ARG1, &ARG2, &ARG3
&ARG0   A  1,  &ARG1
          AIF (&COUNT EQ 1) .FINI
          A  2,  &ARG2
          AIF (&COUNT EQ 2).FINI
          A    3,  &ARG3
.FINI     MEND
          . . .
          . . .
LOOP1   VARY 3, DATA1, DATA2, DATA3
          . . .
          . . .
LOOP2   VARY 2, DATA3, DATA2
          . . .
          . . .
LOOP3   VARY 1, DATA1
          . . .
          . . .
 DATA1   DC  F'5'
 DATA2   DC  F'10'
 DATA3   DC  F'15'
```

Source Code with MACRO

3. Conditional Macro Expansion

Example 4

In above program, if AIF condition is true then control is transferred to Macro label .FINI

```
        MACRO
&ARG0 VARY &COUNT, &ARG1, &ARG2, &ARG3
&ARG0  A  1,  &ARG1
        AIF (&COUNT EQ 1) .FINI
        A  2,  &ARG2
        AIF (&COUNT EQ 2).FINI
        A   3,  &ARG3
.FINI   MEND
         . . .
         . . .
LOOP1  VARY 3, DATA1, DATA2, DATA3
        . . .
        . . .
LOOP2  VARY 2, DATA3, DATA2
        . . .
        . . .
LOOP3  VARY 1, DATA1
        . . .
        . . .
DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

Source Code with MACRO

```
                . . .
                . . .
                . . .
LOOP1   A    1,  DATA1
        A    2,  DATA2
        A    3,  DATA3
. .
                . . .
                . . .
LOOP2   A    1,  DATA3
        A    2,  DATA2
                . . .
                . . .
                . . .

LOOP3   A  1, DATA1
                . . .
                . . .
DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

MACRO Expansion

4. Macro Calls within Macros Example 5

Within definition of the macro 'ADDS' are three separate calls to previously defined macro 'ADD1'. Due to this, length of Macro ADDS is shortened

```
MACRO
ADD1  &ARG
L    1,  &ARG
A    1,   =F'1'
ST  1,   &ARG
MEND
MACRO
ADDS &ARG1, &ARG2, &ARG3
ADD1  &ARG1
ADD1  &ARG2
ADD1  &ARG3
MEND
```

4. Macro Calls within Macros Example 5

```
MACRO
ADD1  &ARG
L    1,  &ARG
A    1,   =F'1'
ST  1,   &ARG
MEND
```

```
MACRO
ADDS &ARG1, &ARG2, &ARG3
ADD1  &ARG1
ADD1  &ARG2
ADD1  &ARG3
MEND
. . .
. . .
ADDS  DATA1, DATA2, DATA3
. . .
. . .
DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

Source Code

Expanded Source Code (Level1)

```
ADD1  DATA1
ADD1  DATA2
ADD1  DATA3
. . .
. . .
DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

Expanded Source Code (Level2)

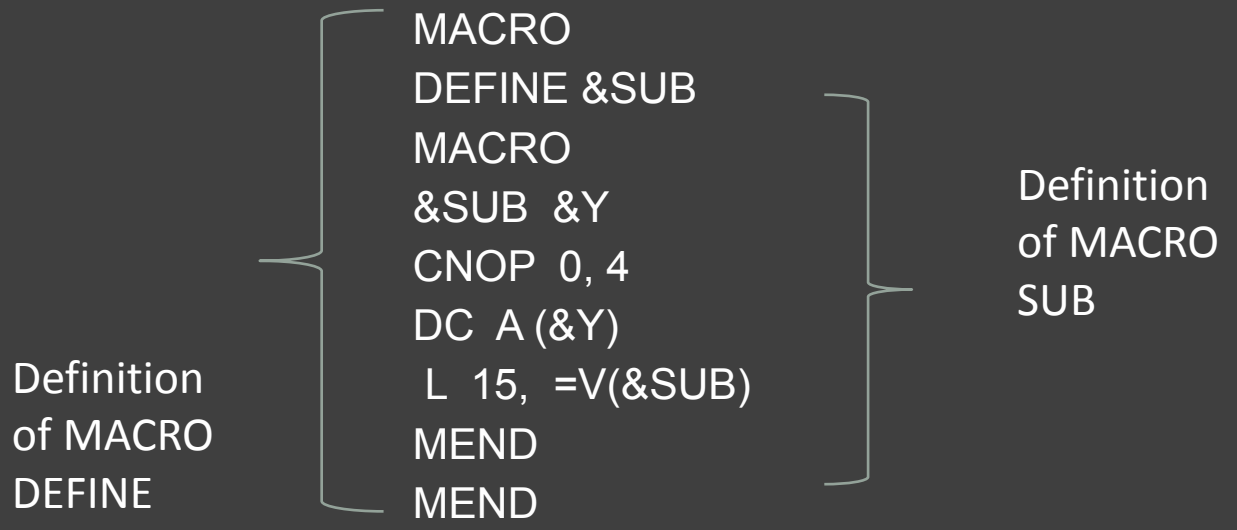
```
L    1,  DATA1
A    1,   =F'1'
ST  1,   DATA1
L    1,  DATA2
A    1,   =F'1'
ST  1,   DATA2
L    1,  DATA3
A    1,   =F'1'
ST  1,   DATA1
. . .
. . .
DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

5. Macro Instruction defining Macros Example 6

Macro definition can also be abbreviated using Macros.

It also be called as 'Macro Definition within Macros'

Single Macro Instructions can be used to simplify the process of defining a group of similar macros



5. Macro Instruction defining Macros Example 6

```
MACRO
DEFINE &SUB
MACRO
&SUB  &Y
CNOP  0, 4
DC  A (&Y)
 L  15,  =V(&SUB)
MEND
MEND
. . .
. . .
DEFINE COS
. . .
. . .
COS AR
```

Source Code

```
MACRO
COS  &Y
CNOP  0, 4
DC  A (&Y)
 L  15,  =V(COS)
MEND
. . .
. . .
COS AR
```

Expanded Source Code (Level1)

```
CNOP 0, 4
DC     A (AR)
L      15, =V(COS)
```

Expanded Source Code (Level2)

	Macro	SubRoutine/Function/procedure
Action if appears in mnemonic field	Expansion Only	Call Statement leads to execution
Handled by	Assembler (Preprocessor) during Assembly time	Hardware at runtime
Size of Resulting Code	Increases	Remains same
Execution Speed	Fast	Slow

	Macro	SubRoutine/Function/procedure
Type Restriction	No One Macro can be served for different data types	Yes Only same data type
Useful for	Small & Simple task	Complex
Flow of Execution	Not Altered	Altered
Calling Process at	Translation Time	Execution Time

Macros and Macro Processor

MODULE 3

Content

- Design of Two Pass Macro Processor
- Data Structures used

Design of Two Pass Macro

IMPLEMENTATION OF TWO PASS MACRO

SPECIFICATION OF DATABASES

FLOWCHART

Implementation

any MACRO Processor

1. Recognize Macro Definition
   - Identified by pseudo-ops MACRO and MEND
   - Complicated task if Macro Definition appears within Macros
2. Save the Definition
   - Macro instructions must be stored
   - It will be used for expanding Macro Calls later
3. Recognize Calls
   - Processor must recognize Macro call that appear as operation mnemonic
   - Macro names are handled as type of Opcode
4. Expand the Calls and Substitute the arguments
   - Processor must substitute for dummy or macro definition arguments
   - Resulting Symbolic text is then substituted for Macro Call

Assumptions

1. Macro Processor is functionally independent of the Assembler
2. Output text from the Macro Processor will be fed into the Assembler
3. The basic Macro Processor algorithm is going to be implemented (No macro calls or Macro Definition inside Macros)
4. Macro Processor never process EQU statement
5. It will make two systematic scans over the task
6. The Macro Processor cannot expand a macro call before having found its definition

FIRST PASS

1. Examine every operation code
2. It will save all Macro Definition in a Macro Definition Table (MDT)
3. Save a copy of input text, minus macro definition on secondary storage for second pass
4. Prepare Macro Name Table (MNT) to recognize macro call

SECOND PASS

1. Examine every operation mnemonic and check whether name is present in MNT
2. Replace each Macro Name with appropriate text from the Macro Definition

Specification of Databases

PASS 1 Database

1. The Input Macro Source Deck
2. The Output macro source deck copy for use by Pass 2
3. The Macro Definition Table (MDT) used to store the body of the Macro definition
4. The Macro Name Table (MNT) used to store the names of defined macros
5. The Macro Definition Table Counter (MDTC) used to indicate the next available entry in the MDT
6. The Macro Name Table Counter (MNTC) used to indicate the next available entry in MNT
7. The Argument List Array (ALA) used to substitute index markers for dummy arguments before storing Macro Definition

PASS 2 Database

1. The copy of the input Macro Source Deck
2. The Output expanded source deck to be used as input to 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 Pointer (MDTP) used to indicate next line of text to be used during Macro Expansion
6. The Argument List Array (ALA) used to substitute macro call argument

Argument List Array (ALA)

- It is used during both Pass 1 and Pass 2
- During Pass 1, dummy arguments in the macro definition are replaced with positional indicator when the definition is stored
- The ith dummy argument on the macro name card is represented in the body of macro by the index marker symbol #
- #i is a symbol reserved for the use of macro processor
- These symbols are used in conjunction with the argument list prepared before expansion of macro call

Argument List Array (ALA)

Macro Definition Table (MDT)

```
&LAB   INCR  &ARG1,  &ARG2,  &ARG3
#0          A      1,  #1
             A      2, #2
             A      3, #3
```

Macro Call: Loop1 INCR   DATA1, DATA2, DATA3

Argument List Array (ALA)

Index	Arguments
0	"Loop1bbb"
1	"DATA1bbb"
2	"DATA2bbb"
3	"DATA3bbb"

Argument List Array (ALA)

Macro Definition Table (MDT)

```
&LAB   INCR  &ARG1,  &ARG2,  &ARG3
#0          A      1,  #1
             A      2, #2
             A      3, #3
```

Macro Call: INCR &ARG1=DATA3, &ARG2=DATA2, &ARG3=DATA1

Argument List Array (ALA)

Index	Arguments
0	"bbbbbbbb"
1	"DATA3bbb"
2	"DATA2bbb"
3	"DATA1bbb"

Macro Definition Table (MDT)

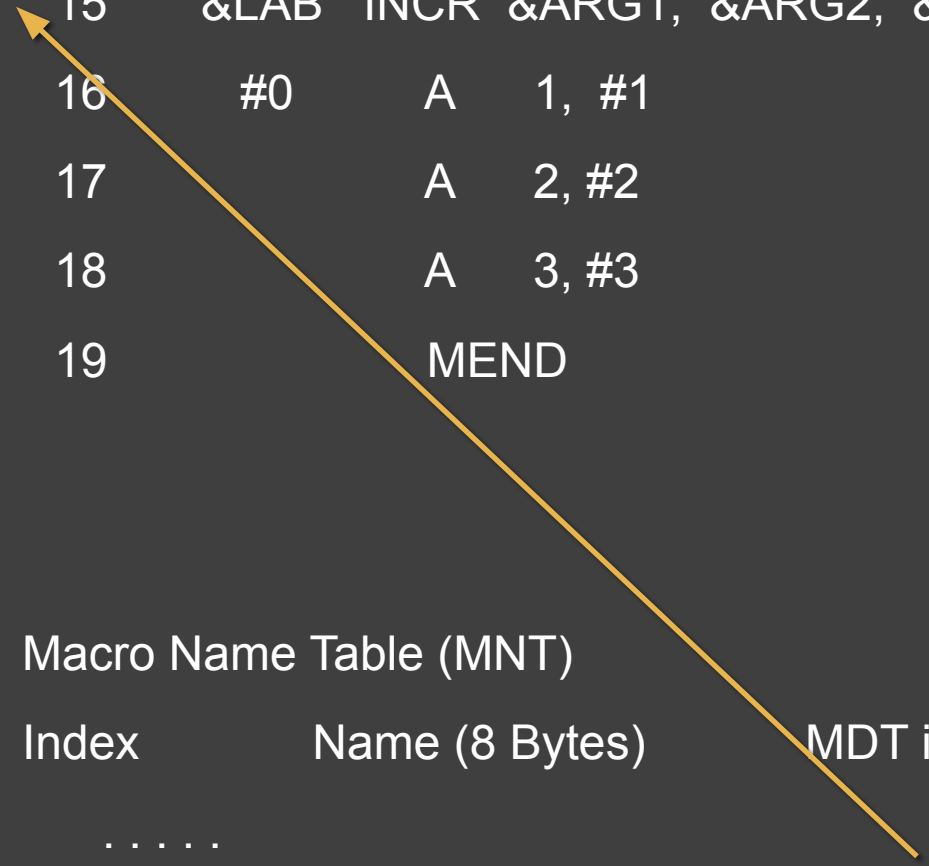
Macro Definition Table (MDT)

Index	Card
15	&LAB INCR &ARG1, &ARG2, &ARG3
16	#0 A 1, #1
17	A 2, #2
18	A 3, #3
19	MEND

Macro Name Table (MNT)

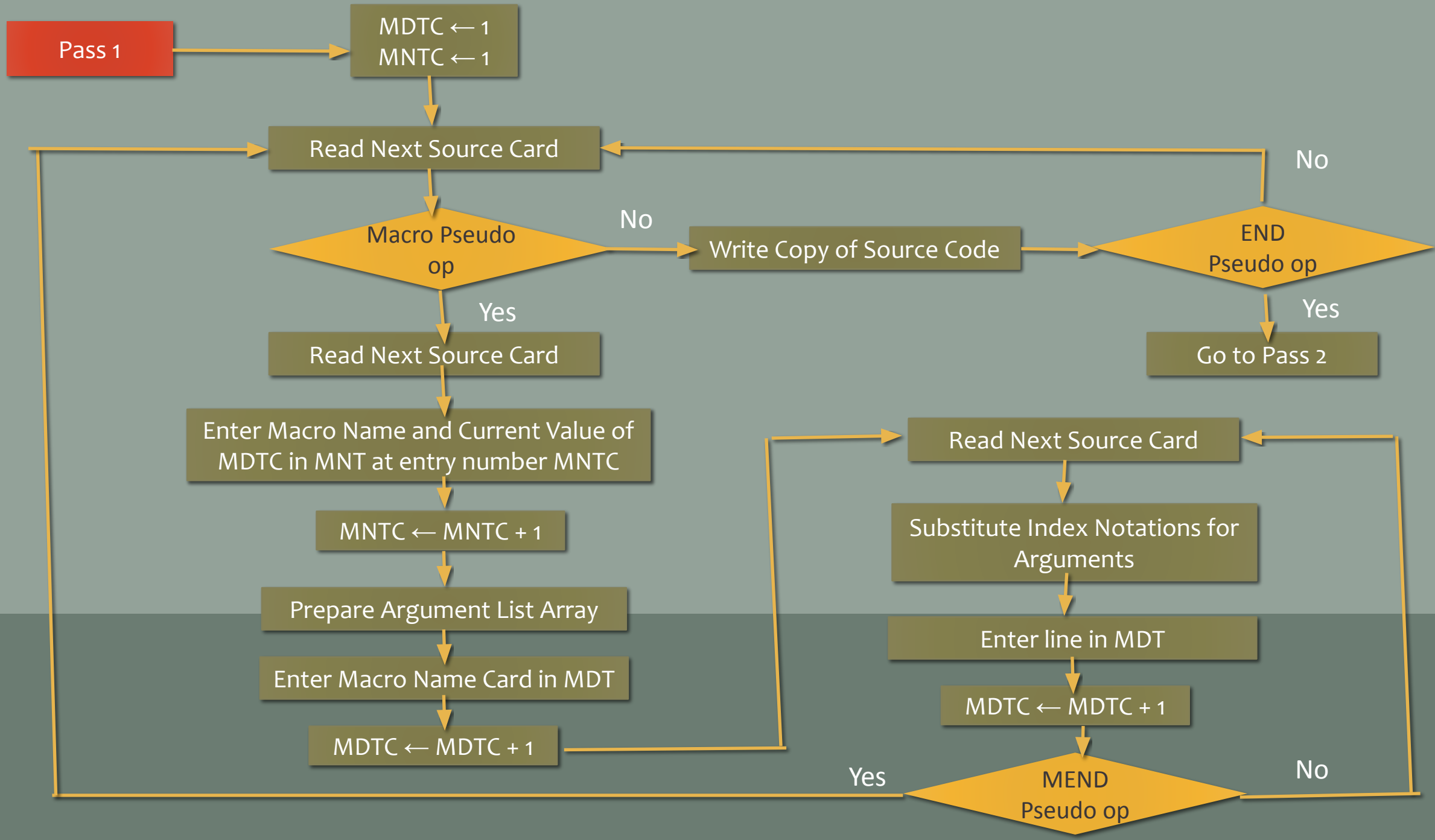
Macro Definition Table (MDT)

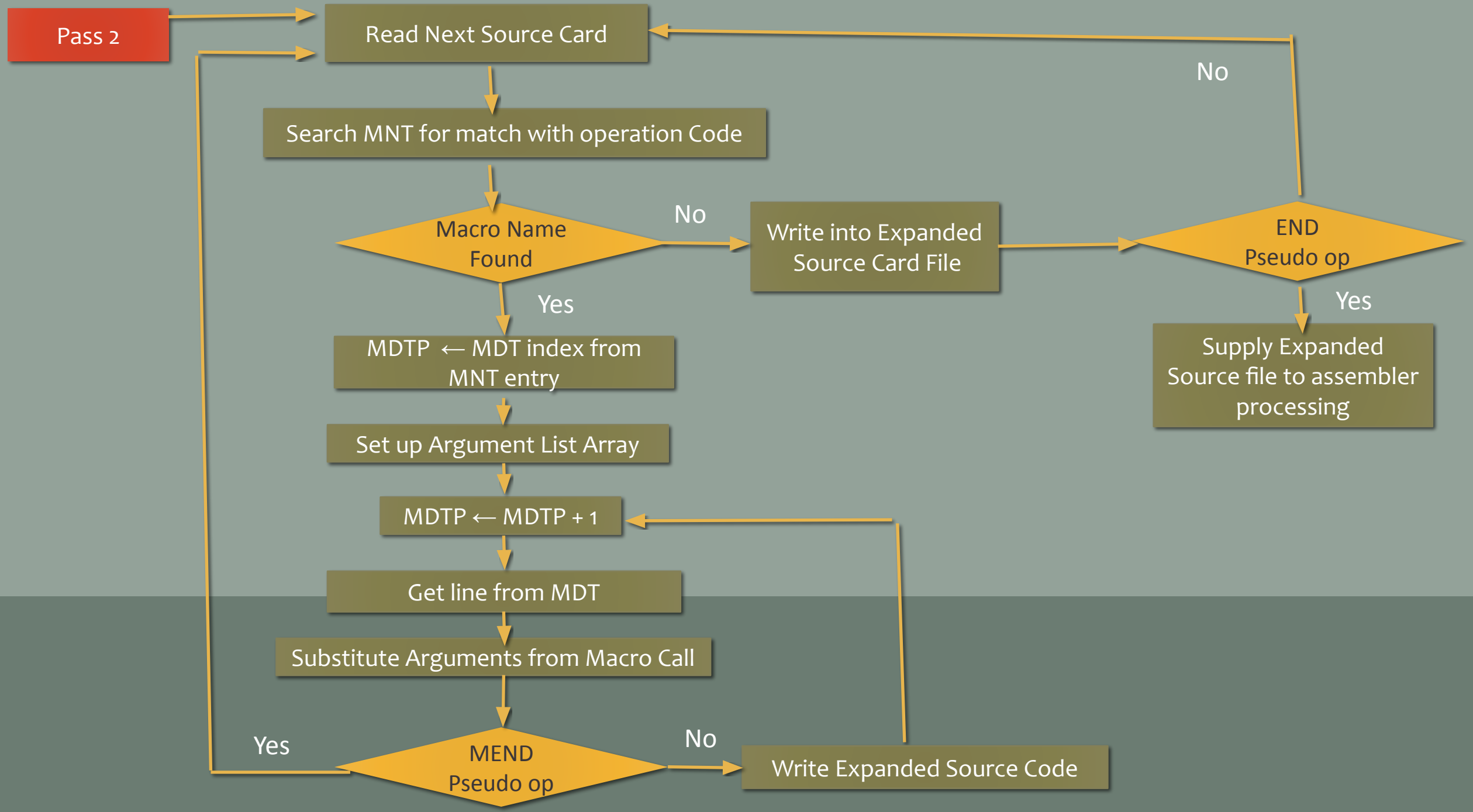
Index	Card
15	&LAB INCR &ARG1, &ARG2, &ARG3
16	#0 A 1, #1
17	A 2, #2
18	A 3, #3
19	MEND



Macro Name Table (MNT)

Index	Name (8 Bytes)	MDT index (4 Bytes)
. . . . .		
3	"INCRbbbb"	15


Pass 1
MDTC ← 1
MNTC ← 1
Read Next Source Card
Macro Pseudo op
No
Write Copy of Source Code
END Pseudo op
No
Yes
Go to Pass 2
Yes
Read Next Source Card
Enter Macro Name and Current Value of MDTC in MNT at entry number MNTC
MNTC ← MNTC + 1
Prepare Argument List Array
Enter Macro Name Card in MDT
MDTC ← MDTC + 1
Read Next Source Card
Substitute Index Notations for Arguments
Enter line in MDT
MDTC ← MDTC + 1
MEND Pseudo op
Yes
No


Pass 2
Read Next Source Card
Search MNT for match with operation Code
Macro Name Found
No
Write into Expanded Source Card File
END Pseudo op
No
Yes
Supply Expanded Source file to assembler processing
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 Code