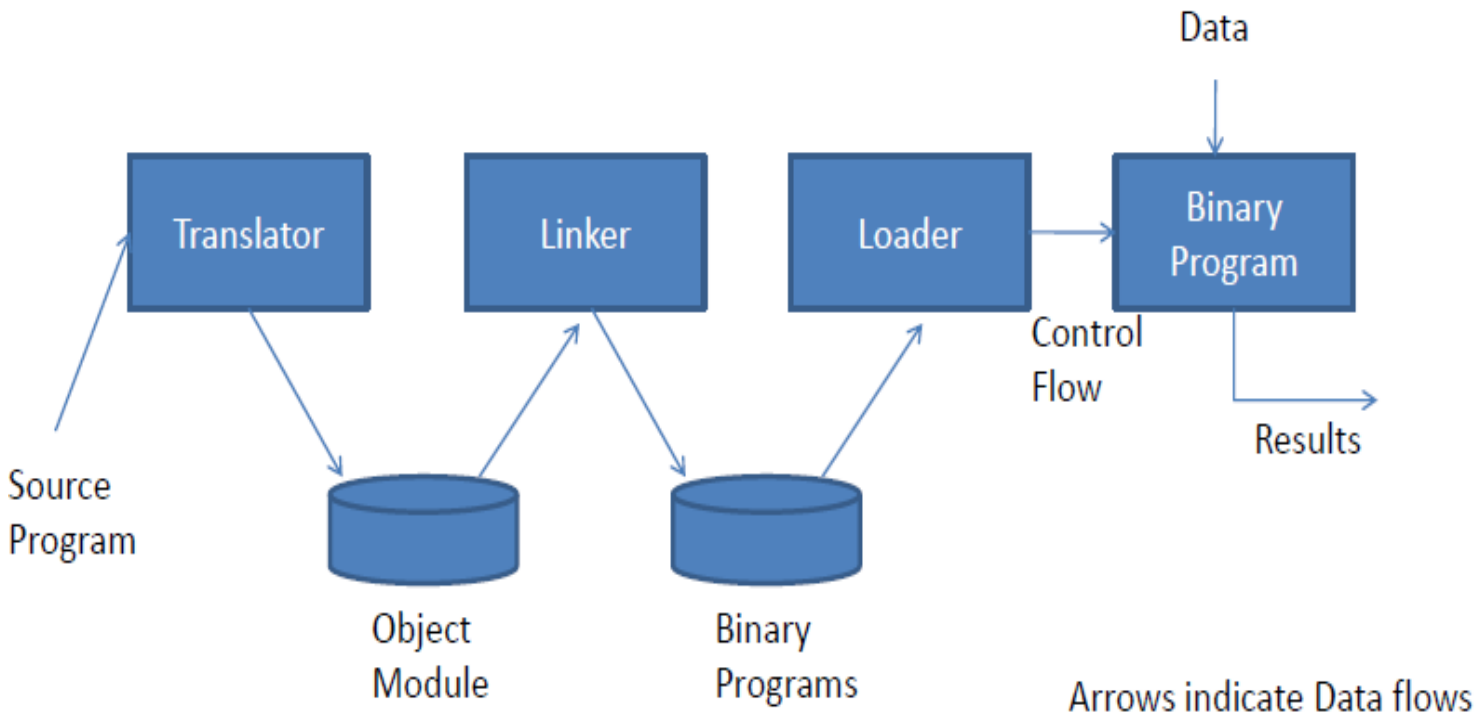
Table of Contents

Linkers

Linker

- **Steps for Execution:**

1. **Translation** of program
2. **Linking** of program with other programs needed for its execution.
3. **Relocation** of the program to execute from the specific memory area allocated to it.
4. **Loading** of the program in the memory for the purpose of execution.


Data
Translator
Linker
Loader
Binary Program
Control Flow
Results
Source Program
Object Module
Binary Programs
Arrows indicate Data flows

Linkers

- Links with other programs needed for its execution
- Processes a set of object modules to produce a ready-to-execute program form called binary program
- Loader loads this program into memory for execution

Linker

- A Linker is a system program that combines the code of a target program with codes of other programs and library routines

Object Module and Binary Program

- Object Module:
  - Contains target code (Machine language) of the program and information about other programs and library routines that it needs to invoke during its execution
- Binary Program:
  - Target code and other program routines combined together to form Binary program

Addresses

Address of a program entity may vary at different times:

1. Translation time address: Address assigned by translator (ORIGIN or START)
2. Linked address: Address assigned by linker
3. Load time address: Address assigned by loader

Addresses

Origin of a program may have to be changed by linker or loader because

1. Same set of translated addresses may have been used in different object modules, resulting in memory conflicts
2. OS MM module may require that a program should be allocated specific area of memory

Example:

	Statement		Address	Code
	START	500		
	ENTRY	TOTAL		
	EXTRN	MAX, ALPHA		
	READ	A	500)	+ 09 0 540
LOOP			501)	
	.			
	.			
	MOVER	AREG, ALPHA	518)	+ 04 1 000
	BC	ANY, MAX	519)	+ 06 6 000
	.			
	.			
	BC	LT, LOOP	538)	+ 06 1 501
	STOP		539)	+ 00 0 000
A	DS	1	540)	
TOTAL	DS	1	541)	
	END			

The translated origin

The Translated Time Address of LOOP

Relocation concept

- Program relocation is the process of modifying the addresses used in the address sensitive instructions of a program such that the program can execute correctly from the designated area of memory
- If linked origin ≠ translated origin, relocation must be performed by the linker
- If load origin ≠ linked origin, relocation must be performed by the loader

Sample Assembly Language Program and generated code (Program P)

```
        START    500
        ENTRY    TOTAL
        EXTERN   MAX,ALPHA
        READ     A              500 +400)          09  00   540+400=940
LOOP                            501+400)  901
        ----
        MOVER    AREG, ALPHA    518+400) 918       04   01   000
        BC       ANY, MAX       519+400) 919       07   06   000
        -----
        BC       LT, LOOP       538)       07   01  501
        STOP                    539)       00   00  000
A       DS       1              540)
TOTAL   DS       1              541)
        END
```

Performing Relocation

t_origin_p - translated origin of program P

l_origin_p - linked origin of program P

t_{symb} - translation time address of a symbol symb

l_{symb} - link time address of a symbol symb

Relocation factor of P is defined as

$$relocation\_factor_p = l\_origin_p - t\_origin \qquad \text{------1}$$

Performing Relocation

$relocation\_factor_p = l\_origin_p - t\_origin$ ------1

$t_{symb} = t\_origin_p + d_{symb}$ -----2

$l_{symb} = l\_origin_p + d_{symb}$ -----3

From 1 and 3

$l_{symb} = t\_origin_p + relocation\_factor_p + d_{symb}$ -----4

From 2 and 4

$l_{symb} = t_{symb} + relocation\_factor_p$ -----5

For Program P:

- Translated origin is 500
- Suppose l_origin = 900
- Relocation_factor = 900- 500 = 400
- Relocation will be performed for instructions with translated time address 500 and 538
  - For instruction with translated time address 500: address 540 in operand field will change to (540+400) = 940
  - For instruction with translated time address 538: address 501 in operand field will change to (501+400) = 901

Linking

- Program interacts with another program unit using its instructions & data in its own instructions
- Public definitions & external references required
  - ENTRY: Public definitions
  - A symbol defined in a program unit that may be referenced in other program unit.
  - EXTRN: External references
  - A reference to a symbol that is not defined in the program unit containing the reference (defined in other program)
- Linking is the process of binding an external reference to the correct link time address

Program Q

```
        START   200
        ENTRY   ALPHA
        --
ALPHA   DS      25          231) 00 00 025
        END
```

Linkers

Object module contains all info necessary to relocate & link program with other programs

1. Header : has translated origin, size & execution start address
2. Program : has machine code
3. Relocation table (RELOCTAB) : each entry contains translated address of an address sensitive instruction
4. Linking table (LINKTAB): contains PD/EXT symbols

Linker generates the linked addresses of all the symbols & instructions

RELOCTAB and LINKTAB

- Program P:
- RELOCTAB:
  - 500) 09 00 540
  - 538) 07 01 501
- LINKTAB:
  - TOTAL PD
  - MAX EXT
  - ALPHA EXT

RELOCTAB and LINKTAB

- Program Q:
- LINKTAB:
  - ALPHA PD

NTAB:

Symbol	Linked addr
P	900
TOTAL	941
Q	942
ALPHA	973

Linking of Program P and Q

Program P:

```
            START 500
            ENTRY       TOTAL
            EXTERN      MAX,ALPHA
            READ        A                   900)        09  00   940
            LOOP                            901)
        ----
            MOVER       AREG, ALPHA         918)        04   01   973
            BC          ANY, MAX            919)        07   06   000
        -----
            BC          LT, LOOP            938)        07    01  901
            STOP                            939)        00    00  000
            A           DS 1                940)
            TOTAL       DS 1                941)
            END
```

Program Q:

```
            START       200
            ENTRY       ALPHA
            --                              942)
            ALPHA   DS  25                  973)        00  00  025
            END
```

Self-Relocating Programs

Programs can be classified into

1. Non relocatable programs

- cannot be executed in any memory area other than its translated origin

-due to lack of information pertaining to address sensitive instructions in program

2. Relocatable programs

- Has info available related to address sensitive instructions in program

3. Self-relocating programs

Self-Relocating Programs

- Performs relocation of its own address sensitive instructions
- 2 provisions for this

1. Table containing address sensitive instructions exists as part of program

2. Relocating logic: Code to perform relocation of address sensitive instructions also exists as part of program

- Can execute in any area of memory
- Useful in time sharing operating systems

Static Link Libraries

- Is the process of copying all library modules used in the program into the final executable image.
- This is performed by the linker and it is done as the last step of the compilation process.
- The linker combines library routines with the program code in order to resolve external references, and to generate an executable image suitable for loading into memory.
- When the program is loaded, the operating system places into memory a single file that contains the executable code and data.
- This statically linked file includes both the calling program and the called program.
- Statically linked files are significantly larger in size because external programs are built into the executable files.

Static Link Libraries

- If any of the external programs change then they have to be recompiled and re-linked again else the changes won't reflect in existing executable file.
- Statically linked program takes constant load time every time it is loaded into the memory for execution
- Programs that use statically-linked libraries are usually faster
- In statically-linked programs, all code is contained in a single executable module. Therefore, they never run into compatibility issues.

Dynamic Link Libraries

- In dynamic linking the names of the external libraries (shared libraries) are placed in the final executable file while the actual linking takes place at run time when both executable file and libraries are placed in the memory.
- Several programs use a single copy of an executable module.
- Is performed at run time by the operating system
- Only one copy of shared library is kept in memory. This significantly reduces the size of executable programs, thereby saving memory and disk space
- Individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers.

Dynamic Link Libraries

- Load time might be reduced if the shared library code is already present in memory.
- Programs that use shared libraries are usually slower than those that use statically-linked libraries.
- Dynamically linked programs are dependent on having a compatible library. If a library is changed (for example, a new release may change a library), applications might have to be reworked to be made compatible with the new version of the library. If a library is removed from the system, programs using that library will no longer work.

MIT-WPU
Final Year (B.Tech)

System Software and Compiler Design

Module II

- Macro processor: Macro Definition, Macro expansion and nested macros
- **Loaders: Loader schemes: Types of loaders, direct linking loaders.**
- Linkers: Relocation and linking concepts, self-relocating programs, Static and dynamic link libraries.

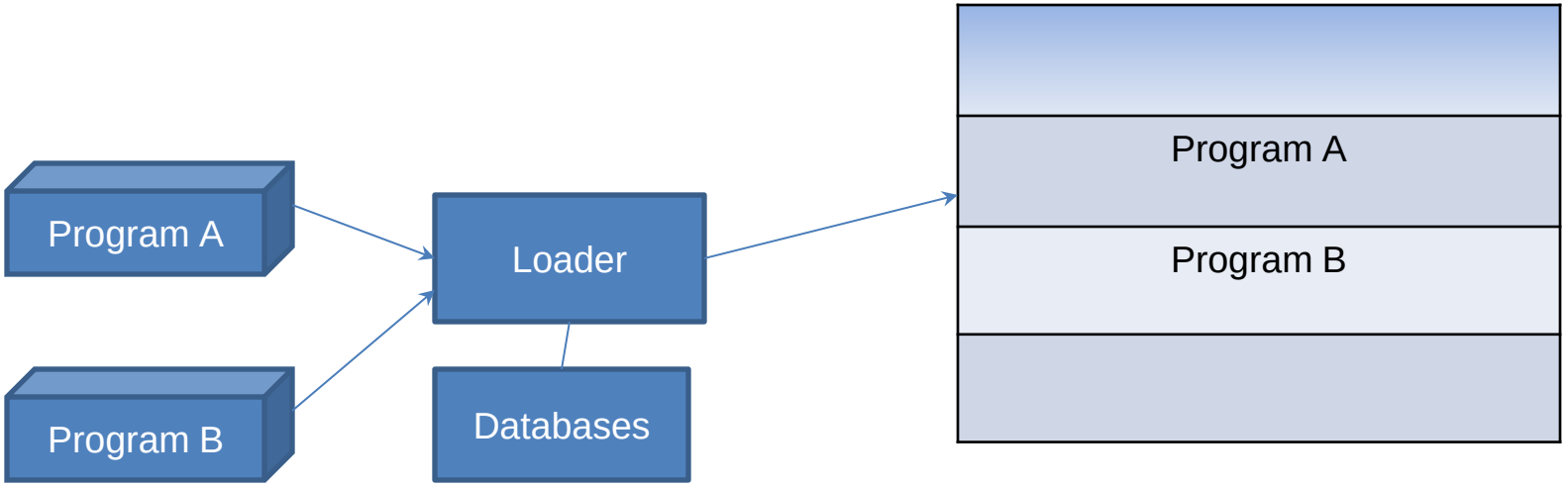
Loader

Loaders

Loader is a program that accepts the object program, prepares these programs for execution by the computer and initiates the execution of the program.

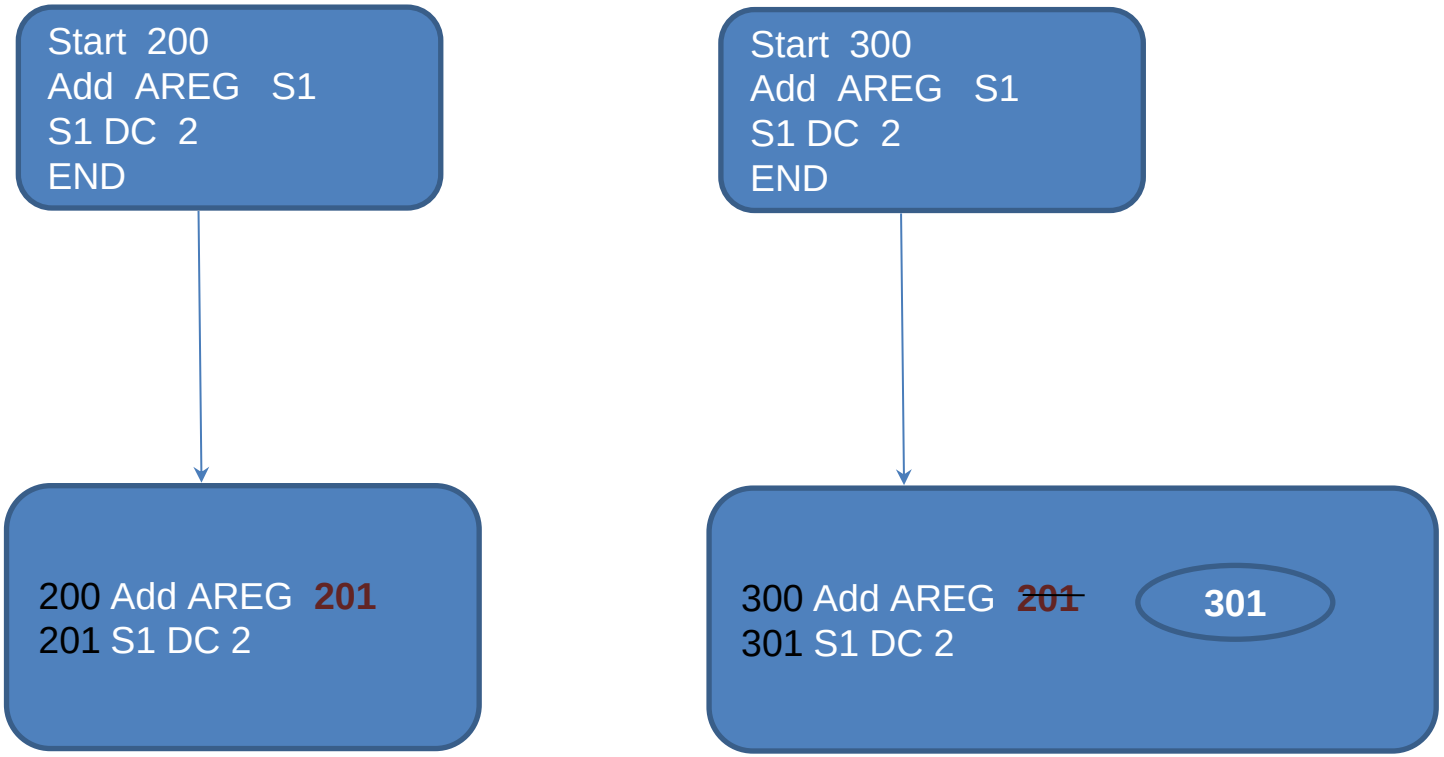
Functions of the loader

1. Allocate space in memory for the programs.(**Allocation**)
2. Resolve symbolic references between object decks.(**Linking**)
3. Adjust all addr dependent locations, such as addr constants, to correspond to the allocated space.(**Relocation**)
4. Physically place the m/c instr and data into memory.(**Loading**)

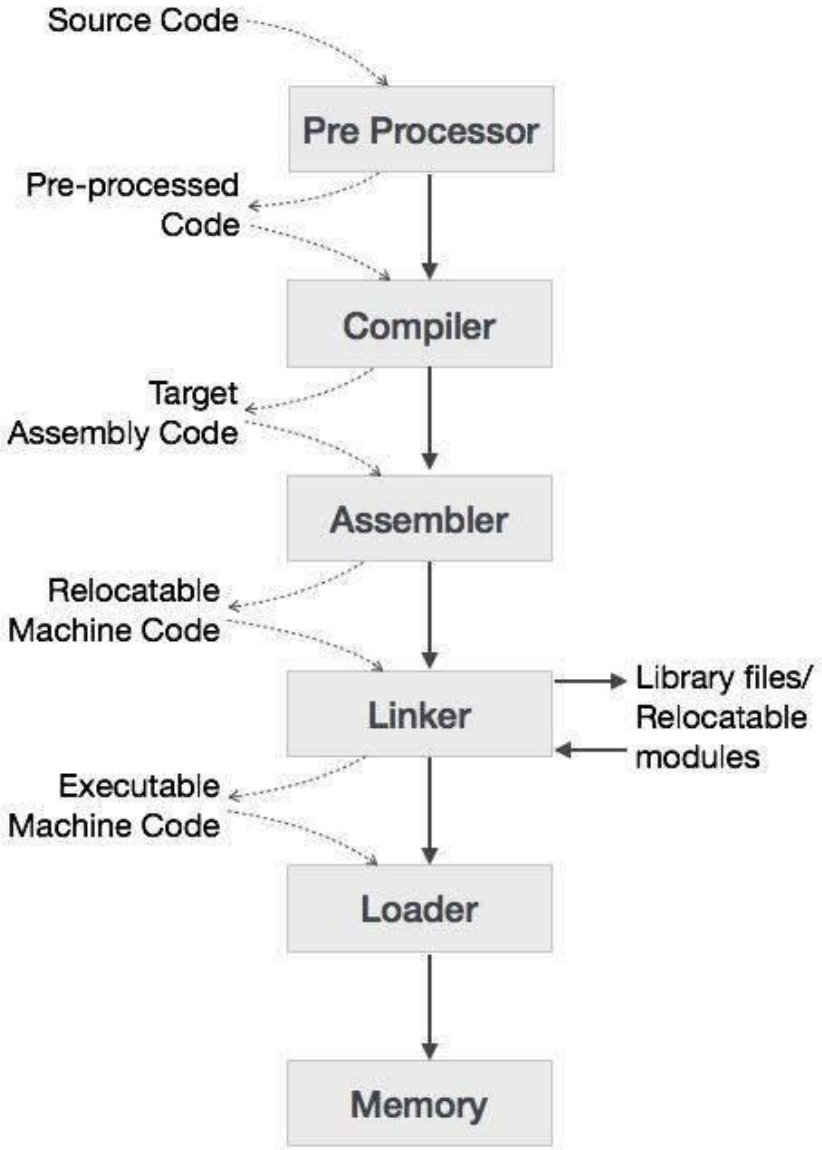


Programs loaded in memory ready for execution

Concept of Relocation



Language Processing System



Different Types of Loader Schemes

1. Compile and Go Loaders
2. General Loader Scheme
3. Absolute Loaders
4. Relocating Loaders
5. Direct Linking Loaders

1. Compile and Go Loaders

-Assembler places the code into core

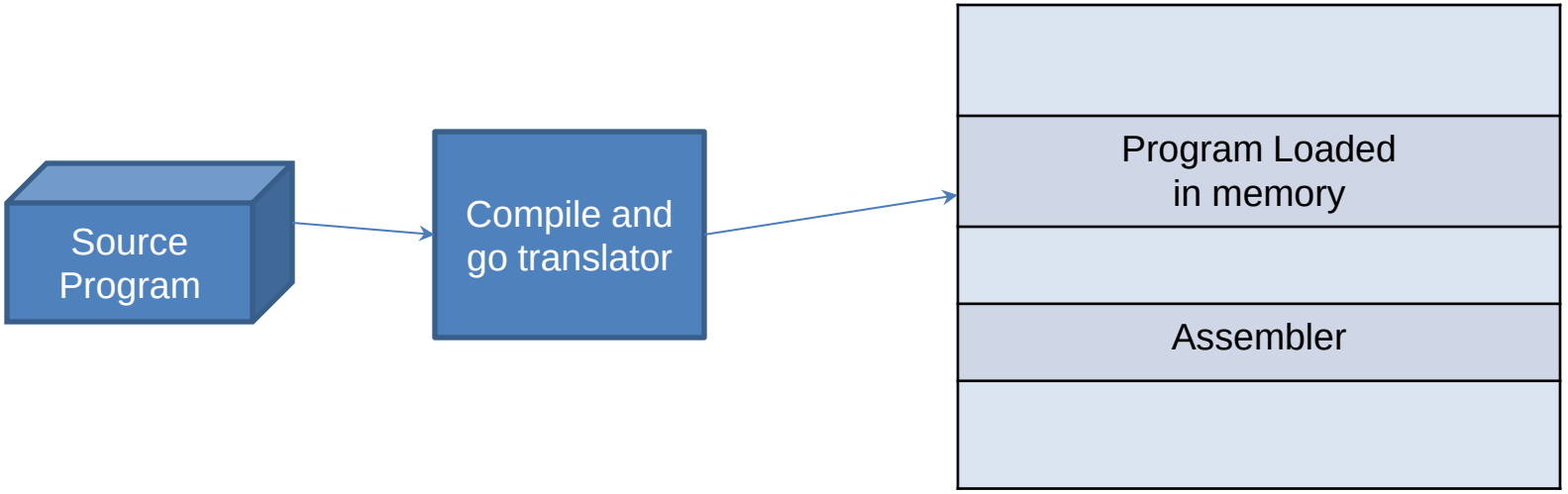
-Loader consists of one instr that transfers to the starting instr of the newly assembled program

-easy to implement

-portion of memory is wasted because of assembler

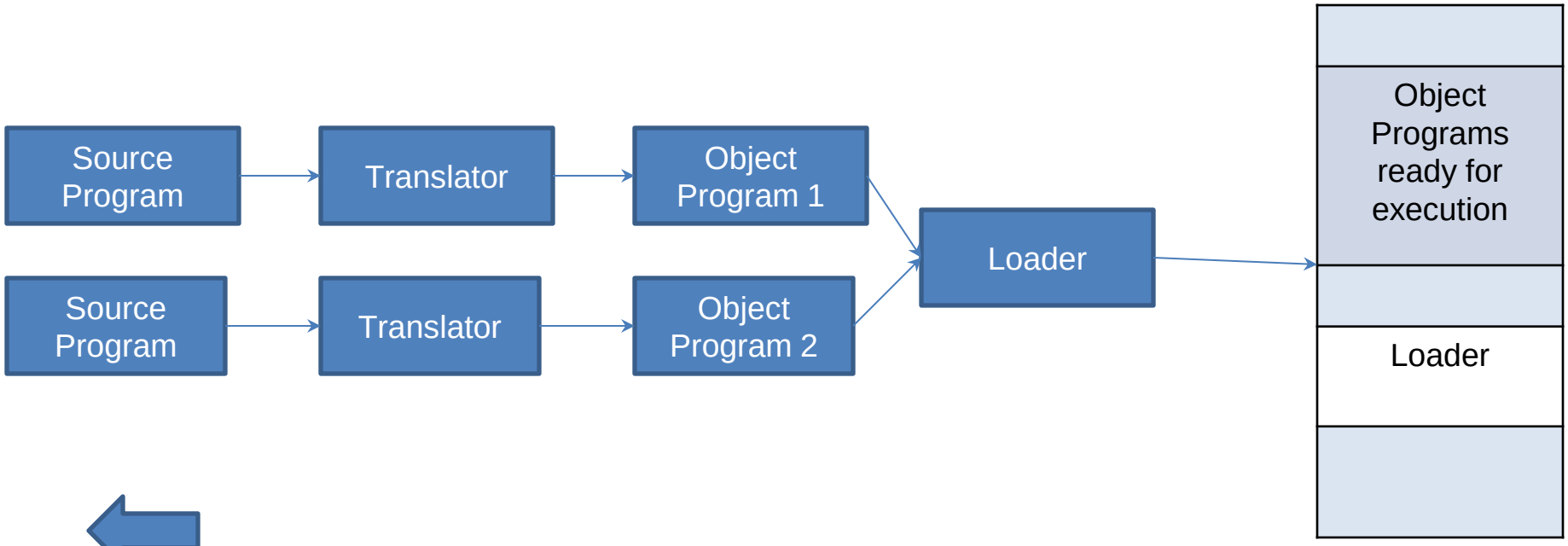
-every time the program is run it has to be retranslated

-difficult to handle multiple segments



2. General Loader Scheme

- As loader is smaller than assembler more memory is available
- Reassembling of program is not required to run the program later.
- Loader is present in memory.



3. Absolute Loader

-Same as “compile and go “ loader except data is punched on cards instead of memory.

-Loader accepts m/c language text and places it into memory at the location specified by the assembler

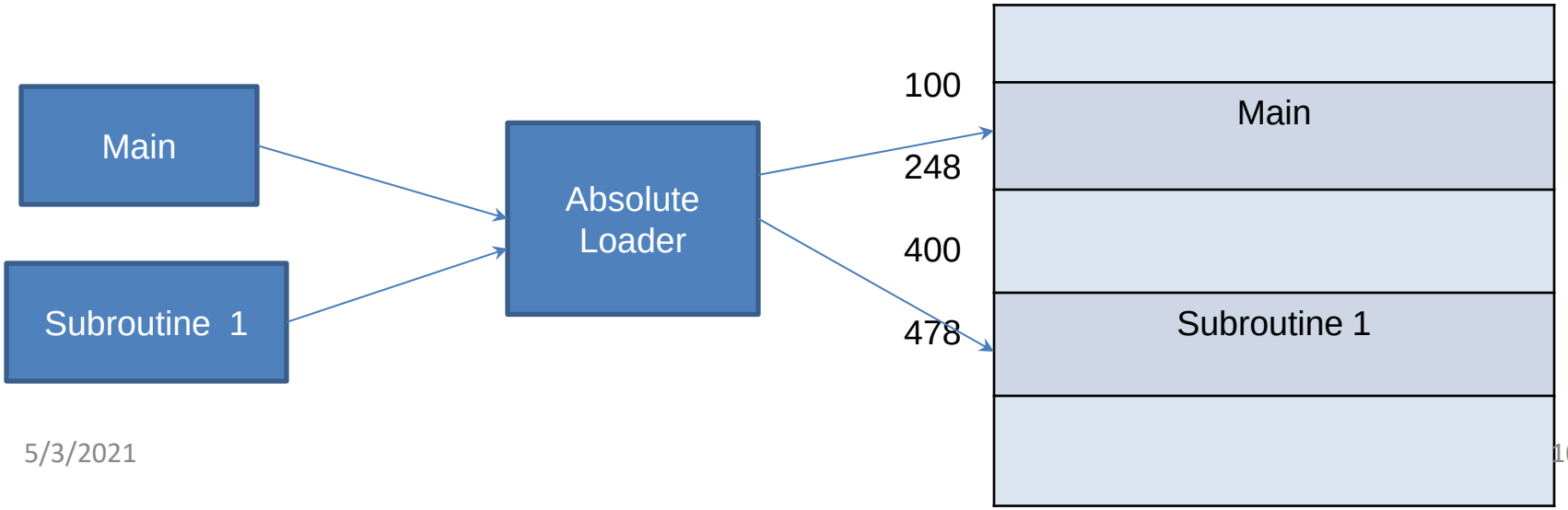
Advantages

-More memory is available, -simple to implement

Disadvantages

-Programmer must specify address to the assembler where the program is loaded.

-In case of multiple subroutines, programmer has to remember address of each subroutine.



Relocating Loader (Binary Symbolic Subroutine)

-To **avoid** possible **reassembling** of all subroutines when a single subroutine is changed.

-To perform **task of allocation and linking** for the programmer.

-Allows **many procedure segments** but only one data segment.

-Translated code segments and the information regarding relocation and intersegment references is passed to the loader.

Information provided by the assembler to the BSS loader.

- ***Transfer Vector***
- Contains the address and names related to the subroutines referenced in the program.
- Total length of the program
- length of transfer vector
- ***Relocation Bits***
- relocation bit is associated with every instruction
- Relocation bits can be 0 or 1.
- If 1 then address field needs relocation
- If 0 then address field does not need relocation

ST	14	SAVE
ST	14	SAVE
Relocation Bit=0		Relocation Bit=1

4. Relocating Loader (Binary Symbolic Subroutine)

- In BSS
- All four functions of loader (allocation, linking, relocation and loading)are performed automatically by the BSS loader.

- **Relocation bits** are used to solve the problem of relocation.

- The **transfer vector** is used to solve the problem of linking.

- The **program length** information is used to solve the problem of allocation.

4. Relocating Loader (Contd..)

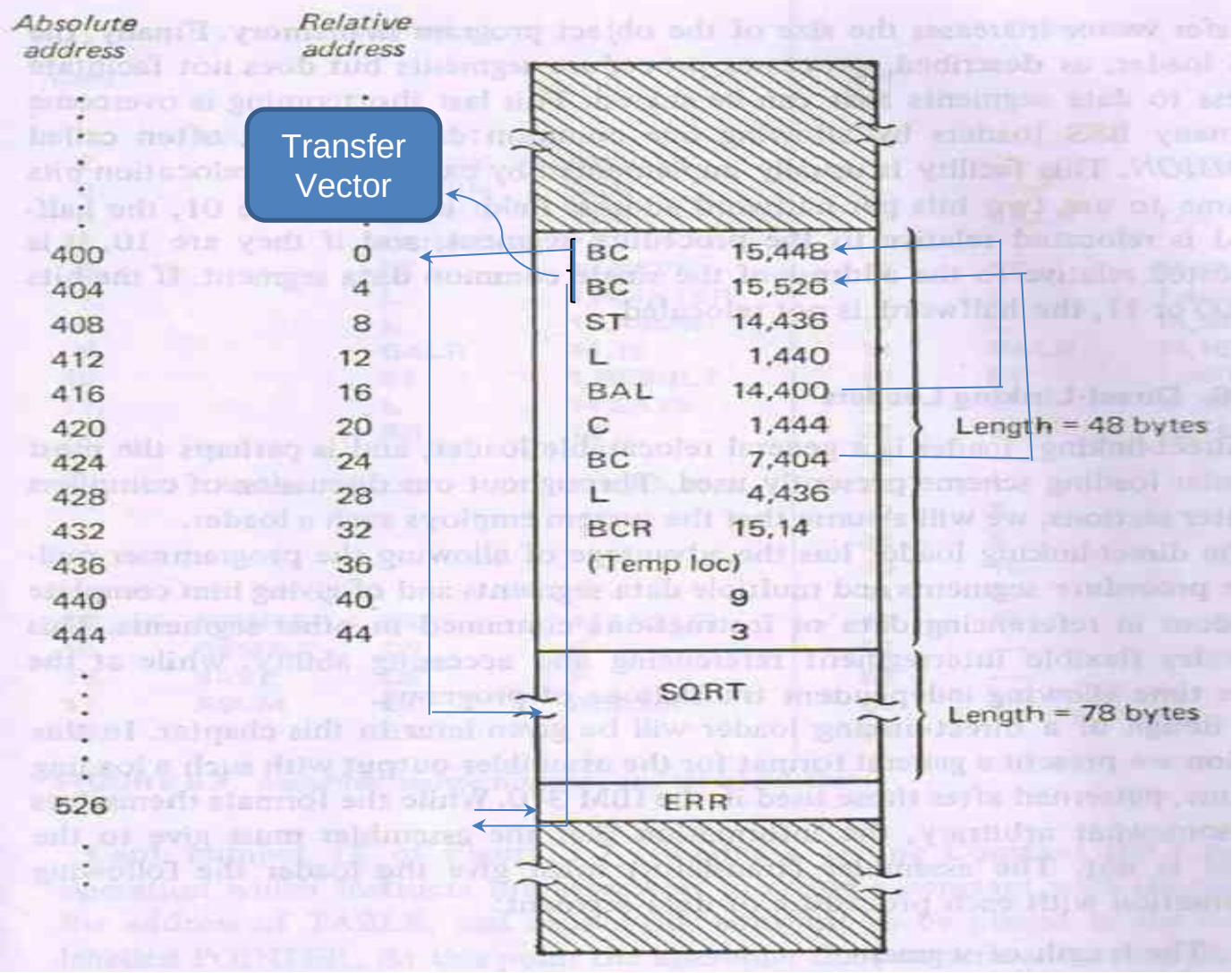
Source program

Program Length = 48 bytes

Transfer Vector = 8 bytes

			Rel. Addr.	Rel Bits	Object Code	
MAIN	START					
	EXTRN	SQRT	0	00	'SQRT'	
	EXTRN	ERR	4	00	'ERRb'	
	ST	14,SAVE	8	01	ST	14,36
	L	1,=F'9'	12	01	L	1,40
	BAL	14,SQRT	16	01	BAL	14,0
	C	1,=F'3'	20	01	C	1,44
	BNE	ERR	24	01	BC	7,4
	L	14,SAVE	28	01	L	14,36
	BR	14	32	0	BCR	15,14
SAVE	DS	F	36	00	(Temp location)	
	END		40	00	9	
			44	00	3	

4. Relocating Loader (Contd..)



Disadvantages of Relocating Loader

- The transfer vector linkage is only useful for transfers and **not well suited for loading** or storing external data.
- The transfer vector **increases the size of the object program** in memory.
- BSS loader processes procedure segments but **does not facilitate access to data segments** that can be shared.

5. Direct Linking Loader

- *Flexible intersegment referencing and accessing ability.*
- *Allows independent translation of programs.*

Information provided by the assembler with each procedure or data segment

- *Length of the segment.*
- *List of symbols and relative locations.*
- List of symbols not defined but referenced.
- Information where address constants are located.
- M/c translation of source program and relative address assigned.

Assembler produces 4 types of cards in the object deck.

ESD External Symbol Dictionary.

TXT Actual Object Code.

RLD Relocation and Linkage Directory.

END End of object deck.

5. Direct Linking Loader(contd…)

ESD cards

- Contains info related to all the symbols defined and referenced in the program.
- Values for ESD cards are

  SD (Segment Definition) name on START card

  LD (Local Definition) Specified on ENTRY card

  ER (External Reference) specified on EXTRN card

TXT cards

- Contains actual object code translated version of program.

RLD cards

- The location constant that needs to be changed due to relocation
- By what is has to be changed
- The operation to be performed(+/-)

END cards

- End of object deck and specifies the starting address for execution if the assembled routine is the main program.

5. Direct Linking Loader(contd…)

Card No	ALP	Rel Loc	Translation
1.	JOHN START		
2.	ENTRY RESULT		
3.	EXTRN SUM		
4.	BALR 12, 0	0	BALR 12,0
5.	USING *, 12		[12]<-- current value of LC
6	ST 14, SAVE	2	ST 14, 54(0,12)
7.	L 1, POINTER	6	L 1, 46(0,12)
8.	L 15, ASUM	10	L 15, 58(0,12)
9.	BALR 14, 15	14	BALR 14, 15
10.	ST 1, RESULT	16	ST 1, 50(0,12)
11.	L 14, SAVE	20	L 14, 54(0,12)
12.	BR 14	24	BCR 15, 14
13.	TABLE DC F '1, 7, 9, 10, 3'	28 32 36 40 44	1 7 9 10 3
14.	POINTER DC A(TABLE)	48	28
15.	RESULT DS F	52	-
16.	SAVE DS F	56	-
17.	ASUM DC A(SUM)	60	?
18.	END	64	

Offset + [index reg] + [base reg]
54 + 0 + [12] = 54 + 0 + 2 = 56

ESD And RLD Cards

ESD Cards				
Ref No	Symbol	Type	Relative Loc	Length
1.	JOHN	SD	0	64
2.	RESULT	LD	52	-
3.	SUM	ER	-	-

RLD Cards				
Ref No	Symbol	Flag	Length	Rel Loc
14	JOHN	+	4	48
17	SUM	+	4	60

TXT Cards

TXT Cards			
Ref No	**Rel Loc**	**Object Code**	
4	0	BALR	12,0
6	2	ST	14, 54(0,12)
7	6	L	1, 46(0,12)
8	10	L	15, 58(0,12)
9	14	BALR	14, 15
10	16	ST	1, 50(0,12)
11	20	L	14, 54(0,12)
12	24	BCR	15, 14
13	28	1	
13	32	7	
13	36	9	
13	40	10	
13	44	3	
14	48	28	
17	60	0	

```
1       PG1         START
2                   ENTRY       PG1ENT1, PG1ENT2
3                   EXTRN       PG2ENT2, PG2
4       PG1ENT1
5       PG1ENT2     -
6                   DC      A (PG1ENT1)
7                   DC      A (PG1ENT2+15)
8                   DC      A (PG1ENT2-PG1ENT1-3)
9                   DC      A (PG2)
10                  DC      A (PG2ENT1+PG2-PG1ENT1+4)
11                  END

12      PG2         START
13                  ENTRY       PG2ENT1
14                  EXTRN       PG1ENT1, PG1ENT2
15      PG2ENT1     -
16                  DC      A (PG1ENT1)
17                  DC      A (PG1ENT2+15)
18                  DC      A (PG1ENT2-PG1ENT1-3)
19                  END
```

```
Source      Relative
Card        Address
Ref no
1           0       PG1         START
2                               ENTRY       PG1ENT1, PG1ENT2
3                               EXTRN       PG2ENT2, PG2
4           20      PG1ENT1
5           30      PG1ENT2     -
6           40                  DC      A (PG1ENT1)
7           44                  DC      A (PG1ENT2+15)
8           48                  DC      A (PG1ENT2-PG1ENT1-3)
9           52                  DC      A (PG2)
10          56                  DC      A (PG2ENT1+PG2-PG1ENT1+4)
11          60                  END

12          0       PG2         START
13                              ENTRY       PG2ENT1
14                              EXTRN       PG1ENT1, PG1ENT2
15          16      PG2ENT1     -
16          24                  DC      A (PG1ENT1)
17          28                  DC      A (PG1ENT2+15)
18          32                  DC      A (PG1ENT2-PG1ENT1-3)
19          36                  END
```

OBJECT DECK FOR PG1

ESD Cards

Source Card Ref No	Name	Type	ID	Relative Address	Length
1	PG1	SD	01	0	60
2	PG1ENT1	LD	01	20	
2	PG1ENT2	LD	01	30	
3	PG2	ER	02		
3	PG2ENT1	ER	03		

TXT Cards (Those having address constants)

Source Card Ref No	Relative Address	Contents	Comments
6	40-43	20	
7	44-47	45	30 +15
8	48-51	7	30-20-3
9	52-55	0	UNKNOWN TO PG
10	56-59	-16	-20+4

RLD Cards

Source Card Ref No	ESD-ID	Length in bytes	FLAG + or -	Relative Address
6	01	4	+	40
7	01	4	+	44
9	02	4	+	52
10	03	4	+	56
10	02	4	+	56
10	01	4	-	56

- PG2

ESD Cards

Source Card Ref No	Name	Type	ID	Relative Address	Length
12	PG2	SD	01	0	36
13	PG2ENT1	LD		16	
14	PG1ENT1	ER	02		
14	PG1ENT2	ER	03		

Txt Cards (Those having address constants)

Source Card Ref No	Relative Address	Contents	Comments
16	24-27	0	
17	28-31	15	
18	32-35	-3	

RLD Cards

Source Card Ref No	ESD-ID	Length in bytes	FLAG + or -	Relative Address
16	02	4	+	24
17	03	4	+	28
18	03	4	+	32
18	02	4	-	32

MIT-WPU
Final Year (B.Tech)

System Software and Compiler Design

Module II

- Macro processor: Macro Definition, Macro expansion and nested macros
- Loaders: Loader schemes: Types of loaders, direct linking loaders.
- Linkers: Relocation and linking concepts, self-relocating programs, Static and dynamic link libraries.

Introduction to Macro

- In the mid-1950s, when assembly language programming was commonly used to write programs for digital computers, the use of **macro instructions** was initiated for two main purposes:
  - to reduce the amount of program coding that had to be written by generating several assembly language statements from one macro instruction and
  - to enforce program writing standards, e.g. specifying input/output commands in standard ways.
  - Macro instructions were effectively a middle step between assembly language programming and the high-level programming languages that followed, such as FORTRAN and COBOL
  - Eg.
  - **#define** PI 3.14159 //"PI" to be replaced with "3.14159" wherever it occurs
  - **#define** pred(x) ((x)-1) // What this macro expands to depends on what argument *x* is passed to it. Here are some possible expansions:
    - pred(2) → ((2) -1)
    - pred(y+2) → ((y+2) -1)
    - pred(f(5)) → ((f(5))-1)

Introduction

- Macro instructions (macros) are single -line abbreviations for group of instructions.
- Macros are used to provide a program generation facility through **macro expansion**.
- Many languages provide built in facilities for writing macros.

  e.g. PL/I, C, Ada and C++, Assembly languages.
- Generating preprocessors or software tools like Awk of Unix has an equivalent effect.
- **A macro is a unit of specification for program generation through expansion.**

Introduction (contd…)

Macros are defined at the start of the program.

A Macro Definition consists of

- MACRO Pseudo opcode
- Name of macro
- List of formal parameters
- Body of macro(instruction) or Sequence to be abbreviated
- MEND Pseudo opcode

Macro Definition Syntax :-

1) Macro header :- It contains keyword ‘MACRO’.
2) Macro prototype statement syntax :-

   < Macro Name > [& < Formal Parameters >]
3) Model Statements :- It contains 1 or more simple assembly statements, which will replace MACRO CALL while macro expansion.
4) MACRO END MARKER :- It contains keyword ‘MEND’.

Introduction

- Macro name with a set of actual parameters, is replaced by some code, generated from macro body. **This is called macro expansion.**
- **Two types of expansions:**
- **Lexical expansion:** It implies replacement of a character string by another character string during program generation.
- **Semantic expansion:** Generation of type specific instructions for manipulation of byte and word operands.

Example 4.1 (Benefits of semantic expansion) The following sequence of statements is used to increment the value stored in a memory word by a given constant:

1. Move the value from the memory word into a CPU register.
2. Increment the value in the CPU register.
3. Move the new value into the memory word.

Macro-processor

- Many times some blocks of code is repeated in the course of a program
- They may consists of code

-- to save or exchange set of registers

-- to set up linkages

-- to perform a series of arithmetic operations

- In this situation macro instruction facility is useful.
- Programmer defines a single instruction to represent a block of code.
- For every occurrence of this one line instruction the assembler will substitute the entire block.

Macros and Functions

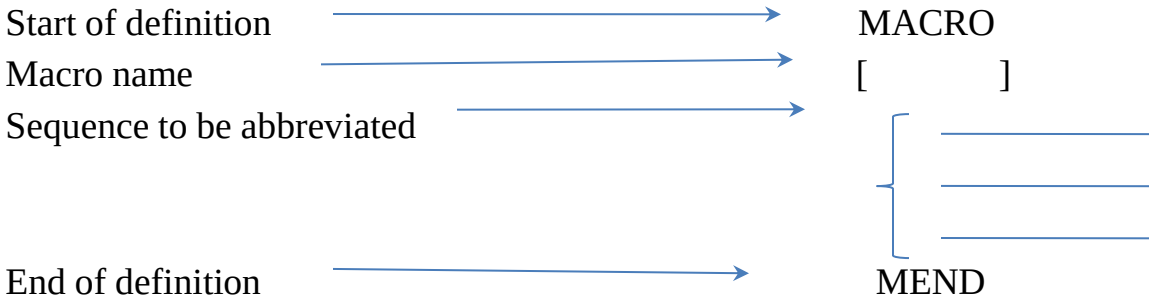
Macros

- Macro is Preprocessed
- No Type Checking is done in Macro
- Using Macro increases the code length
- Speed of Execution using Macro is Faster
- Before Compilation, macro name is replaced by macro value
- Macros are useful when small code is repeated many times
- Macro does not check any Compile-Time Errors
- Does not require CALL and RETURN

Functions

- Function is Compiled
- Type Checking is done in functions
- Using Function keeps the code length unaffected
- Speed of Execution using Function is Slower
- During function call, transfer of control takes place
- Functions are useful when large code is to be written
- Function checks Compile-Time Errors
- Requires CALL and RETURN

Macro Instruction



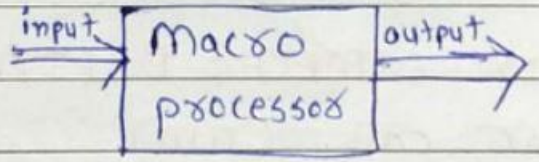
Macro Example

```
MACRO
INCR
        ADD     1,DATA
        ADD     2,DATA
        ADD     3,DATA
MEND
:
INCR
:
INCR
```

Example of macro

* Source Program ⇒

```
MACRO
My MACRO
    ADD AREG,X
    ADD BREG,X
MEND
 START 100
 My MACRO
 MyMACRO
 MyMACRO
  STOP
```



Target Program with Macro expansion ⇒

```
START 100
ADD AREG,X
ADD BREG,X
ADD AREG,X
ADD BREG,X
ADD AREG,X
ADD BREG,X
STOP
```

Example-Macro

- Macro instructions are **single line abbreviations for group of instructions**

e.g.

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

DATA   DC  F'5'
```

	EXPANDED SOURCE
MACRO	
INCR	
A 1,DATA	
A 2,DATA	
A 3,DATA	
MEND	
……	A 1,DATA
INCR	A 2,DATA
……...	A 3,DATA
	……...
……	A 1,DATA
INCR	A 2,DATA
……	A 3,DATA
DATA DC F'5'	
……	

Contd…

Source program with Macro definitions & macro calls

```
MACRO
    INCR  &ARG
    ADD AREG ,&ARG
    ADD BREG,&ARG
MEND
MOVER CREG,LABEL1
INCR DATA1
SUB CREG,LABEL1
---
---
---
```

expanded source without definition & call

```
 MOVER CREG,LABEL1
ADD AREG,DATA1
ADD BREG,DATA1
SUB CREG ,LABEL1
```

Macro features

1.Macro instruction arguments

2.Conditional macro expansion

3. Macro calls within macros

4.Macro instructions defining macros

Macro Instruction Arguments

1.Macro instruction arguments

- Macro facility lacks in flexibility: there is no way for a specific macro call to modify the coding that replaces it.
- So extension of this facility consists of providing for arguments or parameters in macro call
  - dummy arguments
  - positional arguments
  - keyword arguments
  - label arguments

these arguments are preceded by &.

Macro Instruction Arguments Contd…

e.g.

```
……
……
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’
```

Source

```
        MACRO
        INCR  &ARG
        A     1,&ARG
        A     2,&ARG
        A     3,&ARG
        MEND
        ……
        ……
        INCR  DATA1
        …….
        …….
       INCR  DATA2
        ……
        ……
DATA1  DC  F’5’
DATA2   DC F’10’
……
……
```

Expanded source

```
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’
```

Macro Instruction Arguments (Label Arguments)

```
        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’
…….
```

```
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 Instruction Arguments

Positional arguments

arguments are matched with dummy arguments according to the order in which they appear

e.g. INCR A B C

Keyword arguments:

it allows reference to dummy arguments by name as well as by position

e.g. INCR &ARG1=A,&ARG3=C,&ARG2=B

Macro Instruction Arguments (Positional Arguments)

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

2 Pass Macroprocessor

- Recognize macro definitions
- Save the definitions
- Recognize calls
- Expand calls and substitute arguments

2 Pass Macroprocessor structure

- **Pass I**

1. **Input** macro source deck
2. **Output** macro source deck copy for use by pass 2
3. Macro definition table (**MDT**) ,used to store body of macro def.
4. Macro name table (**MNT**) ,used to store names of defined macros
5. **MDTC**-MDT counter ,used to indicate next available entry in MDT
6. **MNTC**-MNT counter, used to indicate next available entry in MNT
7. Argument List Array(**ALA**) used to substitute index markers for dummy arguments before storing macro definition

Example

MACRO
M1 &ARG1,&ARG2
ADD AREG &ARG1
ADD BREG &ARG2
MEND
MACRO
M2 &ARG3,&ARG4
SUB AREG &ARG3
SUB BREG &ARG4
MEND
START 300
MOVER AREG S1
MOVEM BREG S2
M1 D1 D2
MOVER AREG S1

M2 D3 D4
PRINT S1
PRINT S2
S1 DC 5
S2 DC 6
END

Contd…

- **Pass 2**

1. Copy of **input** macro source deck
2. **Output** expanded source deck to be used as input to assembler
3. **MDT**, created by Pass 1
4. **MNT**, created by pass 1
5. **MDTP**- MDT pointer used to indicate next line of text to be used during macro expansion
6. **ALA** ,used to substitute macro call arguments for the index markers in the stored macro definition.

MDT

- **Macro definition table(MDT)**
- Used to store macro definition
- Created by pass-1
- Pass 1 identifies and stores all macro definitions in MDT
- Pass 2 can identify macro calls and expand these calls by using their macro definitions stored in MDTs
- Every line of macro definition except MACRO is stored in MDT as MACRO is not used to expand macro
- MEND indicates end of macro definition so it is stored in MDT
- MDT has 80 bytes per entry

Index	instruction

MNT

- **Macro name table(MNT)**
- Used to store name of macros and corresponding MDT index
- Created by Pass 1 and used by Pass 2
- So that pass 2 can decide whether opcode of this source instruction is macro call or not by searching through name field of MNT.

MNT index	Macro Name	MDT index

- **MDTC(Macro def table counter)**

  this variable stores the last count from the MDT table
- **MNTC(Macro name table counter)**

  this variable stores the no. of macros defined in the program

ALA

- **Argument List Array(ALA)**
- Used for association of dummy arguments and actual arguments
- Partially table is created by pass 1 where pass 1 associates an unique integer number with each dummy arguments in the order in which they appear
- Macros are stored in MDT by using these numbers
- It is partially constructed and used by pass 2
- It keeps track of dummy parameters when the macro is being defined and maintain the actual arguments when expanding the call

Integer index	dummy argument	Actual argument

Example

```
MACRO
   INCR &ARG1
   MOVER AREG,& ARG1
   ADD AREG,& ARG1
   MOVEM AREG,& ARG1
MEND

START
MOVEM BREG, A
ADD CREG,=‘1’
SUB CREG,A
INCR DATA1
MUL CREG,=‘1’
END
```

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,=‘1’
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	

MNT

MNT Index	Macro Name	MDT index
1		

ALA

ALA Index	Formal Argument
#1	

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A
3	ADD AREG,B

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A
3	ADD AREG,B
4	MOVEM AREG,A

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Macro processor example

START 200
MACRO
INCR &ARG1,&ARG2
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
MEND
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A
3	ADD AREG,B
4	MOVEM AREG,A
5	MEND

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Output of Pass 1

Output file

START 200
MOVER AREG,='1'
MOVEM BREG,M
INCR DATA1,DATA2
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A
3	ADD AREG,B
4	MOVEM AREG,A
5	MEND

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Output of Pass 2

Final Expansion

START 200
MOVER AREG,='1'
MOVEM BREG,M
MOVER AREG, A
ADD AREG,B
MOVEM AREG,A
DATA1 DC 5
DATA2 DC 10
END

MDT

MDT index	Instruction
1	INCR &ARG1,&ARG2
2	MOVER AREG, A
3	ADD AREG,B
4	MOVEM AREG,A
5	MEND

MNT

MNT Index	Macro Name	MDT index
1	INCR	1

ALA

ALA Index	Formal Argument
#1	&ARG1
#2	&ARG2

Contd…

```
        MACRO
&LAB  INCR   &ARG1,&ARG2,&ARG3
&LAB   MOVER   AREG,  &ARG1
         A DD        AREG, &ARG2
         MOVEM    BREG,  &ARG3
        MEND

  …….
LOOP1  INCR   DATA1,DATA2,DATA3
  …….
LOOP2  INCR   DATA3,DATA2,DATA1
  ……..
DATA1  DC F’5’
DATA2   DC F’10’
DATA3   DC F’15’
……..
```

Example Contd…

- MDT

Index	instruction
1	&LAB INCR &ARG1,&ARG2,&ARG3
2	#0 MOVER AREG,#1
3	ADD AREG,#2
4	MOVEM BREG ,#3
5	MEND

ALA

Index	arguments
#0	“LOOP1BBB”
#1	“DATA1BBB”
#2	“DATA2BBB”
#3	“DATA3BBB”

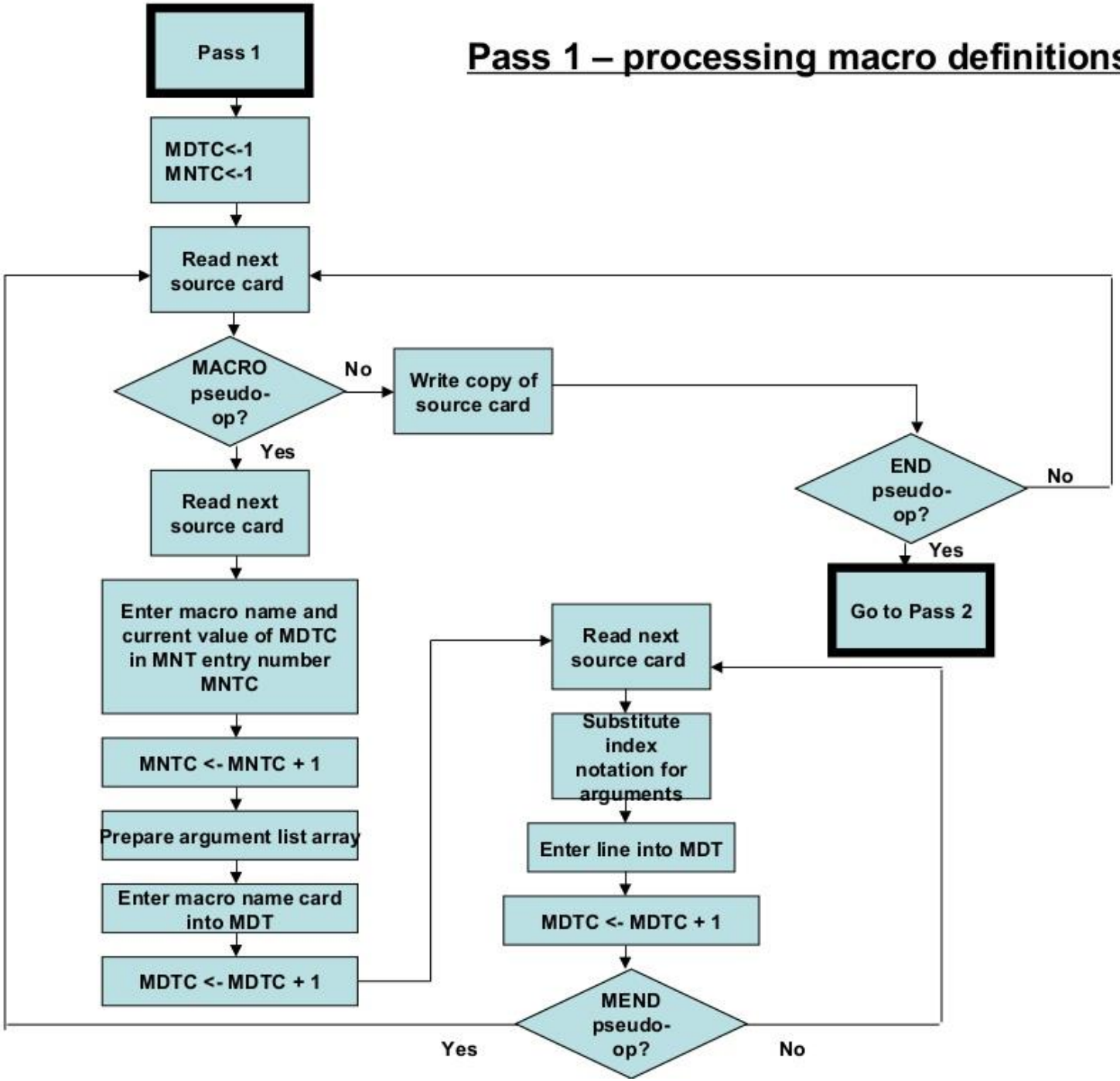
- MNT

Index	Name(8 bytes)	MDT index (4 bytes)
1	“INCRbbbb”	1

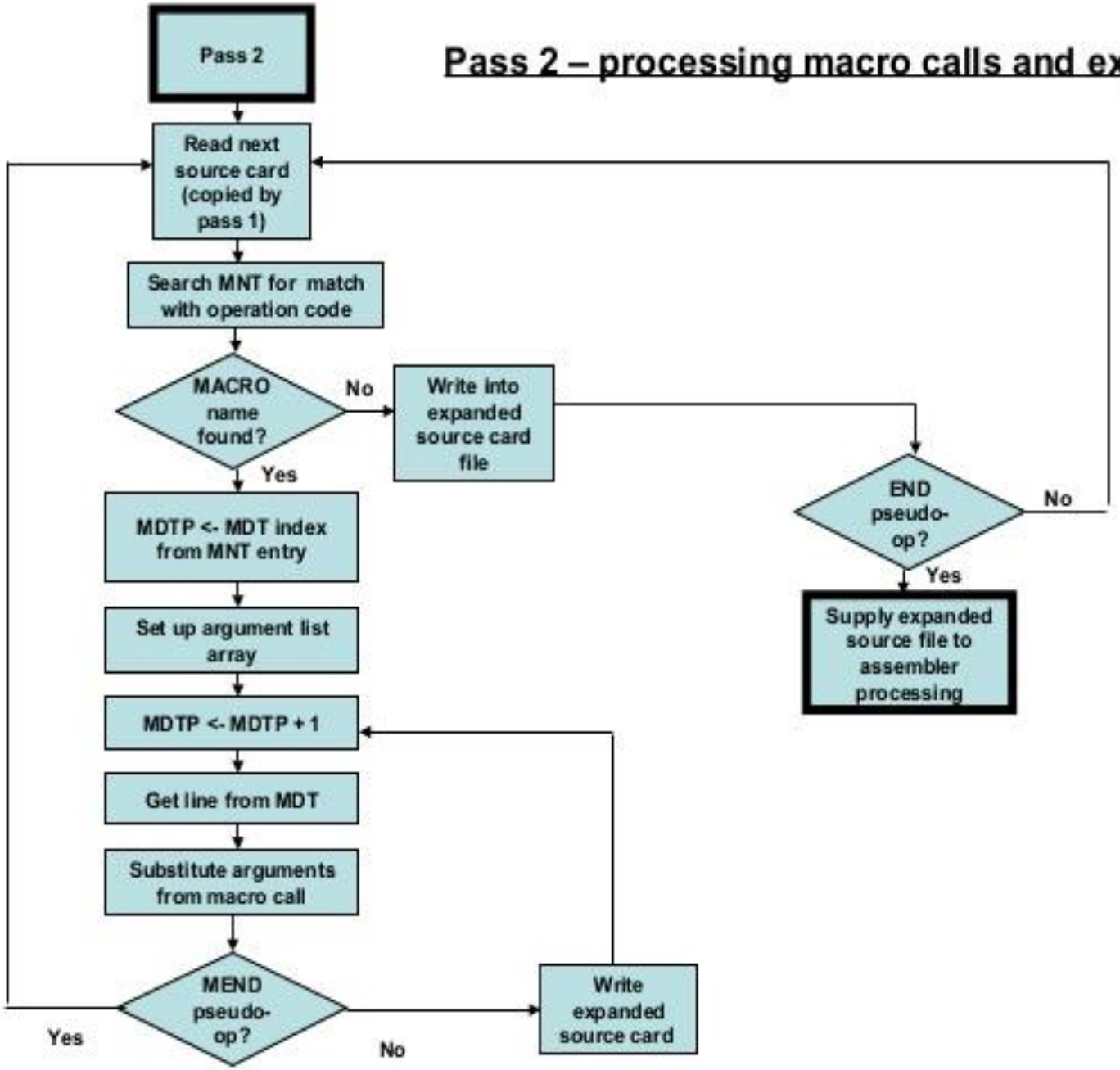
ALA

Index	arguments
#0	“BBBBBBB”
#1	“DATA3BBB”
#2	“DATA2BBB”
#3	“DATA1BBB”

Pass 1 – processing macro definitions



Pass 2 – processing macro calls and expansion



Contd..

```
START
……..
MACRO
INCR    &ARG1,&ARG2,&ARG3
 A          1,  &ARG1
 A          2, &ARG2
 A          3,  &ARG3
MEND
 …….
 INCR   DATA1,DATA2,DATA3
 …….
 INCR   DATA3,DATA2,DATA1
 ……..
 DATA1  DC F’5’
 DATA2   DC F’10’
 DATA3   DC F’15’
 …….
 END
```

Contd…

```
START
  MOVER  AREG,A
  MOVEM AREG,B
   ……
  MACRO
  VARY    &ARG1,&ARG2
   L      1,F'5'
   A      1,&ARG1
   A      1,&ARG2
   MEND
  MACRO
  INCR    &ARG1,&ARG2,&ARG3
   A          1,  &ARG1
   A          2, &ARG2
   A          3,  &ARG3
   MEND
    …….
   INCR   DATA1,DATA2,DATA3
    …….
   VARY    DATA3,DATA2
    ……..
   DATA1  DC F'5'
   DATA2   DC F'10'
   DATA3   DC F'15'
    …….
    END
```

Conditional macro expansion

```
            MACRO
&ARG0    VARY    &COUNT,&ARG1,&ARG2,&ARG3
&ARG0     A          1,  &ARG1
          A IF       (&COUNT EQ 1).FINI          TEST IF &COUNT=1
          A          2,  &ARG2
          A IF       (&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 code

```
      ……
Loop1  A     1,DATA1
       A     2,DATA2
       A     3,DATA3
  …….
Loop2  A     1,DATA3
       A     2,DATA2
       ……
Loop1  A     1,DATA1
  …….
DATA1  DC F’5’
DATA2  DC F’10’
DATA3  DC F ’15’
```

Contd…

- Pseudo opcodes AIF & AGO
- .FINI (labels starting with a period(.) are macro labels)and do not appear in the output of the macro processor
- AIF (conditional branch pseudo opcodes)
- AGO (unconditional branch pseudo opcodes) or goto statement.
- AIF & AGO control the sequence in which the macro processor expands the statements in macro instruction

Macro instructions defining macros

Def.of Macro define

def of macro &SUB

```
MACRO
DEFINE &SUB          macro name: DEFINE
MACRO
&SUB &Y              dummy macro name
CNOP 0,4
BAL 1,*+8            set register 1 to parameter list ptr
DC  A(&Y)            parameter list ptr
L   15,=V(&SUB)      address of subroutine
BALR  14,15          transfer control to subroutine
MEND
MEND
…..
…..
DEFINE COS           outer macro
…..
……
COS AR               inner macro
```

A single pass algorithm for macro processor

- **MDI** : macro definition input indicator
  works like switch
  keeps track of macro call

  MDI is ON
  -during expansion of a macro call
  -lines are read from MDT

  MDI is OFF
  all other times

  --the reading of MEND line indicates end of macro and terminates expansion of a call ,MDI is set off next line is obtained from regular i/p stream.

- **MDLC** : macro definition level counter
  -- keeps track of macro definition.
  -- MDLC is incremented by 1 when a MACRO is encountered and decremented by 1 when MEND occurs
  -- MDLC is used to insure that the entire macro def. including MACROs & MENDs get stored in MDT

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1		

ALA

ALA Index	Formal Argument	Actual Argument
#1		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	DEFINE &SUB
2	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	DEFINE &SUB
2	MACRO
3	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	DEFINE &SUB
2	MACRO
3	#1 &Y
4	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	DEFINE &SUB
2	MACRO
3	#1 &Y
4	CNOP 0,4
5	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions
1	DEFINE &SUB
2	MACRO
3	#1 &Y
4	CNOP 0,4
5	BAL 1,*+8
6	DC A(&Y)
7	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9		
10		
11		
12		
13		
14		
15		
16		
17		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11		
12		
13		
14		
15		
16		
17		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2		

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2		

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14	DC	A(#1)
15		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14	DC	A(#1)
15	L	15,=V(COS)
16		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14	DC	A(#1)
15	L	15,=V(COS)
16	BALR	14,15
17		

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
…….
END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14	DC	A(#1)
15	L	15,=V(COS)
16	BALR	14,15
17	MEND	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	

Expansion of Define Cos call

```
START
   ….
MACRO
   DEFINE &SUB
   MACRO
   &SUB &Y
   CNOP 0,4
   BAL 1,*+8
   DC  A(&Y)
   L   15,=V(&SUB)
   BALR  14,15
   MEND
   MEND
   …..
   …..
   DEFINE COS
   …..
   ……
   COS AR
 …….
 END
```

MDT

Index	instructions	
1	DEFINE	&SUB
2	MACRO	
3	#1	&Y
4	CNOP	0,4
5	BAL	1,*+8
6	DC	A(&Y)
7	L	15,=V(#1)
8	BALR	14,15
9	MEND	
10	MEND	
11	COS	&Y
12	CNOP	0,4
13	BAL	1,*+8
14	DC	A(#1)
15	L	15,=V(COS)
16	BALR	14,15
17	MEND	

MNT

MNT INDEX	MACRO NAME	MDT INDEX
1	DEFINE	1
2	COS	11

ALA

ALA Index	Formal Argument	Actual Argument
#1	&SUB	COS
#2	&Y	AR

Expansion of COS AR call

```
CNOP    0,4
BAL     1,*+8
DC      A(#1)
L       15,=V(COS)
BALR    14,15
```

Macro calls within macros

```
MACRO
ADD1     &ARG
MOVER  AREG, &ARG
ADD AREG, =‘1’
MOVEM   AREG,&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’
……..
```

Contd…

```
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'
END
```

expanded source level 1

```
ADD1 DATA1

ADD1 DATA 2

ADD1 DATA3
```

expansion of ADDS

expanded source level 2

```
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,DATA3
```

expansion of ADD1

```
START
…..
…..
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’
……..
END
```

```
START
…..
…..
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'
……..
END
```

Index	instructions
1	ADD1 &ARG
2	L 1,#1
3	A 1,F'1'
4	ST 1 ,#1
5	MEND
6	ADDS &ARG1,&ARG2,&ARG3
7	ADD1 #1
8	ADD1 #2
9	ADD1 #3
10	MEND

MNT INDEX	MACRO NAME	MDT INDEX
1	ADD1	1
2	ADDS	6

Contd…

- MDTP= 6 when ADDS is called
- And MDI is set ON .
- Then READ function increments MDTP , gets line from the MDT(line 7)
- Then MDTP = MDTP+1 =7
- So it is ADD1 DATA1

Then MDTP =1 so here previous value of MDTP i.e. 7 will be lost.

This the problem with macro calls within macros

- So it will work recursively.
- means to process one macro before it is finished with another then to continue with the previous or outer.
- Recursive procedures usually operate by means of stack.
- Each stack frame is associated with each recursive call
- Here status of unfinished computations is preserved.

References

- System Programming & Operating System - TE: Macro Definition and call, Nested Macro Calls, (wikinote.org)