



Lecture – 5



Section-B

Macro Language and Macro processor

References: John. J. Donovan



Introduction

- Features of Macro facility

Features of a Macro Facility

1) Macro Instruction Arguments:

- The macro facility presented thus far is capable of inserting blocks of instructions in place of macro calls.
- All of the calls to any given macro will be replaced by identical blocks.
- This macro facility lacks flexibility: there is no way for a specific macro call to modify the coding that replaces it.
- An important extension of this facility consists of providing for arguments, or parameters, macro calls. Corresponding *macro dummy arguments* will appear in macro definitions. Consider the following program:

• **Example 2:**

A
A
A

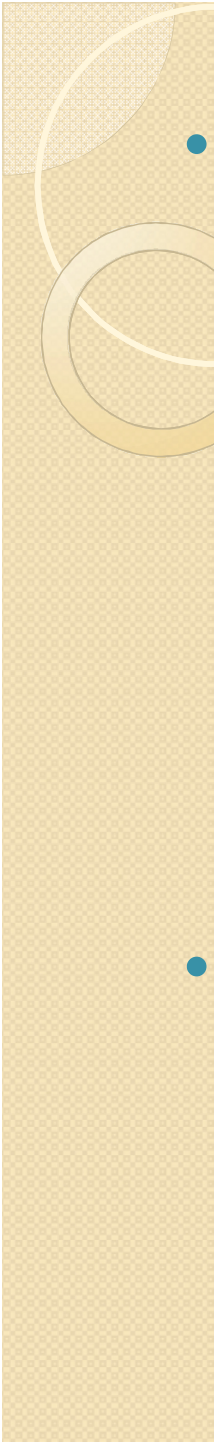
1, DATA1
2, DATA1
3, DATA1

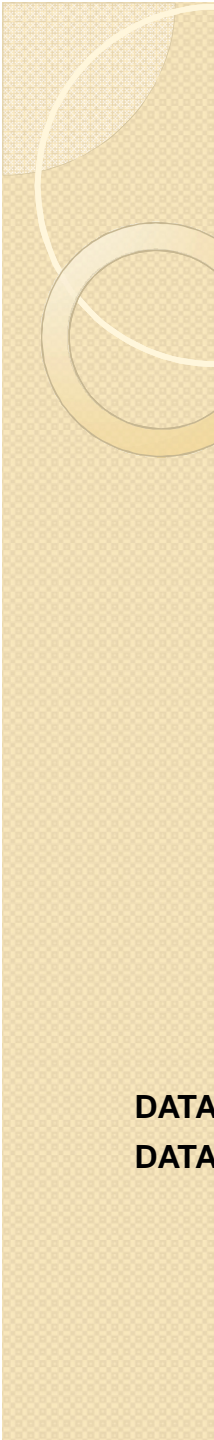
A
A
A

1, DATA2
2, DATA2
3, DATA2

DATA1 DC
DATA2 DC

F'5'
F'10'

- 
- 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 parameter or argument. Such a parameter is called a macro instruction argument or dummy argument.
 - It is specified on the macro name line and distinguished (as a macro instruction symbol rather than an assembly language symbol) by the ampersand (&), which is always its first character. The preceding program could be written as :



• **Source
MACRO**

INCR & ARG
A 1, & ARG
A 2, & ARG
A 3, & ARG
MEND

.
. .
. . .

INCR DATA 1 Use DATA1 as operand

.
. .
. . .

INCR DATA 2 Use DATA2 as operand

.
. .
. . .

DATA 1 DC
DATA 2 DC
. . .

**Macro INCR has
one argument**

A 1,DATA 1
A 2,DATA1
A 3. DATA1

A 1,DATA 2
A 2,DATA2
A 3. DATA2

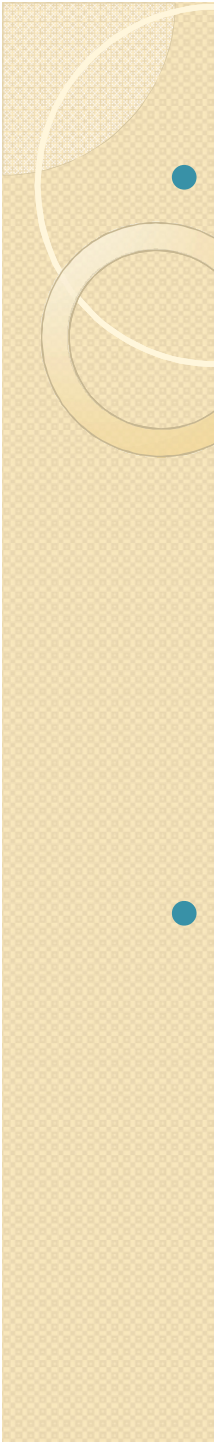
DATA1 DC
DATA2 DC

Expanded Source

.
. .
. . .

.
. .

F'5'
F'10'

- 
- It is possible to supply more than one argument in a macro call. Each argument must correspond to a definition (“dummy”) argument on the macro name line of the macro definition.
 - When a macro call is processed, the arguments supplied are substituted for the respective dummy arguments in the macro definition.

• **Example 3:**

LOOP1

A
A
A

1, DATA1
2, DATA2
3, DATA3

.

.

.

LOOP 2

A
A
A

1, DATA2
2, DATA2
3, DATA2

.

DATA1 DC
DATA2 DC
DATA 3 DC

F'5'
F'10'
F'15'

.

(2) Conditional Macro Expansion :

- Two important macro processor pseudo –ops , AIF and AGO, permit conditional reordering of the sequence of macro expansion.
- This allows conditional selection of the machine instruction that appear in expansions of a macro call.
- Consider the following program:

• **Example 4:**

```

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

```

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

.

Macro Calls Within Macros

- Some macro calls are “abbreviations” of instruction sequences, it seems reasonable that such “abbreviations” should be available within other macro definitions. For example

- Example 5:

MACRO

ADD1

&ARG

L

1, &ARG

A

1, = F'1'

ST

1, &ARG

MEND

MACRO

ADDS

&ARG1, &ARG2, &ARG3

ADD1

&ARG1

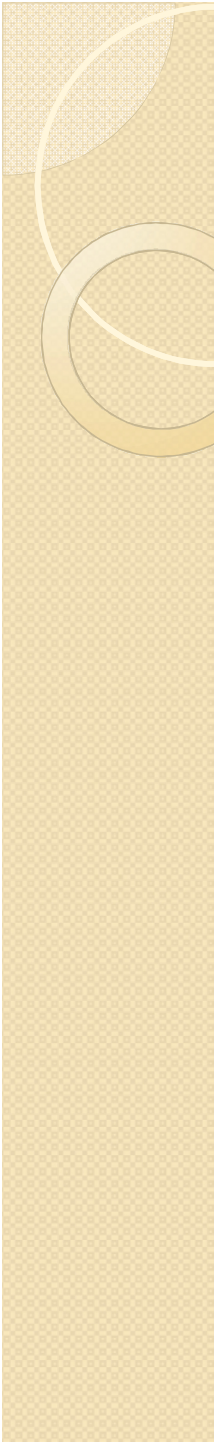
ADD1

&ARG2

ADD1

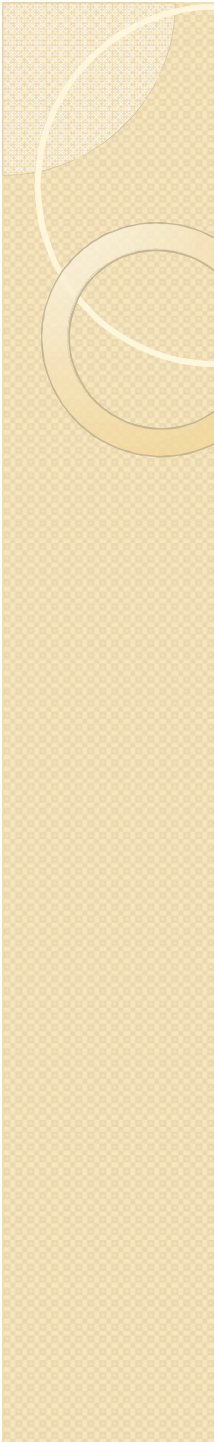
&ARG3

MEND

- 
- With the definition of the macro 'ADDS' are three separate calls to a previously defined macro 'ADD1'. The use of the macro 'ADD1' has shortened the length of the definition of 'ADDS' and thus has made it more easily understood.
 - Such use of macros results in macro expansions on multiple 'levels'.

(4) Macro Instructions Defining Macros

- In this manner a single macro instruction might be used to simplify the process of defining a group of similar macros.
- It is important to realize that the inner macro definition is not defined (i.e. callable) until after the outer macro has been called.
- This is because of the method by which definitions are implemented. For example, a user might wish to define a group of macros for subroutine calls with some standardized calling sequence.
- The following example defines a macro instruction DEFINE, which when called with a subroutine name defines a macro with the same name as the subroutine.



The individual macros generated bear the names (given through the argument &SUB) of their associated subroutines.

- For this program please refer John J. Donovan.
- Under the topic Macro instructions defining macros..
- **(Program is important)**

Applications

- In older operating systems such as those used on IBM mainframes, full operating system functionality was only available to assembler language programs, not to high level language programs (unless assembly language subroutines were used, of course), as the standard macro instructions did not always have counterparts in routines available to high-level languages.
- In modern operating systems such as Unix and its derivatives, operating system access is provided through subroutines, usually invoking DLL routines. High-level languages such as C offer comprehensive access to operating system functions, obviating the need for assembler language programs for such

Scope of research

- Making of a Macro

A Good example nicely explained at:

http://www.wowwiki.com/Making_a_macro

This is an article on **making a macro**. A [macro](#) is a list of slash commands. Common slash commands include the following:

- /say (/s)
- /whisper (/w, /talk, /t)
- /reply (/r)
- /emote (/e, /em, /me)
- /dance