**Chapter-3**

**Macro Language and Macro processor**

 Sometimes need to repeat some blocks of code several times in our program/task.

In such cases to avoid this repetition the system software will be provided one special component

i.e., MACRO

 Macro instructions is a notational convenience for the programmer, it allows the programmer to write a short hand version of a program

**Definition:**

 Macro instructions are the single line of abbreviation for group of instructions

**OR**

 The design of macro processor is generally machine independent.

 Every programming language must be uses the macro processor

**Ex:** C-programming uses a macro processor to support for defining symbolic constant #define, #include

**Basic Macro-Processor Functions**

There are 3 main functions of macro processor. They are

1. Macro definition

2. Macro calls or invocation

3. Macro expansion

**Macro Definition:**

Macro definition attaches a name to a sequence of instruction

**Structure of macro-definition**

MACRO starting of macro definition

[ ] give a macro\_name

…………….

……………. sequence of instruction

…………….

MEND ending of macro definition

The macro definition starts with MACRO pseudo op code. It indicates beginning of the macro definition. The Macro-definition terminated with the MEND pseudo op code.

**Ex:** Sequence of instruction will be

A 1, data

A 2, data

A 3, data

…..

…..

A 1, data

A 2, data

A 3, data

…..

…..

A 1, data

A 2, data

A 3, data

…..

…..

Data DC f „5‟

In the above example the sequence of

A 1, data

A 2, data

A 3, data

It repeats 3 times MACRO permits as to attach a name to this sequence and to use the name in its place then the macro definition will be

MACRO

Add

A 1, data

A 2, data

A 3, data

MEND

Where

MACRO=> is a pseudo op indicates beginning of definition

Add => is the name of the macro

MEND => is a pseudo op indicate end of the macro definition

Between the name of the Macro Add and MEND we have the sequence of instruction.

**2 MACRO Calls or Invocation:**

Once the macro has been defined, the use of the macro name in the place of sequence

**Definition:**

Sequence of instruction are simply substituted at the point of call or macro name is referred as macro call

Ex: In the above mentioned example sequence will be repeated thrice, then we need to replace sequence by macro name like:

MACRO

Add

A 1, data

A 2, data

A 3, data

MEND

…….

Add

……

Add

……

Data DC f ‘5’

**3 Macro Expansions:**

Whenever the program needs the instruction in the place of macro name then we need macro expansion.

**Definition:**

The macro-processor replace each macro calls with the defined set of instructions. This process of replacement is called macro expansion.

**Source Expanded source**

MACRO

Add

A 1, data

A 2, data does not appear

A 3, data

MEND

……. A 1, data

Add A 2, data

…… A 3, data

……

……

……

Add A 1, data

A 2, data

A 3, data

Data DC f ‘5’

**Features of macro facility:**

Thereare mainly 4 important feature of macro. They are

1. Macro instruction argument

2. Conditional macro expansion

3. Macro calls within macros

4. Macro instruction defining macros

**1 Macro instruction argument**:

As we have seen before, macro will be replaced by identical blocks. In such case no way for aspecific macro call to modifying, coding etc. operations

To overcome this problem we use macro instructions arguments where these arguments are appears in macro call. The corresponding macro dummy arguments are appears in macro

definition.

**Ex:**

A 1, data1

A 2, data2

……..

A 1, data1

A 2, data2

……..

In the above example operations are the same with different parameter value

The first sequence performs an operation using data1 as operand.

The second sequence performs an operation using data2 as operand.

**Keyword argument:**

The arguments which are presented in the macro definition known as **Keyword argument** or **dummy argument** these arguments must be preceded by & symbol

MACRO

Add &arg1,&arg2, &arg3

A 1,&arg1

A 2,&arg2

A 3,&arg3

MEND

**Positional argument:**

The arguments which are presented along with the macro call outside the definition referred as positional argument or actual argument.

Positional and keyword argument must be match according to the number.

**Ex:** Add a, b, c

Where

a replaces the first keyword argument

b replaces the second keyword argument

c replaces the third keyword argument

MACRO

Add &arg

A 1 &arg Keyword argument

A 2 &arg

MEND

……. A 1, a

Add a A 2, a

…….

Add b A 1, b

…… A 2, b

Data DC f ‘5’

**More than one in a Macro calls:**

**Source Expanded source**

MACRO

Add &arg1, &arg2, &arg3 Keyword argument

A 1, &arg1

A 2, &arg2

A 3, &arg3

MEND

……… A 1, a

Add a, b, c A 2, b

…… A 3, c

……

A 1, x

Add x, y, z A 2, y

…… A 3, z

Data DC f ‘5’

**2 Conditional Macro Expansions:**

The sequence of macro expansion can be reordered or change based on some conditions.

There are 2 Important macro processor pseudo op. they are

i. AIF

ii. AGO

**AIF:** It is a conditional branching pseudo op the format of AIF is

AIF<expression>**.**<sequencing label>

Where expression is a relational expression, it involves strings, numbers etc.

If the expression evaluates to true them the control transferred to the statements containing the sequencing label Otherwise, the control transferred to the next statement followed by AIF

AIF statement does not appears in the expanded source code

**AGO**:It is an unconditional branching pseudo op the format of AGO is

AGO**.**<Sequencing label>

It is conditional transfer control to the statement containing sequencing label

Each and every label must be starting with a .(dot) operator.

AGO statement does not appear in the expanded source code.

**Ex:**

Loop1 A 1,data1

A 2, data2

A 3, data3

Loop2 A 1,data3

A 2, data2

Loop3 A 1, data1

**3. Macro calls within the macros:**

Macro calls can be called only after being defined. Sometimes one macro will call another macro such a concept is called macro calls.

“Macro can be called within the definition of the another macro this is also referred as nested macros”.

The definition of called macro should appear before the macro definition which contains the macro call.

Ex:

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

Here we have the macro definition for the Add macro.

After this we will define another macro definition i.e.,Add1

Inside the definition of Add1.we have 3keyword argument to precisely defined macro Add

Expansion code of such macro’s leads to multiple levels

**4 Macro definitions within Macro definition**

This feature helps to defining a group of similar macro’s using a single macro instruction

It is possible to have macro definition with in the body of macro. The inner macro definition is not defined until the outer macro has been called.

**Ex:**

MACRO

DEFINE &fun

MACRO

&fun &arg

…………..

…………..

MEND

………….

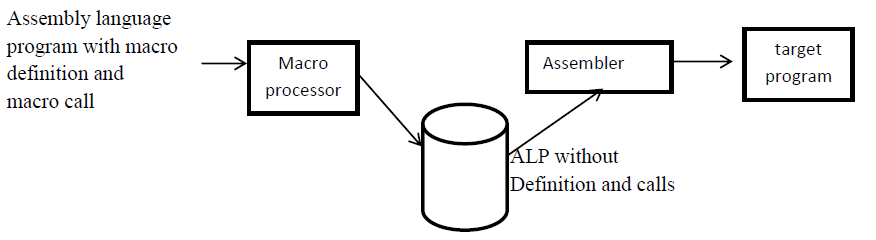
MEND

Above example defines a macro definition “DEFINE” with an argument & fun

Inside this macro definition we have another macro definition &fun

Using this feature we can dynamically generate the definitions for new macros.

**Implementation of a macro processor:**



The macro process taken as input an ALP which contains macro definition and macro calls. Then it transforms to expanded source without consisting macro definition and macro calls is through the translator it will be convert as an object code.

**Statement of a problem/ basic functions/task of a macro processor:**

There are 4 basic functions performed by macro processor.

They are:

**1 Recognize macro definition:** The macro processor must be recognizing macro definition by the MACRO and MEND pseudo op.

**2 Save the definition:** The macro processor must store the definition in memory which is required for expanding macro call.

**3 Recognize macro call:**The macro processor must organize macro names appears as operations mnemonics.

**4 Expanded calls and substitute arguments:**The macro substitute dummy/ macro definition arguments with the corresponding positional arguments in a macro call.

**Database Specification**

**Pass1 [Processing macro-definition and calls]**

1. The input macro source code
2. The output macro code copy to pass2
3. MDT[macro definition table] which is used to store the body of the macro definition
4. MNT[macro name table], which is used to store names of the defined macro
5. MDTC[Macro Definition table Counter] which is used to indicate the next available entry in MDT
6. MNTC[Macro Name Table Counter] which is used to indicate the next available entry in MNT
7. ALA[Argument List array] to substitute index marker for the dummy argument before storing a macro definition

**Pass2 [Processing Macro Expansions]**

1. Copy of the output macro source code from pass1

2. Output expanded source code to be used as input to the assembler

3. MDT created by pass1

4. MNT created by pass1

5. MDTP[Macro Definition Table Pointer] which is used to indicate the next line of text during expansion

6. ALA is the reverse function of pass1 to substitute macro call arguments(positional arguments) for the index markers.

**Arguments List array:**

It maintains the details about the parameters. It is used in both pass1 and pass2, but the functions are reverse in both the passes.

**ALA in pass1**

In this when the macro definition are stored the arguments in the definition are replaced by index markers.

# is the index marker, which is preceded by the dummy argument.

MACRO MACRO

Loop Add &arg1, &arg2, &arg3 #0 Add &arg1, &arg2,&arg3

A 1, &arg1 A 1, #1

A 2, &arg2 A 2, #2

A 3, &arg3 A 3, #3

MEND MEND

……..

……..

Add data1, data2, data3

In the above example loop is a label i.e., replaced by #0

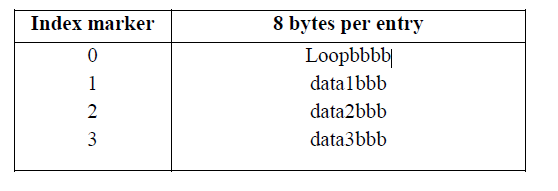
Arg1 replaced by #1 like so on

**ALA in pass2**

In this argument in the macro call are substituted for the index marker stored in macro definition

In the above example the macro call is

Loop Add data1, data2, data3



**MDT[Macro Definition table]**

* It is used to store the body of macro definition
* The size of MDT is 80-bytes per entry
* It will be read every line in the definition except MACRO

MACRO

Loop Add &arg1, &arg2, &arg3

A 1, &arg1

A 2, &arg2

A 3, &arg3

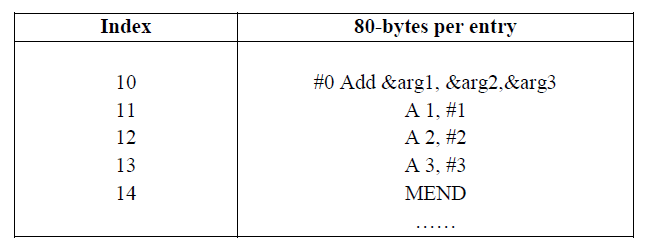
MEND

……..

……..

Add data1, data2, data3

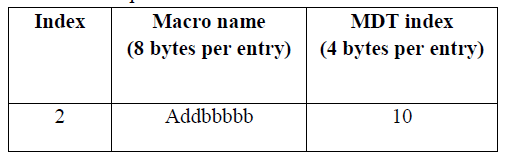
**The macro definition table will be**



**MNT[Macro Name Table]**

* It is used to store the names of the defined macros
* It has 3 fields, but the 2nd and 3rd field has the 12-bytes capacity
* The 2nd field is stored macro name it has 8 bytes and 3rd field MDT index for the macro name it has 4 bytes
* Total size of MNT is 12-bytes per entry

MNT for the above example is



**Pass1 Algorithm and Flowchart**

Pass1 is used to store and processing macro definitions

1. Initialize or set MDTC as well as MNTC=1

2. Read the first line from the source code

3.

a. If it is a macro pseudo op

b. The entire macro definition except macro is stored in MDT

c. Read next line from source

d. Enter macro name and MDTC value stored in MNT

e. Then increment MNTC value

f. Prepare ALA

g. Enter the macro name line in MDT then increment MDTC value

h. Read next line from source i.e., sequence then activate ALA in pass1

i. Substitute index marker for the dummy argument

j. Enter these values in MDT then increment MDTC value

k. Repeat these procedures until we get MEND pseudo op code

4.

a. If it is a MEND pseudo op code read the next line from source code

otherwise the same procedure will be continue.

b. If it is not a macro pseudo op code then copy the source code into pass2



**Pass2 Algorithm and Flowchart**

Pass2 is used for specifying macro calls and expansion

1. Read next line from the source code copied by pass1

2. Search macro name table for match with that code

a)

\*. Check whether you have encountered macro name then the MDT

index of that macro name entered into MDTP

\*. Then activate argument list array

\*. Increment MDTP value by one

\*. Read next line from MDT and substitute arguments for macro call

\*. Repeat this process until we get MEND pseudo op

\*. If it is a MEND pseudo op then, read the next statement from the

Source otherwise write the expanded source code

b) If it is not a macro name directly write into expanded source code

**Pass2 Flowchart**



**Simple One Pass Macro Processor**

There are 2 additional variables are used in simple one pass macro processor.

They are:

**MDI [Macro Definition Indicator]**

\*. It is used to keep track of macro calls

\*. It has 2 states ON and OFF

\*. While expanding a macro calls, then MDI=ON otherwise MDI=OFF

**MDLC [Macro Definition Level Counter]**

\*. It is used to keep track of macro definition

\*. It is a counter for MACRO and MEND pseudo op

\*. When macro pseudo op is encountered then MDLC is incremented by 1 and It is decremented by 1 when MEND pseudo op is encountered