System Programming *For* BCA 3rd Semester Unit 3	Compiled *By* Sakhi Bandyopadhyay Dept. of Computer Science & BCA, Kharagpur College, Kharagpur 721305

Macro and Macro Processor

Macro

- Formally, macro instructions (often called macro) are single-line abbreviations for groups of instructions.
- For every occurrence of this one-line macro instruction within a program, the instruction must be replaced by the entire block.
- The advantages of using macro are as follows:

  o Simplify and reduce the amount of repetitive coding.

  o Reduce the possibility of errors caused by repetitive coding.

  o Make an assembly program more readable.

Macro Processors

- A processor can be any program that processes its input data to produce output, which is used as an input to another program.
- The outputs of the macro processors are assembly programs that become inputs to the assembler.
- The macro processor may exist independently and be called during the assembling process or be a part of the assembler implementation itself.

Difference between Macro and Subroutine

Macro	Subroutine
Macro name in the mnemonic field leads to expansion only.	Subroutine name in a call statement in the program leads to execution.
Macros are completely handled by the assembler during assembly time.	Subroutines 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. The hardware knows nothing about macros.	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 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 processing increases the size of the resulting code but results in faster execution of program for expanded programs.	Use of subroutines does not result into bulk object codes but has substantial overheads of control transfer during execution.

Macro Definition and Call

- It has been aforementioned that a macro consists of a name, a set of formal parameters, and a body of codes.
- A macro can be defined by enclosing a set of statements between a macro header and a macro end statement.
- The formal structure of a macro includes the following features:

o Macro prototype statement: Specifies the name of the macro and name and type of formal parameters.

o Model statements: Specify the statements in the body of the macro from which assembly language statements are to be generated during expansion.

o Macro preprocessor statement: Specifies the statement used for performing auxiliary function during macro expansion.

- A macro prototype statement can be written as follows:
  <name_of_macro> [<formal parameter spec> [,...]]
  where [<formal parameter spec> [,...]] defines the parameter name and its kind, which are of the following form:
  &<name_of_parameter> [<parameter_type>]
- A macro can be called by writing the name of the macro in the mnemonic field of the assembly language. The syntax of a typical macro call can be of the following form:
  <name_of_macro> [<actual_parameter_spec> [,…]]
- The MACRO directive in the mnemonic field specifies the start of the macro definition and it should compulsorily have the macro name in the label field.
- The MEND directive specifies the end of the macro definition.
- The statements between MACRO and MEND directives define the body (model statements) of the macro and can appear in the expanded code.

• Eg.

Macro Definition

```
MACRO
INCR          &MEM_VAL, &INC_VAL, &REG
```

```
MOVER       &REG        &MEM_VAL
ADD         &REG        &INC_VAL
MOVEM       &REG        &MEM_VAL
MEND
```

Macro Call

```
INCR A, B
```

Macro Expansion

A macro call in a program leads to macro expansion. To expand a macro, the name of the macro is placed in the operation field, and no special directives are necessary. During macro expansion, the macro name statement in the program is replaced by the sequence of assembly statements. Let us consider the following example:

```
        START     100
        A         DS     1
        B         DS     1
+       MOVER     REG    A
+       ADD       REG    B
+       MOVEM     REG    A
        PRINT     A
        STOP
        END
```

The statements marked with a '+' sign in the preceding label field denote the expanded code and differentiate them from the original statements of the program.

Attributes of formal parameter

- An attribute is written using the syntax

  <attribute name> ' <formal parameter spec>
- It represents information about the value of the formal parameter, i.e. about the corresponding actual parameter.
- The type, length and size attributes have the names T, L and S.
- Example

```
MACRO
DCL_CONST       &A
AIF             (L'&A EQ 1) .NEXT
--
```

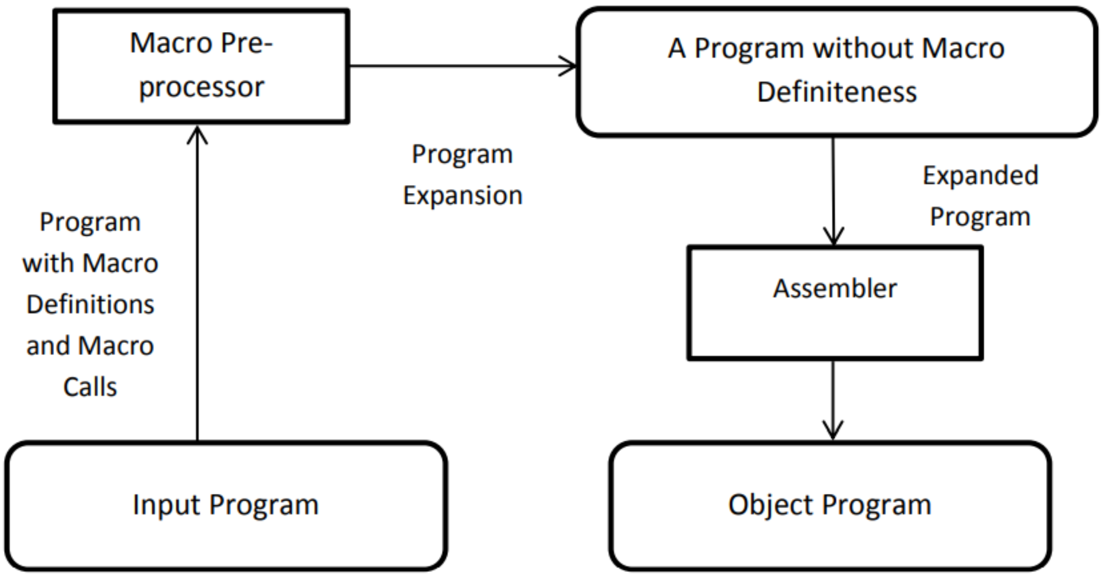
```
        .NEXT    --
                 --
                 MEND
```

Here expansion time control is transferred to the statement having .NEXT field only if the actual parameter corresponding to the formal parameter length of ' 1'.

Design of Macro Preprocessor

Macro preprocessors are vital for processing all programs that contain macro definitions and/or calls. Language translators such as assemblers and compilers cannot directly generate the target code from the programs containing definitions and calls for macros. Therefore, most language processing activities by assemblers and compilers preprocess these programs through macro processors. A macro preprocessor essentially accepts an assembly program with macro definitions and calls as its input and processes it into an equivalent expanded assembly program with no macro definitions and calls. The macro preprocessor output program is then passed over to an assemble to generate the target object program.

The general design semantics of a macro preprocessor is shown as below



The design of a macro preprocessor is influenced by the provisions for performing the following tasks involved in macro expansion:

• **Recognize macro calls:** A table is maintained to store names of all macros defined in a program. Such a table is called Macro Name Table (MNT) in which an entry is made for every macro definition being processed. During processing program statements, a match is done to compare strings in the mnemonic field with entries in the MNT. A successful match in the MNT indicates that the statement is a macro call.

• **Determine the values of formal parameters:** A table called Actual Parameter Table (APT) holds the values of formal parameters during the expansion of a macro call. The entry into this table will be in pair of the form (,). A table called Parameter Default Table (PDT) contains information about default parameters stored as pairs of the form (,) for each macro defined in the program. If the programmer does not specify value for any or some parameters, its corresponding default value is copied from PDT to APT.

• **Maintain the values of expansion time variables declared in a macro:** A table called Expansion time Variable Table (EVT) maintains information about expansion variables in the form (,). It is used when a preprocessor statement or a model statement during expansion refers to an EV.

• **Organize expansion time control flow:** A table called Macro Definition Table (MDT) is used to store the body of a macro. The flow of control determines when a model statement from the MDT is to be visited for expansion during macro expansion. MEC {Macro Expansion Counter) is defined and initialized to the first statement of the macro body in the MDT. MDT is updated following an expansion of a model statement by a macro preprocessor.

• **Determine the values of sequencing symbols:** A table called Sequencing Symbols Table (SST) maintains information about sequencing symbols in pairs of the form
(<sequencing symbol name>, <MDT entry #>)
Where **<MDT entry #>** denotes the index of the MDT entry containing the model statement with the sequencing symbol. Entries are made on encountering a statement with the sequencing symbol in their label field or on reading a reference prior to its definition.

• **Perform expansion of a model statement:** The expansion task has the following steps:

o MEC points to the entry in the MDT table with the model statements.

o APT and EVT provide the values of the formal parameters and EVs, respectively.

o SST enables identifying the model statement and defining sequencing.

Functions of Macro Processor

The design and operation of a macro processor greatly influence the activities performed by it. In general, a macro processor will perform the following tasks:

• Identifies macro definitions and calls in the program.

• Determines formal parameters and their values.

• Keeps track of the values of expansion time variables and sequencing symbols declared in a macro.

• Handles expansion time control flow and performs expansion of model statements.

Design of Two-pass Macro Preprocessor

Pass 0 of Assembler

The activities of pass-0 macro processor is given in the following steps:

1. Read and examine the next source statement.

2. If MACRO statement, continue reading the source and copy the entire macro definition to the MDT. Go to Step 1.

3. If the statement is a pass-0 directive, execute it. Go to Step 1. (These directives are written to the new source file in a unique manner (different from normal directives). They are only needed for the listing in pass 2.

4. If the statement contains a macro name, it must perform expansion, that is, read model statements from the MDT corresponding to the call, substitute parameters, and write each statement to the new source file (or execute it if it is a pass-0 directive). Go to Step 1.

5. For any other statement, write the statement to the new source file. Go to Step 1.

6. If the current statement contains the END directive, stop (end of pass 0).

The assembler will be in one of the three modes:

• In the normal mode, the assembler will read statement lines from the source file and write them to the new source file. There is no translation or any change in the statements. In the macro definition mode, the assembler will continuously copy the source file to the MDT.

• In the macro expansion mode, the assembler will read statements from the MDT, substitute parameters, and write them to the new source file. Nested macros can be implemented using the Definition and Expansion (DE) mode.

Pass 1 of Macro Processor - Processing Macro Definitions

1. Initialize MDTC and MNTC.

2. Read the next source statement of the program.

3. If the statement contains MACRO pseudo-op. go to Step 6.

4. Output the instruction of the program.

5. If the statement contains END pseudo-op, go to Pass 2, else go to Step 2.

6. Read the next source statement of the program.

7. Make an entry of the macro name and MTDC into MNT at location MNTC and increment the MNTC by 1.

8. Prepare the parameter (arguments) list array.

9. Enter macro name into the MDT and increment the MTDC by 1.

10. Read the next card and substitute index for the parameters (arguments).

11. Enter the statement into the MDT and increment the MDT by 1.

12. If MEND pseudo-op found, go to Step 2, else go to Step 10.

Pass 2 of Macro Processor - Processing for Calls and Expansion of Macro

1. Read the next source statement copied by pass 1.

2. Search into the MNT for record and evaluate the operation code.

3. If the operation code has a macro name, go to Step 5.

4. Write the statement to the expanded source file.

5. If END pseudo-op found, pass the entire expanded code to the assembler for assembling and stop. Else go to Step 1.

6. Update the MDTP to the MDT index from the MNT entry.

7. Prepare the parameter (argument) list array.

8. Increment the MDTP by 1.

9. Read the statement from the current MDT and substitute actual parameters (arguments) from the macro call.

10. If the statement contains MEND pseudo-op, go to Step 1, else write the expanded source code and go to Step 8.