

System Programming

Second Class

مدرس مساعد: منال مطلوب

CHAPTER SIX (Macro processors)

Introduction:

* Macro is a unit of specification for program generation through expansion.

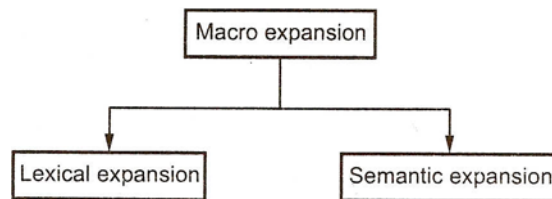
* A macro represents a commonly used group of statements in the source programming language. The macro processor replaces each macro instruction with the corresponding group of source language statements. This is called expanding of macros.

Basic Macro Processor Functions:

We study the fundamental functions that are commonly used by all macro

Macro Definition and Expansion

- Macro is a unit of specification for program generation through expansion.
- A macro consists of a name, a set of formal parameters and body of code.



- **Macro expansion** is a macro name with a set of formal parameters is replaced by some code.
- **Lexical expansion** is replacement of a character string by another character string during program generation.
- **Semantic expansion** is generation of instructions tailored to the requirements of specific usage.

- Macro definition consists of
 - i) macro prototype ii) one of more model iii) macro preprocessor
- Macro definition is in between a macro header statement and a macro e statement.
- Syntax of macro prototype statement
 - < macro name> [< formal parameter spec> [;
 - ...]] where
 - < macro name> = It is in the mnemonic field of an assembly statement < formal parameter spec > is in the following form
 - < parameter name > [< parameter kind >]
- Syntax of the macro call is
 - < macro name> [< actual parameter spec> [; ...]]

Macro Expansion

- Macro call leads to macro expansion.
- Macro call statement is replaced by a sequence of assembly statement in macro expansion.
- Two key notions are used in macro expansion
 - a)Expansion time control flow
 - b)Lexical substitution
- Algorithm for macro expansion
 1. Initialise the macro expansion counter (MEC)
 2. Check the statement which is pointed by MEC is not a MEND statement a) if the statement is model statement
 - then
 - expand the statement
 - and increment the MEC
 - by 1
 - else
 - MEC := new value specified in the statement;
 3. Exit from the macro expansion
- RDBUFF and WRBUFF are the two macro instruction used in the Fig. 4.2 SIC/XE program. MACRO and MEND are the two new assembler directives also used. RDBUFF is the name of macro which is in the label field. The entry in the operand field identify the parameters of the macro instruction.

- Each parameter begins with the character & which facilitates the substitution of parameters during macro expansion.
- The macro name and parameters define a pattern or prototype for the macro instruction used by the programmer. Body of the macro definition is defined by the MACRO¹ directive statements (Line no. from 15 to 90). Macro expansion generate these statements. End of the macro definition is defined by the MEND assembler directive.
- The main program start from line 180. Macro invocation statement that gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro (Line 190).
- Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from the macro invocation substituted for the parameters in the macro prototype. Parameters and arguments, both are associated with one another according to their positions.
- The argument F1 is substituted for the parameter &INDEV whenever it occurs in the body of the macro. BUFFER is substituted for and &BUFADR &LENGTH is substituted for and &RECLTH.

After macro processing, the expanded file can be used as input to the assembler.

Macro Processor Algorithm and Data Structures:

- For designing two pass macro processor, all macro definitions are processed during the first pass and all macro invocation statements are expanded during the second pass. This type of two pass macro processor would not allow the body of one macro instruction to contain definitions of other macros. The reason behind is that, all macros defined during the first pass before any macro invocation were expanded.
- Following example shows the definition of macros within a macro body.

```

1  MACROS  MACRO    //define SIC standard version macros
2  RDBUFF  MACRO
      .
      .
3          MEND    //END OF RDBUFF
4  WRBUFF  MACRO
      .
5          MEND    //END OF WRBUFF
      .
6          MEND    //END OF MACROS

```

figure (A)

```

1   MACROX  MACRO
2   RDBUFF  MACRO
      .
      .
3           MEND      //END OF RDBUFF
4   WRBUFF  MACRO
      .
      .
5           MEND      //END OF WRBUFF
      .
      .
6           MEND      //END OF MACROX

```

figure (B)

- The body of first macro (MACROS) contains statements that define RDBUFF WRBUFF and other macro instructions for a SIC system. Second macro instruction body (MACROX) defines these same macros for a SIC\XE system. The same program could run on either a standard SIC machine or a SIC\XE machine. Invocation of MACROS or MACROX is only changed for use. Defining MACROS or MACROX does not define RDBUFF and other macro instructions. These definitions are processed only when an invocation 0MACROS or MACROX is expanded.
- Macro processor uses three main data structure.
 1. Definition table (DEFT AB)
 2. Name table (NAMTAB)
 3. Argument table (ARGT AB)
- Definition table stores macro definitions. It contains the macro prototype and the statements that make up the macro body. The macro names are also entered into NAMT AB, which serves as an index to DEFT AB.
- When macro instruction defined, NAMTAB contains pointers to the beginning and end of the definition in DEFTAB.
- Argument table (ARGT AB) is used during the expansion of macro invocations. When a macro invocation statement is recognized the argument are stored in ARGTAB according to their position in the argument list.