Macro Processors (MP)

- Macro:
  - A commonly used group of statements.
- Macro Processors (MP):
  - Expands macros, i.e, each macro reference is replaced by its corresponding group of the statements
- Examples:
  - SAVEREGS
  - LOADREGS
- In general, a MP does not perform analysis of the text. So can be considered to be independent from machines.
- MP for assembler, compiler, OS command language
- Also general-purpose MP.
- Macros are also part of an assembly program.

Basic Macro processor Functions

- Macro definition, invocation, and expansion.
- Assembler Directives for Macro definition
  - MACRO and MEND
  - name & params forms a prototype for the macro.
- Use of Macros in a SIC/EX program (Figure 4.1)
  - RDBUF and WRBUF
  - Line 190: Macro invocation, name arguments
- Program from Figure 4.1 with Macros expanded (Figure 4.2)
- Macro invocation and function call:
  - MP expands a macro for each invocation
  - Replaces a parameter with its corresponding argument.
  - Macro is removed from the object code.
  - In contrast, a function is kept in the program, a function call will jump to the function, and the call returns after the function is finished.
  - Similarly, arguments are used to replace parameters.
  - Comparison of Macro and function call
- Notes:
  - No labels in a macro body. Why?
  - JEQ *-3 in Line 140 and JLT *-14 in Line 155.
  - An inconvenient and error-prone method. How to avoid it?

MP algorithm and Data structures

- Typical two-pass MP:
  - First pass: process all macro definitions
  - Second pass: expand all macro invocations.
  - But, it will not allow the body of one macro to contain definitions of other macros.
  - Nested definition of macros is useful some time.
    - Example of the definition of macros within a macro body (Figure 4.3)
    - Change the invocation to MACROS or MACROX
- One-pass MP
  - Suppose allowing alternation between macro definition and macro expansion
  - Macro definition must be before its invocation.
    - satisfy programmers' expectation.
  - Three Data structures: NAMTAB, DEFTAB, and ARGTAB
    - NAMTAB points to both the beginning and end of a macro
    - ?1, ?2, … are placed in the macro definition in DEFTAB
    - ?1, ?2…, are replaced by the corresponding indexed arguments from ARGTAB during expansion
  - Contents of MP tables for Program in Fig. 4.1 (Figure 4.4)
  - Algoritm for a one-pass MP (Figure 4.5) and Figure 4.5 (Cont'd)
    - Procedure DEFINE to process a macro definition and enter symbols and macro in NAMTAB and DEFTAB.
    - Procedure EXPAND to enter arguments to ARGTAB and expand the macro.
    - Procedure GETLINE to read a line from source file or DEFTAB.
    - LEVEL to record the level of macro definition for nested macros.
- Standard macro library.

Machine-independent MP features

- Concatenation of MACRO parameters
  - &ID is a parameter and X&ID1, X&ID2, …
  - When argument is A, get XA1, XA2, …
  - If &ID and &ID1 are both parameters,
    - X&ID->1, X&ID->2, …
  - Concatenation of macro parameters
- Generation of unique labels for macro
  - Too long relative jump may be inconvenient, error prone, and difficult to read.
  - $ prefixed to a symbol to indicate a dynamic label
  - $ is replaced with $AA, $AB when expansion.
  - So $Loop will be $AALoop for the first expansion and $ABLoop for the second expansion.
  - Generation of unique labels within macro expansion
- Conditional Macro expansion
  - Modify the sequence of statements during expansion depending on the arguments provided.
  - More generally, called conditional assembly.
  - &EOR, &MAXLTH: parameters to control conditional macro expansion.
  - IF … ELSE …ENDIF to indicate the expansion depending on the condition
  - SET is a MP directive, and a symbol beginning with & but not in parameters, is a macro time symbol, or a set symbol
  - Use of macro-time conditional statements (Figure 4.8) and Figure 4.8 (Cont'd) (b,c,d) with three different invocations
  - Maintain a macro time symbol table
  - Evaluate the Boolean expression in IF
  - Use the statements in IF or ELSE part for expansion.
  - No nested IF. But can be extended to allow it.
  - All conditional macro expansion must be finished before assembly (only on sequence of source statements without any conditional expansion directive)
  - WHILE … ENDW to generate repeated expansion of statements within WHILE until the Boolean expression in WHILE is evaluated to FALSE.
  - %NITEMS(…) : a macro processor function.
  - Use of macro time looping statements
  - Does not allow nested WHILE. But can be extended.
- Keyword parameters
  - Previously, positional parameters, arguments match parameters by positions.
  - Except the last arguments which may be omitted, other omitted argument must have ,, to indicate its position.
  - So keyword parameters. &key-work-para=default-value.
  - Use of keyword parameters in macro instructions (Figure 4.10) and Figure 4.10 (Cont'd)

MP Design Options --Recursive Macro Expansion

- Example of nested macro invocation (Figure 4.11)
  - RDCHAR is a macro and is invoked in macro RDBUFF.
  - Unfortunately, the MP algorithm discussed so far cannot process this. Why?
    - When RDBUFF is invoked, procedure EXPAND is called, EXPANDING is set to true, and arguments (BUFFER, LENGTH, F1) are entered to ARGTAB. During expansion, RDCHAR is invoked, EXPAND is called again, new argument (F1) enters ARGTAB (to erase the arguments of RDBUFF). After returning from EXPAND, EXPANDING is set to false, which stops the expansion of RDBUFF.
  - EXPAND is not called recursively.
  - If recursive call is allowed, it will work.
  - The essential technique dealing with recursive calls is *Stack*.
  - Chapter 5, compiler will determine whether a language allows recursive function call.

MP Design Options
--MP within language translators

- MP as an input subroutine (pre-processing):
  - Read the source program line by line
  - Process the macro if definition and invocation are encountered
  - The output lines are passed to assembler or compiler.
  - Advantages:
    - Avoid passing source program again
    - Share Data structures.
    - Share utility functions.
    - Easier to diagnose errors.
- Integrated MP:
  - Assembler or compiler takes main control to do the analysis
    - DO 100 I = 1, 20 (which is a loop statement) and DO 100 I =100, which is an assignment.
  - When encountering macros, call MP to finish necessary processing.
  - Can support context-dependent features,
    - For example, a macro can specify a substitution to be applied only to variables or constants of a certain type, or only to variables appearing as loop indices in DO statement.
    - The expansion of a macro could also depend on a variety of characteristics of its arguments.
  - Disadvantages:
    - Specially designed
    - Development cost
    - Larger and more complex.

MP Design Options --General-purpose MPs

- Advantages:
  - No need to care for different facilities for different languages.
  - Once developed (even cost more), can be used for any language, and also for a long time.
- Disadvantages:
  - Many details associated with a real language, so a general purpose MP must provide some way for a user to define specific set of rules to be followed.
    - For examples, different comment tags for different language.
    - Grouping delimiter: Begin … End, {…}
    - Length of identifiers, format of constants, :=, ** (double symbols OP), blanks as delimiter?, line continuation, statement formatting.
    - Different syntax for macro definitions and invocations in different languages.
  - Briefly discuss a general purpose MP in 4.4.3.

Implementation examples --Microsoft MASM MP

- Integrated with the pass 1 of MASM assembler
- All basic functions of a MP, including nested macro definition.
- Also recursive macro invocation
- A macro may be redefined within a program.
- Main differences from SIC MP:
  - Conditional macro expansion, more generally conditional assembly
  - Not only related to macros but also outside macros.
  - Examples of MASM macro and conditional statements (Figure 4.12)
    - Word or double word, so register AX or EAX
    - No need & for parameters
    - EXIT is a local label. When the macro is expanded, local label is replaced by unique name ??xxxx.
    - Line 4 to 8: nested conditional statements.
    - Line 10, & is a concatenation operator.
    - ;; comments
  - Iteration statement IRP:
    - Example of MASM iteration statement (Figure 4.13)
    - Macro time variable S, which can have values from <…>
    - For each value of S, statements between IRP and ENDM are generated once.

Implementation examples --ANSI C Macro language

- C high level language
- Called pre-processor
- For examples:
  - #define NULL 0, #define EOF (-1), #define EQ ==
  - Also called constant definition
- More complicate, macro with parameters:
  - #define ABSDIFF(x,y)   ((x)>(y)? (x)-(y): (y)-(x))
  - So ABSDIFF (I+1,J-5), ABSDIFF(I,3.1415), ABSDIFF(‘D’,’A’).
  - #define ABSDIFF(x,y)   x>y? x-y: y-x will result in problem.
  - Note: the importance of parentheses, due to the simple string substitution.
- Conditional compilation
  - Example1
    - #ifndef BUFFER_SIZE
    - #define BUFFER_SIZE 1024
    - #endif
  - Example 2
    - #define DEBUG 1
    - …
    - #if DEBUG==1
    - printf(….);
    - #endif
    - pintf(…) will appears in the program. If change #define DEBUG 0, then the printf(…) will not.
    - Also #ifdef DEBUG
    -     printf(…);
    -     #endif
    - In this case, #define DEBUG will have printf(…) in the program and printf(…) will not appear in program without #define DEBUG

Implementation examples
--The ELENA Macro Processor (general-purpose)

- Macro definition: a header and a body
  - Header is not required to have any special form.
  - Consists of a sequence of keywords and parameter markers (beginning with %) and at least one of the first two tokens must be a keyword.
  - Body can be defined in any language specific form.
- For examples:
  - A macro header: %1 = %2 +%3 (=, + are keywords)
    - Can be invoked as ALPHA=BETA+GAMMA
  - A header: ADD %1 TO THE VALUE OF %2
    - Can be invoked as ADD 10 TO THE VALUE OF INDEX
- Examples of ELENA macro definition and invocation with different languages (Figure 4.14)
  - Same header but the body is C or x86 assembly language.
- Example of ELENA macro-time variable and repeat expansion (Figure 4.15)
  - Very generic in term of body
- Complexity:
  - No macro "name" in the headers
    - ADD %1 TO %2 or ADD %1 TO THE FIRST ELEMENT OF %2
  - Even not clear from macro invocation which token is keyword and which is parameter
    - An invocation: DISPLAY TABLE can be for a macro with header DISPLAY %1 or %1 TABLE
    - Note: DISPLAY and TABLE cannot be both parameters, since at least one must be keyword.
  - ELENA deals with it
    - By constructing an index of all macro headers according to the keywords in the first two tokens of the header.
    - Potential macro invocations are compared against all headers with keywords that match at least one of their first two tokens.
  - For example
    - A SUM B, C would be compared with all macro headers in which the first token is A or the second token is SUM.
    - When more than one headers match, the header with the fewest parameters is selected.
      - A=B+1 can match %1=%2+%3 or %1=%2+1, in this case, %1=%2+1 is selected.
    - If same amount of parameters, the most recently defined one is selected.

Summary

- Basic functions
  - Definitions and expansions
- Features
  - Labels, nested definitions, recursive invocations.
  - Conditional macro processing.
    - IF…ENDIF, WHILE…ENDW, macro-time variables and instructions
  - Keyword parameters
- Data structures and algorithms
  - NAMTAB, DEFTAB, ARGTAB
  - For recursive invocation, STACK.
- Relation between macro processors and assemblers