Assignment No. 3

Study Assignment for Macro Processor. (Consider all aspects of Macro Processor)

Objective:

1. To understand macro facility, features and its use in assembly language programming.
2. To study how the macro definition is processed and how macro call results in the expansion of code.

Theory:

An assembly language macro facility is to extend the set of operations provided in an assembly language.

In order that programmers can repeat identical parts of their program macro facility can be used. This permits the programmer to define an abbreviation for a part of program & use this abbreviation in the program. This abbreviation is treated as macro definition & saved by the macro processor. For all occurrences the abbreviation i.e. macro call, macro processor substitutes the definition.

Macro definition part:

It consists of

1. Macro Prototype Statement - this declares the name of macro & types of parameters.
2. Model statement - It is statement from which assembly language statement is generated during macro expansion.
3. Preprocessor Statement - It is used to perform auxiliary function during macro expansion.

Macro Call & Expansion:

The operation defined by a macro can be used by writing a macro name in the mnemonic field and its operand in the operand field. Appearance of the macro name in the mnemonic field leads to a macro call. Macro call replaces such statements by sequence of statement comprising the macro. This is known as macro expansion.

Macro Facilities:

1. Use of AIF & AGO allows us alter the flow of control during expansion.
2. Loops can be implemented using expansion time variables.

Design Procedure:

1. Definition processing - Scan all macro definitions and for each macro definition enter the macro name in macro name table (MNT). Store entire macro definition in macro definition table (MDT) and add auxiliary information in MNT such as no of positional parameters (#PP) no of key word parameters (#KP), macro definition table position (MDTP) etc.
2. Macro expansion - Examine all statement in assembly source program to detect the macro calls. For each macro call locate the macro in MNT, retrieve MDTP, establish the correspondence between formal & actual parameters and expand the macro.

Data structures required for macro definition processing -

1. Macro Name Table [MNT] - Fields-
   Name of Macro, #pp (no of positional parameters), # kp(no of keyword parameters),

, MDTP (Macro Definition Table Pointer), Keyword Parameters Default Table Position (KPDTP),

2. Parameter Name Table [PNTAB] -
   Fields - Parameter Name
3. Keyword parameter Default Table [KPDTAB] -
   Fields - Parameter Name, Default value
4. Macro Definition Table [MDT] -
   Model Statement are stored in the intermediate code from as:
   Opcode, Operands.

Algorithm for definition processing:

Before processing any definition initialize KPDTAB_ptr, MDT_ptr to 0 and MNT_ptr to -1. These table pointers are common to all macro definitions. For **each** macro definition perform the following steps.

1. Initialize PNTAB – ptr to 0 & fields of MNT, # pp, # kp to 0 and increment MNT_ptr by 1.
2. For macro prototype statement from MNT entry.
   a. Entry name into name field.
   b. For each position parameter field
   i. Enter name in parameter name table.
   ii. Increment PNTAB – ptr by 1.
   iii. Increment # pp by 1.
   c. KPDTP ← KPDTAB - ptr
   d. For each keyword parameter
   i. Enter name & default value in KPDTAB.
   ii. Increment KPTAB –ptr by 1.
   iii. Enter name in PNTAB & increment PNTAB – ptr by 1.
   iv. Increment # kp by 1.
   e. MDTP ← MDT – ptr (current MDT Ptr)
3. Read next statement
   a. Model statement
      i. For parameter generate specification (p, # n).
      ii. Record intermediate code in MDT.
      iii. Increment MDT - ptr by 1.
   end
   b. If MEND statement
      Begin
      Enter MEND in MDT, increment MDT_ptr by 1.
      If #kp == 0 then KPDTP = 0
      Return to main logic ie step 6 of main logic.

Data structures required for expansion processing :-

1. Actual parameter table: APTAB
2. Macro expansion counter : MEC.

Algorithm for macro expansion:

1. Initialization
   i. MEC ← MDTP from MNT.
   ii. Create APTAB with # pp & # kp entries and set APTAB ptr accordingly.
   iii. Copy keyword parameter defaults from KPDTAB in APTAB[pp] to APTAB[#pp + #kp -1].

iv. Process actual positional parameters in call and copy them in APTAB from 0 to # pp-1.
v. For keyword parameter specification search name in parameter name field of KPDTAB, get matching entry in q & enter value in APTAB [#pp + q – KPDTP].

2. While Statement pointed by MEC in MDT is not MEND.
i. If model statement then replace operands of the form (p, #n) by values in APTAB.
ii. Increment MEC by one.
iii. Write the model statement on expanded code file.

3. Exit from macro expansion

Main Program Logic :

1. Initialize KPDTAB_ptr, MDT_ptr to 0 and MNT_ptr to -1. These table pointers are common to all macro definitions (There could be more than one macro definition in program)
2. Read the statement from source file, one line at time
3. Separate the words from that line and count the no of words. Save the separated words in the array say word which is a array of strings
4. If count for words in a line is one then check if that only word matches with "MACRO" keyword, if MACRO keyword found then perform definition processing.
5. If it does not match then check whether first word of the line matches with any of the entries in the macro name table. (Search the complete macro name table for presence of macro call), if so then perform macro expansion routine.
6. If no Macro call or no definition then enter the line as it is in the expanded code file.
7. If not end of file go to step 3.

Sample Input: -

The assembly language program with macro definitions & macro calls

```
        MACRO
        MAC1
        MOVER  AREG,  M
        ADD    BREG,  M
        MOVEM  CREG,  M
        MEND
        MACRO
        EVAL &X,&Y,&Z
        MOVER AREG, &X
        SUB AREG, &Y
        ADD AREG, &Z
        MOVER AREG, &Z
        MEND
        MACRO
        CALC &X,&Y,&OP=MULT,&LAB=
&LAB    MOVER AREG, &X
        &OP AREG, &Y
        MOVEM AREG, &X
        MEND
        START
```

```
                MOVEM AREG, B
                EVAL A, B, C
                ADD AREG, N
                MOVEM AREG, N
                CALC P, Q, LAB=LOOP:
                MOVEM AREG, N
                MAC1
                CALC P,Q,OP=DIV, LAB=NEXT
M               DS    1
A               DS   5
B               DS   1
C               DS   1
N               DS   1
P               DS   1
Q               DS   1
                END
```

Sample output: -

Macro name Table

Name	#pp	#kp	mdtp	kpdtp
MAC1	0	0	0	0
EVAL	3	0	4	0
CALC	2	2	9	1

MACRO DEFINITION TABLE

Index	Statements of the macros
0	MOVER AREG, M
1	ADD BREG, M
2	MOVEM CREG, M
3	MEND
4	MOVER AREG, (P,0)
5	SUB AREG, (P,1)
6	ADD AREG, (P,2)
7	MOVER AREG, (P,2)
8	MEND
9	(P,3) MOVER AREG, (P,0)
10	(P,2) AREG, (P,1)
11	MOVEM AREG, (P,0)
12	MEND

Expanded code with no macro definition & macro calls.

```
                START
                MOVEM AREG, B
               + MOVER AREG, A // expanded code of EVAL
                +SUB AREG, B
                +ADD AREG, C
                 +MOVER AREG, C
```

```
                ADD AREG, N
                MOVEM AREG, N
        + LOOP MOVER AREG, P // expanded code for CALC
                +MULT AREG, Q
                +MOVEM AREG, P
                MOVEM AREG,N
                +MOVER AREG, M // expanded code for MAC1
                +ADD        BREG, M
                 +MOVEM CREG, M
    + NEXT MOVER AREG, P // expanded code for CALC
                +DIV AREG, Q
                +MOVEM AREG, P
    A            DS    5
    B            DS    1
    C            DS    1
    N            DS    1
    P            DS    1
    Q            DS    1
                END
```

Conclusion: Ewe have understood the working of macro processor and how to generate expanded code.