System Programming : Macro Processor

Prof. Reshma Pise & Swati Shriyal

Comp Engg. Dept

Vishwakarma University

Course Outcomes :

On completion of the course, the students will be able to –

1. Discriminate among different System software and their functionalities.

2. Design language translators like Macro processor and Assembler.

3. Develop approaches and methods for implementing compiler, linker and loader.

4. Use LEX tool for lexical analysis.

5. Interpret the techniques of implementing utility software.

Outline

❑Macro definition and call

❑macro expansion

❑Nested Macro calls

❑Design of macro processor

❑Design issues of macro processors. (Data structures for design of MDT, MNT)

❑Design of two-pass macro processors

❑Advanced Macro Facilities.

Introduction

A macro instruction is a notational convenience for the programmer

It allows the programmer to write shorthand version of a program (*module programming*)

The macro processor replaces each macro invocation with the corresponding sequence of statements (*expanding*)

 "A macro is a unit of specification for program generation through expansion.

 Macro consist of name, a set of formal parameters and a body of code.

 "The use of macro name with a set of actual parameters is replaced by some code generated from its body,   this is called macro expansion."

Introduction

❑Two kind of expansion

1. Lexical expansion:
   ➢Lexical expansion implies replacement of character string by another character string during program generation.
   ➢Lexical expansion is typically employed to replace occurrences of formal parameter by corresponding actual parameters.

2. Semantic Expansion:
   ➢Semantic expansion implies generation of instructions tailored to the requirements of a specific usage
   ➢Example: generation of type specific instruction for manipulation of byte and word operands.

Example

- Macro

- INCR &MEM_VAL, &INCR_VAL, &REG   Prototype

- MOVER &REG, &MEM_VAL   **Model**

- ADD &REG, &INCR_VAL   **Model**

- MOVEM &REG, &MEM_VAL   **Model**

- MEND

Macro Processor`

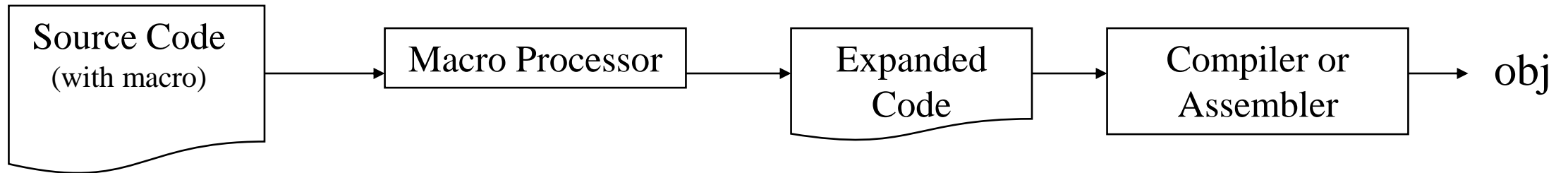Recognize macro definitions

Save the macro definition

Recognize macro calls

Expand macro calls

1) Macro Assembler
2) Macro Processor

Source Code (with macro) → Macro Processor → Expanded Code → Compiler or Assembler → obj

Macro Definition

copy code

parameter substitution

conditional macro expansion

macro instruction defining macros

Copy code -- Example

Source

```
STRG     MACRO
         STA      DATA1
         STB      DATA2
         STX      DATA3
         MEND
  .
STRG
  .
STRG
  .
  .
```

Expanded source

```
.
.
.
      STA      DATA1
      STB      DATA2
      STX      DATA3
.
      STA      DATA1
      STB      DATA2
      STX      DATA3
.
```

Macro vs. Subroutine

Macro
◦ the statement of expansion are generated each time the macro are invoked

Subroutine
◦ the statement in a subroutine appears only once

Macro definition and call

- Macro definition:
  - A macro definition is enclosed between a macro header statement and macro end statement.
  - Macro definition are typically located at the start of program .
  - Macro definition consist of
    - A macro prototype statement
    - One or more model statement
    - Macro preprocessor statement

☐ Example: the following sequence of instruction is used to increment the value in a memory word by a constant:

☐ Move the value from the memory word into a machine register.

☐ Increment the value in the machine register.

☐ Move the new value into the memory word.

☐ Using lexical expansion the macro call INCR A,B,AREG can lead to the generation of a MOVE-ADD-MOVE instruction sequence to increment A by the value B using AREG.

Macro definition and call

- A macro prototype statement

  - The macro prototype statement declares the name of a macro and the names and kinds of its parameters.

  - <macro name> [<formal parameter spec>, ...]

  - Where name appears in the mnemonic field of assembly statement and <formal parameter spec> is of the form &<parameter name>[<parameter kind>]

- Model statement

  - A model statement is a statement from which an assembly language statement may be generated during macro expansion.

- Macro preprocessor statement

  - A preprocessor statement is used to perform auxiliary functions during macro expansion.

Example

- Macro
- INCR &MEM_VAL, &INCR_VAL, &REG
- MOVER &REG, &MEM_VAL
- ADD &REG, &INCR_VAL
- MOVEM &REG, &MEM_VAL
- MEND

Macro call

- A macro is called by writing the macro name in the mnemonic field of an assembly statement.

- <macro name> [<actual parameter spec>,...]

- Where an actual parameter typically an operand specification in an assembly language statement.

Macro Expansion

- A macro call leads to macro expansion, during macro expansion, the macro call statement is replaced by a sequence of assembly statements.

- „+" is used to differentiate between the original statement of program and macro statement.

Macro Expansion

□ Two key notions concerning macro expansion are:

   □ Expansion time control flow:

   ■ This determines the order in which model statements are visited during macro expansion.

   □ Lexical substitution:

   ■ Lexical substitution is used to generate an assembly statement from a model statement.

Flow of control during expansion

☐ Flow of control during expansion

   ☐ The default flow of control during macro expansion is sequential. its start with statement following the macro prototype statement and ending with the statement preceding the MEND statement.

   ☐ A preprocessor statement can alter the flow of control during expansion such that some model statements are never visited during expansion is called **conditional expansion**.

   ☐ Same statement are repeatedly visited during expansion is called **loops expansion**.

Algorithm – Micro Expansion

- Macro expansion is implemented using a macro expansion counter (MEC).
- Algorithm: (Outline of macro expansion)
  - *MEC=statement number of first statement following the prototype statement;*
  - *While statement pointed by MEC is not a MEND statement*
    - *(a) if a model statement then*
      - *(i) Expand the statement*
      - *(ii) MEC=MEC+1;*
    - *(b) Else (i.e. a preprocessor statement)*
      - *(i) MEC= new value specified in the statement;*
  - *Exit from macro expansion.*

Lexical Substitution

☐ Lexical Substitution:

  ☐ Model statement consists of 3 type of strings

   ■ An ordinary string, which stands for itself.

   ■ The name of a formal parameter which is preceded by the character „&".

   ■ The name of preprocessor variable, which is also preceded by the character „&".

  ☐ During lexical expansion, string of type 1 are retained without substitution.

  ☐ String type 2 and 3 are replaced by the corresponding actual parameter values.

  ☐ The value of formal parameter depends on the kind of parameter.

Quiz

1. The translator which perform macro expansion is called a _____
2. A _____ statement declare the name of macro.
3. During macro expansion each statement is replaced by _____
4. Each macro statement is marked with the _____ sign preceded it.
5. The flow control during macro expansion is_____
6. A model statement contains call for another macro is called as_____
7. Expansion time variables are used _____
8. Macro processor is an inbuilt function of ? _____.
9. If a number of instructions are repeating through the main program, then to reduce the length of the program, _____ is used.
10. The process of assigning a label or macro name to the string is called _____

Types of Parameters

- Positional parameters

- Keyword parameters

- Default specification of parameter

- Macro with mixed parameter lists

- Other uses of parameters

Positional parameters

- Positional parameters
  - A positional formal parameter is written as &<parameter name>. The <actual parameter spec> in call on a macro using positional parameters is simply an <ordinary string>.
  - Step-1 find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.
  - Step-2 find the actual parameter specification occupying the same ordinal position in the list of actual parameters in macro call statement.

Positional parameters – Example

- INCR A, B, AREG
- The rule of positional association values of the formal parameters are:
- Formal parameter

MEM_VAL INCR_VAL
REG

value
A B
AREG

Lexical expansion of model statement now leads to the code

+ MOVER   AREG,A
+ ADD   AREG, B
+ MOVEM   AREG, A

Keyword parameters

- Keyword parameters

  - <parameter name> is an ordinary string and <parameter kind> is the string „=„ in syntax rule.

  - The <actual parameter spec> is written as <formal parameter name>=<ordinary string>.

  - The keyword association rules:

    - Step-1 find the actual parameter specification which has the form XYZ=<ordinary string>

    - Step-2   Let <ordinary string> in the specification be the string ABC. Then the value of formal parameter XYZ is ABC.

Keyword parameters

☐ Example :

```
INCR_M          MEM_VAL=A, INCR_VAL=B, REG=AREG
...
INCR_M          INCR_VAL=B, REG=AREG, MEM_VAL=A

MACRO
INCR_M          &MEM_VAL=, &INCR_VAL=, &REG=
MOVER           &REG, &MEM_VAL
ADD             &REG, &INCR_VAL
MOVEM           &REG, &MEM_VAL
MEND
```

Default specification of parameters

- Default specification of parameters
  - A default is a standard assumption in the absence of an explicit specification by programmer.
  - Default specification of parameters is useful in situations where a parameter has the same value in most calls.
  - When desired value is different from the default value, the desired value can be specified explicitly in a macro call.

Default specification of parameters

☐ Example:

Call the macro

    INCR_D   MEM_VAL=A, INCR_VAL=B

    INCR_D   INCR_VAL=B,   MEM_VAL=A

    INCR_D   INCR_VAL=B,   MEM_VAL=A,   REG=BREG

MARCO DIFINITION

MACRO

    INCR_D   &MEM_VAL=, &INCR_VAL=, &REG=AREG

    MOVER   &REG,   &MEM_VAL

    ADD   &REG, &INCR_VAL

    MOVEM   &REG， &MEM_VAL

    MEND

Macro with mixed parameter lists

☐ Macro with mixed parameter lists

  ☐ A macro may be defined to use both positional and keyword parameters.

  ☐ All positional parameters must precede all keyword parameters.

  ☐ Example: SUMUP A,B,G=20,H=X

  ☐ Where A,B are positional parameters while G,H are keyword parameters.

Other uses of parameters

- Other uses of parameters
  - The model statements have used formal parameters only in operand fields.
  - Formal parameter can also appear in the label and opcode fields of model statements.

Other uses of parameters-Example

```
        MACRO
        CALC                    &X, &Y, &OP= MULT, &LAB=

&LAB            MOVER           AREG, &X
                &OP             AREG, &Y
                MOVEM           AREG, &X
                MEND
```

Expansion of the call CALC A, B, LAB=LOOP leads to the following code:

```
+   LOOP        MOVER           AREG, A
+               MULT            AREG, B
+               MOVEM           AREG, A
```

Nested Macro Call

- A model statement in macro may constitute a call on another macro, such calls are known as nested macro calls.

- The macro containing the nested call is called outer macro.

- The called macro called inner macro.

- Expansion of nested macro calls follows the last-in- first-out(LIFO) rule.

Nested Macro Call – Example

```
MACRO
COMPUTE      &FIRST, &SECOND
MOVEM        BREG, TMP
INCR_D       &FIRST, &SECOND, REG=BREG
MOVER        BREG, TMP
MEND
```

$$
\text{COMPUTE} \quad X,Y
\begin{cases}
+ \text{MOVEM} \quad \text{BREG, TMP} \boxed{1} \\
+ \text{INCR\_D} \quad X,Y \quad
\begin{cases}
+ \text{MOVER} \quad \text{BREG, X} \boxed{2} \\
+ \text{ADD} \quad \text{BREG, Y} \boxed{3} \\
+ \text{MOVEM} \quad \text{BREG, X} \boxed{4}
\end{cases} \\
+ \text{MOVER} \quad \text{BREG, TMP} \boxed{5}
\end{cases}
$$

```
+            MOVEM        BREG, TMP
+            MOVER        BREG, X
+            ADD          BREG, Y
+            MOVEM        BREG, X
+            MOVER        BREG, TMP
```

Advanced Macro Facilities

☐ Advance macro facilities are aimed at supporting semantic expansion.

  ☐ Facilities for alteration of flow of control during expansion.

  ☐ Expansion time variables

  ☐ Attributes of parameters.

Alteration of flow of control during expansion

- Alteration of flow of control during expansion:
  - Expansion time sequencing symbols (SS).
  - Expansion time statements **AIF, AGO and ANOP.**
  - Sequencing symbol has syntax

    .<ordinary string>

  A SS is defined by putting it in the label field of statement in the macro body.

  It is used as operand in an AIF, AGO statement for expansion control transfer.

Advanced Macro Facilities

□ An AIF statement has syntax

AIF (<expression>) <sequencing symbol>

□ Where, <expression> is relational expression involving ordinary strings, formal parameters and their attributes, and expansion time variables.

□ If the relational expression evaluates to true, expansion time control is transferred to the statement containing <sequencing symbol> in its label field.

Advanced Macro Facilities

- An AGO statement the syntax

  AGO <sequencing symbol>

- Unconditionally transfer expansion time control to the statement containing <sequencing symbol> in its label field.

- An ANOP statement is written as

  <sequencing symbol> ANOP

- Simply has the effect of defining the sequencing symbol.

Quiz

1. What are x and y in the following macro definition? macro Add x, y Load y Mul x Store y end macro.
2. What is the value of X printed by the following program ? program COMPUTE (input, output); var X : integer ; procedure FIND (X: real) ; begin X : = sqrt (X) ; end ; begin X : = 2 FIND(X);
   writeln(X);
   end.
3. A macro definition consists of_____.
4. The process of assigning a label or macroname to the string is called_____.
5. A macro within a macro is called_____.
6. The beginning of the macro can be represented as_____.
7. Which of the following statements is incorrect?
   a) complete code of instruction string is inserted at each place, wherever the macroname appears
   b) macro requires less time of execution than that of procedure
   c) macro uses stack memory
   d) macroname can be anything except registers and mnemonics
8. Inserting the statements and instructions represented by macro, directly at the place of the macroname, in the program, is known as_____

Expansion Time Variable (EV's)

- Expansion Time Variable

  - Expansion time variable are variables which can only be used during the expansion of macro calls.

  - Local EV iss created for use only during a particular macro call.

  - Global E V exists across all macro calls situated in program and can be used in any macro which has a declaration for it.

LCL <EV specification> [,<EV specification>…]

GBL<EV specification> [,<EV specification>…]

Expansion Time Variable (EV's)

- <EV specification> has syntax &<EV name>, where EVname is ordinary string.

- Initialize EV by preprocessor statement SET.

  <EV Specification> SET < SET- expression>

EV's Example

```
           MACRO
           CONSTANTS
           LCL              &A
    &A     SET              1
           DB               &A
    &A     SET              &A+1
           DB               &A
           MEND
```

A call on macro CONSTANTS is expanded as follows: The local EV A is created. The first SET statement assigns the value '1' to it. The first DB statement thus declares a byte constant '1'. The second SET statement assigns the value '2' to A and the second DB statement declares a constant '2'.

Attributes of formal parameters

☐ **Attributes of formal parameters:**

<attribute name>"<formal parameter spec>

Represents information about the value of the formal parameter about corresponding actual parameter.

The type, length and size attributes have the name 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 in its label field only if the actual parameter corresponding to the formal parameter A has the length of '1'.

□ **Conditional expansion:**

□ Conditional expansion helps in generating assembly code specifically suited to the parameters in macro call.

□ A model statement is visited only under specific conditions during the expansion of a macro.

□ AIF and AGO statement used for this purpose.

Example:

☐ evaluate A - B + C in AREG.

```
        MACRO
         EVAL              &X,  &Y,  &Z
         AIF               (&Y  EQ  &X)        .ONLY
         MOVER             AREG ,  &X
         SUB               AREG ,  &Y
         ADD               AREG,   &Z
         AGO               .OVER
.ONLY    MOVER             AREG,    &Z
.OVER    MEND
```

- Expansion time loop
  - To generate many similar statements during the expansion of a macro.
  - This can be achieved by similar model statements in the macro.
  - Example:

```
MACRO
        CLEAR           &A
        MOVER           AREG,= "0"
        MOVEM           AREG, &A
        MOVEM           AREG, &A+1
        MOVEM           AREG, &A+2
        MEND
```

□ Expansion time loops can be written using expansion time variables and expansion time control transfer statement AIF and AGO.

Example:

```
                MACRO
                        CLEAR                   &X, &N
                        LCL                     &M
&M                      SET                     0
                        MOVER AREG,=„0"
.MOVE        MOVEM                 AREG, &X+&M
&M                      SET                     &M+1
                        AIF                     (&M NE N)        .MORE
                        MEND
```

Comparison with execution time loops:

- Most expansion time loops can be replaced by execution time loops.

- An execution time loop leads to more compact assembly programs.

- In execution time loop programs would execute slower than programs containing expansion time loops.

☐ Other facilities for expansion time loops:

☐ REPT statement
- Syntax: REPT**<expression>**
- <expression> should evaluate to a numerical value during macro expansion.
- The statements between REPT and an ENDM statement would be processed for expansion <expression> number of times.

☐ Example		
	MACRO	
	CONST10	
	LCL	&M
&M	SET	1
	R EPT	10
	DC	„&M"
&M	SET	&M+1
	ENDM	
	MEND	

- IRP statement

    IRP <formal parameter>, <argument-list>

- Formal parameter mentioned in the statement takes successive values from the argument list.

- The statements between the IRP and ENDM statements are expanded once.

☐ **Example:**

```
MACRO

CONSTS          &M, &N, &Z

IRP             &Z, &M, 7, &N

DC              '&Z'

ENDM

MEND
```

A MACRO call CONSTS 4, 10 leads to declaration of 3 constants with the values 4,7 and 10.

□ Semantic Expansion:

▢ Semantic expansion is the generation of instructions tailored to the requirements of a specific usage.

Example:

	MACRO	
	CREATE_CONST	&X, &Y
	AIF	(T'&X EQ B) .BYTE
&Y	DW	25
	AGO	.OVER
.BYTE	ANOP	
&Y	DB	25
.OVER	MEND	

Assignments

Explain macro with macro-processor expansion of macro?

2. Describe the features offered by Macro facility. Give example.

3. Define Macro. How it is different from subroutine?

4. Write short note on: Nested Macro call

5. Write short note on: Macro processor

6. What is lexical & semantic expansion? Explain with example, how macro & subroutine differ?

7. Discuss different kind of parameter in macros.

8. Discuss macro definition, call and expansion in detail with examples.

9. Define macro & macro expansion. What is lexical & semantic expansion?

10. Explain advanced macro facilities with example.

DESIGN OF A MACRO PREPROCESSOR

□ The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definition or call.

_____ _____	Macro preprocessor	Assembler	_____ _____

Program with macro definitions and calls

Program without macros

Target program
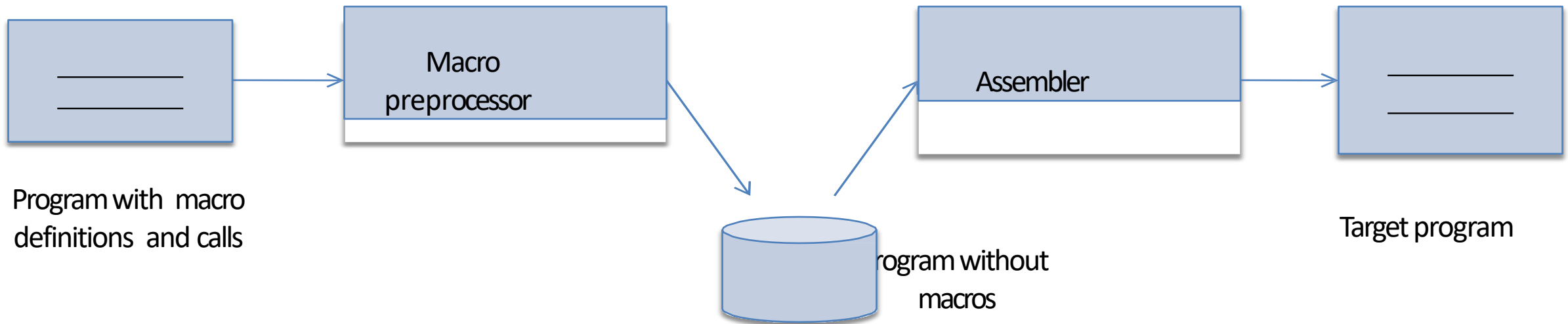
Design overview

- Listing all tasks involved in macro expansion
  - Identify macro calls in the program.
  - Determine the values of formal parameters.
  - Maintain the values of expansion time variables declared in a macro.
  - Organize expansion time control flow.
  - Determine the values of sequencing symbols.
  - Perform expansion of a model statement.

- The following 4 step procedure is followed to arrive at a design specification for each task:
  - Identify the information necessary to perform a task.
  - Design a suitable data structure to record the information.
  - Determine the processing necessary to obtain the information.
  - Determine the processing necessary to perform the task.

- Identify macro calls:
  - A table called the macro name table (MNT) is designed to hold the name of all macro defined in program.

- Determine values of formal parameters
  - A table called actual parameter table (APT) is designed to hold the values of formal parameters during the expansion of a macro call.
  - It contains (<formal parameter name>,<value>)
  - A table called parameter default table(PDT) is used for each macro.
  - Accessible from the MNT entry of macro.
  - It contain pairs of the form (<formal parameter name>,<default value>).
  - If macro call statement does not specify a value for some parameter then its default value would be copied from PDT to APT.

□ Maintain expansion time variables:

□ An expansion time variables table (EVT) is maintained for this purpose.

□ Table contain pairs of the form

□ (<EV name>,<value>)

□ It accessed when a preprocessor statement or model statement under expansion refers to an EV.

□ Organize expansion time control flow

□ The body of macro contained set of model statements and preprocessor statement in it, is stored in a table called the macro definition table (MDT) for use during macro expansion.

□ The flow of control during macro expansion determines when a model statement is to be visited for expansion.

□ Determine values of sequencing symbols:

    □ A sequencing symbol table (SST) is maintained to hold this information

    □ Table contains pairs of the form

    □ (<sequencing symbol name>,<MDT entry#>)

    □ Where <MDT entry#> is the number of the MDT entry which contains the model statement defining the sequencing symbol.

☐ Perform expansion of a model statement

☐ Task are as follow
- MEC points to the MDT entry containing the model statement.
- Values of formal parameters and EV's are available in APT and EVT, respectively
- The model statement defining a sequencing symbol can be identified from SST.

☐ Expansion of a model statement is achieved by performing a lexical substitution for the parameters and EV's used in the model statement.

Data structures

□ To obtain a detailed design of the data structure it is necessary to apply the practical criteria of processing efficiency and memory requirements.

□ The table APT,PDT and EVT contain pairs which are searched using the first component of the pairs as a key- the formal parameter name is used as the key to obtain its value from APT.

- This search can be eliminated if the position of an entity within a table is known when its value is accessed.
- The value of formal parameter ABC is needed while expanding a model statement using it

    MOVER   AREG, &ABC

- Let the pair (ABC,5) occupy entry #5 in APT. the search in APT can be avoided if the model statement appears as

    MOVER   AREG, (P,5)

- In the MDT, where (P,5) stand for the word „parameter #5".

- The first component of the pairs stored in APT is no longer used during macro expansion e.g. the information (P,5) appearing in model statement is sufficient to access the value of formal parameter ABC.

- APT containing (<formal parameter name>,<value>) pairs is replaced by another table called APTAB which only contains <value>'s.

- Ordinal number are assigned to all parameters of macro, a table named *parameter name table* (PNTAB) is used for this purpose.

- Parameter name are entered in PNTAB in same order in which they appear in the prototype statement.

☐ The information (<formal parameter name>,<value>) in APT has been split into two tables

PNTAB-which contains formal parameter names

APTAB-which contains formal parameter values PNTAB is used while

processing a macro definition while APTAB is used during macro expansion.

- The positional parameter of macro appear before keyword parameters in the prototype statement.

- If macro have p positional parameter and k keyword parameters, then keyword parameters have the ordinal number p+1, p+2 …P+k

- Due to this numbering redundancies appear in PDT.

- Entry only needs to exist for parameter number p+1, P+2 …P+k.

- So, replace parameter default table(PDT) by a keyword parameter default table (KPDTAB), this table have only k entries.

- MNT has entries for all macros defined in a program, each entry contains three pointers MDTP, KPDTP and SSTP which are pointers to MDT, KPDTAB and SSNTAB for the macro respectively.

- Similar analysis leads to splitting of EVT into EVNTAB and EVTAB and SST into SSNTAB and SSTAB.

- EV name are entered in EVNTAB while processing EV declarations.

- SS name are entered in SSNTAB while processing an SS reference or definition, whichever occur earlier.

- Macro preprocessor data structure can be summarized as follows:
    - PNTAB and KPDTAB are constructed by processing the prototype statement.
    - Entries are added to EVNTAB and SSNTAB as EV declarations and SS definitions/references are encountered.
    - MDT entries are constructed while processing model statements and preprocessor statements in macro body.
    - SSTAB entries, when the definition of sequencing symbol is encountered.
    - APTAB is constructed while processing a macro.
    - EVTAB is constructed at the start of expansion of macro.

Tables of the macro preprocessor

Table	Fields in each entry
Macro name Table(MNT)	Macro name, Number of positional parameter(#PP), Number of keyword parameter(#KP), Number of expansion time variables(#EV), MDT pointer (MDTP). KPDTABpointer (KPDTP). SSTABpointer (SSTP)
Parameter Name Table(PNTAB)	Parameter name
EVName Table (EVNTAB)	EV name
SSName Table (SSNTAB)	SSname
Keyword Parameter Default Table(KPDTAB)	Parameter name, default value
Macro Definition Table(MDT)	Label, Opcode, Operands
Actual Parameter Table(APTAB)	Value
EVTable (EVTAB)	Value
SSTable (SSTAB)	MDT entry #

```
              MACRO
              CLEARMEM           &X, &N, &REG=AREG
                LCL                 &M
&M              SET                 0
                MOVER               &REG,="0"
.MORE           MOVEM               &REG, &X + &M
&M              SET                 &M+1
                AIF                 (&M   NE  &N)  .MORE
                MEND
```

PNTAB

X
N
REG

EVNTAB

M

SSNTAB

MORE

		#PP	#KP	#EV	MDTP	KPDTP	SSTP
MNT	CLEARMEM	2	1	1	25	10	5

KPDTAB   10

REG	AREG

SSTAB   5

28

MDT

Macro Call : CLEARMEM AREA, 10, &REG=BREG

APTAB
AREA
10 BREG

25	LCL	(E,1)
26	(E,1)	SET	0
27		MOVER	(P,3)=,,0"
28		MOVEM (P,3),(P,1)+(E,1)
29	(E,1)	SET	(E,1)+1
30		AIF	((E,1) NE (P,2)) (S,1)
31		MEND

Processing of Macro definitions : Algorithm

- KPDTAB_pointer = 1
- SSTAB_ptr = 1;
- MDT_ptr = 1;
- Algorithm :(Processing of a macro definition)
  1. *SSNTAB_ptr=1; PNTAB_ptr=1;*
  2. *Process the macro prototype statement and form the MNT entry*
     - *(a) name= macro name*
     - *(b) for each positional parameter*
       - *(i) Enter parameter name in PNTAB[PNTAB_ptr]*
       - *(ii) PNTAB_ptr= PNTAB_ptr+1;*
       - *(iii) #PP=#PP+1;*

Processing of Macro definitions : Algorithm

(c)   KPDTP=KPDTAB_ptr;

(d)   for each keyword parameter

     (i)   Enter parameter name and default value (if any), in KPDTAB[KPDTAB_ptr].

(i)   Enter parameter name in PNTAB[PNTAB_ptr].

(ii)   KPDTAB_ptr=KPDTAB_ptr+1;

(iii)   PNTAB_ptr=PNTAB_ptr+1;

(iv) #KP=#KP+1;

(e)   MDTP=MDT_ptr;

(f)   #EV=0;

(g)   SSTP=SSTAB_ptr;

Processing of Macro definitions : Algorithm

3. While not a MEND statement

(a) if an LCL statement then

       (i) Enter expansion time variable name in EVNTAB.

       (ii) #EV=#EV+1;

  (b) if a model statement then

       I) if label field contains a sequencing symbol then if symbol is present in

       SSNTAB then

                  q= entry number in SSNTAB;

          else

                  Enter symbol in SSNTAB[SSNTAB_ptr]; q= SSNTAB_ptr;

                  SSNTAB_ptr=SSNTAB_ptr+1;

                  SSTAB[SSTP+q-1]=MDT_ptr;

      (ii) For a parameter, generate the specification (P,#n) : n is index of parameter in PNTAB

      (iii) For an expansion variable, generate the specification (E,#m);

Processing of Macro definitions : Algorithm

(iv) Record the IC in MDT[MDT_ptr];

(v) MDT_ptr=MDT_ptr+1;

(c) If Preprocessor statement then

(i) if a SET statement search each expansion time variable name used in the statement in EVNTAB and generate the spec(E,#m).

(ii) if an AIF or AGO statement then

if sequencing symbol used in the statement is present in SSNTAB then

q=entry number in SSNTAB;

else

enter symbol in SSNTAB[SSNTAB_ptr] q=SSNTAB_ptr;

SSNTAB_ptr=SSNTAB_ptr+1

replace the symbol by (S, SSTP+q-1)

Processing of Macro definitions : Algorithm

(iii) Record the IC in MDT[MDT_ptr]

(iV) MDT_ptr=MDT_ptr+1

4. (MEND statement)

if SSNTAB_ptr =1 (SSNTAB is empty) then SSTP=0

else

SSTAB_ptr=SSTAB_ptr+SSNTAB_ptr-1

if #KP=0 then KPDTP=0;

Macro expansion

- We use the following data structure to perform macro expansion:
  - APTAB – Actual parameter table
  - EVTAB – EV table
  - MEC – Macro expansion counter
  - APTAB_ptr – APTAB pointer
  - EVTAB_ptr – EVTAB pointer
- Number of entries in APTAB equals to the sum of values in the #PP and #KP fields of the MNT entry of macro.

☐ Algorithm 5.3 (Macro Expansion)

1. Perform initialization for the expansion of a macro

   a) MEC= MDTP field of MNT entry;

   b) Create EVTAB with #EV entries and set EVTAB_ptr.

   c) Create APTAB with #PP+#KP entries and set APTAB_ptr.

   d) Copy keyword parameter defaults from the entries KPDTAB[KPDTP] to KPDTAB[KPDTP+#KP-1] into APTAB[#PP+1] to APTAB[#PP+#KP].

   e) Process positional parameters in the actual parameter list and copy them into APTAB[1] to APTAB[#PP].

f) For keyword parameters in the actual parameter list search the keyword name in parameter name field of

KPDTAB[KPDTP] to KPDTAB[KPDTP+#KP-1]. Let KPDTAB[q]
contain a matching entry. Enter value of keyword parameter in the call (if any) in APTAB[#PP+q-KPDTP+1].

2) While statement pointed by MEC is not MEND statement
   a) if a model statement then
      (i) Replace operands of the form (P,#n) and
      (E,#m) by values in APTAB[n] and EVTAB[m]   respectively.
      (ii) Output the generated statement.
      (iii) MEC=MEC+1;

(b) If a SET statement with the specification (E,#m) in the label field then

   (i) Evaluate the expression in the operand field and set an appropriate value in EVTAB[m].

     (ii) MEC=MEC+1;

(c) If an AGO statement with (S,#s) in operand field MEC=SSTAB[SSTP+s-1]; then

(d) If an AIF statement with (S,#s) is operand field if condition in AIF then statement is true then

          MEC=SSTAB[SSTP+s-1];

(3) Exit from macro expansion.

Assignments

1. Explain the design of macroprocessor.

2. Write the algorithm for processing of macro definition and explain with the help of example.

3. Write the algorithm for macro expansion and explain with the help of example.

4. Explain the following facilities for expansion time loops: REPT and IRP

5. List all tasks involved in macro expansion.

6. Explain with the help of example data structure required for design of macro preprocessor.

7. Compare the two features - macro and subroutines in a programming language.

8. What do you understand by *conditional expansion* during macro processing ?

Nested Macro Call

- A model statement in macro may constitute a call on another macro, such calls are known as nested macro calls.

- The macro containing the nested call is called outer macro.

- The called macro called inner macro.

- Expansion of nested macro calls follows the last-in- first-out(LIFO) rule.

Nested Macro Call – Example

```
MACRO
COMPUTE       &FIRST, &SECOND
MOVEM         BREG, TMP
INCR_D        &FIRST, &SECOND, REG=BREG
MOVER         BREG, TMP
MEND
```


COMPUTE   X,Y
- + MOVEM   BREG, TMP [1]
- + INCR_D X,Y
  - + MOVER   BREG, X [2]
  - + ADD   BREG, Y [3]
  - + MOVEM   BREG, X [4]
- + MOVER   BREG, TMP [5]

```
+           MOVEM         BREG, TMP
+           MOVER         BREG, X
+           ADD           BREG, Y
+           MOVEM         BREG, X
+           MOVER         BREG, TMP
```
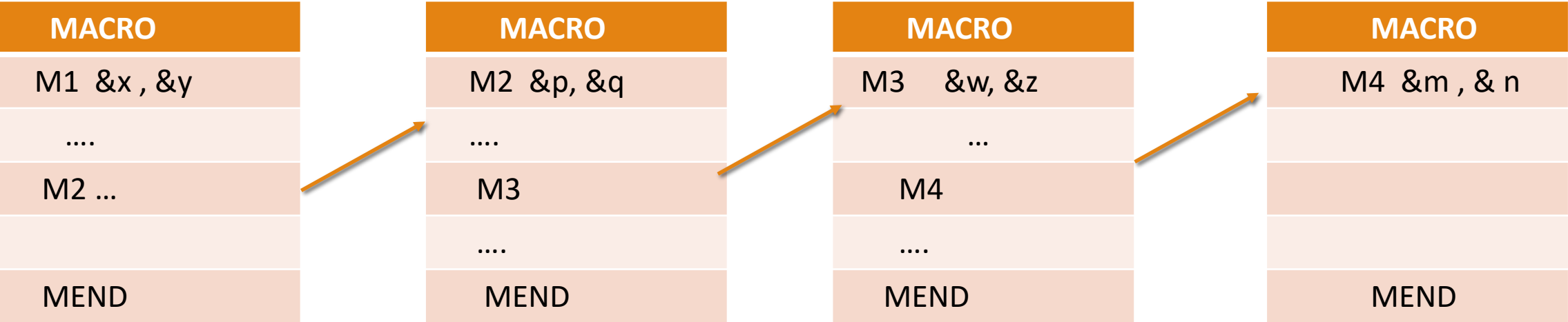
Nested macro calls

- Two basic alternatives exist for processing nested macro calls.
    - In this code macro calls appearing in the source program have been expanded but statements resulting from the expansion may themselves contain macro calls.
    - This first level expanded code to expand these macro calls, until we obtain a code form which dose not contain any macro calls.
    - This scheme would require a number of passes of macro expansion, which makes it quite expensive.

Nested macro calls

- Efficient alternative would be to examine each statement generated during macro expansion to see if it is itself macro call.

- A provision can be made to expand this call before continuing with the parent macro call.

- This avoid multiple passes of macro expansion.

- Two provisions are required to implement the expansion of nested macro calls:
  - Each macro under expansion must have its own set of data structures,
  - (MEC, APTAB, EVTAB, APTAB_ptr and EVTAB_ptr).
  - An expansion nesting counter (Nest_cntr) is maintained to count the number of nested macro calls. Nest_cntr is incremented when a macro call is recognized and decremented when MEND statement is encountered.

MACRO	MACRO	MACRO	MACRO
M1 &x , &y	M2 &p, &q	M3 &w, &z	M4 &m , & n
….	….	…	
M2 …	M3	M4	
	….	….	
MEND	MEND	MEND	MEND

(MEC , APTAB,   EVTAB, APTAB_ptr and EVTAB_ptr)

Nest_cntr =1	Nest_cntr = 2	Nest_cntr = 3	Nest_cntr = 4	

Nested macro calls

- Creating many copies of the expansion time data structure, this arrangement provides access efficiency but it is expensive in terms of memory requirements.
- Difficult in design decision- how many copies of the data structures should be created?
- If too many copies are created then some may never be used.
- If too few are created, some assembly programs may have to be rejected.

Nested macro calls

- Macro calls are expanded in LIFO manner, the stack consists of expansion records, each expansion record accommodating one set of expansion time data structures.

- Expansion record at the top of stack corresponds the macro call currently being expanded.

- When a nested macro call is recognized, a new expansion record is pushed on the stack to hold the data structures for the call.

- At MEND an expansion record is popped off the stack.

Nested macro calls

- Record base (RB) is a pointer pointing to the start of this expansion record.

- TOS point to the last occupied entry in stack.

- When nested macro call is detected, another set of data structure is allocated on the stack.

Nested macro calls (Expansion Record)
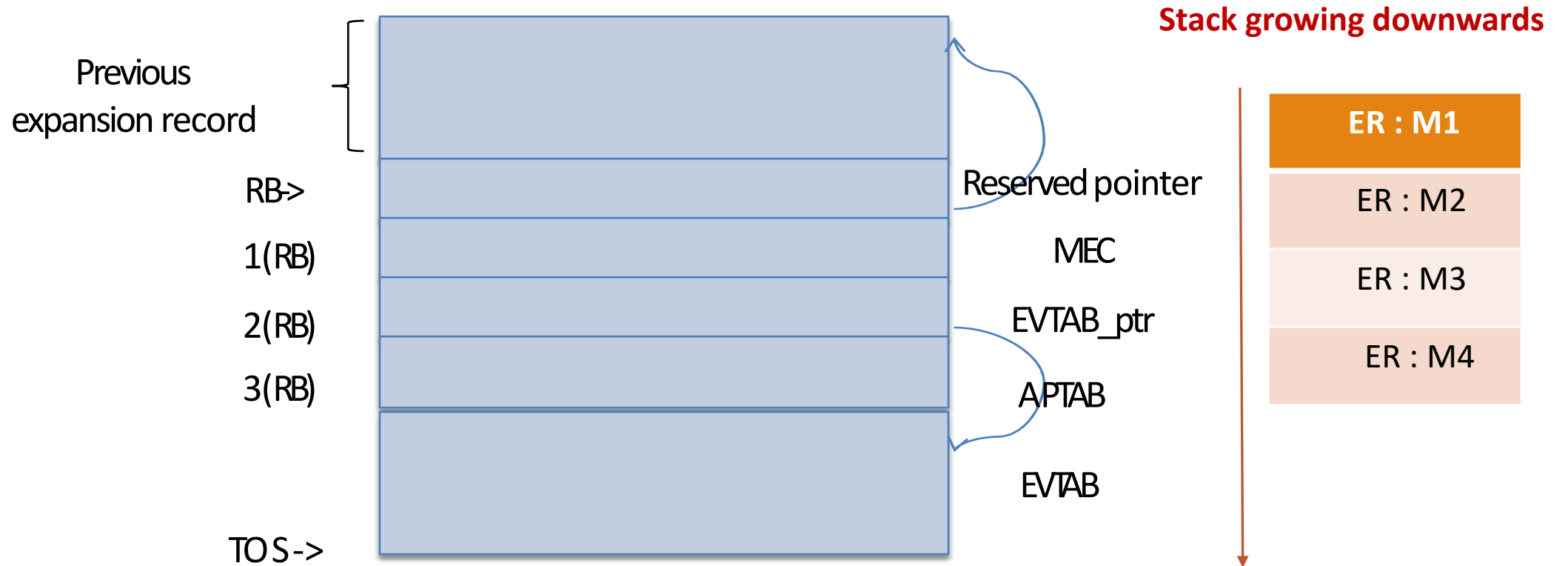
Data structure	Address
Reserved pointer	0(RB)
MEC	1(RB)
EVTAB_ptr	2(RB)
APTAB	3(RB) to e_{APTAB}+2(RB)
EVTAB	Contents of EVTAB_ptr

☐ The start of expansion

No.	Statement
1.	TOS=TOS+1;
2.	TOS* = RB;
3.	RB=TOS;
4.	1(RB)=MDTP entry of MNT;
5.	2(RB)= RB+$\#e_{APTAB}$;
6.	TOS=TOS+$\#e_{APTAB}$+$\#e_{EVTAB}$+2

- First statement increment TOS to point at the first word of the new expansion record. This is reserved pointer.

- Second statement deposits the address of the previous record base into this word.

- New RB is established in statement 3.

- MEC and EVTAB_ptr set in statement 4 and 5 respectively.

At the end of Expansion

No.	Statement
1.	TOS=RB-1;
2.	RB= RB*;

☐ The first statement pops an expansion record off the stack by resetting TOS to the value it had while the outer macro was being expanded.

☐ RB is then made to point at the base of previous record.

Design of macro assembler

- Macro preprocessor followed by conventional assembler is an expensive way of handling macro since the number of passes over the source program is large and many function get duplicated.

- Example:
  - A source statement to detect macro calls require us to process the mnemonic field. Similar function is required in first pass of the assembler. Similar functions of the preprocessor and assembler can be merged if macros are handled by a macro assembler which perform macro expansion and program assembly simultaneously.

- Macro expansion perform in single pass is not true, as certain kinds of forward references in macros cannot be handled in a single pass.

- This problem leads to the classical two pass organization for macro expansion.

  - First pass collects information about the symbols defined in a program.

  - second pass perform macro expansion.

- Passstructure of a macro-assembler
  - First merge the function of macro preprocessor with the function of conventional assembler, then the functions can be structured into passes of the macro assembler.
- Pass-I
  - Marco definition processing
  - SYMTAB construction
- Pass-II
  - Macro expansion
  - Memory allocation and LCprocessing
  - Processing of literals
  - Intermediate code generation.
- Pass-III
  - Target code generation.

□ The pass structure can be simplified if attributes of actual parameter are not to be supported.

□ Pass-I

   □ Macro definition processing

   □ Macro expansion

   □ Memory allocation, LC Processing and SYMTAB Construction

   □ Processing of Literals

   □ Intermediate code generation.

□ Pass-II

   □ Target code generation.

Examples Questions

□ Construct all data structure for the MACRO

```
        MACRO
        BECE6              &X, &Y, &REG=BREG
        AIF                (&Y EQ 0)    .EXIT
        MOVER              &REG, &X
        MUL                &REG, &Y
.EXIT   MEND
```

□ Generate the statement for these macro calls.

```
        BECE6              6, 8, REG=AREG
        BECE6              3,0
```

□ Generate the statement for these macro calls.

BECE6 6, 8, REG=AREG

BECE6 3,0

□ For first one BECE6 6, 8, REG=AREG

+ MOVER AREG, 6

+ MUL AREG, 8

Assignments

1. Give a small example to show the use of a *macro* in some hypothetical assembly language. Your example should contain *parameters* and *conditional expansion statements*.

2. What are positional parameters, keyword parameters and expansion time variables in macros ? Give a sample example to show their usage.

3. Write the following program using expansion time loops.

    MACRO

    CLEAR &A

    MOVER AREG, ='0'

    MOVEM AREG, &A

    MOVEM AREG, &A+1

    MOVEM AREG, &A+2

    MOVEM AREG, &A+3

    MEND

4. How are "expansion time variables" in macros different from normal program variables?