Define a macro. Explain macro definition and macro call with an example

Macro :

- A macro is a named block of assembly language statements. Macro allows a sequence of source language code to be defined.
- Once defined , it can be called one or more times. When the name is encountered in the code, it is replaced by the actual code it represents. Macros are typically used to define reusable pieces of code or to introduce abstractions for common tasks.

Macro Definition:

- A macro defination is enclosed between a macro header statement and a macro end statement.
- Macro definition involves creating a named sequence of code that can be used as a single statement. Macro definations are typically located at the start of the program.
- A macro defination consists of :
    1. A macro prototype statement
    2. One or more model statements
    3. Macro preprocessor statements
- The syntax for defining a macro is as follows :

MACRO

macro_name Formal Parameters

body

MEND

- The body of macro consists of the macro statements that either define operations or data. the body of a macro ends with the MEND keyword.
- MACRO – This keyword identifies the beginning of the macro definition.
- MEND – This keyword identifies the end of a macro definition.
- Parameters – It includes the parameters passed to the macro. Every parameter has to begin with ‘&’.
- Body – The body includes all the statements that the processor will substitute in response to the macro call in the program.

Macro Call:

- Macro call is the act of invoking or using the macro in your code. When a macro is called, the code associated with it gets substituted at the point where the macro is called.
- The syntax for calling a macro often looks like: **MACRO_NAME(argument_values)**

Macro Examples

#Macro DefinitionMACRO

MACNAME &R1, &R2, &R3
LOAD &R1, &R2
ADD &R1, &R3
STORE &R1, &R2

MEND

#Macro Invocation

MACNAME A, B, C

#Macro Expansion

MACNAME A, B, C
LOAD A, B
ADD A, C
STORE A, B

Formal Parameter	Actual Parameter
&R1	A
&R2	B
&R3	C

Advantages

- Using macro reduces the size of code.
- Programs with macros don't reduce their execution speed.
- Reduces programmer's effort in writing repetitive code.

Disadvantages

- When the macro processor replaces the abbreviation with the macro definition the length of the code increases prior to compilation.

What is Macro Expansion. Discuss two different ways of Macro Expansion.

- A macro is a named block of assembly language statements. Macro allows a sequence of source language code to be defined.
- Once defined , it can be called one or more times. When the name is encountered in the code, it is replaced by the actual code it represents

Macro Expansion :

- Macro expansion is the process of replacing a macro call with the actual code defined in the macro. When a macro is invoked, the macro expansion substitutes the parameters with their corresponding values and incorporates the macro body into the code.
- To expand a macro, the name of the macro is placed in the operation field, and no special directives are necessary.

Macro Expansion can be performed by using two kinds of language processors :

  - macro assembler
  - macro pre-processor

- **Macro Assembler** : It performs expansion of each macro call in a program into a sequence of assembly language statements and also assembles the resultant assembly language program.
- In this method, the expansion of macros is performed by the assembler itself during the assembly phase.
- The assembler recognizes and processes macro calls, replacing them with the corresponding macro body.

Process:

- The assembler encounters a macro call in the source code.
- It replaces the macro call with the expanded code from the macro definition.
- The resultant code, with macros expanded, is then assembled.

Advantages:

- Simplifies the overall assembly process.
- The programmer only sees the final expanded code.

Disadvantages:

- Limited flexibility as macro expansion is tied to the assembler.

- **Macro pre-processor:** It only processes the macro call . Other statements are processes with the help of assembler a macro pre-processor merely performs expansion of macro in program.

Description:

- In this method, a separate tool called a preprocessor is used to perform macro expansion before the assembly phase.
- The preprocessor scans the source code for macros, expands them, and produces an intermediate code that is then passed to the assembler.

Process:

- The preprocessor scans the source code for macro calls.
- It expands the macro calls, replacing them with the corresponding macro body.
- The preprocessed code, with macros expanded, is then passed to the assembler for further processing.

Advantages:

- Provides greater flexibility as it separates the macro expansion process from the assembler.
- Allows for more complex preprocessing operations.

Disadvantages:

- Adds an additional step in the compilation process.
- Requires coordination between the preprocessor and the assembler.

Example

Macro call

```
START  100
A      DS      1
B      DS      1
INCR   A, B, AREG
PRINT  A
STOP
END
```

Macro call

Macro definition

```
MACRO
INCR  &A,  &B, AREG
MOVER   REG   &A
ADD     REG   & B
MOVEM   REG   & A
MEND
```

Expanded code

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

- A and B are an actual parameters
- &A and &B are formal parameters

Explain with example Positional Parameters, Keyword Parameters and Default Specification Of Parameters.

- In programming, parameters are used in functions and procedures to pass values into the code. There are different ways to specify and use parameters
- Rules for determining the value of a formal parameter depends on the kind of the parameter following are the parameters :
  - **Positional Parameters**
  - **Keyword Parameters**
  - **Default Specification Of Parameters**

POSITIONAL PARAMETERS

- A positional formal parameter is written as &<parameter name>,
- e.g. &SAMPLE
- where SAMPLE is the name of parameter.
- <parameter kind> of syntax rule is omitted.
- The value of a positional formal parameter XYZ is determined by the rule of positional association as follows:
- Find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.
- Find the actual parameter specification occupying the same ordinal position in the list of actual parameters in the macro call statement.

EXAMPLE

- Consider the call:

  INCR A,B,AREG
- On macro INCR, following rule of positional association, values of formal parameters are:

Formal parameter	value
MEM_VAL	A
INCR_VAL	B
REG	AREG

- Lexical expansion of the model statements now leads to the code

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

KEYWORD PARAMETER

- For keyword parameter,
  - <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>.
- Note that the ordinal position of the specification XYZ=ABC in the list of actual parameters is immaterial.
- This is very useful in situations where long lists of parameters have to be used.
- Let us see example for it.

EXAMPLE:

- Following are macro call statement:

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

- Both are equivalent.
- Following is macro definition using keyword parameter:

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

DEFAULT SPECIFICATIONS OF PARAMETERS

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

&<parameter name> [<parameter kind> [<default value>]]

EXAMPLE

- The macro can be redefined to use a default specification for the parameter REG
- 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
- First two calls are equivalent but third call overrides the default value for REG with the value BREG in next example. Have a look.
- MACRO
- INCR_D &MEM_VAL=, &INCR_VAL=, &REG=AREG
- MOVER &REG, &MEM_VAL
- ADD &REG, &INCR_VAL
- MOVEM &REG, &MEM_VAL
- MEND

Explain Nested macro calls with example

- A macro is a named block of assembly language statements. Macro allows a sequence of source language code to be defined.
- Once defined , it can be called one or more times. When the name is encountered in the code, it is replaced by the actual code it represents
- Nested macro calls occur when a macro body contains macro calls, and these macro calls may further contain other macro calls, forming a nested structure.
- The nested nature of these instructions enhances code modularity and readability by encapsulating specific functionalities within the macro's scope.
- In the realm of nested macros, inner macro definitions act as "black boxes," maintaining a level of abstraction from the outermost macro definition.
- Variables or sequence symbols within these nested macros are not visible externally, fostering encapsulation and preventing unintended interference or parameterization.

Example :

```
INSIDE MACRO
SUBB A,R3
ENDM

OUTSIDE MACRO
MOV A,#42
INSIDE
MOV R7,A
ENDM
```

- In the body of the macro OUTSIDE, the macro INSIDE is called.

OUTSIDE Macro Invocation:

- The OUTSIDE macro is called in the main program.

Contents of OUTSIDE Macro:

- MOV A,#42: Moves the immediate value 42 into register A.
- INSIDE: Calls the INSIDE macro.

Contents of INSIDE Macro:

- SUBB A,R3: Performs a subtraction operation between registers A and R3.

MOV R7,A :

- After the INSIDE macro call, the OUTSIDE macro continues with the instruction to move the contents of register A into register R7.
- So, the expansion of the OUTSIDE macro results in a sequence of instructions that include the initialization of register A with 42, a subtraction operation, and finally, moving the result into register R7.
- The INSIDE macro is essentially a subroutine called within the OUTSIDE macro, adding modularity to the code.

Which are the advanced macro facilities for alteration of flow of control during expansion

- Advanced macro facilities are aimed to supporting semantic expansion.
- These facilities are used for performing conditional expansion of model statements and in writing expansion time loops.
- Following Two features are provided to facilitate alteration of flow of control during expansion.
  1. Expansion time sequencing symbol
  2. Expansion time statements

Expansion time sequencing symbol :

- Expansion time sequencing symbols (SS) are special markers used in macro processing to control the flow of expansion.
- They are employed in conjunction with AIF (Assembler IF) and AGO (Assembler GO TO) statements to direct the expansion process based on certain conditions.
- 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
- An AIF statement has syntax **AIF (<expression>) <sequencing symbol>** Where, 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 in its label field.
- An AGO statement the syntax **AGO <sequencing symbol>**
- Unconditionally transfer expansion time control to the statement containing in its label field.
- An ANOP statement is written as **<sequencing symbol> ANOP**
- Simply has the effect of defining the sequencing symbol.

Example

```
        MACRO
        TEST &X, &Y, &Z
        AIF   (&Y EQ &X)  .ONLY
        MOVER AREG, &Y
        AGO .LAST
.ONLY   MOVER   AREG, &Z
.LAST   MEND
```

REPRESENTING SS

If condition is true (Y= X) then control will be transferred to .ONLY

Explain Advanced Macro Facilities with example

- Advanced macro facilities are aimed to supporting semantic expansion.
- These facilities are used for performing conditional expansion of model statements and in writing expansion time loops.
- These facilities can be grouped into following.
  1. Facilities for alteration of flow of control during expansion.
  2. Expansion time variables.
  3. Attributes of parameters.

Facilities for alteration of flow of control during expansion : Explained in previous question

Expansion time variables :

- Expansion time variables (EVs) are exclusive variables used only during the expansion of macro calls in assembly language programming.
- Two types: Local EV (created for a specific macro call) and Global EV (exists across all macro calls in the program).

Declaration of EVs:

- Local and global EVs are declared using specific syntax:
  LCL <EV specification>[,<EV specification>…]
  GBL <EV specification>[,<EV specification>…]
- <EV specification> has syntax & <EV name>, where EV name is ordinary string.
- We initialize EV by preprocessor statement SET.

<EV Specification> SET <SET-expression>

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 :

- In macro programming, formal parameters can be associated with attributes that provide information about their corresponding actual parameters. The three common attributes are Type (T), Length (L), and Size (S).

Type (T) Attribute:

- Syntax: <T' formal parameter spec>
- Represents the data type of the corresponding actual parameter.
- Example: If a parameter is expected to be a numerical value, T' might indicate its numeric type.

Length (L) Attribute:

- Syntax: <L' formal parameter spec>
- Provides information about the length of the corresponding actual parameter.
- Example: If a parameter is expected to represent a string, L' might specify its character length.

Size (S) Attribute:

- Syntax: <S' formal parameter spec>
- Indicates the size or dimensionality of the corresponding actual parameter.
- Example: In an array parameter, S' might denote the size or number of elements.

EXAMPLE

- 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'.

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

Explain semantic expansion with example

- Semantic expansion implies generation of instructions tailored to the requirements of a specific usage.
- Semantic expansion is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.
- Eg: Generation of type specific instructions for manipulation of byte and word operands.
- The following sequence of instructions is used to increment the value in a memory word by a constant.
  1. Move the value from the memory word into a machineregister.
  2. Increment the value in the machine register.
  3. Move the new value into the memory word.
- Since the instruction sequence MOVE-ADD-MOVE may be used a number of times in a program, it is convenient to define a macro named INCR.
- Use of Semantic expansion can enable the instruction sequence to be adapted to the types of A and B.
- For example an INC instruction could be generated if A is a byte operand and B has the value 1.

```
; Macro Definition
MACRO INCR &MEM_VAL, &INCR_VAL, &REG
MOVER &REG, &MEM_VAL
ADD &REG, &INCR_VAL
MOVEM &REG, &MEM_VAL
MEND
```

Key Advantages of Semantic Expansion:

1. **Customization:** Semantic expansion allows for tailoring instructions to specific use cases, enhancing the flexibility and adaptability of macros.
2. **Efficiency:** By adapting the instruction sequence to the specific requirements of each macro call, semantic expansion can generate more efficient and optimized code.

Write an Algorithm for processing of Macro Definition.

- A macro defination is enclosed between a macro header statement and a macro end statement.
- Macro definition involves creating a named sequence of code that can be used as a single statement. Macro definations are typically located at the start of the program.

PROCESSING OF MACRO DEFINITIONS

- **The following initializations are performed before initiating the processing of macro definitions in a program**
- **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;
   c. KPDTP:=KPDTAB_ptr;
   d. For each keyword parameter
      i. Enter parameter name and default value (if any) in KPDTAB[KPDTAB_ptr].
      ii. Enter parameter name in PNTAB[PNTAB_ptr].
      iii. KPDTAB_ptr:=KPDTAB_ptr+1;
      iv. PNTAB_ptr:=PNTAB_ptr+1;
      v. #KP:=#KP+1;
   e. MDTP:=MDT_ptr;
   f. #EV:=0;
   g. SSTP:=SSTAB_ptr;

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)
      iii. For an expansion variable, generate the specification (E,#m).
      iv. Record the IC in MDT[MDT_ptr];
      v. MDT_ptr:=MDT_ptr + 1;

   c. If a 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).
      iii. Record the IC in MDT[MDT_ptr]
      iv. MDT_ptr:=MDT_ptr+1;

4. (MEND statement)
   If SSNTAB_ptr=1 (i.e. SSNTAB is empty)
   then
      SSTP:=0;
   Else
      SSTAB_ptr:=SSTAB_ptr+SSNTAB_ptr-1;
   If #KP=0 then KPDTP=0;

Explain Data structures of macro pre-processor

- The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definitions and calls.
- The macro preprocessor uses several data structures to manage and process macro definitions efficiently.

Macro Name Table (MNT):

- Purpose: To store information about each macro defined in the program.
  Content:
- Macro name
- MDT pointer (to locate the start of macro definition in MDT)
- KPDT pointer (to locate the keyword parameter default values in KPDT)
- SST pointer (to locate the start of the SST for the macro)

Macro Definition Table (MDT):

- Purpose: To store the actual instructions and statements of a macro.
  Content:
- Instructions and statements of the macro definition.

Keyword Parameter Default Table (KPDT):

- Purpose: To store default values for keyword parameters.
  Content:
- Keyword parameter name
- Default value (if any)

Positional Parameter Table (PNTAB):

- Purpose: To store the positional parameters of a macro.
  Content:
- Positional parameter names

Sequencing Symbol Name Table (SSNTAB):

- Purpose: To store sequencing symbols encountered during macro processing.
  Content:
- Sequencing symbols

Expansion Time Variable Name Table (EVNTAB):

- Purpose: To store expansion time variable names encountered during macro processing.
  Content:
- Expansion time variable names

Symbolic Stack Table (SSTAB):

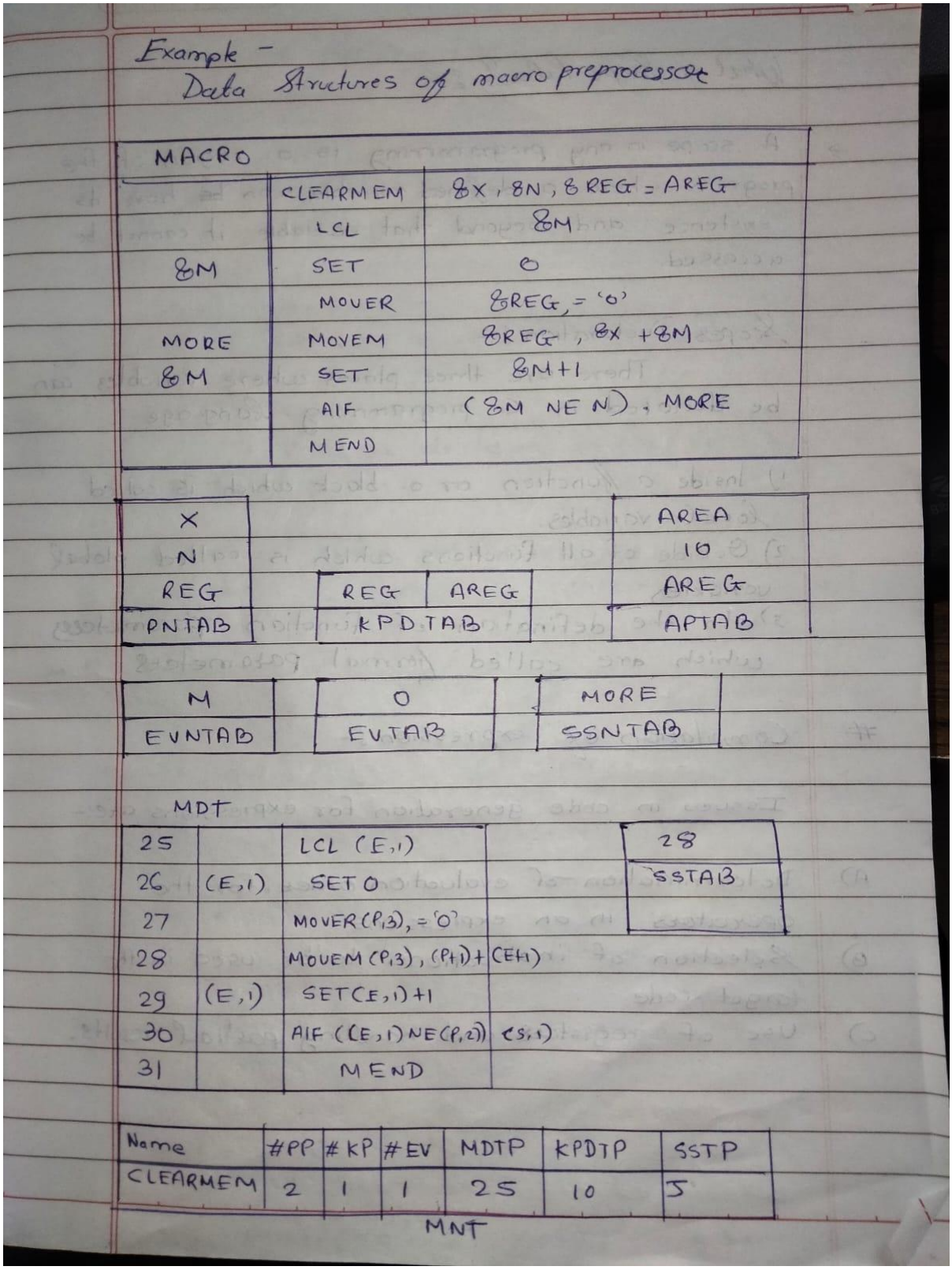
- Purpose: To manage symbolic stack entries.
  Content:
- MDT pointers corresponding to the positions of sequencing symbols in MDT

Local Expansion Time Variable Table (LEVNTAB):

- Purpose: To store local expansion time variables.
  Content:
- Local expansion time variable names

Table	Fields
MNT (Macro Name Table)	Macro Name
	Number of Positional Parameter (#PP)
	Number of keyword parameter (#KP)
	Number of Expansion Time Variable (#EV)
	MDT pointer (MDTP)
	KPDTAB pointer (KPDTABP)
	SSTAB pointer (SSTP)
PNTAB (Parameter Name Table)	Parameter name
EVNTAB (EV Name Table)	EV Name
SSNTAB (SS Name Table)	SS Name
KPDTAB (Keyword Parameter Default Table)	Parameter name, default value
MDT (Macro Definition Table)	Label, Opcode, Operands Value
APTAB (Actual Parameter Table)	Value
EVTAB (EV Table)	Value
SSTAB (SS Table)	MDT entry #

Example :



Example -
Data Structures of macro preprocessor

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	

X		AREA
N		10
REG	REG AREG	AREG
PNTAB	KPDTAB	APTAB

M	0	MORE
EVNTAB	EVTAB	SSNTAB

MDT

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	

28
SSTAB

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

MNT