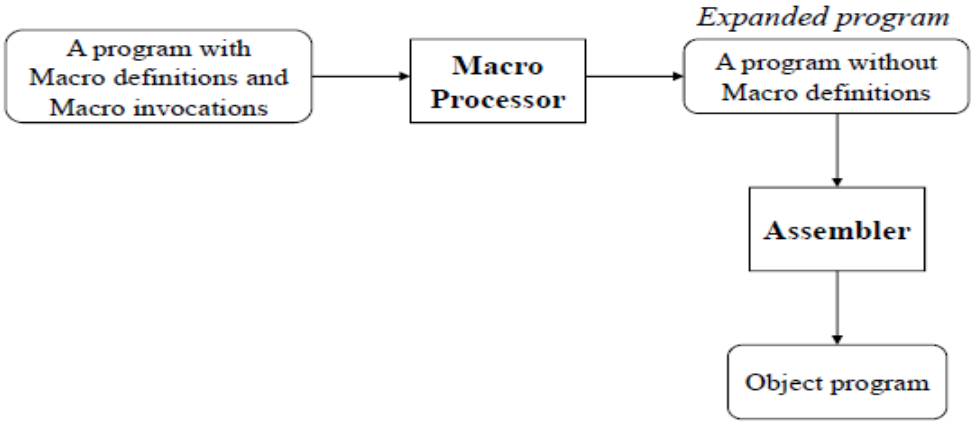
MODULE V

- **Macro Preprocessor**
  - **Macro Instruction Definition and Expansion.**
  - **One pass Macro processor Algorithm and data structures**
  - **Machine Independent Macro Processor Features**
    - Concatenation of Macro Parameters
    - Generation of unique labels
    - Conditional Macro Expansion
    - Keyword Macro Parameters
  - **Macro processor design options**
    - Recursive Macro Expansion
    - General-Purpose Macro Processors
    - Macro Processing within Language Translators

- **Macro Instruction (Macro)**
  - It is simply a notational convenience for the programmer to write a shorthand version of a program.
  - It represents a commonly used group of statements in the source program.

- **Macro Preprocessor**
  - Function: Substitution of one group of characters or lines for another.
  - It does not perform analysis of the text it handles.
  - It doesn't concern the meaning of the involved statements during macro expansion.
  - The design of macro pre-processor is machine independent.
  - Macro processors are used in
    - Assembly language
    - High-level programming languages, e.g., C or C++
    - OS command languages
    - General purpose



- **<u>BASIC MACROPROCESSOR FUNCTIONS</u>**
  - The fundamental functions common to all macro processors are:
    - Recognize Macro Definition
    - Recognize Macro Invocation/ Macro Calls
    - Expand Macro Calls

- **Macro Definition**
  - Assembler directives used in macro definition
    - **MACRO:** specifies the beginning of a macro definition
    - **MEND:** specifies the end of a macro definition
  - Syntax of Macro:

```
Macro_Name   MACRO      parameters
             -----
             -----      } Body
             -----
             MEND
```

    - **Macro Prototype statement**: The first line of the macro definition is called macro prototype statement.
    - The symbol in the label field of the prototype statement is the macro name.
    - Parameters are begins with '&'
    - Body: The statements that will be generated as the expansion of the macro
- **Macro Invocation (Macro Call)**
  - A macro invocation statement contains the name of the macro being invoked and the arguments to be used in expanding the macro.

```
Macro_Name  parameters
```

  - Difference between macro call and procedure call
    - Macro call: Statements of the macro body are expanded each time the macro is invoked.
    - Procedure call: statements of the subroutine appear only one, regardless of how many times the subroutine is called.
- **Macro Expansion**
  - Each macro invocation statement will be expanded into the statements that form the body of the macro.
  - Arguments from the macro invocation are substituted for the parameters in the macro prototype (according to their positions).
    - In the definition of macro: parameter
    - In the macro invocation: argument
  - Comment lines within the macro body will be deleted.
  - Macro invocation statement itself has been included as a comment line
  - The label on the macro invocation statement has been retained as a label on the first statement generated in the macro expansion
  - Example:

Source Code			Expanded Code			
PGM	START	0	PGM	START	0	
ABC	MACRO	&A,&B		. ABC	P,Q	//Comment line
	STA	&A		STA	P	
	STB	&B		STB	Q	
	MEND			.ABC	R,S	//Comment line
	ABC	P,Q		STA	R	
	ABC	R,S		STB	S	
	END			END		

```
5       COPY    START   0               COPY FILE FROM INPUT TO OUTPUT
10      RDBUFF  MACRO   &INDEV,&BUFADR,&RECLTH
15      .
20      .       MACRO TO READ RECORD INTO BUFFER
25      .
30              CLEAR   X               CLEAR LOOP COUNTER
35              CLEAR   A
40              CLEAR   S
45              +LDT    #4096           SET MAXIMUM RECORD LENGTH
50              TD      =X'&INDEV'      TEST INPUT DEVICE
55              JEQ     *-3             LOOP UNTIL READY
60              RD      =X'&INDEV'      READ CHARACTER INTO REG A
65              COMPR   A,S             TEST FOR END OF RECORD
70              JEQ     *+11            EXIT LOOP IF EOR
75              STCH    &BUFADR,X       STORE CHARACTER IN BUFFER
80              TIXR    T               LOOP UNLESS MAXIMUM LENGTH
85              JLT     *-19              HAS BEEN REACHED
90              STX     &RECLTH         SAVE RECORD LENGTH
95              MEND
100     WRBUFF  MACRO   &OUTDEV,&BUFADR,&RECLTH
105     .
110     .       MACRO TO WRITE RECORD FROM BUFFER
115     .
120             CLEAR   X               CLEAR LOOP COUNTER
125             LDT     &RECLTH
130             LDCH    &BUFADR,X       GET CHARACTER FROM BUFFER
135             TD      =X'&OUTDEV'     TEST OUTPUT DEVICE
140             JEQ     *-3             LOOP UNTIL READY
145             WD      =X'&OUTDEV'     WRITE CHARACTER
150             TIXR    T               LOOP UNTIL ALL CHARACTERS
155             JLT     *-14              HAVE BEEN WRITTEN
160             MEND
165
170     .               MAIN PROGRAM
175     .
180     FIRST   STL     RETADR          SAVE RETURN ADDRESS
190     CLOOP   RDBUFF  F1,BUFFER,LENGTH READ RECORD INTO BUFFER
195             LDA     LENGTH          TEST FOR END OF FILE
200             COMP    #0
205             JEQ     ENDFIL          EXIT IF EOF FOUND
210             WRBUFF  05,BUFFER,LENGTH WRITE OUTPUT RECORD
215             J       CLOOP           LOOP
220     ENDFIL  WRBUFF  05,EOF,THREE    INSERT EOF MARKER
225             J       @RETADR
230     EOF     BYTE    C'EOF'
235     THREE   WORD    3
240     RETADR  RESW    1
245     LENGTH  RESW    1               LENGTH OF RECORD
250     BUFFER  RESB    4096            4096-BYTE BUFFER AREA
255             END     FIRST
```

- After macro expansion the code will be as follows

```
  5     COPY      START   0                 COPY FILE FROM INPUT TO OUTPUT
180     FIRST     STL     RETADR            SAVE RETURN ADDRESS
190     .CLOOP    RDBUFF  F1,BUFFER,LENGTH READ RECORD INTO BUFFER
190a    CLOOP     CLEAR   X                 CLEAR LOOP COUNTER
190b              CLEAR   A
190c              CLEAR   S
190d             +LDT     #4096             SET MAXIMUM RECORD LENGTH
190e              TD      =X'F1'            TEST INPUT DEVICE
190f              JEQ     *-3               LOOP UNTIL READY
190g              RD      =X'F1'            READ CHARACTER INTO REG A
190h              COMPR   A,S               TEST FOR END OF RECORD
190i              JEQ     *+11              EXIT LOOP IF EOR
190j              STCH    BUFFER,X          STORE CHARACTER IN BUFFER
190k              TIXR    T                 LOOP UNLESS MAXIMUM LENGTH
190l              JLT     *-19                HAS BEEN REACHED
190m              STX     LENGTH            SAVE RECORD LENGTH
195               LDA     LENGTH            TEST FOR END OF FILE
200               COMP    #0
205               JEQ     ENDFIL            EXIT IF EOF FOUND

210         .     WRBUFF  05,BUFFER,LENGTH  WRITE OUTPUT RECORD
210a              CLEAR   X                 CLEAR LOOP COUNTER
210b              LDT     LENGTH
210c              LDCH    BUFFER,X          GET CHARACTER FROM BUFFER
210d              TD      =X'05'            TEST OUTPUT DEVICE
210e              JEQ     *-3               LOOP UNTIL READY
210f              WD      =X'05'            WRITE CHARACTER
210g              TIXR    T                 LOOP UNTIL ALL CHARACTERS
210h              JLT     *-14                HAVE BEEN WRITTEN
215               J       CLOOP             LOOP

220     .ENDFIL   WRBUFF  05,EOF,THREE        INSERT EOF MARKER
220a    ENDFIL    CLEAR   X                   CLEAR LOOP COUNTER
220b              LDT     THREE
220c              LDCH    EOF,X               GET CHARACTER FROM BUFFER
220d              TD      =X'05'              TEST OUTPUT DEVICE
220e              JEQ     *-3                 LOOP UNTIL READY
220f              WD      =X'05'              WRITE CHARACTER
220g              TIXR    T                   LOOP UNTIL ALL CHARACTERS
220h              JLT     *-14                  HAVE BEEN WRITTEN
225               J       @RETADR
230     EOF       BYTE    C'EOF'
235     THREE     WORD    3
240     RETADR    RESW    1
245     LENGTH    RESW    1                   LENGTH OF RECORD
250     BUFFER    RESB    4096                4096-BYTE BUFFER AREA
255               END     FIRST
```

- Problem of the label in the body of macro:
  - If the same macro is expanded multiple times at different places in the program. There will be *duplicate labels*, which will be treated as errors by the assembler.
  - Solutions:
    - Do not use labels in the body of macro.
    - Explicitly use PC-relative addressing instead.
      - Eg:
        - JEQ *+11 //jump to the location LOCCTR + 11
        - JLT *-14 //jump to the location LOCCTR – 14
    - It is inconvenient and error-prone.

- Example:
  - The following program shows an example of a SIC/XE program using macro Instructions.
  - This program defines two macros:
    - RDBUFF: Similar to RDREC subroutine
    - WRDUFF: Similar to WRREC subroutine
  - Line 10 and 100 are the beginning of first and second macro definition.
  - The instruction on line 55 is JEQ *-3
    - Means jump to 3 location back.
    - This type of PC relative addressing mode is used to avoid labels in the macro body.
  - The MAIN program contains 3 macro calls on line 190, 210 and 220.
  - Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from macro invocation substituted for the parameters in macro prototype.
  - The arguments and parameters are associated with one another according to their positions.
  - The macro definition has been deleted since they have been no longer needed after macros are expanded

- **TYPES OF MACROS**
  - Simple Macro
  - Parameterized Macro
  - Nested Macro
  - Recursive Macro

- **Simple Macro**
  - A macro without argument is called simple macro.

Source Code			Expanded Code	
ABC	MACRO		. ABC	
	STA	A	STA	A
	STB	B	STB	B
	MEND		. ABC	
	ABC		STA	A
	ABC		STB	B

- **Parameterized Macro**
  - A macro with argument is called parameterized macro.
  - Two types of Parameterized Macros
    - Positional Parameters
    - Keyword Parameters
  - **Positional Parameters**
    - The programmer must specify the arguments in proper order.
    - Parameters and arguments are associated according to their position in the macro prototype and invocation.
    - When the macro is called, the parameters will be replaced within the macro body by the value specified.
    - If an argument is to be omitted, a null argument should be used to maintain the order in the macro invocation statement.

Source Code			Expanded Code	
ABC	MACRO	&A,&B	. ABC	P,Q
	STA	&A	STA	P
	STB	&B	STB	Q
	MEND		. ABC	Q,P
	ABC	P,Q	STA	Q
	ABC	Q,P	STB	P

  - **Keyword Parameters**
    - Arguments may appear in any order.
    - Each argument value is written with a keyword that names the corresponding parameter. Each parameter name is followed by =
    - Null arguments no longer need to be used.

Source Code			Expanded Code	
ABC	MACRO	&A=,&B=	. ABC	A=P,B=Q
	STA	&A	STA	P
	STB	&B	STB	Q
	MEND		. ABC	B=Q,A=P
	ABC	A=P,B=Q	STA	P
	ABC	B=Q,A=P	STB	Q

- **Nested Macro**
  - A macro body may contain another macro definition
  - Example: Here the macro SWAP defines another macro STORE inside it.

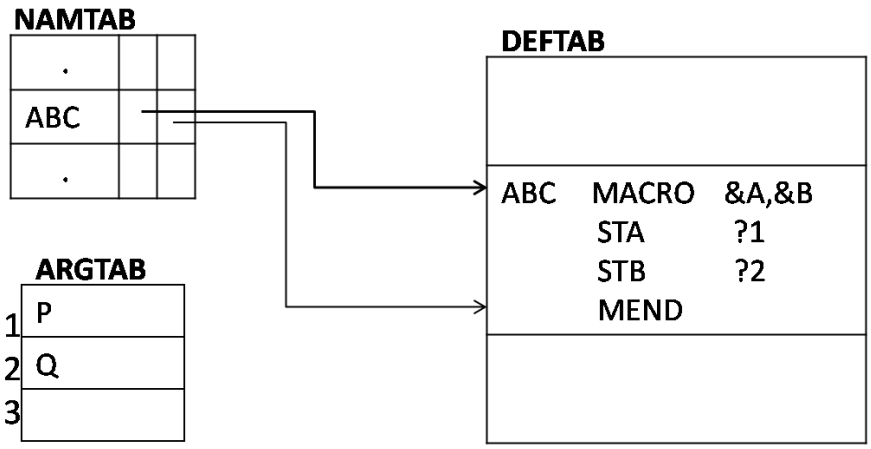
```
SWAP    MACRO    &X,&Y          //Outer Macro Definition
        LDA      &X
        LDX      &Y
STORE   MACRO    &X,&Y          //Inner Macro Definition
        STA      &Y
        STX      &X
        MEND
        MEND
```

  - The expansion of nested macro calls follows the last-in-first-out rule(LIFO).
  - The expansion of latest macro call is completed first.

- **Recursive Macros**
  - A macro definition contains another macro call. This call may be the same macro or a different macro.

```
ABC   MACRO     &A,&B
      --------
      --------
       PQR      X,Y
      --------
      --------
      MEND
PQR   MACRO     &P,&Q
      --------
      --------
      MEND
```

- **<u>MACRO PROCESSOR ALGORITHM AND DATA STRUCTURES</u>**
  - Macro Processors can be implemented in two ways
    - Two Pass Macro Preprocessor
      - Pass 1: All macro definitions are processed
      - Pass 2: All macro invocation statements are expanded
      - Disadvantage: Nested macros definitions are not allowed.
    - Single Pass Macro Preprocessor
      - Nested macro definitions are allowed but nested calls are not allowed.
      - The definition of a macro must appear in the source program before any statements that invoke that macro.
      - **<u>Data Structures for One Pass Macro Preprocessor</u>**
        - Three Data Structures
          - **Definition table (DEFTAB)**
            - The macro definition is stored in definition table (DEFTAB), which contains
              - Macro prototype statement
              - Macro body statements
            - Comment lines from macro definition are not entered into DEFTAB.
          - **Name table (NAMTAB)**
            - Stores macro names
            - For each macro definition, NAMTAB contains pointers to beginning and end of definition in DEFTAB.
          - **Argument table (ARGTAB)**
            - When macro invocation statements are recognized, the arguments are stored in ARGTAB according to their position in argument list.
            - As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.



- The position notation is used for the parameters.
  - &A has been converted to ?1
  - &B has been converted to ?2, and so on.
- When the ?n notation is recognized in a line from DEFTAB, a simple indexing operation supplies the property argument from ARGTAB.

- **Algorithm for One Pass Macro Preprocessor**

```
ONE_PASS_MACRO()
{       EXPANDING= FALSE
        while OPCODE !='END'
        {
                GETLINE()
                PROCESSLINE()
        }
}
 PROCESSLINE()
{       Search NAMETAB for OPCODE
        If found then                           EXPAND()
        Else if OPCODE='MACRO' then             DEFINE()
        Else                                    Write source line to expanded file
}
DEFINE()
{       Enter macro name into NAMTAB
        Enter macro prototype into DEFTAB
        LEVEL = 1
        While LEVEL > 0
        {       GETLINE()
                If this is not a comment line
                {       Substitute positional notation for parameters
                        Enter line into DEFTAB
                        If OPCODE='MACRO'       LEVEL = LEVEL+1
                        Else If OPCODE='MEND'   LEVEL = LEVEL-1
                }
```

```
        }
        Store in NAMETAB pointers to beginning and end of definition
}
EXPAND()
{       EXPANDING = TRUE
        Get prototype statement from DEFTAB
        Set up arguments from macro invocation in ARGTAB
        Write macro invocation to expanded file as comment
        While not end of macro definition
        {       GETLINE()
                PROCESSLINE()
        }
        EXPANDING = FALSE
}
GETLINE()
{       If EXPANDING
        {
                Get next line of macro definition from DEFTAB
                Substitute arguments from ARGTAB for positional notation
        }
        Else    Read next line from input file
}
```

- The procedure DEFINE, which is called when the beginning of a macro definition is recognized, makes the appropriate entries in DEFTAB and NAMTAB.
- When a macro definition is being entered into DEFTAB, the normal approach is to continue until an MEND directive is reached.
- This would not work for nested macro definition because the first MEND encountered in the inner macro will terminate the whole macro definition process.
- To solve this problem, a counter LEVEL is used to keep track of the level of macro definitions.
  - Increase LEVEL by 1 each time a MACRO directive is read.
  - Decrease LEVEL by 1 each time a MEND directive is read.
- A MEND terminates the whole macro definition process when LEVEL reaches 0.
- This process is very much like matching left and right parentheses when scanning an arithmetic expression.

- EXPAND is called to set up the argument values in ARGTAB and expand a macro invocation statement.

- The procedure GETLINE gets the next line to be processed
  - This line may come from DEFTAB or from the input file, depending upon whether the Boolean variable EXPANDING is set to TRUE or FALSE.

- **MACHINE INDEPENDENT MACRO PROCESSOR FEATURES**
  - Following are the features that are not directly related to the architecture of computer for which the macro processor is written
    - Concatenation of Macro Parameters
    - Generation of unique labels
    - Conditional Macro Expansion
    - Keyword Macro Parameters
  - **Concatenation of Macro Parameters**
    - Parameters to be concatenated with other character strings.
    - Suppose a program contains a set of series of variables:
      - XA1, XA2, XA3,…
      - XB1, XB2, XB3,… etc.
    - If similar processing is to be performed on each series of variables, the programmer might want to incorporate this processing into a macro instruction.
    - The parameter to such a macro instruction could specify the series of variables to be operated on (A, B, C …).
    - The macro processor constructs the symbols by concatenating X, (A, B, …), and (1,2,3,…) in the macro expansion.
    - Such parameters are begins with & and ends with →. (→ is a concatenation operator. It will not appear in the macro expansion).
    - Example:

```
1   SUM MACRO   &ID
2       LDA     X&ID→ 1
3       ADD     X&ID→ 2
4       ADD     X&ID→ 3
5       STA     X&ID→ S
6       MEND
```

```
SUM     A                    SUM     BETA
 ↓                            ↓
LDA     XA1                  LDA     XBEATA1
ADD     XA2                  ADD     XBEATA2
ADD     XA3                  ADD     XBEATA3
STA     XAS                  STA     XBEATAS
```

  - **Generation of unique labels**
    - Labels in the macro body may cause "duplicate labels" problem if the macro is invocated and expanded multiple times.
    - Use of relative addressing at the source statement level is very inconvenient, error-prone, and difficult to read.
    - It is highly desirable to
      - Let the programmer use label in the macro body
      - Let the macro processor generate unique labels for each macro expansion.
    - Labels used within the macro body should begin with $.
    - During macro expansion, the $ will be replaced with $xx, where xx is a two-character alphanumeric counter of the number of macro instructions expanded.
      - For the first macro expansion in a program, xx will have the value AA.
      - For succeeding macro expansions, xx will be set to AB, AC etc.
      - This allows 1296 macro expansions in a single program

- Example:

Source Code			Expanded Code			
PGM	START	0	PGM	START	0	
ABC	MACRO	&X		……..		
	……..		$AAL1	……..		
$L1	……..			JEQ	$AAL1	1st macro expansion
	JEQ	$L1		……..		
	……..			……..		
	MEND		$ABL1	……..		
	ABC	A		JEQ	$ABL1	2nd macro expansion
	ABC	B		……..		
	END			END		

- **Conditional Macro Expansion**
  - Normally same macro calls will generate same set of statements.
  - Conditional Macro Expansion (Conditional Assembly): Sequence of statements generated for macro expansion is depends on the arguments supplied in the macro invocation.
  - Conditional assembly depends on parameters provides
  - **Macro-time variables**
    - Any symbol that begins with symbol & and not a macro instruction parameter inside a macro definition is considered as macro-time variable.
    - Used to store working values during the macro expansion
      - Usually store the evaluation result of Boolean expression
      - Control the macro-time conditional structures
    - It is initialized to 0
    - SET macro processor directive is used to assign a particular value to a macro time variable.
      - Ex: &EORCK SET 1
        &EORCTR SET &EORCTR + 1

Here EORCK is a Macro-time variable and this statement will not be in the expanded code.

  - **Macro-time conditional structure**
    - IF-ELSE-ENDIF
    - WHILE-ENDW
  - **IF-ELSE-ENDIF structure**
    - The macro processor must maintain a symbol table
    - This table contains the values of all macro-time variables used
    - Entries in this table are made or modified when SET statements are processed.
    - This table is used to look up the current value of a macro-time variable whenever it is required.
    - When an IF statement is encountered during the expansion of a macro, the specified Boolean expression is evaluated. If value is
      - TRUE
        - The macro processor continues to process lines from DEFTAB until it encounters the next ELSE or ENDIF statement.
        - If ELSE is encountered, then skips to ENDIF

- FALSE
  - The macro processor skips ahead in DEFTAB until it finds the next ELSE or ENDIF statement.
  - Ex:

```
MACRO       &COND
…….
IF (&COND  NE  ‘’)
        PART I
ELSE
        PART II
END IF
…….
ENDM
```

- Part I is expanded if condition part is true, otherwise part II is expanded
- Compare operator: NE, EQ, LE, GT

- IF, ELSE and ENDIF statements will not be in the expanded code.

- **WHILE-ENDW structure(Macro-time looping statement)**
  - When a WHILE statement is encountered during the expansion of a macro, the specified Boolean expression is evaluated. If the value is
    - TRUE
      - The macro processor continues to process lines from DEFTAB until it encounters the next ENDW statement.
      - When ENDW is encountered, the macro processor returns to the preceding WHILE, re-evaluates the Boolean expression, and takes action based on the new value.
    - FALSE
      - The macro processor skips ahead in DEFTAB until it finds the next ENDW statement and then resumes normal macro expansion.

```
WHILE  (  cond  )
……
ENDW
```

  - WHILE and ENDW statements will not be in the expanded code.

- Example:

Macro Definition:

```
25        RDBUFF   MACRO    &INDEV, &BUFADR, &RECLTH, &EOR. &MAXLTH
26                 IF       (&EOR NE ' ')
27        &EORCK   SET      1
28                 ENDIF
30                 CLEAR    X                    CLEAR LOOP COUNTER
35                 CLEAR    A
38                 IF       (&EORCK EQ 1)
40                 LDCH     =X'&EOR'             SET EOR COUNTER
42                 RMO      A, S
43                 ENDIF
44                 IF       (&MAXLTH EQ ' ')
45                +LDT      #4096                SET MAX LENGTH = 4096
46                 ELSE
47                +LDT      #&MAXLTH             SET MAXIMUM RECORD LENGTH
48                 ENDIF
50        $LOOP    TD       =X'&INDEV'           TEST INPUT DEVICE
55                 JEQ       $LOOP               LOOP UNTIL READY
60                 RD       =X'&INDEV'           READ CHARACTER INTI REG A
63                 IF       (&EORCK EQ 1)
65                 COMPR    A, S                 TEST FOR END OF RECORD
70                 JEQ      $EXIT                EXIT LOOP IF EOR
73                 ENDIF
75                 STCH     &BUFADR, X           STORE CHARACTER IN BUFFER
80                 TIXR     T                    LOOP UNLESS MAXIMUN LENGTH
85                 JLT      $LOOP                HAS BEEN REACHED
90        $EXIT    STX      &RECLTH              SAVE RECORD LENGTH
95                 MEND
```

Macro-time variable

Macro Call:

```
          RDBUFF   F31 BUF, RECL, 04, 2048
```

Expanded Code:

```
30                 CLEAR    X                    CLEAR LOOP COUNTER
35                 CLEAR    A
40                 LDCH     =X'04'               SET EOR CHARACTER
42                 RMO      A, S
47                +LDT      #2048                SET MAXIMUM RECORD LENGTH
50        $AALOOP  TD       =X'F3'               TEST INPUT DEVICE
55                 JEQ       $AALOOP             LOOP UNTIL READY
60                 RD       =X'F3'               READ CHARACTER INTI REG A
65                 COMPR    A, S                 TEST FOR END OF RECORD
70                 JEQ       $AAEXIT             EXIT LOOP IF EOR
75                 STCH     BUF, X               STORE CHARACTE IN BUFFER
80                 TIXR     T                    LOOP UNLESS MAXIMUM LENGTH
85                 JLT      $AALOOP              HAS BEEN REACHED
90        $AAEXIT  STX      RECL                 SAVE RECORD LENGTH
```

Macro Call:

```
RDBUFF    OE, BUFFER, LENGTH, , 80
```

Expanded Code:

```
30          CLEAR    X          CLEAR LOOP COUNTER
35          CLEAR    A
47         +LDT      #80        SET MAXIMUM RECORD LENGTH
50   $ABLOOP TD       =X'0E'     TEST INPUT DEVICE
55          JEQ      $ABLOOP    LOOP UNTIL READY
60          RD       =X'0E'     READ CHARACTER IN REG A
75          STCH     BUFFER, X  STORE CHARACTER IN BUFFER
80          TIXR     T          LOOP UNLESS MAXIMUM LENGTH
87          JLT      $ABLOOP    HAS BEEN REACHED
90   $ABEXIT STX     LENGTH     SAVE RECORD LENGTH
```

Macro Call:

```
RDBUFF    F1. BUFF, ELENG, 04
```

Expanded Code:

```
30          CLEAR    X          CLEAR LOOP COUNTER
35          CLEAR    A
40          LDCH     =X'04'     SET EOR CHARACTER
42          RMO      A, S
45         +LDT      #4096      SET MAX LENGTH = 4096
50   $ACLOOP TD       =X'F1'     TEST INPUT DEVICE
55          JEQ      $ACLOOP    LOOP UNTIL READY
60          RD       =X'F1'     READ CHARACTER INTI REG A
65          COMPR    A,S        TEST FOR END OF RECORD
70          JEQ      $ACEXIT    EXIT LOOP IF EOR
75          STCH     BUFF,X     STORE CHARACTER IN BUFFER
80          TIXR     T          LOOP UNLESS MAXIMUM LENGTH
85          JLT      $ACLOOP    HAS LOOP REACHED
90   $ACEXIT STX     RLENG      SAVE RECORD LENGTH
```

- Example:

Macro Definition:

```
25   RDBUFF  MACRO   &INDEV, &BUFADR, &RECLTH, &EOR
27   &EORCT  SET     %NITEMS (&EOR)   <-- Macro processor function
30           CLEAR   X          CLEAR LOOP COUNTER
35           CLEAR   A
45          +LDT     #4096              SET MAX LENGTH = 4096
50   $LOOP   TD      =X'&INDEV'         TEST INPUT DEVICE
55           JEQ     $LOOP              LOOP UNTIL READY
60           RD      =X'&INDEV'         READ CHARACTER INTO REG A
63   &CTR    SET     1
64           WHILE   (&CTR LE &EORCT)
65           COMPR   =X'0000&EOR[&CTR]'   <-- List index
70           JEQ     $EXIT
71   &CTR    SET     &CTR+1
73           ENDW
75           STCH    &BUFADR, X         STORE CHARACTER IN BUFFER
80           TIXR    T                  LOOP UNLESS MAXIMUM LENGTH
85           JLT     $LOOP              HAS BEEN REACHED
90   $EXIT   STX     &RECLTH            SAVE RECORTD LENGTH
100          MEND
```

%NITEMS is a macro processor function that returns the number of members in the argument list.

Macro Call:

```
RDBUFF    F2, BUFFER, LENGTH, (00, 03, 04)
                              ↖ List
```

Here &EOR is (00, 03, 04). Then %NITEMS(&EOR) is 3.

On the first iteration the expression &EOR[&CTR] on line 65 has the value 00. On the second iteration it has the value 03, and so on.

Expanded Code:

```
30                CLEAR    X              CLEAR LOOP COUNTER
35                CLEAR    A
45               +LDT      #4096          SET MAX LENGTH = 4096
50      $AALOOP   TD       =X'F2'         TEST INPUT DEVICE
55                JEQ      $AALOOP        LOOP UNTIL READY
60                RD       =X'F2'         READ CHARACTER INTO REG A
65                COMP     =X'000000'
70                JEQ      $AAEXIT
65                COMP     =X'000003'
70                JEQ      $AAEXIT
65                COMP     =X'000004'
70                JEQ      $AAEXIT
75                STCH     BUFFER, X      STORE CHARACTER IN BUFFER
80                TIXR     T              LOOP UNLESS MAXIMUM LENGTH
85                JLT      $AALOOP        HAS BEEN REACHED
90      $AAEXIT   STX      LENGTH         SAVE RECORD LENGTH
```

- **Keyword Macro Parameters**
  - **Positional parameters**
    - Parameters and arguments are associated according to their positions in the macro prototype and invocation.
    - The programmer must specify the arguments in proper order.
    - If an argument is to be omitted, a null argument should be used to maintain the proper order in macro invocation statement.
    - For example: Suppose a macro instruction **ABC** has 10 possible parameters, but in a particular invocation of the macro only the 3rd and 9th parameters are to be specified. The macro call statement is

      ABC ,,DIRECT,,,,,,3
    - Disadvantage: It is not suitable if a macro has a large number of parameters, and only a few of these are given values in a typical invocation.
    - Solution: Use Keyword parameters instead of Positional parameters.
  - **Keyword parameters**
    - Each argument value is written with a keyword that names the corresponding parameter.
    - Arguments may appear in any order.
    - Null arguments no longer need to be used.

- Each parameter name is followed by =
- After the =, a default value can be specified for some of the parameters. The parameters are assumed to have this default value if its name does not appear in the macro invocation statement.
- For example: Suppose a macro instruction **ABC** has 10 possible parameters, but in a particular invocation of the macro only the 3rd and 9th parameters are to be specified. If the 3rd parameter is named &TYPE and 9th parameter is named &CHANNEL. The macro call statement will be

  ABC TYPE=DIRECT,CHANNEL=3

  or

  ABC CHANNEL=3, TYPE=DIRECT
- Advantage:
  - Easier to read
  - Less error-prone than the positional method

▪ Example:

Macro Definition:

The following macro definition contains 5 parameters. Three of them (&INDEV,&EOR,&,MAXLTH) having default value.

```
25        RDBUFF   MACRO   &INDEV=F1, &BUFADR=, &RECLTH=, &EOR=04, &MAXLTH=4096
26                 IF      (&EOR NE ' ')
27        &EORCK   SET     1
28                 ENDIF
30                 CLEAR   X                CLEAR LOOP COUNTER
35                 CLEAR   A
38                 IF      (&EORCK EQ 1)
40                 LDCH    =X'&EOR'         SET EOR CHARACTER
42                 RMO     A, S
43                 ENDIF
47                +LDT     #MAXLTH          SET MAXIMUM RECORD LENGTH
50        $LOOP    TD      =X'&INDEV'       TEST INPUT DEVICE
55                 JEQ     $LOOP            LOOP UNTIL READY
60                 RD      =X'&INDEV'       READ CHARACTER INTI REG A
63                 IF      (&EORCK EQ 1)
65                 COMPR   A, S             TEST FOR END OF RECORD
70                 JEQ     $EXIT            EXIT LOOP IF EOR
73                 ENDIF
75                 STCH    $BUFADR, X       STORE CHARACTER IN BUFFER
80                 TIXR    T                LOOP UNLESS MAXIMUM LENGTH
85                 JLT     $LOOP            HAS BEEN REACHED
90        $EXIT    STX     &RECLTH          SAVE RECORD LENGTH
95                 MEND
```

Parameters with default value

Macro Call:

```
RDBUFF   BUFADR=BUFFER, RECLTH-LENGTH
```

Expanded Code

```
30          CLEAR   X           CLEAR LOOP COUNTER
35          CLEAR   A
40          LDCH    =X'04'      SET EOR CHARACTER
42          RMO     A, S
47         +LDT     #4096       SET MAXIMUM RECORD LENGTH
50  $AALOOP TD      =X'F1'      TEST INPUT DEVICE
55          JEQ     $AALOOP     LOOP UNTIL READY
60          RD      =X'F1'      READ CHARACTER INTI REG A
65          COMPR   A, S        TEST FOR END OF RECORD
70          JEQ     $AAEXIT     EXIT LOOP IF EOR
75          STCH    BUFFER, X   STORE CHARACTER IN BUFFER
80          TIXR    T           LOOP UNLESS MAXIMUM LENGTH
85          JLT     $AALOOP     HAS BEEN REACHED
90  $AAEXUT STX     LENGTH      SAVE RECORD LENGTH
```

- **MACROPROCESSOR DESIGN OPTIONS**
  - Recursive Macro Expansion
  - General-Purpose Macro Processors
  - Macro Processing within Language Translators

  - **Recursive Macro Expansion**
    - Invoke a macro from another macro definition
    - Example:

```
10      RDBUFF  MACRO   &BUFADR, &RECLTH, &INDEV
15      .
20      .       MACRO TO READ RECORD INTO BUFFER
25      .
30              CLEAR   X           CLEAR LOOP COUNTER
35              CLEAR   A
40              CLEAR   S
45              +LDT    #4096       SET MAXIMUN RECORD LENGTH
50      $LOOP   RDCHAR  &INDEV      READ CHARACTER INTO REG A
65              COMPR   A, S        TEST FOR END OF RECORD
70              JEQ     &EXIT       EXIT LOOP IF EOR
75              STCH    &BUFADR, X  STORE CHARACTER IN BUFFER
80              TIXR    T           LOOP UNLESS MAXIMUN LENGTH
85              JLT     $LOOP       HAS BEEN REACHED
90      $EXIT   STX     &RECLTH     SAVE RECORD LENGTH
95              MEND
5   RDCHAR  MACRO   &IN
10  .
15  .   MACROTO READ CHARACTER INTO REGISTER A
20  .
25              TD      =X'&IN'         TEST INPUT DEVICE
30              JEQ     *-3             LOOP UNTIL READY
35              RD      =X'&IN'         READ CHARACTER
40              MEND
```

- RDBUFF and RDCHAR are the 2 macro definitions.
- RDCHAR is used to read a character from an input device to register A.
- Macro Call: **RDBUFF BUFFER, LENGTH, F1**

- One pass macro processor cannot handle such kind of recursive macro invocation and expansion
- Reasons:
  - The procedure EXPAND would be called recursively, thus the invocation arguments in the ARGTAB will be overwritten.
    - The procedure EXPAND would be called when the macro was recognized. The arguments from the macro invocation would be entered into ARGTAB as follows.

Parameter	Value
1	BUFFER
2	LENGTH
3	F1
4	(unused)
-	-

    - The Boolean variable EXPANDING would be set to TRUE, and expansion of the macro invocation statement would begin. The processing would proceed normally until statement invoking RDCHAR is processed. This time, ARGTAB would look like

Parameter	Value
1	F1
2	(Unused)
--	--

  - The Boolean variable EXPANDING would be set to FALSE when the "inner" macro expansion is finished, that is, the macro process would forget that it had been in the middle of expanding an "outer" macro.
    - At the expansion, when the end of RDCHAR is recognized, EXPANDING would be set to FALSE. Thus the macro processor would forget that it had been in the middle of expanding a macro when it encountered the RDCHAR statement. In addition, the arguments from the original macro invocation (RDBUFF) would be lost because the value in ARGTAB was overwritten with the arguments from the invocation of RDCHAR
  - A similar problem would occur with PROCESSLINE since this procedure too would be called recursively.
- Solutions:
  - Write the macro processor in a programming language that allows recursive calls, thus local variables will be retained.
  - Use a Stack to save ARGTAB.
  - Use a counter to identify the expansion

- **Single Pass Macro Processor Algorithm to handle recursive calls**

```
 ONE_PASS_MACRO()
{       EXPANDING= FALSE
        SP = -1
        N=0
        while OPCODE !='END'
        {       GETLINE()
                PROCESSLINE()
        }
}
 PROCESSLINE()
{       Search NAMETAB for OPCODE
        If found then                           EXPAND()
        Else if OPCODE='MACRO' then             DEFINE()
        Else                                    Write source line to expanded file
}
DEFINE()
{       Enter macro name into NAMTAB
        Enter macro prototype into DEFTAB
        LEVEL = 1
        While LEVEL > 0
        {       GETLINE()
                If this is not a comment line
                {       Substitute positional notation for parameters
                        Enter line into DEFTAB
                        If OPCODE='MACRO'       LEVEL = LEVEL+1
                        Else If OPCODE='MEND'   LEVEL = LEVEL-1
                }
        }
        Store in NAMETAB pointers to beginning and end of definition
}
Procedure EXPAND
{       set S ( SP + N +  2) = SP
        set SP = SP + N + 2
        set S ( SP + 1 ) =DEFTAB index from NAMTAB
        setup macro call argument list array in S( SP + 2)…….S(SP + N + 1) where N =
        total number of arguments
        while not end of macro definition and Level !=0 do
        {
                GETLINE
                PROCESSLINE
        }
        N = SP – S( SP) – 2             //reset previous calls number of arguments
        SP = S( SP )                    // previous calls starting index of S
}
```

```
procedure GETLINE
{
        if SP != -1 then
        {
                increment DEFTAB pointer to next entry
                set S ( SP + 1) = S (SP + 1) + 1
                get the line from DEFTAB with the pointer S( SP+1)
                substitute arguments from macro call S (SP + 2)…….. S(SP + N + 1)
        }
        else
                read next line from input file
}
```

- **G**
- **General-Purpose Macro Processors**
  - Macro processors that do not dependent on any particular programming language, but can be used with a variety of different languages.
  - Example: ELENA macro processor
  - **Advantages**
    - Programmers do not need to learn many macro languages.
    - Although its development costs are somewhat greater than those for language specific macro processor, this expense does not need to be repeated for each language, thus save substantial overall cost.
  - **Disadvantages**
    - Large number of details must be dealt with in a real programming language
    - In a typical programming language, there are several situations in which normal macro parameter substitution should not occur
    - Each programming language has its own methods for identifying comments
      - Some languages use special characters to mark the start and end of a comment.
      - Some languages use a special character to mark only the start of a comment. The comment is automatically terminated at the end of the source line.
      - Some languages use a special symbol to flag an entire line as a comment.
      - In most assembly languages, an characters on a line following the end of the instruction operand field are automatically taken as comments
    - Each programming languages having their own facilities for grouping together terms, expressions, or statements
      - Some languages use keywords such as begin and end for grouping statements.
      - Others use special characters such as { and } for grouping statements.
    - A more general problem involves the tokens of the programming language like identifiers, constants, operators, and keywords
      - Languages differ their restrictions on the length of identifiers and the rules for the formation of constants.
      - Some languages support multiple character operators. Eg: **

- Another potential problem with general purpose macro processors involves the syntax used for macro definitions and macro invocation statements. With most special purpose macro processors, macro invocations are very similar in form to statements in the source programming language.

- **Macro Processing within Language Translators**
  - The macro processors we discussed are called "Preprocessors".
    - Process macro definitions
    - Expand macro invocations
    - Produce an expanded version of the source program, which is then used as input to an assembler or compiler
  - Alternative design: Combine the macro processing functions with the language translator
    - Line-by-line macro processor
    - Integrated macro processor
  - **Line-by-Line Macro Processor**
    - Used as a sort of input routine for the assembler or compiler
      - Read source program
      - Process macro definitions and expand macro invocations. The expanded code is not written to an expanded source file.
      - Pass output lines to the assembler or compiler
    - Benefits
      - Avoid making an extra pass over the source program. So it is more efficient than using a macro processor.
      - Data structures required by the macro processor and the language translator can be combined (e.g., OPTAB and NAMTAB)
      - Utility subroutines can be used by both macro processor and the language translator.
        - Scanning input lines
        - Searching tables
        - Data format conversion
      - It is easier to give diagnostic messages related to the source statements.
  - **Integrated Macro Processor**
    - An integrated macro processor can potentially make use of any information about the source program that is extracted by the language translator.
    - Many real programming languages have certain characteristics that create unpleasant difficulties.
      - Ex : Consider the following FORTRAN statement
        - DO 100 I = 1,20
          - It is a normal DO statement
          - 100 is the line number
        - DO 100 I = 1
          - An assignment statement
          - DO100I is variable (blanks are not significant in FORTRAN)

    - The proper interpretation of the characters DO, 100 etc, cannot be decided until the rest of the statement is examined. Such interpretations would be very important for a macro expansion with macro name I.
  - An integrated macro processor can support macro instructions that depend upon the context in which they occur.
    - The expansion of macro could also depend up on a variety of characteristics of its arguments.
- **Disadvantages of Line-by-line and Integrated Macro processor**:
  - More expensive: The cost of macro processor development must be added to the cost of language translator, which result in a more expensive piece of software.
  - More complex: The assembler will be more complex.
  - Size is larger: The size may be problem if the translator is to run on a computer with limited memory.
  - Take more time: The assembler will take more time to assemble the code.

Previous Year University Questions

1. Explain the concept of macro definition and expansion with the help of examples.
2. Differentiate between a macro and a subroutine. Illustrate macro definition and expansion using an example
3. A code segment need to be repeatedly used in various parts of assembly language program and fast execution is also needed. Would you use a macro or a subroutine? Justify your answer with help of examples
4. Describe the data structures used in a one pass macro processor algorithm. Give examples
5. What are the data structures required for a macroprocessor algorithm? Explain the format of each
6. Give the algorithm for a one pass macro processor
7. Explain the working of One pass Macro Processor
8. Write the algorithm for one pass macro processor and explain the process, showing when and how the different data structures are used
9. Certain macro processor features are independent of the machine architecture Give the details of such machine independent macro-processor features
10. Write short note on concatenation of macro parameters within a character string
11. How are unique labels generated in a Macro Expansion
12. Is it possible to include labels in the body of macro definition? Justify your answer.
13. Is it possible to use labels within the macro body? Explain your answer with the help of examples. Also illustrate a possible solution for the same
14. Explain conditional macro expansion with an example
15. Explain the different types of conditional macro expansion statements and their implementation with examples
16. Write notes on keyword macro parameters, giving suitable examples
17. Differentiate between keyword and positional macro parameters
18. List and explain the different design options available for macroprocessors
19. Explain recursive macro expansion with example
20. Write notes on Recursive Macro Expansion
21. What do you mean by recursive macro expansion? What are the possible problems associated with it?

22. What are the important factors considered while designing general purpose macro processors?
23. What is meant by line-by-line macro processor? What are its advantages?