

Chapter 4 Macro Processors.

- A macro instruction is simply a notational convenience for programmer.
- Macro represents commonly used group of statements in the source program
- the macro processor replace each macro instruction with corresponding group of source statements.
 - this operation is called 'expanding the macro'.
- using macros allows programmer to write a shorthand version of a program.
- eg:- before calling a subroutine, the contents of all registers may need to be stored this routine work can be done using a macro.

Macroprocessors.

- its function essentially involves the substitution of one group of lines or another.
- it doesn't analyze the text it handles.
- meaning of the statements are of no concern during macro expansion.
Hence, design of macro processor generally is machine independent.
- Macro mostly are used on assembly language programming, also be used in high level programming language such as C, C++.

Macro definition

- Two directives Macro and MEND are used in macro definition.
- Macro's name appears before the Macro directive. Macro's parameters appear after the MACRO directive.

- each parameter begins with '&'
- between MEND and MEND is the body of macro, here these are the statements that will be generated as the expansion of the macro definition.

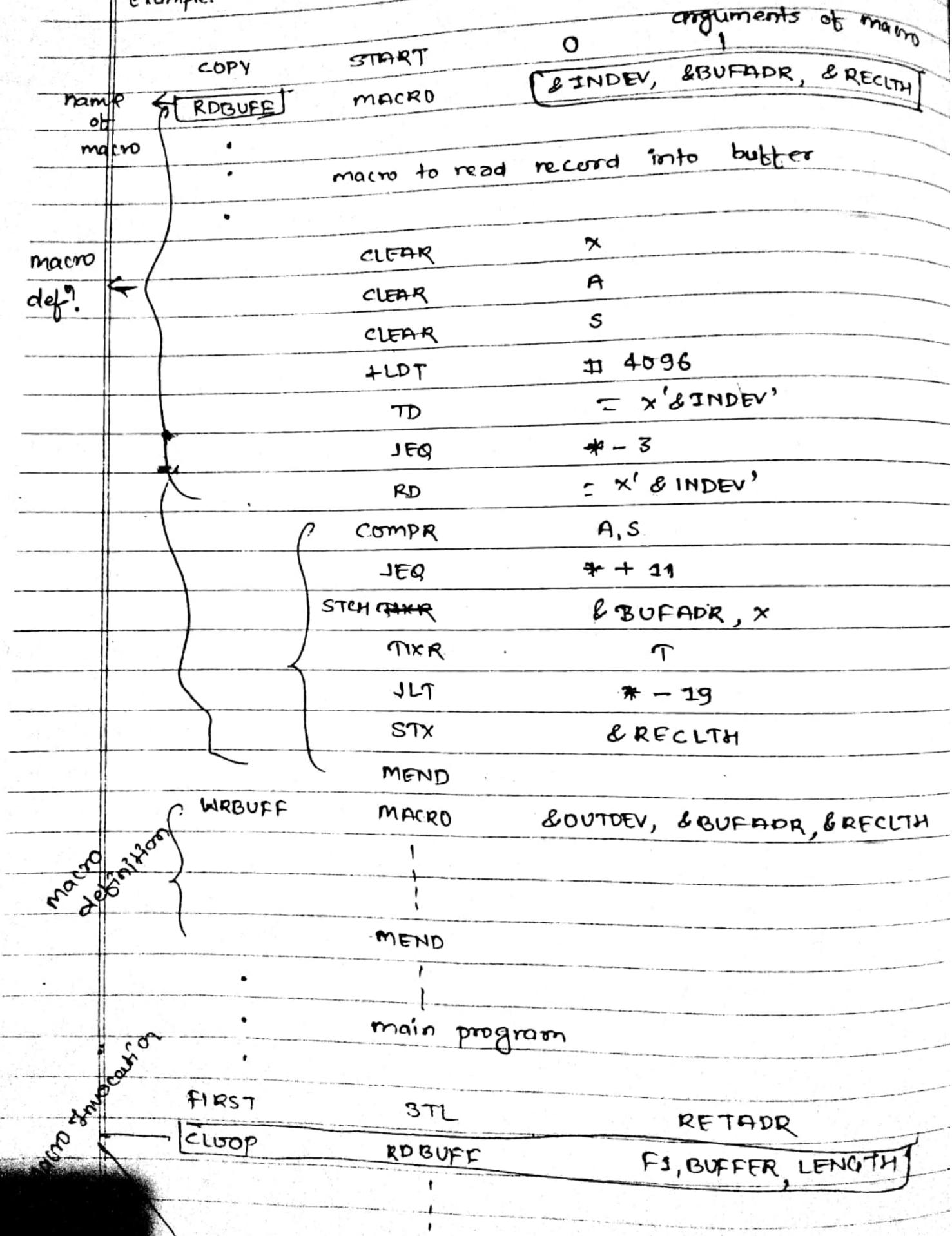
Macro Invocation

- Macro invocation statement (macro call) gives the name of the macro instruction being invoked and the arguments in expanding the macro.

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 macro prototype.
- the arguments and parameters are associated with one another according to their positions
 - the first argument in the macro invocation corresponds to the first parameter in macro prototype
- comment lines within the macro body have been deleted but comments on individual statements have been retained.
- Macro invocation, statement itself has been included as a comment line.

Example:-



Date
Page

WRBUFF

05, BUFFER, LENGTH

|

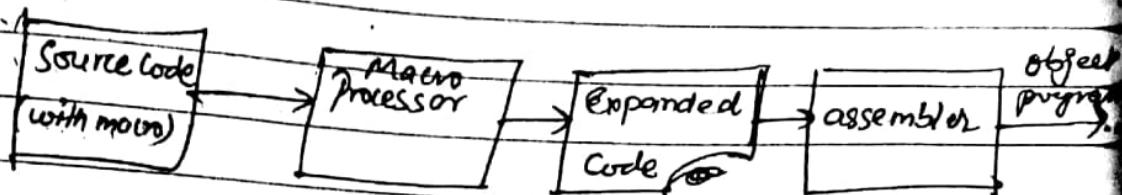
END

FIRST

COPY	START	0
FIRST	STL	RETADR
• CLOOP	RDBUFF	F1, BUFFER, LENGTH
CLOOP	CLEAR	X
	CLEAR	A
	CLEAR	S
	+LDT	# 4096
	TD	= X'F1'
	JEQ	* - 3
	RD	= X'F1'
	COMPR	A, S
	JEQ	* + 11
	STCH	BUFFER, X
	MXR	T
	JLT	* - 19
	STX	LENGTH

Fig:- Macro Expression Example.

- after macro processing the expanded file can be used as input to the assembler.
- the statements expansions generated from the macro had been assembled exactly as they were written directly by the programmers.



Difference between Macro and Subroutine

- statements that form the body of macro are generated each time a macro is expanded.
- statements in subroutine appear only once, regardless of how many times the subroutine is called.

Macro processor algorithm and data structures.

- Two pass macro processor can be chosen where
 - all macro definitions are processed during first pass.
 - all macro invocations statements are expanded during second pass.
 - two pass macro processor wouldn't allow the body of one macro instruction to contain definitions of other ~~macros~~ macros.
 - because all macros would have to be defined during the first pass ~~define~~ before any macro invocation were expanded.

MACROS

RDBUFF

MACRO

MACRO

|

|

|

MEND

MEND

MACRO X

WRCBUFF

MACRO

MACRO

|

|

MEND

|

RDBUFF

MACRO

|

MEND

|

MEND

(a)

(b)

Fig:- Examples of definition of macros within macros body.

- Defining MACROS and MACROX does not define ROBUPF and other macro instructions.
- These definitions are processed only when an invocation of MACROS and MACROX is expanded.
- A one pass macro processor that can alternate between macro definition and macro expansion is able to handle macro like these
- There are 3 main data structures in macro processor
 - ② Definition Table (DEFTAB)
 - The macro definition themselves are stored in definition table (DETTAB) which contains macro prototype and statements that makeup macro body.
 - Comment lines from macro definition are not entered into DETTAB :: they will not be part of macro expansion.

⑥ Name Table (NAMTAB)

- References to macro instructions parameters are connected to a position entered into NAMTAB, which serves index to DEF TAB.
- For each macro instruction defined, NAMTAB ~~contains~~ contains pointers to begining and end of definition in DEFTAB.

⑦ Argument table (ARGTAB)

- is used during expansion of macro invocation
- When macro invocation statements are recognized, the arguments are stored in ARG TAB according to their position in argument list.

- As the macro is expanded, arguments from ARGTAB are substitute for the corresponding parameters in macro body.

DEFTAB

NAMTAB

RDBUFF		

RDBUFF &INDEV, &BUFADR, &RECLEN

CLEAR X

CLEAR A

CLEAR S

TBDT # 4096

TD = x'031'

JEQ *-3

RD = x'?1'

COMPR A,S

JEQ *+11

STCH ?2, X

JIXR T

JLT *-19

STX ?3

MEND

ARGTAB

1	F1
2	BUFFER
3	LENGTH

→ when the ?n notation is recognized in a line form from DEFTAB, a simple indexing operation supplies the property argument from ARGTAB.

Algorithm

1) Procedure DEFINE

- called when the beginning of a macro definition is recognized.
- make appropriate entries in DEFTAB and NAMTAB.

2) Procedure Expand EXPAND

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

3) Procedure GETLINE

- get the nextline to be processed.

Machine Independent Macroprocessor Features.

Concatenation of Macro parameters.

- Most macro processors allow parameters to be concatenated with other character string.
- Say, a program contains one series of variables named by the symbols X_A1, X_A2, X_A3, \dots . another series @ named by X_B1, X_B2, X_B3, \dots etc.
- the body of macro definition might contain statement like

SUM	MACRO	&ID
.	LDA	X&ID1
	ADD	X&ID2
	ADD	X&ID3
	STA	X&ID5
	!	
	MEND	

- here, the begining of macro parameter is identified by starting symbol & however the end of parameter is not marked.
- The problem is that the end of the parameter is not marked so, $X&ID1$ may mean ' $X + &ID + 1$ ' or ' $X + &ID1$ '
- To avoid this ambiguity, a special concatenation operator ' \rightarrow ' is used.
- Now, new form becomes $X&ID \rightarrow 1$
here, ' \rightarrow ' will not appear in macro expansion.

eg:-

SUM

MACRO

d ID

LDA

 $X \& ID \rightarrow 1$

ADD

 $X \& ID \rightarrow 2$

ADD

 $X \& ID \rightarrow 3$

STA

 $X \& ID \rightarrow 5$

MEND

Now,

SUM A	SUM BETA
↓	↓
LDA XA1	LDA XBETAI
ADD XA2	ADD XBETAZ
ADD XB3	ADD XBETAB
STA XA5	STA XBETAG

* Generation of unique labels.

- if a macro is invoked and expanded multiple times, labels in the macro body may cause duplicate labels.
- to generate unique labels for each macro invocation, we must begin a label with \$ while writing ~~macro invocation~~, we must begin a label with \$ while writing with \$ while writing macro definition.
- during macro expansion, the \$ will be replaced with \$xx where xx is a two character alphanumeric counter of the number of macro instruction expanded.
→ xx will start from AA, AB, AC, ...
- Consider definition of WRBUFF

5	COPY	START	0
;			
135	TD	= x'6OUTDEV'	
;			
140	JEG	*-3	
;			
155	JLT	*-14	
;			
255	END	FIRST.	

- here, if a label was placed on TD instruction on line 135, this label would be defined twice, once for each invocation of WRBUFF.
- This duplicate definition would prevent correct assembly of the resulting expanded program.
- the relative addressing on line 140 & 155, may be acceptable for short jumps.

- for longer jumps spanning several instructions such notation is very inconvenient, error prone and difficult to read.
- these problems are avoided using special types of labels within macro instructions.

e.g:-

RDBUFF Definition.

RDBUFF	MACRO	&INDEV, &BUFADR, &RECLTH
CLEAR		X
CLEAR		A
CLEAR		S
+LDT		# 4096
\$LOOP	TD	= x'&INDEV'
	JEQ	\$Loop
	RD	= x'&INDEV'
	COMPR	A,S
	JEQ	\$Exit
	STCH	&BUFADR, X
	TIXR	T
	JLT	\$Loop
\$Exit	STX	&RECLTH
	MEND	

⇒ Expansion.

Macro Expansion.

RDBUFF	F1, BUFFER, LENGTH
CLEAR	X
CLEAR	A
CLEAR	S
+LDT	\$14096
\$AALOOP	TD = X'F1'
JEQ	\$AALOOP
RD	= X'F1'
COMPR	A,S
JEQ	\$AAloop Exit
STM	BUFFER, X
TXR	T
JLT	\$AALOOP
\$AAExit	STX LENGTH,

QX

Conditional Macro Expansion.

- So far, when a macro instruction is invoked, the same sequence of statements are used to expand macro.
- here, depending on the arguments supplied in the macro invocation, the sequence of statements generated for macro expansion can be modified.
- ~~for this~~ this adds greatly to power and flexibility of macro language.
- Macro time variable.
 - is a variable that begins with 'a' and that is not a macro instruction parameter.
 - can be initialized to a value of 0.
 - can be set by macro processor directive SET.
 - can be used to
 - (i) store working values during expansion.
 - (ii) store the evaluation result of Boolean Expansion.
 - (iii) control macro time conditional structures.

→ Macro time conditional structure:

i) IF - ELSE, - END IF

ii) WHILE - ENDW

Implementation of Conditional Macro Expansion.

(i) IF - ELSE - ENDIF

- Firstly a symbol table is maintained by macro processor that 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

macro time variable whenever it is required.

- the testing of condition and looping are done while macro is being expanded.
- 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.
- if value is FALSE
 - the macroprocessor skips ahead in DEFTAB until it finds the next ELSE or ENDIF statement.

(ii) WHILE-ENDW

- When a WHILE statement is encountered during the expansion of a macro, the specified Boolean expression is evaluated.
- if value is true
 - the macroprocessor continues to process lines from DEFTAB until it encounters next ENDW statement.
 - when ENDW is encountered, the macro processor returns to the preceding WHILE, re-evaluates boolean expression and taken action again.
- if value is false.
 - the macro processor skips a head in DEFTAB until it finds the next ENDW statement and then resumes normal macro expression.

keyword Macro parameter.

L Positional Parameter.

→ Parameters

their positions and arguments are associated according to their positions in the macro prototype and invocation.

→ programmer must specify the arguments in proper order.

→ If an argument is to be omitted, a null argument should be used to maintain proper order in macro invocation statement.

→ For eg:- Suppose a macro instruction GENER has 10 possible parameters, but in a particular invocation of the macro, only the 3rd and 9th parameters are to be specified.

Then statement is

GENER „DIRECT„„„, 3,

→ It is not suitable if a macro has a large no. of parameters and only a few of these are given values in a typical invocation.

2. Keyword Parameter.

→ 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.

→ If the 3rd parameter is named &TYPE and 9th parameter is named &CHANNEL, the macro invocation would be

GENER TYPE = DIRECT, CHANNEL = 3

→ It is easier to read and much less error prone than positional method //

Macroprocessor Design option

- Because Recursive macro invocation and expansion can't be handled by previous macro processor design.
- Reasons!-
 - 1) the procedure EXPAND would be called recursively, thus invocation arguments in the ARGTAB will be overwritten.
 - 2) the boolean variable EXPANDING would be set to FALSE when the inner macro expansion is finished, that is, the macro processor would forget that it had been in the middle of expanding 'outer' macro.
 - 3) A similar problem would occur with PROCESSLINE since this procedure too would be called recursively.

→ Solutions!-

- 1) Write the macro processor in a programming language that allows recursive calls, thus local variables will be retained.
- 2) Use a stack to take care of pushing and popping local variables and return address.

③ Two pass macro processor.

→ Pass 1

- process macro definition.

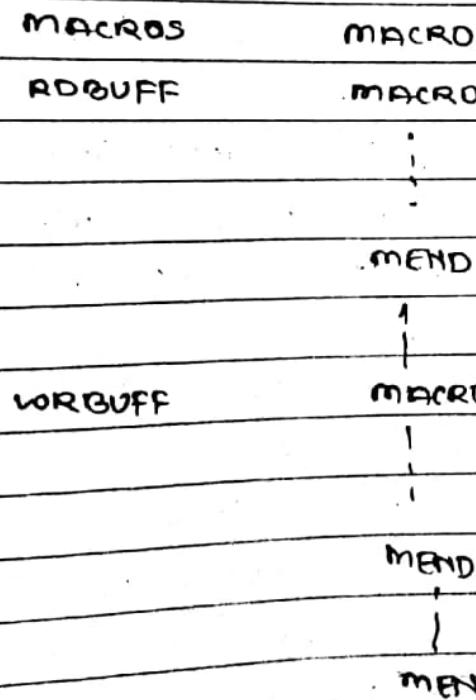
→ Pass 2

- expand all macro invocation statements.

→ problems

- this kind of macro processor can not allow recursive macro definition; i.e. the body of a macro contains definitions of other macros.

- example of recursive macro definition.
- Main MACROS (for SIC)
 - Contains the definition of RDBUFF and WRBUFF written in SIC instruction.
 - MACROX (for SIC/XE)
 - Contains definitions of RDBUFF and WRBUFF written in SIC/XE instructions.
 - A program that is to be run on SIC system could invoke MACROS whereas a program that is to be run on SIC/XE system could invoke MACROX.
 - Defining MACROS or MACROX does not define RDBUFF and WRBUFF. These definitions are processed only when an invocation of MACROS or MACROX is expanded.



③ One pass assembler.

- one pass macroprocessor alternates between macro definition and macro expansion in a recursive way, to handle recursive macro definition.
- Because of one pass structure, the definition of a macro must appear in source program before any statements that invoke that macro.

Handling recursive macro definition.

→ in DEFINE procedure

- 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 recursive macros definition because the first MEND encountered in the inner macro will terminate whole macro definition process.
- to solve this problem, a counter LEVEL is used to keep track of level of macro definition.
- increase level by 1 each time a MACRO directive is read.
- decrease level by 1 each time a MEND directive is read.
- A MEND can terminate the whole macro definition process only when LEVEL reaches 0.

④ General purpose macro processor

Goal:-

- Macro processors that do not depend on any particular programming language, but can be used

with a variety of different languages.

advantages:-

- programmers do not need to learn many macro languages.
- although its dependent development costs are somewhat greater than those for a language specific macro processor, this expense does not need to be repeated for each language, thus save overall cost.

disadvantages.

- large number of details must be dealt with near programming language.
- situation in which normal macro parameter substitution should not occur.
- Syntax.

(A) Macro processing within language translations.

- Macro processor can be

① preprocessors

- process macro definition
- expand macro invocation
- produce an expanded version of source program which is then used as input to an assembler or compiler.

(B) Line by line macro processor.

- used as a sort of input routine for the assembler or compiler.
- read source program.

- process macro definition and expand macro invocations
- pass output lines to the assembler or compiler.
- it avoids making an extra pass over the source program.
- Data structure required by the macro processor and the language translator can be combined.
- utility subroutines can be used by both processor and the language translator.
 - Scanning input lines
 - Searching table.
 - data format conversion

② Integrated macroprocessor:

- an integrated macro processor can potentially make use of any instruction information about the source program that is extracted by language translators.
- it can support macro instructions that depend upon the context in which they occur.

③ Drawback of line by line or integrated macro processor

- they must be specially designed and written to work with a particular implementation of assemble or compiler.
- the cost of macro processor development is added to the costs of language translator, which results expensive development.
- the assemblies or compiler will be large and complex.

M.T.V. = Macro time variable

Date _____
Page _____

* Conditional

macro expansion (example)

RDBUFF

MACRO

&INDEX, &BUFAADR, &RECINT, &EDR, &maxl

&EDRCK

IF

(&EDR NE '')

SET

1

ENDIF

CLEAR

x

CLEAR

a

IF

(&EDRCK EQ 1)

LDCH

= x'&EDR'

RMO

A,S

ENDIF

IF

(&MAXLEN EQ ' ')

+LDT

4096

ELSE

+LDT

&MAXLEN

ENDIF

!

MEND

* condition

RDBUFF F2, BUFFER, LENGTH, 3, ,
1 2 3 4 5

1. checks if (&EDR NE '') → true

2. checks if (&EDRCK EQ 1) → true

3. checks if (&MAXLEN EQ ' ') → true

4. CLEAR x

CLEAR a

LDCH # = x'3'

RMO A,S

+LDT # 4096

Ques Assignment

1. Consider the macro definition given below and show macro expansion for the macro call statement "print 64, FJ". Show all data structures used by macro processor clearly.

```

print MACRO &ch, &od
$Repeat TD &od
    JEQ $repeat
    LDCH #&ch
    WD &od
MEND

```

Ans/ Data structure

DEFTAB

NAMTAB

		Print &ch, &od
Print		\$Repeat TD ?2
		JEQ \$repeat
		LDCH #?1
		WD ?2
		MEND

ARGTAB

1	64
2	FJ

Pg 2
~~Expansion~~

• print

\$AARepeat TD B4, F1

F3

JEG

\$AARrepeat

LDCH

64

WD

F1