MACRO PROCESSORS

Email: nibaran@cse.jdvu.ac.in

INTRODUCTION

- A macro instruction (abbreviated to *macro*) is simply a notational convenience for the programmer.
- Macros are special code fragments that are defined once in the program and are used repetitively by calling them from various places within the program
- A macro represents a commonly used group of statements in the source programming language
- Expanding a macros
  - Replace each macro instruction with the corresponding group of source language statements

Email: nibaran@cse.jdvu.ac.in

INTRODUCTION (CONT'D)

- E.g.
  - On SIC/XE requires a sequence of seven instructions to save the contents of all registers
    - `Write one statement like SAVERGS`
- A macro processor is not directly related to the architecture of the computer on which it is to run
- Macro processors can also be used with high-level programming languages, OS command languages, etc.

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 OR SUBROUTINE WHICH ONE ?

- Size
  - If size is significantly high with substitution for macro,
    - Compiled/assembled program will be very big
      - Slow down the program execution
      - Not preferable ..Subprogram suitable
  - Suitable for small program
- Number of parameters
  - A large number of parameters in a subprogram will necessitate creating all of them in the stack
    - Require large stack-area and may slow down execution

Email: nibaran@cse.jdvu.ac.in

MACRO OR SUBROUTINE WHICH ONE ?

- Debugging
  - Easier with subprogram
    - Execution stopped in entry point
    - For macro source line number where macro is replaced
- Recursion
  - Cannot be implemented with macro
    - Recursion call on run time not in compile time
- Parameter passing
  - Macro normally support call by value not call by references

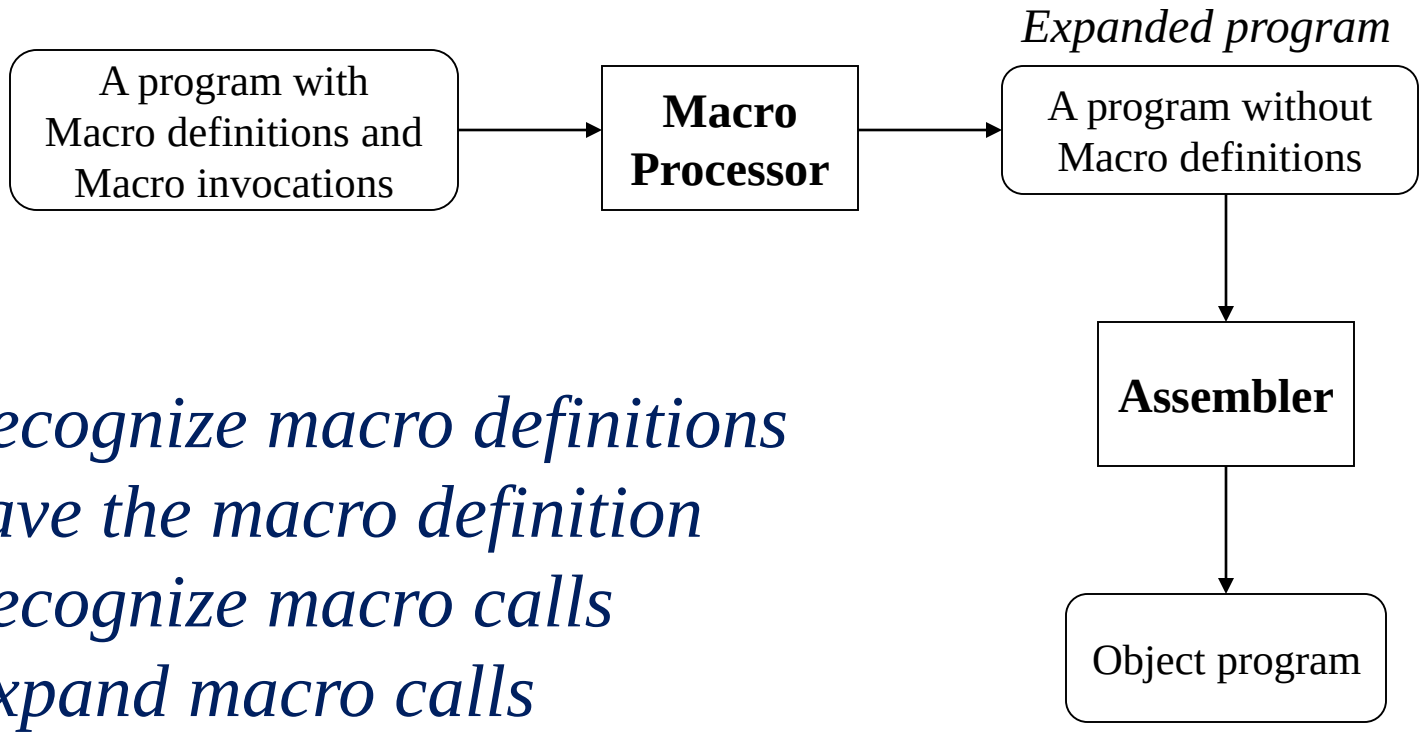
Email: nibaran@cse.jdvu.ac.in

DEFINITIONS

- Macro-processor
  - The task of this system is to replace each macro instruction with the corresponding group of source language statements
- Macro-call
  - A macro name is used in an assembly source program
    - `Macro is called by name`
- Macro-expansion
  - The output of the macroprocessor where macro-call instruction is replaced by its corresponding group of instruction.
  - The output program is termed as macro expanded output

Email: nibaran@cse.jdvu.ac.in

BASIC MACRO PROCESSOR FUNCTIONS



- *Recognize macro definitions*
- *Save the macro definition*
- *Recognize macro calls*
- *Expand macro calls*

Email: nibaran@cse.jdvu.ac.in

BASIC MACRO PROCESSOR FUNCTIONS

Macro Definition

- Two new assembler directives
  - In SIC/XE
    - MACRO
    - MEND
  - In nasm
    - % macro
    - %end macro
- A pattern or prototype for the macro instruction
  - Macro name and parameters
- See next picture

Email: nibaran@cse.jdvu.ac.in

MACRO DEFINITION & CHALLENGES

- Macro Definition
  - copy code
  - parameter substitution
  - conditional macro expansion
  - macro instruction defining macros
- Major Challenge
  - Local Lables in macro
  - Nested Macros
    - Macro defining nesting (most challenging)
    - Macro call nesting

Email: nibaran@cse.jdvu.ac.in

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
    .
```

Email: nibaran@cse.jdvu.ac.in

PARAMETER SUBSTITUTION -- EXAMPLE

Source

```
STRG    MACRO &a1, &a2, &a3
        STA     &a1
        STB     &a2
        STX     &a3
        MEND
    .
STRG    DATA1, DATA2, DATA3
    .
STRG    DATA4, DATA5, DATA6
    .
    .
```

Expanded souce

```
    .
    .
    .
      STA     DATA1
      STB     DATA2
      STX     DATA3
    .
      STA     DATA4
      STB     DATA5
      STX     DATA6
    .
```

Email: nibaran@cse.jdvu.ac.in

```
Line    Source statement

  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
```

Figure 4.1 Use of macros in a SIC/XE program.

BASIC MACRO PROCESSOR FUNCTIONS

- Macro invocation
  - Often referred to as a *macro call*
    - Need the name of the macro instruction begin invoked and the arguments to be used in expanding the macro
- Expanded program
  - Figure 4.2
  - No macro instruction definitions
  - Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from the macro invocation substituted for the parameters in the prototype

Line	Source statement			
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	

Figure 4.2 Program from Fig. 4.1 with macros expanded.

BASIC MACRO PROCESSOR FUNCTIONS

- Macro invocations and subroutine calls are different
- Note also that the macro instructions have been written so that the body of the macro contains no label
  - Why?

- It is easy to design a two-pass macro processor
  - Pass 1:
    - `All macro definitions are processed`
  - Pass 2:
    - `All macro invocation statements are expanded`
- However, a two-pass macro processor would not allow the body of one macro instruction to contain definitions of other macros
  - See Figure 4.3

```
1   MACROS    MACRO      {Defines SIC standard version macros}
2   RDBUFF    MACRO      &INDEV,&BUFADR,&RECLTH
                .
                .        {SIC  standard version}
                .
3             MEND       {End of RDBUFF}
4   WRBUFF    MACRO      &OUTDEV,&BUFADR,&RECLTH
                .
                .        {SIC standard version}
                .
5             MEND       {End of WRBUFF}
                .
                .
                .
6             MEND       {End of MACROS}
```

(a)

```
1   MACROX    MACRO      {Defines  SIC/XE macros}
2   RDBUFF    MACRO      &INDEV,&BUFADR,&RECLTH
                .
                .        {SIC/XE version}
                .
3             MEND       {End of RDBUFF}
4   WRBUFF    MACRO      &OUTDEV,&BUFADR,&RECLTH
                .
                .        {SIC/XE version}
                .
5             MEND       {End of WRBUFF}
                .
                .
                .
6             MEND       {End of MACROX}
```

(b)

Figure 4.3 Example of the definition of macros within a macro body.

- Sub-Macro definitions are only processed when an invocation of their Super-Macros are expanded
  - See Figure 4.3: RDBUFF
- A one-pass macro processor that can alternate between macro definition and macro expansions able to handle macros like those in Figure 4.3

MACRO PROCESSOR ALGORITHM AND DATA STRUCTURES

- Because of the one-pass structure, the definition of a macro must appear in the source program before any statements that invoke that macro
- Three main data structures involved in an one-pass macro processor
  - DEFTAB/ MDT(Macro definition table),
  - NAMTAB/MNT(Macro name table),
  - ARGTAB/ALA(Argument List array)

Email: nibaran@cse.jdvu.ac.in

MAJOR DATA STRUCTURES

- DEFTAB/ MDT(Macro definition table)
  - The body of a newly defined macro is put into MDT
  - One dimensional array of character strings
    - Contains lines of a macro
- NAMTAB/MNT(Macro name table)
  - Name-the name of macro being defined
  - Num-of-parameters
    - Numbers of parameters of the macro
  - MDT-start-index
  - MDT-end-index

Name	Num-of-parameters	MDT-start-index	MDT-end-index

- ARGTAB/ALA(Argument List array)
- Dynamically created

Previous Frame Pointer
Current Frame Pointer
Arguments list

One-Pass Macro Processor

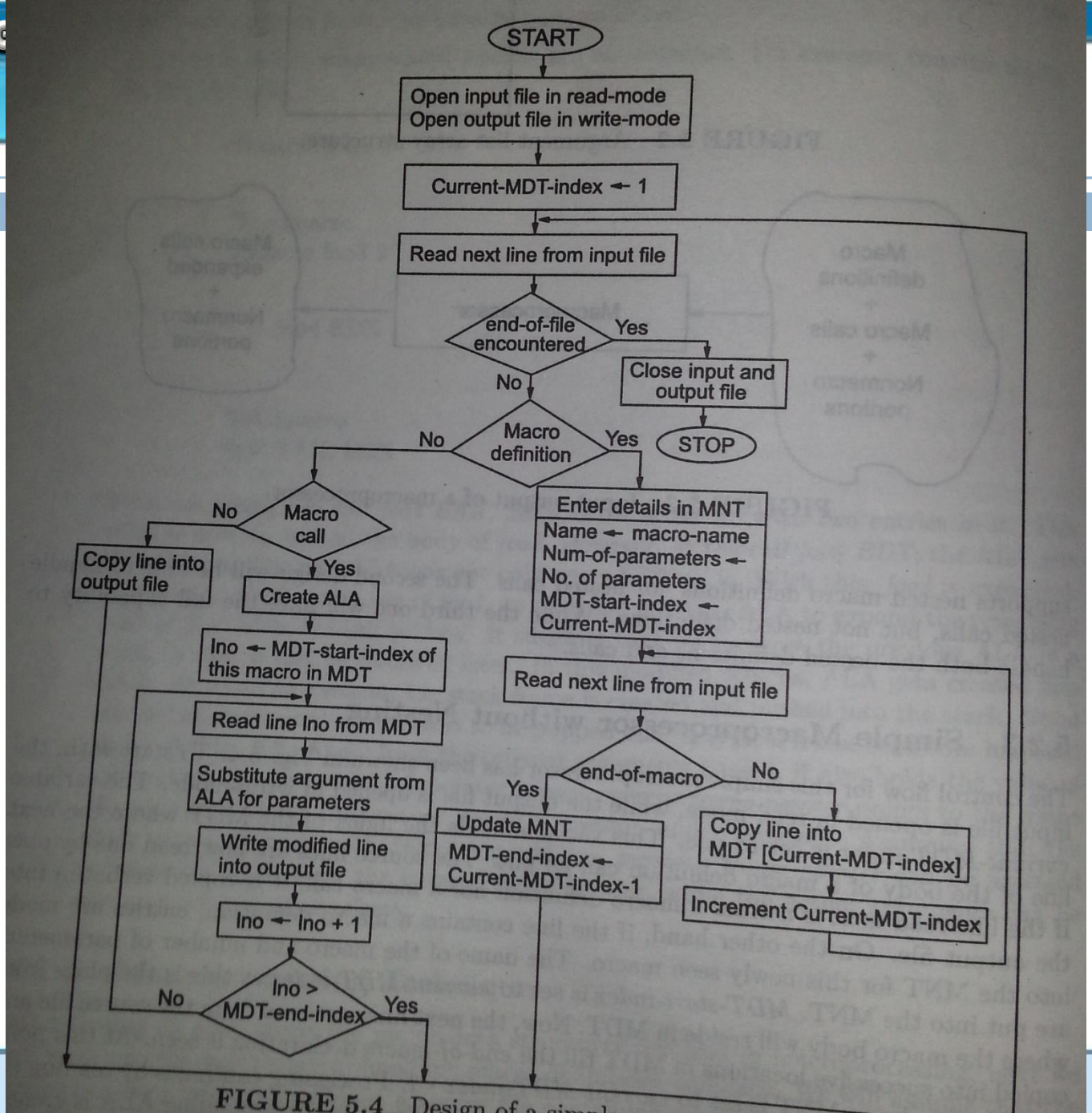


FIGURE 5.4 Design of a simple macroprocessor.

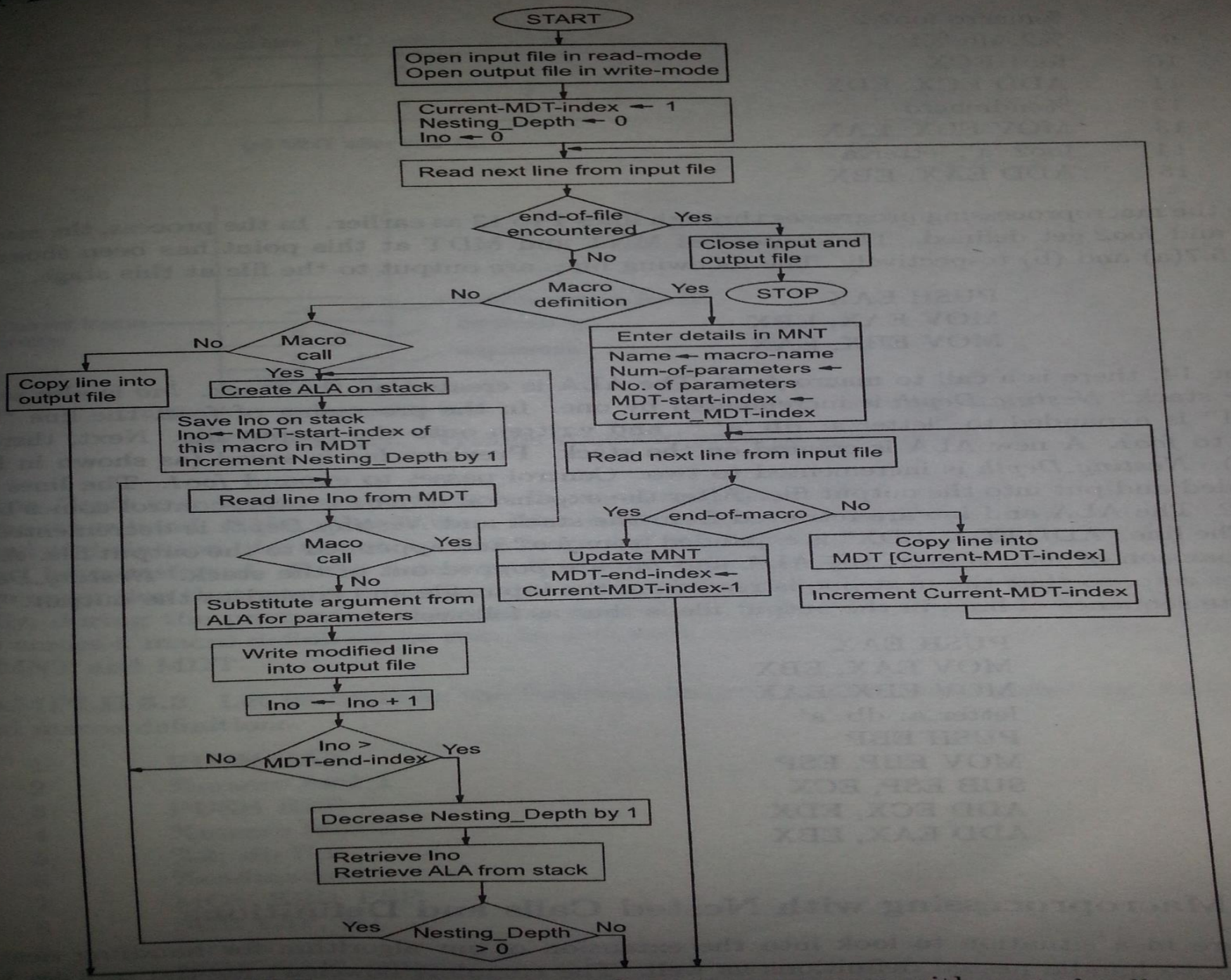
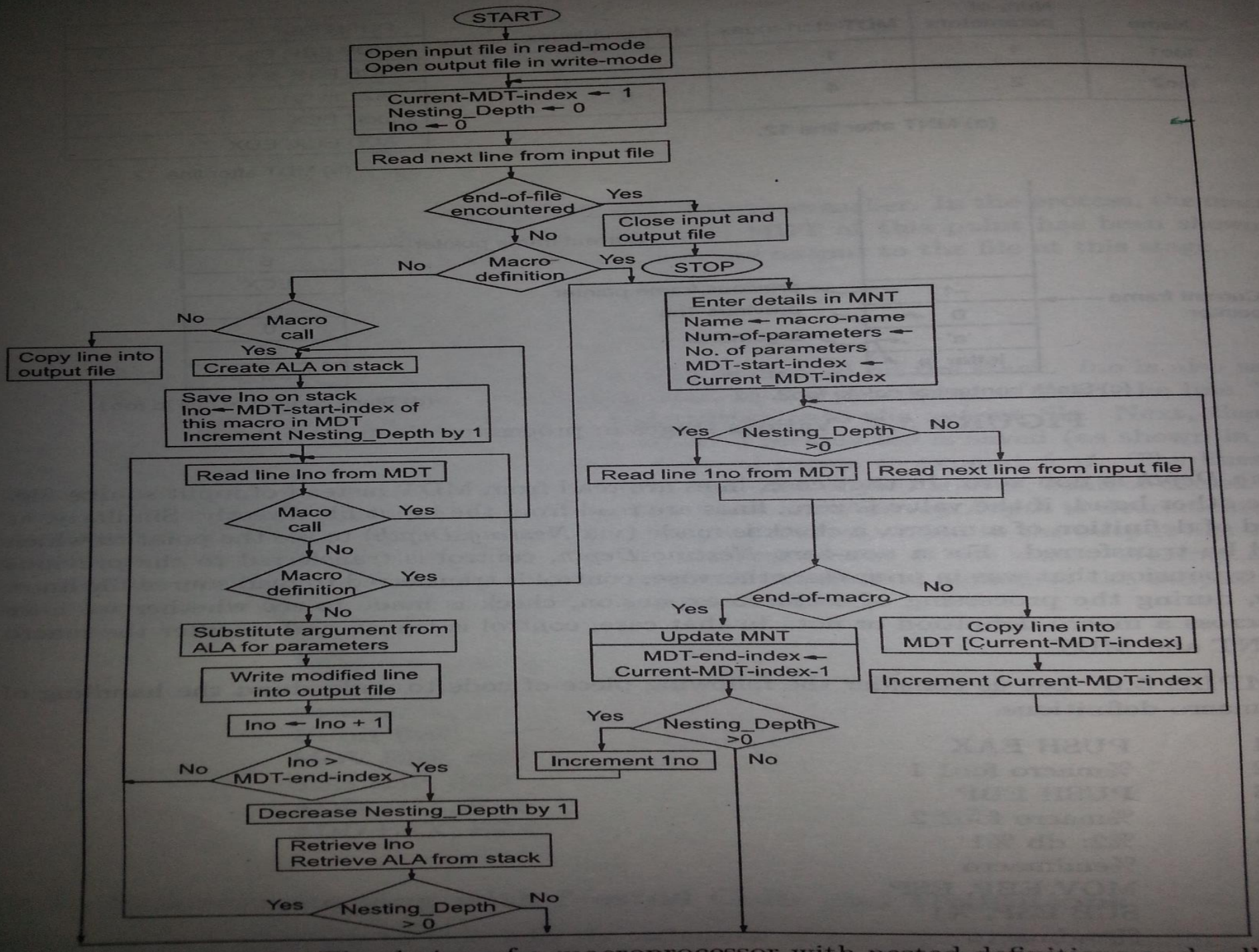


FIGURE 5.6 The Design of a macroprocessor with nested calls.


START
Open input file in read-mode
Open output file in write-mode
Current-MDT-index ← 1
Nesting_Depth ← 0
Ino ← 0
Read next line from input file
end-of-file encountered
Yes
Close input and output file
STOP
No
Macro definition
Yes
Enter details in MNT
Name ← macro-name
Num-of-parameters ← No. of parameters
MDT-start-index ← Current_MDT-index
No
Macro call
Copy line into output file
Yes
Create ALA on stack
Save Ino on stack
Ino ← MDT-start-index of this macro in MDT
Increment Nesting_Depth by 1
Read line Ino from MDT
Maco call
Yes
No
Macro definition
Yes
No
Substitute argument from ALA for parameters
Write modified line into output file
Ino ← Ino + 1
Ino > MDT-end-index
No
Yes
Decrease Nesting_Depth by 1
Retrieve Ino
Retrieve ALA from stack
Nesting_Depth > 0
Yes
No
Yes
Nesting_Depth >0
No
Read line 1no from MDT
Read next line from input file
end-of-macro
Yes
No
Update MNT
MDT-end-index ← Current-MDT-index-1
Copy line into MDT [Current-MDT-index]
Increment Current-MDT-index
Yes
Nesting_Depth >0
No
Increment 1no

COMPARISON OF MACRO PROCESSORS DESIGN

- Single pass
  - every macro must be defined before it is called
  - one-pass processor can alternate between macro definition and macro expansion
  - nested macro definitions may be allowed but nested calls are not
- Two pass algorithm
  - Pass1: Recognize macro definitions
  - Pass2: Recognize macro calls
  - nested macro definitions are not allowed

Email: nibaran@cse.jdvu.ac.in

MACHINE-INDEPENDENT MACRO PROCESSOR FEATURE

- Concatenation of Macro Parameters
- Generation of Unique Labels
- Conditional Macro Expansion
- Keyword Macro Parameters

Email: nibaran@cse.jdvu.ac.in

CONCATENATION OF MACRO PARAMETERS

- Most macro processors allow parameters to be concatenated with other character strings
  - The need of a special catenation operator
    - `LDA X&ID1`
    - `LDA X&ID`
  - The catenation operator
    - `LDA X&ID→1`
- See figure 4.6

```
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
```

(a)

```
SUM      A
```

```
LDA      XA1
ADD      XA2
ADD      XA3
STA      XAS
```

(b)

```
SUM      BETA
```

```
LDA      XBETA1
ADD      XBETA2
ADD      XBETA3
STA      XBETAS
```

(c)

Figure 4.6 Concatenation of macro parameters.

GENERATION OF UNIQUE LABELS

- It is in general not possible for the body of a macro instruction to contain labels of the usual kind
  - Leading to the use of relative addressing at the source statement level
    - `Only be acceptable for short jumps`
- Solution:
  - Allowing the creation of special types of labels within macro instructions
  - See Figure 4.7
    - `Labels used within the macro body begin with the special character $`
  - Programmers are instructed not to use $ in their source programs

```
25    RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH
30              CLEAR    X              CLEAR LOOP COUNTER
35              CLEAR    A
40              CLEAR    S
45              +LDT     #4096          SET MAXIMUM RECORD LENGTH
50    $LOOP     TD       =X'&INDEV'     TEST INPUT DEVICE
55              JEQ      $LOOP          LOOP UNTIL READY
60              RD       =X'&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 MAXIMUM LENGTH
85              JLT      $LOOP            HAS BEEN REACHED
90    $EXIT     STX      &RECLTH        SAVE RECORD LENGTH
95              MEND
```

(a)

```
      .         RDBUFF   F1,BUFFER,LENGTH

30              CLEAR    X              CLEAR LOOP COUNTER
35              CLEAR    A
40              CLEAR    S
45              +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 INTO 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    $AAEXIT   STX      LENGTH         SAVE RECORD LENGTH
```

(b)

Figure 4.7 Generation of unique labels within macro expansion.

CONDITIONAL MACRO EXPANSION

- Most macro processors can modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation
- See Figure 4.8

```
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 CHARACTER
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 INTO 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
```

(a)

```
          .       RDBUFF   F3,BUF,RECL,04,2048

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 INTO REG A
65                COMPR    A,S                  TEST FOR END OF RECORD
70                JEQ      $AAEXIT              EXIT LOOP IF EOR
75                STCH     BUF,X                STORE CHARACTER IN BUFFER
80                TIXR     T                    LOOP UNLESS MAXIMUM LENGTH
85                JLT      $AALOOP                HAS BEEN REACHED
90      $AAEXIT   STX      RECL                 SAVE RECORD LENGTH
```

(b)

Figure 4.8 Use of macro-time conditional statements.

```
.       RDBUFF  0E,BUFFER,LENGTH,,80

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 INTO 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
```

(c)

```
.       RDBUFF  F1,BUFF,RLENG,04

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 INTO 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 BEEN REACHED
90      $ACEXIT STX     RLENG           SAVE RECORD LENGTH
```

(d)

Figure 4.8 (*cont'd*)

CONDITIONAL MACRO EXPANSION

- Most macro processors can modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation
- See Figure 4.8
  - Macro processor directive
    - `IF, ELSE, ENDIF`
    - `SET`
  - Macro-time variable (set symbol)
- WHILE-ENDW
  - See Figure 4.9

```
 25    RDBUFF     MACRO     &INDEV,&BUFADR,&RECLTH,&EOR
 27    &EORCT     SET       %NITEMS(&EOR)
 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               COMP      =X'0000&EOR[&CTR]'
 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 RECORD LENGTH
100               MEND
```

(a)

```
          .       RDBUFF    F2,BUFFER,LENGTH,(00,03,04)

 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
```

(b)

Figure 4.9 Use of macro-time looping statements.

KEYWORD MACRO PARAMETERS

- Positional parameters
  - Parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement
  - Consecutive commas is necessary for a null argument

```
GENER   ,,DIRECT,,,,,,3
```

Email: nibaran@cse.jdvu.ac.in

KEYWORD MACRO PARAMETERS

- Keyword parameters
  - Each argument value is written with a keyword that names the corresponding parameter
  - A macro may have a large number of parameters , and only a few of these are given values in a typical invocation

```
GENER TYPE=DIRECT, CHANNEL=3
```

```
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 INTO 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
```

(a)

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


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 INTO 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      $AAEXIT   STX       LENGTH            SAVE RECORD LENGTH
```

(b)

Figure 4.10 Use of keyword parameters in macro instructions.

```
.          RDBUFF   RECLTH=LENGTH,BUFADR=BUFFER,EOR=,INDEV=F3


30         CLEAR    X              CLEAR LOOP COUNTER
35         CLEAR    A
47        +LDT      #4096          SET MAXIMUM RECORD LENGTH
50  $ABLOOP TD      =X'F3'         TEST INPUT DEVICE
55         JEQ      $ABLOOP        LOOP UNTIL READY
60         RD       =X'F3'         READ CHARACTER INTO REG A
75         STCH     BUFFER,X       STORE CHARACTER IN BUFFER
80         TIXR     T              LOOP UNLESS MAXIMUM LENGTH
85         JLT      $ABLOOP          HAS BEEN REACHED
90  $ABEXIT STX     LENGTH         SAVE RECORD LENGTH
```

(c)

Figure 4.10 (*cont'd*)

GENERAL-PURPOSE MACRO PROCESSORS

- Macro processors have been developed for some high-level programming languages
- These special-purpose macro processors are similar in general function and approach; however, the details differ from language to language

GENERAL-PURPOSE MACRO PROCESSORS

- The advantages of such a general-purpose approach to macro processing are obvious
  - The programmer does not need to learn about a different macro facility for each compiler or assembler language, so much of the time and expense involved in training are eliminated
  - A substantial overall saving in software development cost

GENERAL-PURPOSE MACRO PROCESSORS

- In spite of the advantages noted, there are still relatively few general-purpose macro processors. Why?

1. In a typical programming language, there are several situations in which normal macro parameter substitution should no occur
   - E.g. comments should usually be ignored by a macro processor

GENERAL-PURPOSE MACRO PROCESSORS

2. Another difference between programming languages is related to their facilities for grouping together terms, expressions, or statements
   - E.g. Some languages use keywords such as begin and end for grouping statements. Others use special characters such as { and }.

GENERAL-PURPOSE MACRO PROCESSORS

3. A more general problem involves the tokens of the programming language
   - E.g. identifiers, constants, operators, and keywords
   - E.g. blanks

GENERAL-PURPOSE MACRO PROCESSORS

4. 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

ANSI C MACRO LANGUAGE

Email: nibaran@cse.jdvu.ac.in

ANSI C MACRO LANGUAGE

- Definitions and invocations of macros are handled by a preprocessor, which is generally not integrated with the rest of the compiler.
- Examples:

```
#define  NULL  0
#define  EOF   (-1)
#define  EQ    ==                    ➔syntactic
   modification
#define  ABSDIFF  (X,Y)    ( (X)>(Y) ? (X)-(Y) : (Y)-
   (X) )
```

ANSI C MACRO LANGUAGE

- Parameter substitutions are not performed within quoted strings:

  #define DISPLAY(EXPR) printf("EXPR= %d\n", EXPR)

  - Macro expansion example

    DISPLAY(I*J+1) ➔ printf("EXPR= %d\n", I*J+1)

- A special "stringizing" operator, #, can be used to perform argument substitution in quoted strings:

  #define DISPLAY(EXPR) printf(#EXPR "= %d\n", EXPR)

  - Macro expansion example

    DISPLAY(I*J+1) ➔ printf("I*J+1" "= %d\n", I*J+1)

ANSI C MACRO LANGUAGE

- Recursive macro definitions or invocations
  - After a macro is expanded, the macro processor rescans the text that has been generated, looking for more macro definitions or invocations.
  - Macro cannot invoke or define itself recursively.

```
DISPLAY(ABSDIFF(3,8))
```

scan

```
printf("ABSDIFF(3,8)"  "= %d\n", ABSDIFF(3,8))
```

rescan

```
printf("ABSDIFF(3,8)"  "= %d\n", ( (3)>(8) ? (3)-(8) : (8)-(3) ))
```

Email: nibaran@cse.jdvu.ac.in

ANSI C MACRO LANGUAGE

- Conditional compilation statements
- Example 1:

```
#ifndef   BUFFER_SIZE
#define  BUFFER_SIZE  1024
#endif
```

- Example 2:

```
#define  DEBUG  1
:
#if DEBUG == 1
printf(…)          /* debugging outout */
#endif
```

ANSI C MACRO LANGUAGE

- Miscellaneous functions of the preprocessor of ANSI C
  - Trigraph sequences are replaced by their single-character equipments, e.g., ??< → {
  - Any source line that ends with a backlash, \, and a newline is spliced together with the following line.
  - Any source files included in response to an #include directive are processed.
  - Escape sequences are converted e.g., \n, \0
  - Adjacent string literals are concatenated, e.g., “hello,” “world” → “hello, world”.

PREPROCESSING: AN INTRODUCTION

- Preprocessing
  - Occurs before program compiled
    - Inclusion of external files
    - Definition of symbolic constants
    - Macros
    - Conditional compilation
    - Conditional execution
  - All directives begin with #
    - Can only have whitespace before directives
  - Directives not C++ statements
    - Do not end with ;

THE #INCLUDE PREPROCESSOR DIRECTIVE

- **`#include` directive**
  - Puts copy of file in place of directive
    - `Seen many times in example code`
  - Two forms
    - **`#include <filename>`**
      - For standard library header files
      - Searches predesignated directories
    - **`#include "filename"`**
      - Searches in current directory
      - Normally used for programmer-defined files

THE #INCLUDE PREPROCESSOR DIRECTIVE

- Usage
  - Loading header files
    - `#include <stdio.h>`
  - Programs with multiple source files
  - Header file
    - Has common declarations and definitions
    - Classes, structures, enumerations, function prototypes
    - Extract commonality of multiple program files

THE #DEFINE PREPROCESSOR DIRECTIVE: SYMBOLIC CONSTANTS

- **#define**
  - Symbolic constants
    - Constants represented as symbols
    - When program compiled, all occurrences replaced
  - Format
    - ***#define identifier replacement-text***
    - **#define PI 3.14159**
  - Everything to right of identifier replaces text
    - **#define PI=3.14159**
    - Replaces **PI** with **"=3.14159"**
    - Probably an error
  - Cannot redefine symbolic constants

THE #DEFINE PREPROCESSOR DIRECTIVE: SYMBOLIC CONSTANTS

- Advantages
  - Takes no memory
- Disadvantages
  - Name not be seen by debugger (only replacement text)
  - Do not have specific data type
- `const` variables preferred

THE #DEFINE PREPROCESSOR DIRECTIVE: MACROS

- Macro
  - Operation specified in **`#define`**
  - Intended for legacy C programs
  - Macro without arguments
    - `Treated like a symbolic constant`
  - Macro with arguments
    - `Arguments substituted for replacement text`
    - `Macro expanded`
  - Performs a text substitution
    - `No data type checking`

Email: nibaran@cse.jdvu.ac.in

THE #DEFINE PREPROCESSOR DIRECTIVE: MACROS

- Example

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
area = CIRCLE_AREA( 4 );
```

becomes

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

- Use parentheses
  - Without them,

```
#define CIRCLE_AREA( x )  PI * x * x
area = CIRCLE_AREA( c + 2 );
```

becomes

```
area = 3.14159 * c + 2 * c + 2;
```

which evaluates incorrectly

THE #DEFINE PREPROCESSOR DIRECTIVE: MACROS

- Multiple arguments

```
#define RECTANGLE_AREA( x, y )  ( ( x ) * ( y ) )
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
```

becomes

```
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

- **#undef**
  - Undefines symbolic constant or macro
  - Can later be redefined

CONDITIONAL COMPILATION

- Control preprocessor directives and compilation
  - Cannot evaluate cast expressions, **`sizeof`**, enumeration constants
- Structure similar to **`if`**

```
#if !defined( NULL )
   #define NULL 0
#endif
```

  - Determines if symbolic constant **`NULL`** defined
  - If **`NULL`** defined,
    - **`defined( NULL )`** evaluates to **`1`**
    - **`#define`** statement skipped
  - Otherwise
    - **`#define`** statement used
  - Every **`#if`** ends with **`#endif`**

CONDITIONAL COMPILATION

- Can use else
  - `#else`
  - `#elif` is "else if"
- Abbreviations
  - `#ifdef` short for
    - `#if defined(name)`
  - `#ifndef` short for
    - `#if !defined(name)`

CONDITIONAL COMPILATION

- "Comment out" code
  - Cannot use `/* ... */` with C-style comments
    - Cannot nest `/* */`
  - Instead, use

```
#if 0
   code commented out
#endif
```

  - To enable code, change `0` to `1`

CONDITIONAL COMPILATION

- Debugging

```
#define DEBUG 1
#ifdef DEBUG
  cerr << "Variable x = " << x << endl;
#endif
```

  - Defining **DEBUG** enables code
  - After code corrected
    - Remove **#define** statement
    - Debugging statements are now ignored

THE # AND ## OPERATORS

- # operator
  - Replacement text token converted to string with quotes

```
#define HELLO( x ) cout << "Hello, " #x << endl;
```

  - **HELLO(JOHN)** becomes
    - `cout << "Hello, " "John" << endl;`
    - Same as `cout << "Hello, John" << endl;`
- ## operator
  - Concatenates two tokens

```
#define TOKENCONCAT( x, y )  x ## y
```

  - **TOKENCONCAT(O, K)** becomes
    - OK

ASSERTIONS

- **`assert`** is a macro
  - Header **`<cassert>`**
  - Tests value of an expression
    - If **`0`** (**`false`**) prints error message, calls **`abort`**
      - Terminates program, prints line number and file
      - Good for checking for illegal values
    - If **`1`** (**`true`**), program continues as normal
  - **`assert( x <= 10 );`**
- To remove **`assert`** statements
  - No need to delete them manually
  - **`#define NDEBUG`**
    - All subsequent **`assert`** statements ignored