

Macro

Refer to Book

L.L. Beck: System Software-An Introduction to Systems Programming,
Addition Wesley

Introduction

- A macro instruction (abbreviated to macro) is simply a notational convenience for the programmer.
- Represents a commonly used group of statements in the source programming language
- Expanding a macro
 - Replace each macro instruction with the corresponding group of source language statements
- We will follow SIC (Simplified Instructional Computer) architecture and then
- NASM assembler (x86 arch)

Theoretical Macro – SIC based

Part - I

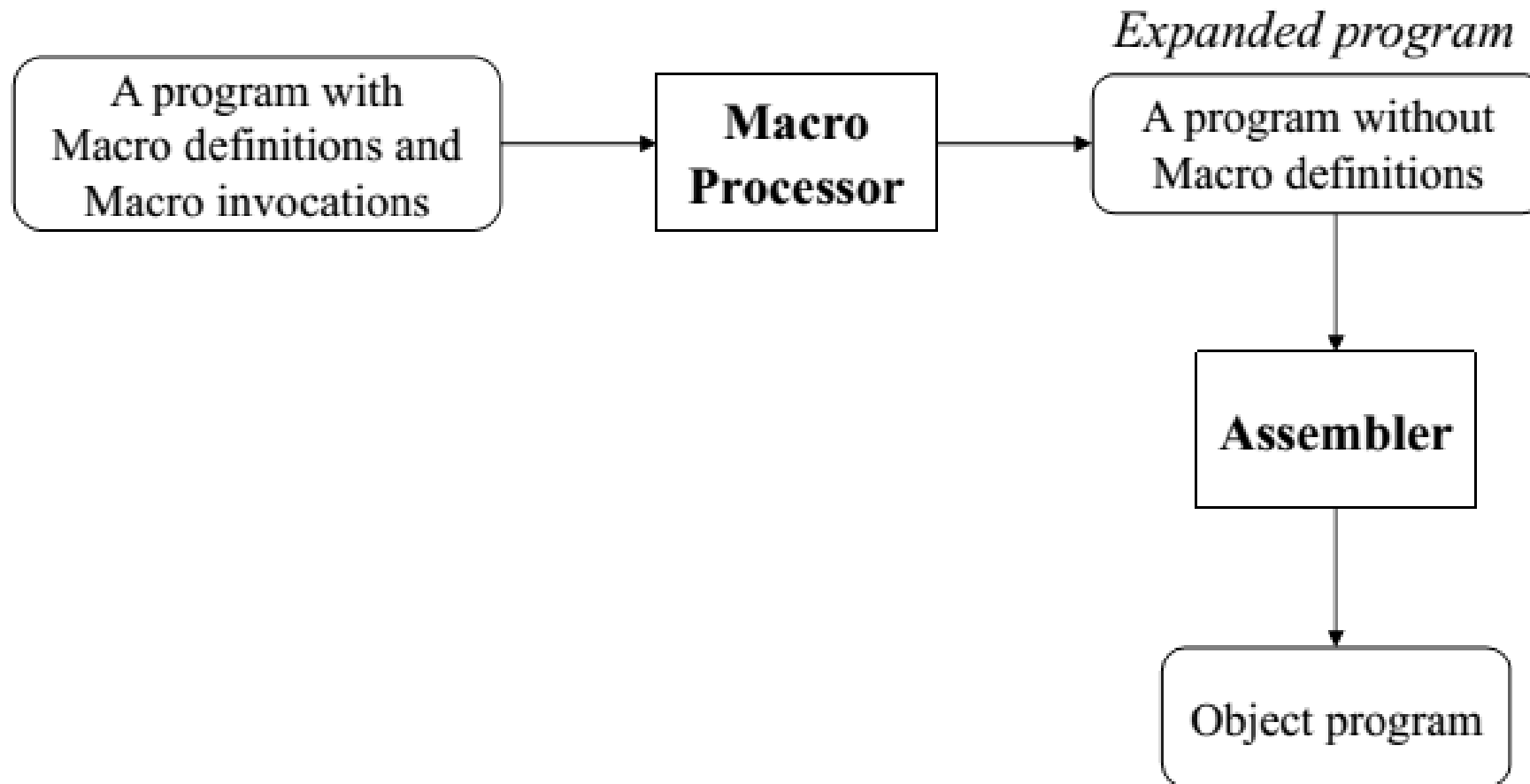
Introduction

- SIC:
 - 24 bit registers
 - A, X, L - (0,1,2)
 - PC
 - SW (Status Word) - (8,9)
- GPR:
 - B, S, T, F - (3,4,5,6)

Introduction

- For example:
- On SIC/XE (Extra Equipment), requires a sequence of seven instructions to save the contents of all registers
- 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.

Basic Macro Processor Functions



Basic Macro Processor Functions

Macro Definition:

1. Two new assembler directives
 - **MACRO**
 - **MEND**
2. A pattern or **prototype** for the macro instruction
3. Macro name and parameters

Basic Macro Processor Functions

- Macro invocation
 - Often referred to as a ***macro call***
 - Need:
 - a) name of the macro instruction being invoked
 - b) arguments to be used in expanding the macro
- Expanded program
 - No macro instruction definitions
 - Each macro invocation statement has been expanded with
 - a) Macro body
 - b) Arguments substituted with the parameters in the prototype

Macro Functionalities

- A. Copy code
- B. Parameter substitution
- C. Macro instruction defining macros
- D. Conditional macro expansion

A. Copy Code

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
.
```

B. Parameter Substitution

Source

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

Expanded source

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

B. Parameter Substitution

- Dummy arguments

- Positional argument

STRG DATA1, DATA2, DATA3

GENER ,,DIRECT,,,,,3

- Keyword argument

STRG &a3=DATA1, &a2=DATA2, &a1=DATA3

GENER TYPE=DIRECT, CHANNEL=3

Example

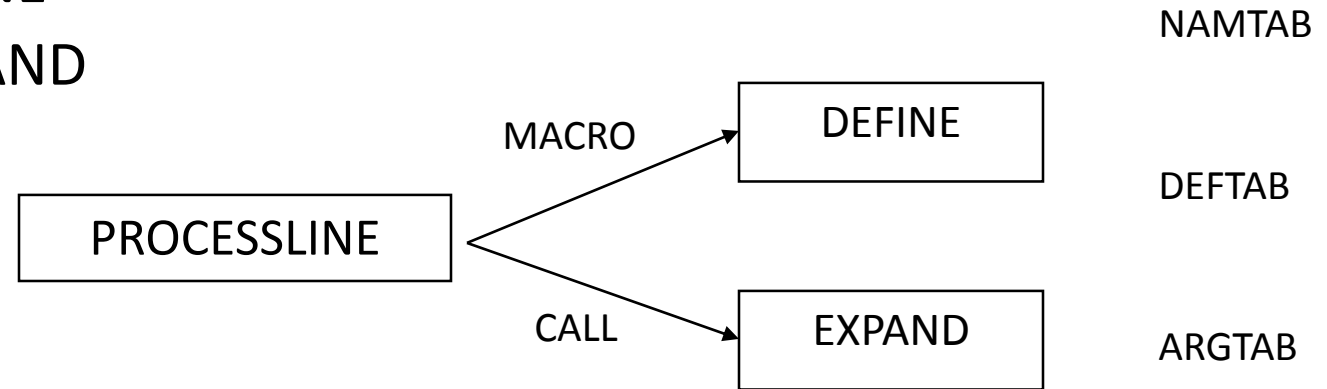
Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	RDBUFF	MACRO	&INDEV, &BUFADR, &RECLTH	
15	.			
20	.	MACRO TO READ RECORD INTO BUFFER		
25	.			
30		CLEAR	X	CLEAR LOOP COUNTER
35		CLEAR	A	
40		CLEAR	S	
45		+LDT	#4096	SET MAXIMUM RECORD LENGTH
50		TD	=X' &INDEV'	TEST INPUT DEVICE
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		

Example

```
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
```

One Pass Macro Processor *(prerequisite for C and D)*

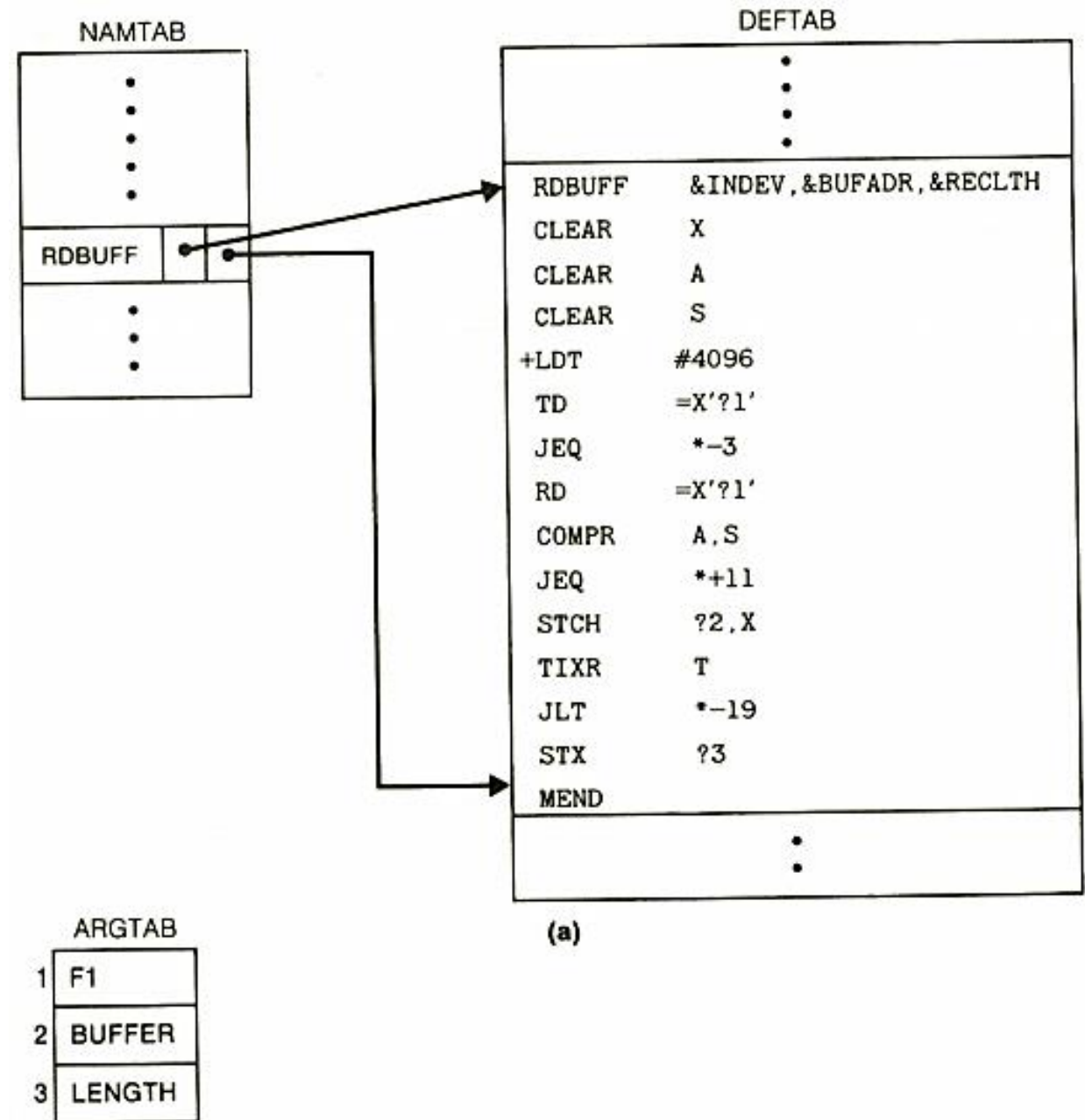
- Prerequisite
 - Every macro must be defined before it is called
- Sub-procedures
 - macro definition: DEFINE
 - macro invocation: EXPAND



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,
 - NAMTAB,
 - ARGTAB

Data Structures



Algo. for one-pass Macro expansion

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE  $\neq$  'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
    end {macro processor}

procedure PROCESSLINE
    begin
        search NAMTAB for OPCODE
        if found then
            EXPAND
        else if OPCODE = 'MACRO' then
            DEFINE
        else write source line to expanded file
    end {PROCESSLINE}
```

C. Nested Macro

```
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}
```

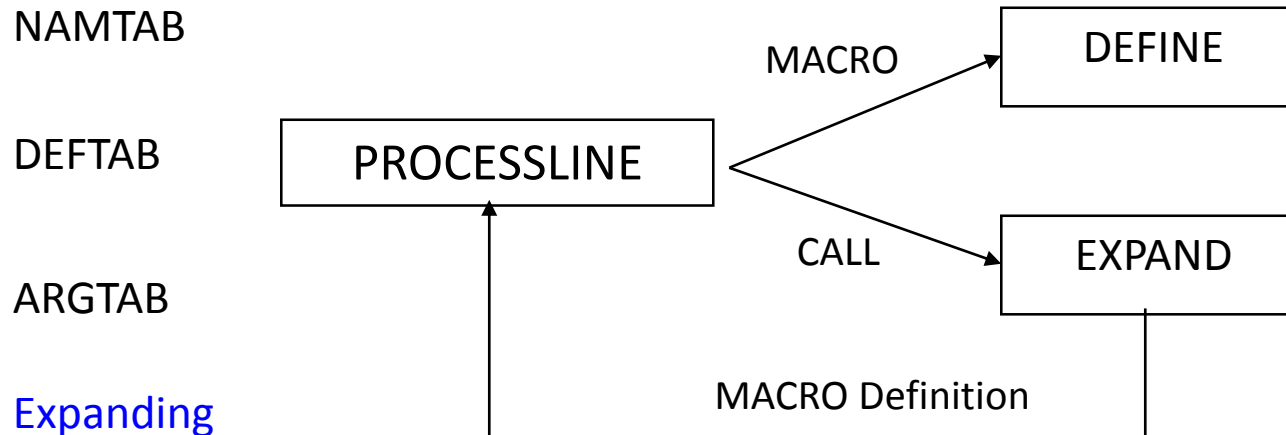
(a)

C. Nested Macro

```
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}
```

One Pass Macro Processor: *for nested macro*

- Sub-procedures
 - macro definition: DEFINE
 - macro invocation: EXPAND
- EXPAND may invoke DEFINE when encounter macro definition



Algo. for one-pass Macro expansion

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE  $\neq$  'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
    end {macro processor}

procedure PROCESSLINE
    begin
        search NAMTAB for OPCODE
        if found then
            EXPAND
        else if OPCODE = 'MACRO' then
            DEFINE
        else write source line to expanded file
    end {PROCESSLINE}
```

```

procedure DEFINE
  begin
    enter macro name into NAMTAB
    enter macro prototype into DEFTAB
    LEVEL := 1
    while LEVEL > 0 do
      begin
        GETLINE
        if this is not a comment line then
          begin
            substitute positional notation for parameters
            enter line into DEFTAB
            if OPCODE = 'MACRO' then
              LEVEL := LEVEL + 1
            else if OPCODE = 'MEND' then
              LEVEL := LEVEL - 1
            end {if not comment}
          end {while}
        store in NAMTAB pointers to beginning and end of definition
      end {DEFINE}

```

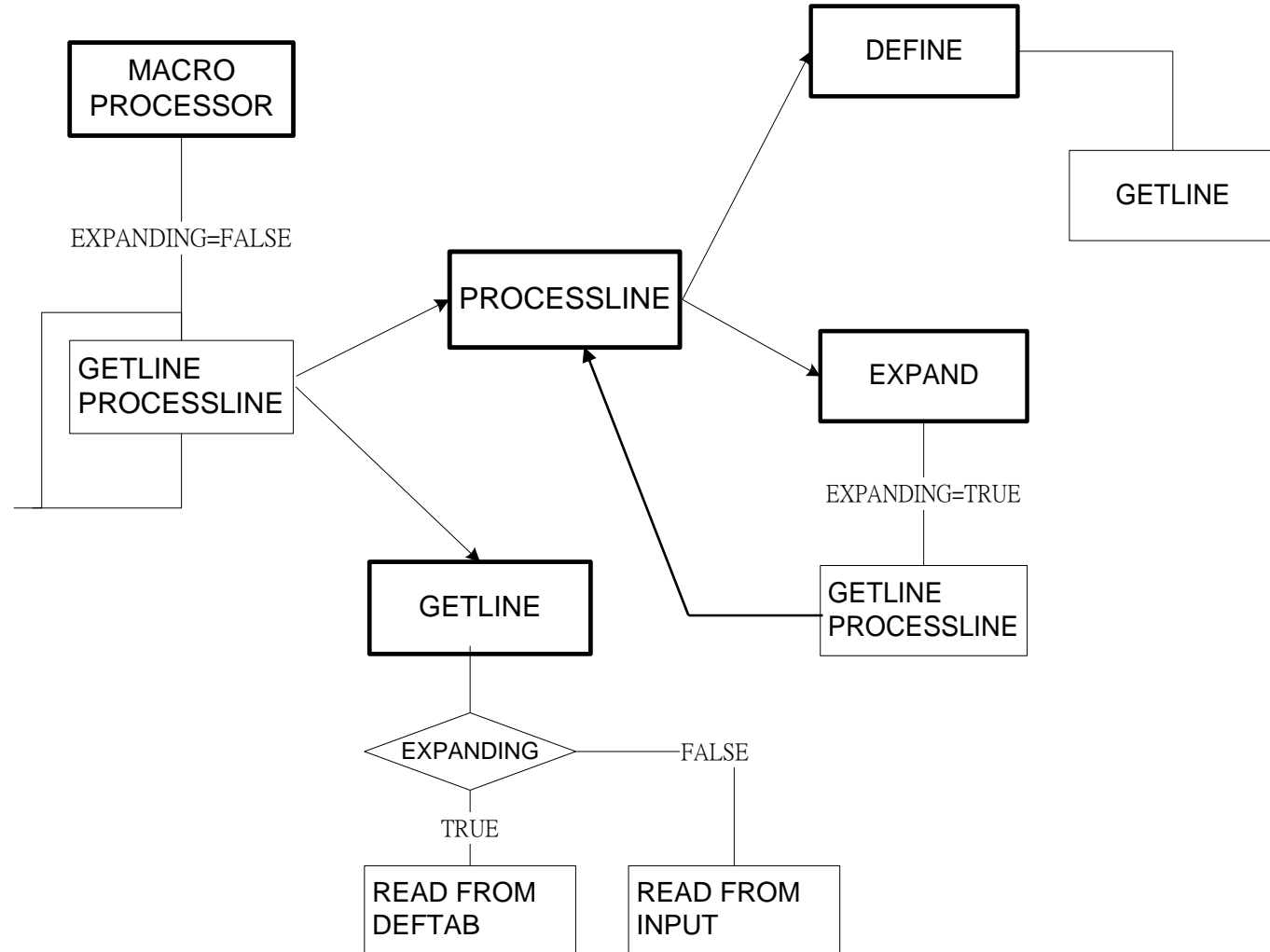
```

procedure EXPAND
  begin
    EXPANDING := TRUE
    get first line of macro definition {prototype} from DEFTAB
    set up arguments from macro invocation in ARGTAB
    write macro invocation to expanded file as a comment
    while not end of macro definition do
      begin
        GETLINE
        PROCESSLINE
      end {while}
    EXPANDING := FALSE
  end {EXPAND}

procedure GETLINE
  begin
    if EXPANDING then
      begin
        get next line of macro definition from DEFTAB
        substitute arguments from ARGTAB for positional notation
      end {if}
    else
      read next line from input file
    end {GETLINE}

```


One Pass Macro Processor



Algo.

```
procedure DEFINE
begin
    enter macro name into NAMTAB
    enter macro prototype into DEFTAB
    LEVEL := 1
    while LEVEL > 0 do
        begin
            GETLINE
            if this is not a comment line then
                begin
                    substitute positional notation for parameters
                    enter line into DEFTAB
                    if OPCODE = 'MACRO' then
                        LEVEL := LEVEL + 1
                    else if OPCODE = 'MEND' then
                        LEVEL := LEVEL - 1
                    end {if not comment}
                end {while}
            store in NAMTAB pointers to beginning and end of definition
        end {DEFINE}
```

Algo.

```
procedure EXPAND
begin
    EXPANDING := TRUE
    get first line of macro definition {prototype} from DEFTAB
    set up arguments from macro invocation in ARGTAB
    write macro invocation to expanded file as a comment
    while not end of macro definition do
        begin
            GETLINE
            PROCESSLINE
        end {while}
    EXPANDING := FALSE
end {EXPAND}

procedure GETLINE
begin
    if EXPANDING then
        begin
            get next line of macro definition from DEFTAB
            substitute arguments from ARGTAB for positional notation
        end {if}
    else
        read next line from input file
    end {GETLINE}
```

Macro Expansion Types

- **Lexical Substitution:**

- Replacement of a character string by another character string during program generation
- Replacement of Formal Parameters with Actual Parameters
- Formal Parameter => Macro name and/or parameter list
 - E.g. for the macro **STRG** **MACRO &a1, &a2, &a3**
 - The call **STRG** DATA1, DATA2, DATA3
- Actual Parameter => Macro body that replaces the formal parameters
 - E.g. After replacement, the macro body
 - **STA** **DATA1**
 - **STB** **DATA2**
 - **STX** **DATA3**

Macro Expansion Types

- **Semantic Expansion:**

- Generation of instructions tailored to the requirements of a specific usage.

- ***Characteristics:***

- Different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.

- Eg: Generation of type specific instructions for manipulation of byte and word operands.

Example

- The following sequence of instructions is used to increment the value in a memory word by a constant.
 1. Move the value (A) from the memory word into a machine register.
 2. Increment the value in the machine register.
 3. Move the new value into the memory word.
- Since the instruction sequence **MOVE-ADD-MOVE** may be used a number of times in a program, it is convenient to define a **macro** named INCR.

Example

- Using **Lexical expansion** the macro call **INCR A, B, AREG** can lead to the generation of a **MOVE-ADD-MOVE** instruction sequence
- Increments **A** by the value of **B** using **AREG** (register) to perform the arithmetic.
- Use of **Semantic expansion** can enable the instruction sequence to be adapted to the **types** of **A** and **B**.
- For example: an **INC** instruction (in Intel **8088**) could be generated if **A** is a **byte** operand and **B** has the value **"1"**.

Macro vs. Subroutine

- Use of a macro name in the mnemonic field of an assembly statement leads to its expansion,
- Whereas, use of subroutine name in a call instruction leads to its execution.
- So there is difference in
 - **Size**
 - **Execution Efficiency**
- Macros can be said to trade program size for execution efficiency.

D. Machine-Independent Macro Processor Feature

1. Concatenation of Macro Parameters
2. Generation of Unique Labels
3. Conditional Macro Expansion
4. Keyword Macro Parameters

1. Concatenation of Macro Parameters

- Most macro processors allow parameters to be concatenated with other character strings
- The need of a special concatenation operator
 - LDA **X&ID1**
 - LDA **X&ID**
- The concatenation operator
 - LDA **X&ID -> 1**

1. 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
↓	
LDA	XA1
ADD	XA2
ADD	XA3
STA	XAS

SUM	BETA
↓	
LDA	XBETA1
ADD	XBETA2
ADD	XBETA3
STA	XBETAS

2. Generation of Unique Labels

- It is, in general, **not possible** for the body of a macro instruction to contain labels of the usual kind
- Leads 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

2. Generation of Unique Labels

Solution:

- Allowing the creation of special types of labels within macro instructions
- Labels used within the macro body begin with the special character **\$**
- Programmers are instructed **no to use \$** in their source programs

2. Example

```
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
```

2. Example

```
.          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
```

3. 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
- Macro processor directive
 - IF, ELSE, ENDIF
 - SET
 - Macro-time variable (set symbol)
- WHILE-ENDW

3. Example for IF-ELSE- ENDIF

```

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

```

	.	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

```
.      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

RDBUFF F1, BUFF, RLENG, 04

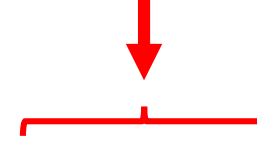
30		CLEAR	X	CLEAR LOOP COUNTER
35		CLEAR	A	
40		LDCH	=X'04'	SET EOR CHARACTER
42		RMO	A, S	
45		+LDT	#4096	SET MAX LENGTH = 4096
50	\$ACLOOP	TD	=X'F1'	TEST INPUT DEVICE
55		JEQ	\$ACLOOP	LOOP UNTIL READY
60		RD	=X'F1'	READ CHARACTER INTO REG A
65		COMPR	A, S	TEST FOR END OF RECORD
70		JEQ	\$ACEXIT	EXIT 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

3. Example for **WHILE- ENDW**

```

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

```

.	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

4. Keyword Macro Parameters

- Positional Parameters
- Keyword Parameters
 - As discussed earlier