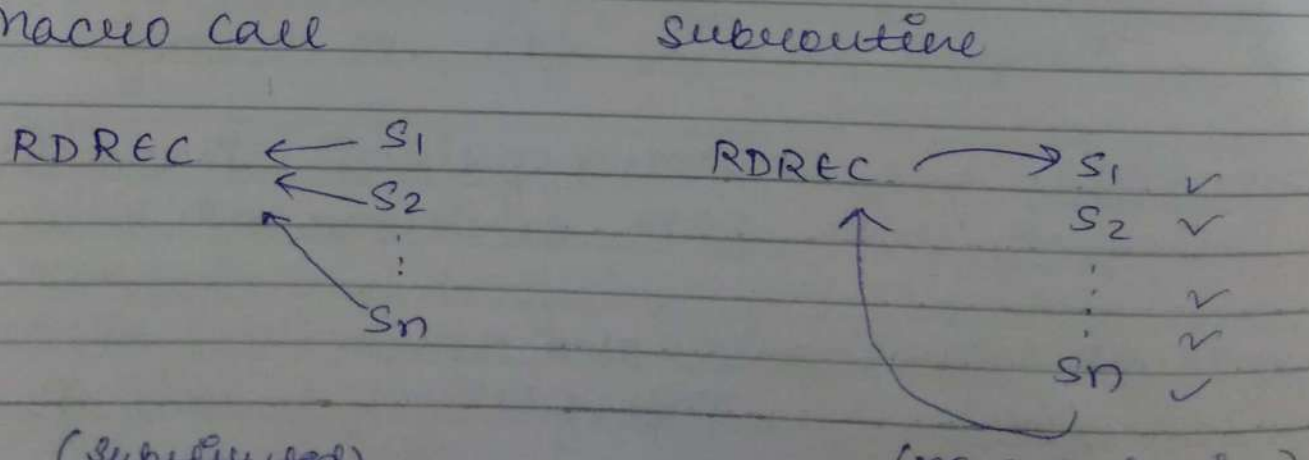
Chapter 05
MACROS

★ Contents:

- Basic Macroprocessor functions.
- Macroprocessor Algorithm & data structures
- Machine Independent Macros features.

* Video 1:

- A function can be written either in the form of subroutine or macros.
- In case of subroutine call, the control is transferred to the set of instructions & after execution control is returned back to the subroutine call.
- However, in case of macros, the macro call substitutes the macro name a set of instructions.



- In case of function call, same copy of inst is executed again & again however in case of macros, the macro call is substituted by a set of instructions thus, the size of program increases.

- So, if the modules are small we use macros else subroutines.

- In case of subroutine, time overhead is there i.e. it takes some time to switch the control however, in case of ~~sub~~ macros there is no such overhead, the instructions are substituted before execution.

*Video 2 : Working of macros

Two Assembler directives : MACRO & MEND

MACRO → beginning of macro

MEND → end of MACRO

MACRO DEFINITION

```
SUM  MACRO  &V1, &V2, &V3
     LDA    V1
     ADD    V2
     STA    V3
     MEND
```

parameters ~~arguments~~

MACRO CALL

```
S1
S2
SUM  X1, X2, X3
S4
S5
S6
```

parameters ~~arguments~~

- Line SUM X1, X2, X3 is replaced by the body of the MACRO i.e. LDA V1
  ADD V2
  STA V3

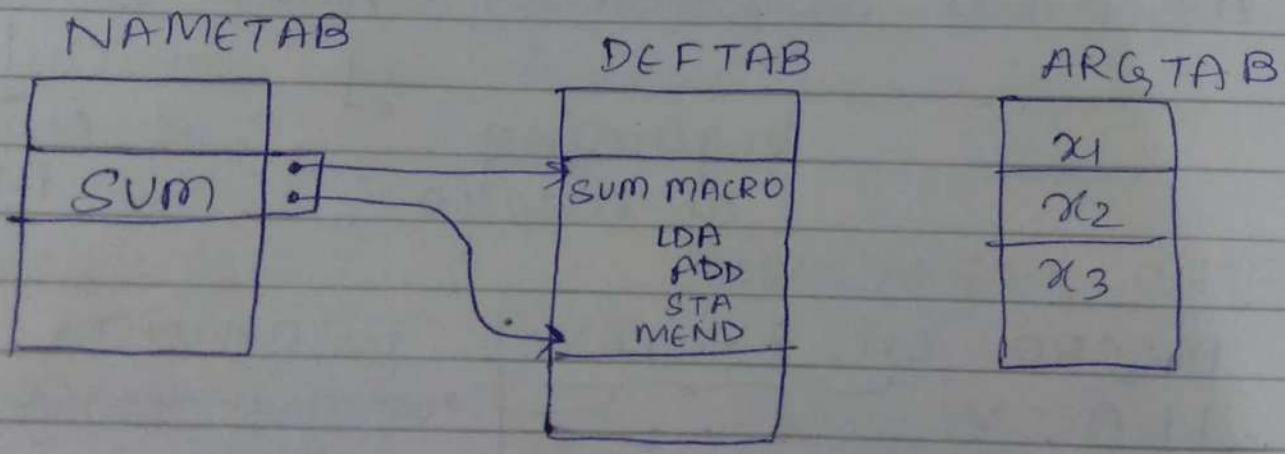
- X1 is copied in V_1
  X_2 ——— V_2
  X_3 ——— V_3

* Data Structures :-

1). NAMETAB ⇓
name of the macro with two pointers (start & end) are present.

2). DEFTAB ⇓
entire MACRO definition for all macros are present.

3). ARGTAB ⇓
used to hold the arguments or the actual parameters.



NOTES

★ MACROS :

- A macro represents a commonly used group of statements in the source program.
- Macro processor replaces each macro instruction with the corresponding group of statements.
- This is called Macro Expansion.

Syntax :

```
<macro name>  MACRO   &p1, &p2, ...., &pn
              /*
                 Body of MACRO
              */
              MEND
```

- The first MACRO statement identifies the beginning of the macro.
- &p1, &p2, . . . &pn are the set of parameters used. Each parameter starts with '&' symbol that signifies that these parameters are to be substituted during macro expansion.
- The body of the MACRO contains the statements that are to be substituted at the time of macro expansion.

- 'MEND' is the assembler directive that marks end of the macro definition.

Example of MACRO definition:

```
SUM   MACRO   &V1, &V2, &V3
      LDA     V1
      ADD     V2
      STA     V3
      MEND
```

Invocation: It can be referred to as macro call.

- It is the statement that takes the name of the macro instruction being invoked & the arguments to be passed in expanding the macro.

E.g: SUM X1, X2, X3

- V1, V2 & V3 are the arguments.
- The arguments an parameters are associated with one another ~~as per~~ according to their position.

★ MACROS DATASTRUCTURES:

There are three main data structures involved in macro processor-

a). DEFTAB b). NAMETAB c). ARGTAB

1). DEFTAB

- It stands for definition table.
- It cotains macro definitions.
- The definition has the MACRO prototype and the statements in body.
- The comments are not included in DEFTAB.

2). NAMTAB

- It stands for NAMETABLE.
- It has the macro names that act as an index to DEFTAB.
- It contains pointers to the beginning & end of the definition in DEFTAB.

3). ARGTAB

- It stands for Argument table.
- It is used during macro expansion.
- When macro invocation is found, the arguments are stored in their position in ARGTAB.
- While expansion, these arguments are substituted into the parameters according to their position.

★ MACRO PROCESSOR ALGORITHM:

- It is easy to design two pass macro processor in which first pass is used to process all macro definitions and second pass is used for macro expansion.

- However, such a pass macro processor will not allow the nested macros condition i.e. the body of one macro to contain other macros definition.

- In SIC machine, we have MACROS as the standard macro that contains all other macro definitions.

```
MACROS   MACRO
RDBUFF   MACRO
          :
         MEND    // End of RDBUFF
WRBUFF   MACRO
          :
         MEND    // End of WRBUFF
         MEND    // End of MACROS
```

- In SIC/XE machine, we have MACROX as the standard version that contains all other macro definitions.

```
MACROX  MACRO
RDBUFF  MACRO
        :
        MEND
WRBUFF  MACRO
        :
        MEND
        MEND
```

- In such nested macro definitions, the 2 pass assembler will fail because all the macros would not be able to be defined during the first pass before expansion.

- Thus, a one-pass macro processor can be helpful to handle such conditions if the macro definition is present in the source program before any statement that invoke that macro.

- In other words, macro needs to be defined before they are called in the program.

// Algorithm of one pass (from book).

* MACHINE INDEPENDENT MACROS FEATURES:

1). Concatenation of Macro Parameters

- Most of the macro processors allow the concatenation of character with different strings.
- let we have one set of parameters as XA1, XA2, XA3, ... & other set as XB1, XB2, XB3, ...
- Macro processors can make use of (A,B) to generate the symbols (XA1, XB1, ...).
- For this concatenation, '&' is used.

```
LDA   X&ID1.
```

// Here, X is concatenated with ID1. So, '&' mark the starting of the concatenation.

- To mark the end of the concatenation as well, we use '→'.

Eg:

```
LDA   X&ID→1
ADD   X&ID→2
 .
 .
 .
```

So, if ID = A

```
LDA   XA1
ADD   XA2
```

2). Generation of Unique labels:

- If we have some label in the macro definition, then every time that macro is invoked that label will be re-defined.
- This can prevent the proper functioning of the program.
- To avoid this, we can use JEQ or JLT instructions. However, these are not permanent solutions as these can be confusing.
- So, macro processors create unique labels with '$' symbol which on each iteration is followed by xx which can be two alpha-numeric characters. So, for the first time xx is AA, followed by AB, AC, and so on.

```
$label   TD    =X'&IDEV'
         :
$EXIT    STX   &RECLTH
```

↓ expansion

```
$AAlabel . . .
         :
$AAEXIT. - - -
```

4). Keyword Macro Parameters:

Macro processors also allow change in the position of the arguments passed.

For some cases, we need not pass all the arguments while invoking the macro.

There may be a condition to define the parameters in the macro prototype with some constant values.

To handle such conditions, one of the possible way could be that we can pass the parameters required & leave the rest of the parameters b/w two comma.

E.g: GENER ,,,3,,,,,,,DIRECT.

Let GENER is a macro that has 9 parameters & we need to pass only 3rd & 9th parameter.

The above representation can also be error-prone sometimes.

So, the better way could be if we assign a name to each position & then pass only those arguments w.r.t name.

E.g: GENER TYPE=DIRECT, CHANNEL=3.