MACRO AND MACRO PROCESSORS

* Macro are used to provide a program generation facility through macro expansion.
* Many programming language provide built in facilities for writing macros. E.g. Ada,C and C++.
* Higher version of processor family also provide such facility.
* “A macro is a unit of specification for program generation through expansion.
* Macro consist of name, a set of formal parameters and a body of code.
* “The use of macro name with a set of actual parameters is replaced by some code generated from its body, this is called **macro expansion**.”
* Two kind of expansion
  + Lexical expansion:
    - Lexical expansion implies replacement of character string by another character string during program generation.
    - Lexical expansion is typically employed to replace occurrences of formal parameter by corresponding actual parameters.
  + Semantic Expansion:
    - Semantic expansion implies generation of instructions tailored to the requirements of a specific usage
    - Example: generation of type specific instruction for

manipulation of byte and word operands.

* Example: the following sequence of instruction is used to increment the value in a memory word by a constant:
  + Move the value from the memory word into a machine register.
  + Increment the value in the machine register.
  + Move the new value into the memory word.
* Using lexical expansion the macro call INCR A,B,AREG can lead to the generation of a MOVE-ADD-MOVE instruction sequence to increment A by the value B using AREG.
* Macro
* INCR &MEM\_VAL, &INCR\_VAL, &REG
* MOVER &REG, &MEM\_VAL
* ADD &REG,&INCR\_VAL
* MOVEM &REG, &MEM\_VAL
* MEND
* Macro definition:
  + A macro definition is enclosed between a macro header statement and macro end statement.
  + Macro definition are typically located at the start of program .
  + Macro definition consist of
    - A macro prototype statement
    - One or more model statement
    - Macro preprocessor statement
  + A macro prototype statement
    - The macro prototype statement declares the name of a macro and the names and kinds of its parameters.
    - <macro name> [<formal parameter spec>, …]
    - Where name appears in the mnemonic field of assembly statement and <formal parameter spec> is of the form &<parameter name>[<parameter kind>]
  + Model statement
    - A model statement is a statement from which an assembly language statement may be generated during macro expansion.
  + Macro preprocessor statement
    - A preprocessor statement is used to perform auxiliary functions

during macro expansion.

* A macro is called by writing the macro name in the mnemonic field of an assembly statement.
* <macro name> [<actual parameter spec>,…]
* Where an actual parameter typically an operand specification in an assembly language statement.
* A macro call leads to macro expansion, during macro expansion, the macro call statement is replaced by a sequence of assembly statements.
* „+‟ is used to differentiate between the original statement of program and macro statement.
* Two key notions concerning macro expansion are:
  + Expansion time control flow:
    - This determines the order in which model statements are visited during macro expansion.
  + Lexical substitution:
    - Lexical substitution is used to generate an assembly

statement from a model statement.

* Flow of control during expansion
  + The default flow of control during macro expansion is sequential. its start with statement following the macro prototype statement and ending with the statement preceding the MEND statement.
  + A preprocessor statement can alter the flow of control during expansion such that some model statements are never visited during expansion is called **conditional expansion**.
  + Same statement are repeatedly visited during expansion is called **loops expansion**.
* Macro expansion is implemented using a macro expansion counter (MEC).
* Algorithm: (Outline of macro expansion)
  + *MEC=statement number of first statement following the prototype statement;*
  + *While statement pointed by MEC is not a MEND statement*
    - *(a) if a model statement then*
      * *(i) Expand the statement*
      * *(ii) MEC=MEC+1;*
    - *(b) Else (i.e. a preprocessor statement)*
      * *(i) MEC= new value specified in the statement;*
  + *Exit from macro expansion.*
* Lexical Substitution:
  + Model statement consists of 3 type of strings
    - An ordinary string, which stands for itself.
    - The name of a formal parameter which is preceded by the character „&‟.
    - The name of preprocessor variable, which is also preceded by the character „&‟.
  + During lexical expansion, string of type 1 are retained

without substitution.

* + String type 2 and 3 are replaced by the corresponding actual parameter values.
  + The value of formal parameter depends on the kind of

parameter.

* Positional parameters
* Keyword parameters
* Default specification of parameter
* Macro with mixed parameter lists
* Other uses of parameters
* Positional parameters
  + A positional formal parameter is written as &<parameter name>. The <actual parameter spec> in call on a macro using positional parameters is simply an

<ordinary string>.

* + Step-1 find the ordinal position of XYZ in the list of formal parameters in the macro prototype statement.
  + Step-2 find the actual parameter specification occupying the same ordinal position in the list of actual parameters in macro call statement.
* INCR A, B, AREG
* The rule of positional association values of the formal parameters are:
* Formal parameter value

MEM\_VAL A

INCR\_VAL B

REG AREG

Lexical expansion of model statement now leads to the code

+ MOVER AREG, A

+ ADD AREG, B

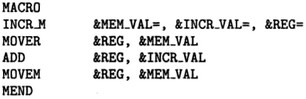
+ MOVEM AREG, A

* Keyword parameters
  + <parameter name > is an ordinary string and

<parameter kind> is the string „=„ in syntax rule.

* + The <actual parameter spec> is written as <formal parameter name>=<ordinary string>.
  + The keyword association rules:
    - Step-1 find the actual parameter specification which has the form XYZ=<ordinary string>
    - Step-2 Let <ordinary string> in the specification be the string ABC. Then the value of formal parameter XYZ is ABC.
* Example :





* Default specification of parameters
  + A default is a standard assumption in the absence of an explicit specification by programmer.
  + Default specification of parameters is useful in situations where a parameter has the same value in most calls.
  + When desired value is different from the default value, the desired value can be specified explicitly in a macro call.
* Example:

Call the macro

INCR\_D MEM\_VAL=A, INCR\_VAL=B INCR\_D INCR\_VAL=B, MEM\_VAL=A

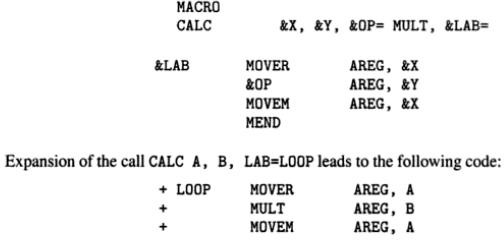
INCR\_D INCR\_VAL=B, MEM\_VAL=A, REG=BREG MARCO DIFINITION

MACRO

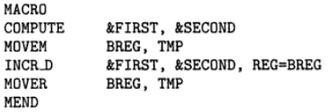
INCR\_D &MEM\_VAL=,&INCR\_VAL=,&REG=AREG MOVER &REG, &MEM\_VAL

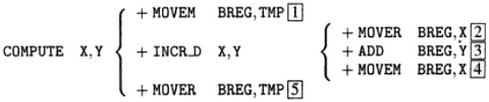
ADD &REG, &INC\_VAL MOVEM &REG, &MEM\_VAL MEND

* Macro with mixed parameter lists
  + A macro may be defined to use both positional and keyword parameters.
  + All positional parameters must precede all keyword parameters.
  + Example: SUMUP A,B,G=20,H=X
  + Where A,B are positional parameters while G,H are keyword parameters.
* Other uses of parameters
  + The model statements have used formal parameters only in operand fields.
  + Formal parameter can also appear in the label and opcode fields of model statements.



* A model statement in macro may constitute a call on another macro, such calls are known as nested macro calls.
* The macro containing the nested call is called outer macro.
* The called macro called inner macro.
* Expansion of nested macro calls follows the last-in- first-out(LIFO) rule.

**Nested Macro Call - Example**





# Advanced Macro Facilities

* Advance macro facilities are aimed at supporting semantic expansion.
  + Facilities for alteration of flow of control during expansion.
  + Expansion time variables
  + Attributes of parameters.

## Alteration of flow of control during expansion

* Alteration of flow of control during expansion:
  + Expansion time sequencing symbols (SS).
  + Expansion time statements AIF, AGO and ANOP. Sequencing symbol has syntax

**.<ordinary string>**

A SS is defined by putting it in the label field of statement in the macro body.

It is used as operand in an AIF, AGO statement for expansion control transfer.

* An AIF statement has syntax

### AIF (<expression>) <sequencing symbol>

* Where, <expression> is relational expression involving ordinary strings, formal parameters and their attributes, and expansion time variables.
* If the relational expression evaluates to true, expansion time control is transferred to the statement containing <sequencing symbol> in its label field.
* An AGO statement the syntax

### AGO <sequencing symbol>

* Unconditionally transfer expansion time control to the statement containing <sequencing symbol> in its label field.
* An ANOP statement is written as

### <sequencing symbol> ANOP

* Simply has the effect of defining the sequencing symbol.

# Expansion Time Variable (EV‟s)

* Expansion Time Variable
  + Expansion time variable are variables which can only be

used during the expansion of macro calls.

* + Local EV is created for use only during a particular macro call.
  + Global EV exists across all macro calls situated in program and can be used in any macro which has a declaration for it.

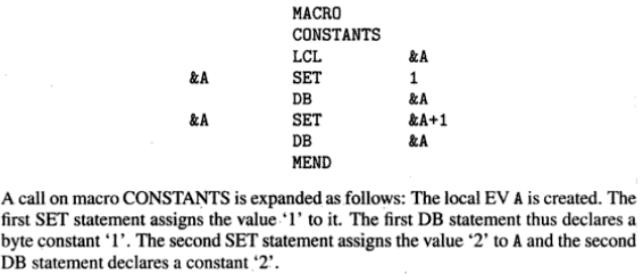
**LCL <EV specification>[,<EV specification>…] GBL <EV specification>[,<EV specification>…]**

# Cont.

* <EV specification>has syntax &<EV name>, where EV name is ordinary string.
* Initialize EV by preprocessor statement SET.

### <EV Specification> SET <SET-expression>

EV‟s Example

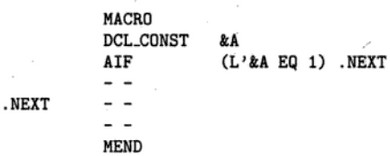


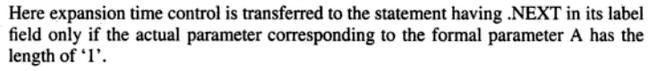
* **Attributes of formal parameters:**

<attribute name>‟ <formal parameter spec>

Represents information about the value of the formal parameter about corresponding actual parameter.

The type, length and size attributes have the name T,L and S.





### Conditional expansion:

* + Conditional expansion helps in generating assembly code specifically suited to the parameters in macro call.
  + A model statement is visited only under specific conditions during the expansion of a macro.
  + AIF and AGO statement used for this purpose.
* Example: evaluate A-B+C in AREG. MACRO

EVAL &X, &Y, &Z

AIF (&Y EQ &X) .ONLY

MOVER AREG, &X

SUB AREG, &Y

ADD AREG, &Z

AGO .OVER

.ONLY MOVER AREG, &Z

.OVER MEND

* Expansion time loop
  + To generate many similar statements during the expansion of a macro.
  + This can be achieved by similar model statements in the macro.
  + Example:

MACRO

CLEAR &A

MOVER AREG, =„0‟

MOVEM AREG, &A

MOVEM AREG, &A+1

MOVEM AREG, &A+2

MEND

* Expansion time loops can be written using expansion time variables and expansion time control transfer statement AIF and AGO.

**Example:**

MACRO

|  |  |  |
| --- | --- | --- |
|  | CLEAR | &X, &N |
| LCL | &M |
| &M | SET | 0 |
| MOVER AREG,=„0‟ | | |
| .MOVE | MOVEM | AREG, &X+&M |
| &M | SET | &M+1 |
|  | AIF  MEND | (&M **NE** N) .MORE |

* Comparison with execution time loops:
  + Most expansion time loops can be replaced by execution time loops.
  + An execution time loop leads to more compact assembly programs.
  + In execution time loop programs would execute slower than programs containing expansion time loops.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

* Example:

MACRO

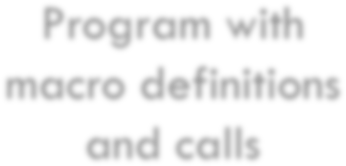
CONSTS &M, &N, &Z IRP &Z, &M=7, &N

DC „&Z‟

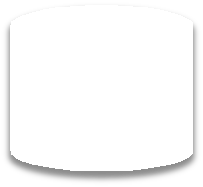
ENDM MEND

A MACRO call CONSTS 4, 10 leads to declaration of 3 constants with the values 4,7 and 10.

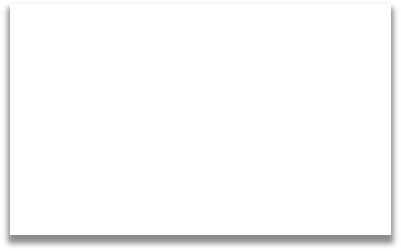
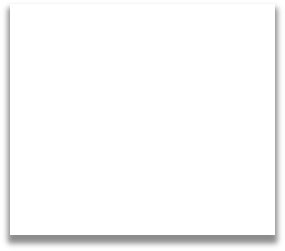
DESIGN OF A MACRO PREPROCESSOR



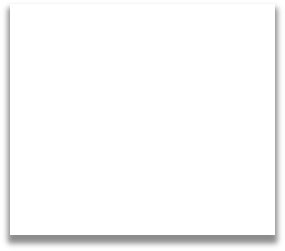
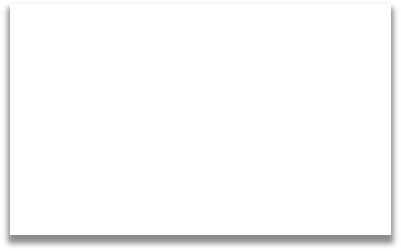
Program with macro definitions and calls



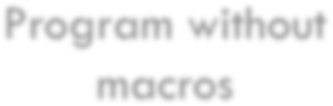
* The macro preprocessor accepts an assembly program containing definitions and calls and translates it into an assembly program which does not contain any macro definition or call.



Macro preprocessor



Assembler



Program without macros



Target program

* Design overview
  + Listing all tasks involved in macro expansion
    - Identify macro calls in the program.
    - Determine the values of formal parameters.
    - Maintain the values of expansion time variables declared in

a macro.

* + - Organize expansion time control flow.
    - Determine the values of sequencing symbols.
    - Perform expansion of a model statement.
* The following 4 step procedure is followed to arrive at a design specification for each task:
  + Identify the information necessary to perform a task.
  + Design a suitable data structure to record the information.
  + Determine the processing necessary to obtain the information.
  + Determine the processing necessary to perform the task.
* Identify macro calls:
  + A table called the macro name table (MNT) is designed to hold the

name of all macro defined in program.

* Determine values of formal parameters
  + A table called actual parameter table (APT) is designed to hold the values of formal parameters during the expansion of a macro call.
  + It contains (<formal parameter name>,<value>)
  + A table called parameter default table(PDT) is used for each macro.
  + Accessible from the MNT entry of macro.
  + It contain pairs of the form (<formal parameter name>,<default value>).
  + If macro call statement does not specify a value for some parameter

then its default value would be copied from PDT to APT.

* Maintain expansion time variables:
  + An expansion time variables table (EVT) is maintained for this purpose.
  + Table contain pairs of the form
  + (<EV name>,<value>)
  + It accessed when a preprocessor statement or model statement under expansion refers to an EV.
* Organize expansion time control flow
  + The body of macro contained set of model statements and preprocessor statement in it, is stored in a table called the macro definition table (MDT) for use during macro expansion.
  + The flow of control during macro expansion determines when a model statement is to be visited for expansion.
* Determine values of sequencing symbols:
  + A sequencing symbol table (SST) is maintained to hold this information
  + Table contains pairs of the form
  + (<sequencing symbol name>,<MDT entry#>)
  + Where <MDT entry#> is the number of the MDT entry which contains the model statement defining the sequencing symbol.
* Perform expansion of a model statement
  + Task are as follow
    - MEC points to the MDT entry containing the model statement.
    - Values of formal parameters and EV‟s are available in APT

and EVT, respectively

* + - The model statement defining a sequencing symbol can be identified from SST.
  + Expansion of a model statement is achieved by performing a lexical substitution for the parameters and EV‟s used in the model statement.
* To obtain a detailed design of the data structure it is necessary to apply the practical criteria of processing efficiency and memory requirements.
* The table APT,PDT and EVT contain pairs which are searched using the first component of the pairs as a key- the formal parameter name is used as the key to obtain its value from APT.
* The information (<formal parameter name>,<value>) in APT has been split into two tables

PNTAB- which contains formal parameter names APTAB- which contains formal parameter values PNTAB is used while processing a macro definition while APTAB is used during macro expansion.

* Similar analysis leads to splitting of EVT into EVNTAB and EVTAB and SST into SSNTAB and SSTAB.
* EV name are entered in EVNTAB while processing EV declarations.
* SS name are entered in SSNTAB while processing an SS reference or definition, whichever occur earlier.
* The positional parameter of macro appear before keyword parameters in the prototype statement.
* If macro have p positional parameter and k keyword parameters, then keyword parameters have the ordinal number p+1, p+2 …P+k
* Due to this numbering redundancies appear in PDT.
* Entry only needs to exist for parameter number p+1, P+2 …P+k.
* So, replace parameter default table(PDT) by a keyword parameter default table (KPDTAB), this table have only k entries.
* MNT has entries for all macros defined in a program, each entry contains three pointers MDTP,KPDTP and SSTP which are pointers to MDT,KPDTAB and SSNTAB for the macro respectively.
* Macro preprocessor data structure can be summarized as follows:
  + PNTAB and KPDTAB are constructed by processing the prototype statement.
  + Entries are added to EVNTAB and SSNTAB as EV declarations and SS definitions/references are encountered.
  + MDT entries are constructed while processing model

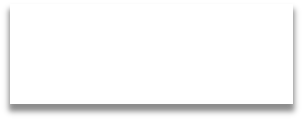
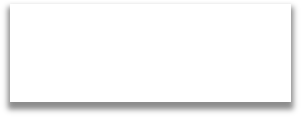
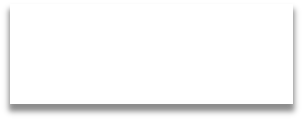
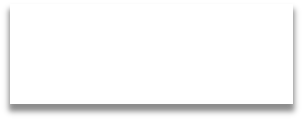
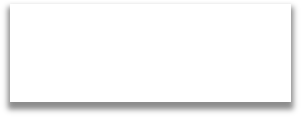
statements and preprocessor statements in macro body.

* + SSTAB entries, when the definition of sequencing symbol in encountered.
  + APTAB is constructed while processing a macro.
  + EVTAB is constructed at the start of expansion of macro.

|  |  |  |
| --- | --- | --- |
| **Table** | **Fields in each entry** |  |
| Macro name Table(MNT) | Macro name,  Number of positional parameter(#PP), Number of keyword parameter(#KP), Number of expansion time variables(#EV), MDT pointer (MDTP).  KPDTAB pointer (KPDTP). SSTAB pointer (SSTP) |  |
| Parameter Name Table(PNTAB) | Parameter name |
| EV Name Table (EVNTAB) | EV name |
| SS Name Table (SSNTAB) | SS name |
| Keyword Parameter Default  Table(KPDTAB) | Parameter name, default value |
| Macro Definition Table(MDT) | Label, Opcode, Operands |
| Actual Parameter Table(APTAB) | Value |
| EV Table (EVTAB) | Value |
| SS Table (SSTAB) | MDT entry # |

|  |  |  |
| --- | --- | --- |
|  | MACRO  CLEARMEM | &X, &N, &REG=AREG |
| LCL | &M |
| &M | SET | 0 |
|  | MOVEM | &REG, =„0‟ |
| .MORE | MOVEM | &REG, &X+&M |
| &M | SET | &M+1 |
|  | AIF | (&M NE N) .MORE |
|  | MEND |  |

#PP #KP #EV MDTP KPDTP SSTP



PNTAB

EVNTAB

SSNTAB

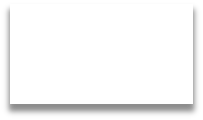
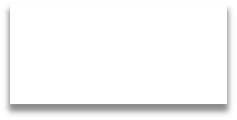
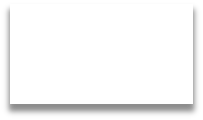
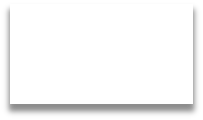
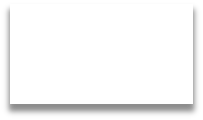
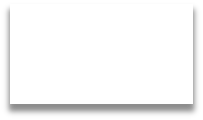
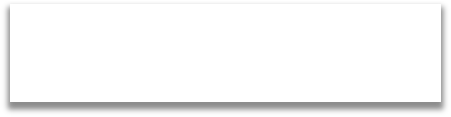
REG

N

X

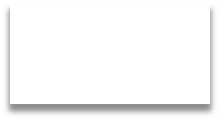
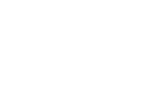
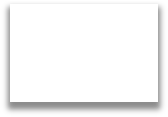
MORE

M



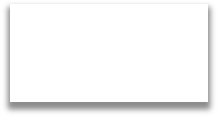
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CLEARMEM | 2 | 1 | 1 | 25 | 10 | 5 |

MNT



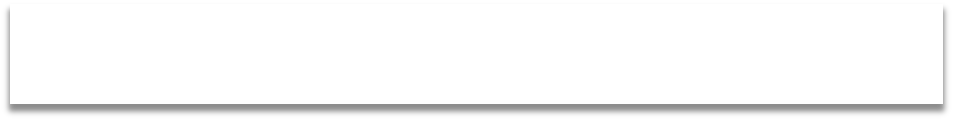
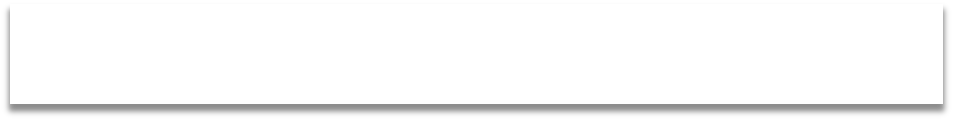
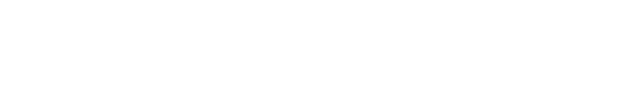
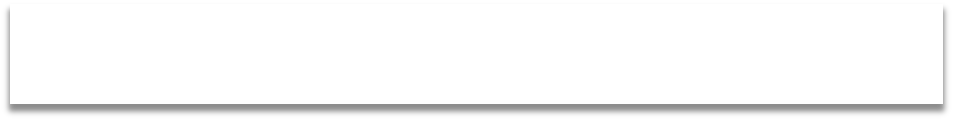
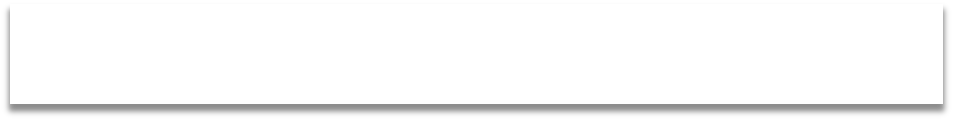
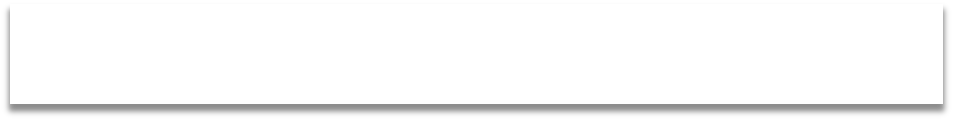
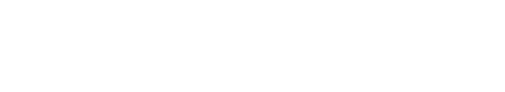
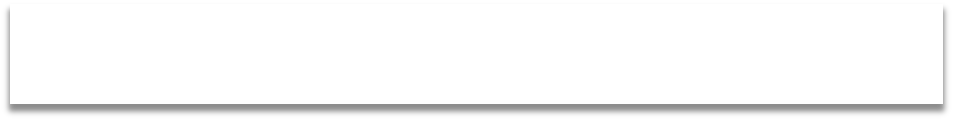
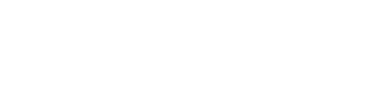
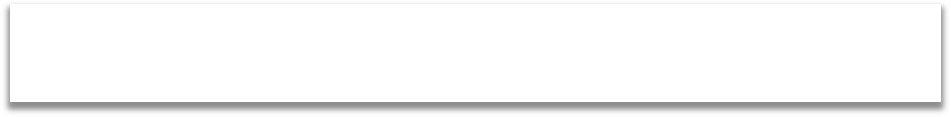
AREG

REG



28

MDT 25



MEND

((E,1) NE (P,2)) (S,1)

AIF

(E,1)+1

SET

(E,1)

MOVEM (P,3),(P,1)+(E,1)

MOVER (P,3)=„0‟

0

SET

(E,1)

(E,1)

LCL

26

27

28

29

30

31

We use the following data structure to perform macro expansion:

* + APTAB – Actual parameter table
  + EVTAB – EV table
  + MEC – Macro expansion counter
  + APTAB\_ptr – APTAB pointer
  + EVTAB\_ptr – EVTAB pointer
* Number of entries in APTAB equals to the sum of values in the #PP and #KP fields of the MNT entry of macro.
* Algorithm 5.3 (Macro Expansion)

1. Perform initialization for the expansion of a macro
   1. MEC= MDTP field of MNT entry;
   2. Create EVTAB with #EV entries and set EVTAB\_ptr.
   3. Create APTAB with #PP+#KP entries and set APTAB\_ptr.
   4. Copy keyword parameter defaults from the entries KPDTAB[KPDTP] to KPDTAB[KPDTP+#KP-1] into APTAB[#PP+1] to APTAB[#PP+#KP].
   5. Process positional parameters in the actual parameter list

and copy them into APTAB[1] to APTAB[#PP].

* 1. For keyword parameters in the actual parameter list search the keyword name in parameter name field of

KPDTAB[KPDTP] to KPDTAB[KPDTP+#KP-1]. Let KPDTAB[q]

contain a matching entry. Enter value of keyword parameter in the call (if any) in APTAB[#PP+q-KPDTP+1].

1. While statement pointed by MEC is not MEND statement
   1. if a model statement then
      1. Replace operands of the form (P,#n) and

(E,#m) by values in APTAB[n] and EVTAB[m] respectively.

* + 1. Output the generated statement.
    2. MEC=MEC+1;

1. If a SET statement with the specification (E,#m) in the label field then
   1. Evaluate the expression in the operand field and set an appropriate value in EVTAB[m].
   2. MEC=MEC+1;
2. If an AGO statement with (S,#s) in operand field then MEC=SSTAB[SSTP+s-1];
3. If an AIF statement with (S,#s) is operand field then if condition in AIF statement is true then

MEC=SSTAB[SSTP+s-1];

(3) Exit from macro expansion.

* Construct all data structure for the MACRO MACRO

BECE6 &X, &Y, &REG=BREG AIF (&Y EQ 0) .EXIT

MOVER &REG, &X

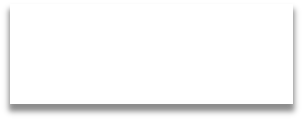
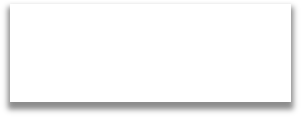
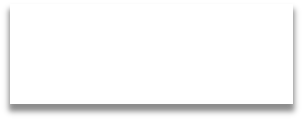
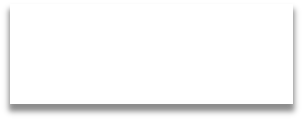
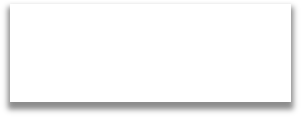
MUL &REG, &Y

.EXIT MEND

* Generate the statement for these macro calls. BECE6 6, 8, REG=AREG

BECE6 3,0

#PP #KP #EV MDTP KPDTP SSTP



PNTAB

EVNTAB

SSNTAB

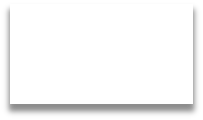
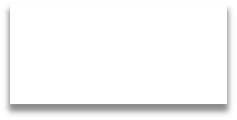
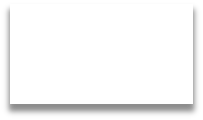
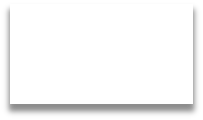
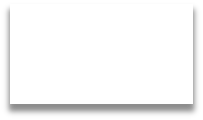
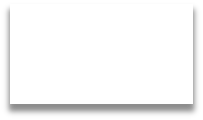
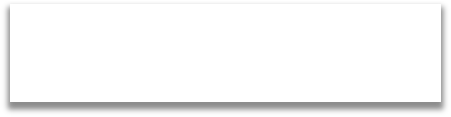
REG

Y

X

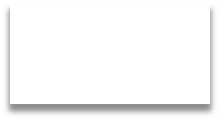
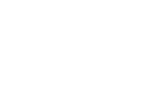
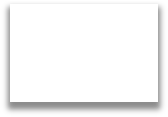
EXIT

-



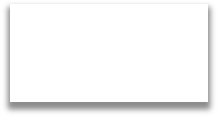
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BECE6 | 2 | 1 | 0 | 35 | 20 | 6 |

MNT



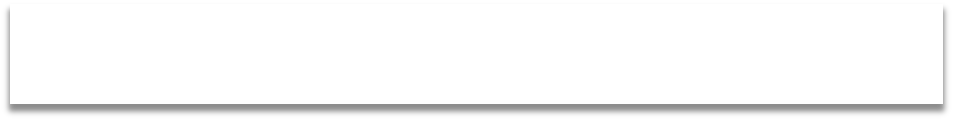
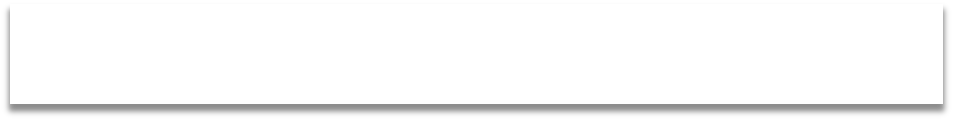
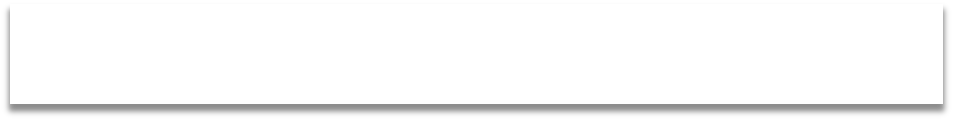
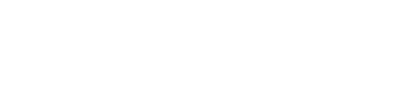
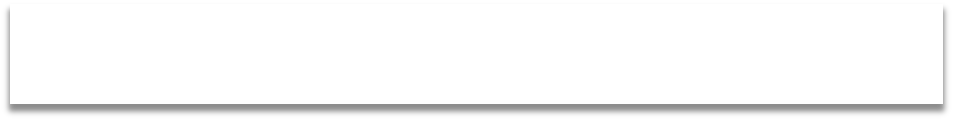
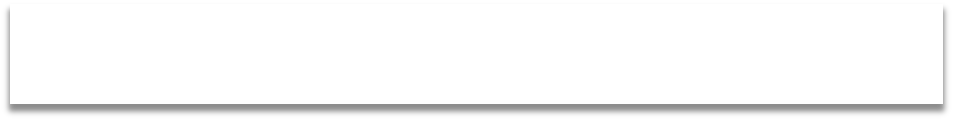
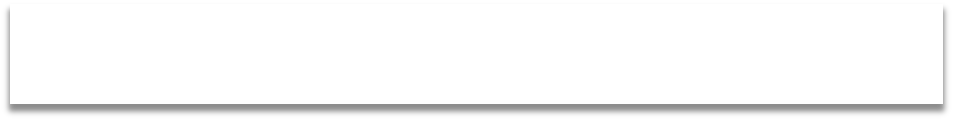
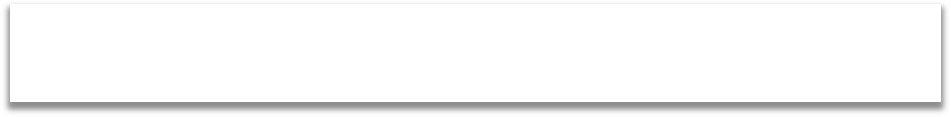
BREG

REG



35

MDT 35



(S,1)

AIF (P,2) EQ 0 .(S,1)

MOVER (P,3), (P,1)

MUL (P,3), (P,2) MEND

36

37

38

39

40

41

* Generate the statement for these macro calls. BECE6 6, 8, REG=AREG
* For first one BECE6 6, 8, REG=AREG

+ MOVER AREG, 6

+ MUL AREG, 8



COMPUTE X, Y

+ MOVEM BREG, TMP [1]

+INCR\_D X, Y

+MOVER BREG, TMP [5]

+ MOVER BREG, X [2]

+ ADD BREG, Y [3]

+ MOVEM BREG, X [4]

* Two basic alternatives exist for processing nested macro calls.
  + In this code macro calls appearing in the source program have been expanded but statements resulting from the expansion may themselves contain macro calls.
  + This first level expanded code to expand these macro calls, until we obtain a code form which dose not contain any macro calls.
  + This scheme would require a number of passes of macro expansion, which makes it quite expensive.
* Efficient alternative would be to examine each statement generated during macro expansion to see if it is itself macro call.
* A provision can be made to expand this call before continuing with the parent macro call.
* This avoid multiple passes of macro expansion.
* Two provisions are required to implement the expansion of nested macro calls:
  + Each macro under expansion must have its own set of data structures, (MEC,APTAB,EVTAB,APTAB\_ptr and EVTAB\_ptr).
  + An expansion nesting counter(Nest\_cntr) is maintained to count the number of nested macro calls. Nest\_cntr is incremented when a macro call is recognized and decremented when MEND statement is encountered.