N.V.PHILIPS Natuurkundig Laboratorium

V.O.S.
NR : MPDEF
DD : 2024-07-18
PAR : JH

MACRO-INSTRUCTIONS

1. Introduction

When programs are constructed, often the same or nearly the same sequences of instructions are written. In these cases macro-instructions can be used to reduce the number of written instructions. When macro-instructions are processed, these instructions are replaced by the corresponding sequences of machine instructions.

To replace a macro-instruction by the corresponding sequences of machine-instructions, a macro-definition is needed. The macro-definition contains the data to be generated.

In this document the syntax and semantics of macro-instructions and macro-definitions, as developed for the "TWIN", are described.

A macro-instruction is a reference to a macro-definition. Besides it may contain the values of certain variables used in the macro-definitions. Two types of variables are distinguished:

1. positional variables
   The values of positional variables are given by the character strings at the associated positions in the macro-instruction;

2. keyword variables
   The values of keyword variables are identified by the corresponding keywords in the macro-instructions. The combinations of keywords and values do not have pre-defined positions.

When both types of variables are used in one macro-instruction, the positional variables are to be given first.

Example. The following macro-instruction contains only the values of positional variables:

```
  LBL    MOVE    P,Q,3
```

The character string following the first blank in the line (i.e. 'MOVE') identifies the macro-instruction. It is similar to the operation code of a normal machine instruction. The character strings 'LBL', 'P', 'Q' and '3' represent the values of certain variables in the related macro-definition. The macro-definition corresponding to this macro-instruction can be:

```
  .MACRO
  &L    MOVE    &A,&B,&N
  &L    LODI,1  &N
        LODA,0  &A,1,-
        STRA,0  &B,1
        BRNR,1  $-6
  .MEND
```

Each macro-definition begins with the control statement '.MACRO' and it is terminated by the control statement '.MEND'. The second line in a definition is called "prototype". It specifies the format of the macro-instruction and it declares the positional variables and the keyword variables. Processing of the given macro-instruction results in the following output:

```
  LBL    LODI,1  3
         LODA,0  P,1,-
         STRA,0  Q,1
         BRNR,1  $-6
```

The control instructions and the prototype do not appear in the output and the variables are replaced by their values in the other statements.

2. Syntax of macro-definitions

The syntax of the macro-definitions is expressed in a slightly modified Backus Normal Form (BNF). Items between a pair of brackets ('<>') are non-terminals; items between a pair of rectangular parentheses ('[]') are optional. The syntax is:

```
<macrodef>          ::= <macrobegin> <macrobody> <macroend>

<macrobegin>        ::= .MACRO <r>

<macroend>          ::= .MEND <r>

<macrobody>         ::= <prototype> <data>

<prototype>         ::= [& <name>] <b> <type> [<b> <operands>] <r>

<type>              ::= <char> <type>                                max 5 char
                      ! <char>

<operands>          ::= & <name> [, <operands>]
                      ! & <name> = <text> [, <operands>]
                      ! & <name> = [, <operands>]

<data>              ::= <macrlbl> <b> <line> <r> <data>
                      ! <macrlbl> <b> <instr> <r> <data>
                      ! nill

<line>              ::= <text> <line>
                      ! & ( <expr> ) <line>
                      ! & <name> <schar> <line>
                      ! <tabchar> <line>
                      ! nill

<instr>             ::= .I[F] <expr> <macrlbl>
                      ! .J[UMP] <macrlbl>
                      ! .G[ENERATE] & <name> = <expr> , <expr>
                      ! .E[ND]
                      ! .N[OP]
                      ! .D[ECLARE] <set>
                      ! .A[SSIGN] & <name> = <expr>
                      ! .S[UBSTRING] & <name> = & <name> ( <expr> , <expr> )
                      ! .T[AB] <char>
                      ! .M[ESSAGE] <text>
                      ! .* <text>

<macrlbl>           ::= . & <name>
                      ! nill

<set>               ::= & <name> [ , <set> ]

<name>              ::= <letter> <restname>                          max 6 char.
                      ! <letter>

<restname>          ::= <char> <restname>
                      ! nill
```

```
<text>               ::= <char> <text>
                       ! <char>

<expr>               ::= <expr1> .OR. <expr>
                       ! <expr1> .XOR. <expr>
                       ! <expr1>

<expr1>              ::= <expr2> .AND. <expr1>
                       ! <expr2>

<expr3>              ::= .NOT. <expr4>
                       ! <expr4>

<expr4>              ::= <expr5> <rel> <expr4>
                       ! <expr5>

<expr5>              ::= <term> + <expr5>
                       ! <term> - <expr5>
                       ! <term>

<term>               ::= <factor> * <term>
                     ! <factor> / <term>
                       ! <factor> .SHR. <term>
                       ! <factor> .SHL. <term>
                       ! <factor>

<factor>             ::= ( <expr> )
                       ! <constant>
                       ! A ' <text> '
                       ! & <name>

<rel>                ::= .EQ.
                       ! .NE.
                       ! .GT.
                       ! .LT.
                       ! .GE.
                       ! .LE.

<constant>           ::= H ' <hexconst> '
                       ! O ' <octconst> '
                       ! B ' <binconst> '
                       ! D ' <decconst> '
                       ! <decconst>

<hexconst>           ::= <hexdigit> <hexconst>
                       ! nill

<octconst>           ::= <octdigit> <octconst>
                       ! nill

<binconst>           ::= <bindigit> <binconst>
                       ! nill

<decconst>           ::= <digit> <decconst>
                       ! nill
```

```
<schar>             ::= <tabchar>
                      ! <b>
                      ! <r>
                    ! ( ! ) ! * ! / ! + ! - ! = ! . ! , ! _

<tabchr>            ::= any character except '&'

<char>              ::= any character except '&' and <tabchr>

<hexdigit>          ::= 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 ! 8 ! 9 ! A ! B ! C ! D ! E ! F

<octdigit>          ::= 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7

<bindigit>          ::= 0 ! 1

<digit>             ::= 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 ! 8 ! 9

<r>                 ::= end of line, such as "CR"

<b>                 ::= one or more consecutive blanks
```

3. Semantics of macro-definitions

During the processing of macro-instructions the text conform to <data> in the corresponding macro-definitions is generated. Only the macro labels (syntax element <macrlbl>) and the macro control instructions (element <instr>) will not appear in the produced output. Also the expressions and variables are replaced by their (current) values. The meaning of the language elements is:

1. <name>
   This item identifies a variable. The name itslef may consist of at most 6 characters. To each variable a value is assigned. Two types of values are distinguished:

   - binary
     The value is a binary number in the range -32767 - +32767.

   - character
     The value is a character string. The string may be empty and the maximum number of characters in a string is 14. Any character may occur in the string.

2. <prototype>
   A prototype contains at least the item <type>. This is the word following the first blank. The item <type> identifies the macro-instruction. When the prototype is labeled, the macro-instruction may contain also a label. When the prototype has no label, the macro-instruction may not have a label.

   The type is (optionally) followed by a string of variables. The variables are separated by commas. When both positional variables and keyword variables are used, the positional variables are to be given first. With keyword variables, identified by '=', a default value can be specified. When the macro-definition contains no defalut value, an empty character string is used as the default value. No default value can be specified for positional variables. Their default value is, by definition, an empty character string. The default value will be used when the macro-instruction does not specify a value.

3. <data>
   The item <data> consists of a number of statements. Each statement is terminated by an end-of-line indicator, such as "EOL" (H'0D'). Two types of statements are distinguished: data lines (syntax element <line>) and control instructions (element <instr>). Each statement may be preceded by a macro-label (element <macrlbl>). The macro-label is not a part of the statement. The first character of a control instruction is '.', a data line begins with another character.

4. <line>
   The data lines contain the data which will appear in the output file. All characters in a data line, starting with the first one, are copied into the current output line, except the current tab character and '&'. The current output line is terminated when the end of the current data line is encountered.

   When the tab character is found, the following data continues at the next tab position (see section 4).

   The character '&' identifies the beginning of either a single variable or an expression. These itmes are distinguished for practical reasons, but a single variable is also a valid expression. An expression shall be enclosed by a pair of parentheses, a single variable is terminated by a special character. All special characters except

'_' are treated as normal data characters. The character '_' at the end of a single variable is used as concatenation operator, it will not appear in the output line.

5. <instr>
The control instructions can be used to jump (conditionally) to any statement in the definition, to declare additional variables, to assign other values to variables, to define the tab character and to display messages. The instructions are:

1. IF
The expression is evaluated. When the result is "true", the first statement in the definition identified by <macrlbl> is treated as the next statement; otherwise the following statement of the definition is considered as the next statement. "true" means in this context:

- binary value:
  Not 0;

- character value:
  The string contains at least one non-blank character.

2. JUMP
This is an unconditional branch-instruction. The first statement in the macro-definition identified by <macrlbl> is treated as the next statement.

3. GENERATE
This instruction forms the beginning of a "generate"-loop. This loop is terminated by the next "END"-instruction. Besides, this instruction declares a new variable together with its initial value (given by the first expression) and its final value (indicated by the second expression). Both expressions shall have a binary value or it must be possible to convert the value into a binary number.

The following statements, up to the next "END"-instruction, will be processed with increasing values of the declared variable. Each time the "END"-instruction is encountered, a new value is assigned to this variable and the statement following the "GENERATE"-instruction is considered as the next statement. The new value is the value of the first expression increased by the number of times the corresponding END-instruction has been executed. The loop is terminated when the new value exceeds the final value. Then the variable is deleted and the statement following the "END"-instruction is treated as the next statement.

A "generate"-loop is also terminated by a (conditional) jump to a statement outside the loop, but then the additional variable is not deleted. Such jumps should only be used to terminate the interpretion of a macro-definition in case of errors.

"generate"-loops cannot be nested.

4. END
This instruction forms the end of a "generate"-loop. A new value is assinged to the "generate"-variable and, when the new value does not exceed the final value, the statement following the corresponding "GENERATE"-instruction is treated as the next statement.

When the new value exceeds the final value, the "generate"-symbol and all following variables (i.e. the variables declared after the execution of the "generate"-instruction) are deleted and the subsequent statement of the definition is considered as the next statement.

5. NOP
This is a dummy instruction which causes not any operation. This instruction is useful to jump to the end of a macro-definition (the "MEND"-statement cannot be labeled).

6. DECLARE
This instruction declares a set of variables. The initial value of these variables is an empty character string. When this instruction appears inside an "generate"-loop, the variables are deleted when the "generate"-loop is terminated.

7. ASSIGN
The expression is evaluated and its value is assigned to the indicated variable.

8. SUBSTRING
This instruction is used to assign a part of a character string as the value of a variable. Both expressions must have a binary value or it must be possible to convert the value into a binary number. The first expression gives the index of the first byte in the original character string, the second expression designates the number of bytes of the new string. The length of the new string cannot exceed the length of the remainder of the original string.

Example. Let the variable &A have the value 'STRING'. The instruction:

```
  .S &B=&A(2,4)
```

assigns the value 'RING' to the variable &B.

9. TAB
This instruction defines the current tab character. This tab character is used only inside the current macro-definition until the next "TAB"-instruction is encountered. The tab character can be any character, except '&'.

10. MESSAGE
This instruction causes <text> to be displayed on "CONO". It can be used to signal errors and exceptional conditions. <text> will be followed, on the next line, by the macro-instruction.

11. *
This instruction identifies only comments and causes not any operation.

6. <expr>
Expressions may occur in control instructions as well as in data lines. Expressions are nearly identical to those used by the Assembler, only the operators '<' and '>' do not exist. An expression is constructed with operators, parentheses and factors. A factor can be a:

- constant
  Several types of constants are possible:

  H : hexadecimal constants;
  O : octal constants;
  B : binary constants;
  D : decimal constants;
  self defining constants.

All constants, except self defining constants, are enclosed by quotation marks and are preceded by the appropriate code character.

A constant is internally expressed in 16 bits and it can have a value in the range -32767 - +32767.

- character string
  A character string is enclosed by quotation marks and it is preceded by the code character 'A'. A character string may contain up to 14 characters. Any character, except a single quotation mark, may occur within a character string. A quotation mark in a character string is to be specified by two consecutive quotation marks. These two characters are interpreted as a single character and not as a string delimiter.

- variable
  A variable is identified by '&' as the first character. The value of a variable is either a character string or a constant.

- expression between parentheses.

The operators, ordered on increasing priority, are:

.OR. .XOR.	: logical OR and EXCLUSIVE OR;
.AND.	: logical AND;
.NOT.	: logical NOT (1-complement);
.EQ. .NE. .GE. .GT. .LE. .LT.	: relations;
+ -	: aritmetic addition and subtraction;
* / .SHL. .SHR.	: arithmetic multiplication, division and shifts.

The requirements and results of the various operations are:

- arithmetic operations
  The operands are to be binary numbers or it must be possible to convert the operands into binary numbers. The result is a binary number which shall be in the range -32767 - +32767.

- relations
  The operands are character strings or binary numbers which are converted into character strings. Negative numbers are preceded by a "minus"-sign and leading zeros are omitted. These character strings are compared from left to right. The result of the comparison is:

  1. EQUAL
     Both strings have the same length and contain the same characters in an identical sequence.

  2. GREATER THAN
     The number of characters of the second operand is less than the number of characters of the first operand or the binary number, which represents the current character of the first operand, is greater than the binary number that represents the current character of the second operand.

  3. LESS THAN
     The length of the second operand exceeds the length of the first operand or the binary number, corresponding to the current character of the first operand, is less than the binary number corresponding to the current character of the second

operand.

The final result of a relation operation is a binary number conform to the following matrix:

	EQ	NE	LE	LT	GE	GT
EQAL	1	0	1	0	1	0
GREATER THAN	0	1	0	0	1	1
LESS THAN	0	1	1	1	0	0

- logical operations
  The operands of logical operations are to be binary numbers. Character strings are, if applicable, converted to binary numbers.

4. Use of the macro-processor

The macro-processor can be invoked with the following system command:

```
MACRO <ifile> <oflie> <mfile>
```

The first operand specifies the input file, the second operand the output file and the last operand the macro-definition file. All operands are required.

When the macro-processor has opened the files, has read the macro definitions from <mfile> and it has not detected any failure, it displays the message:

```
** 2650 MACRO **
```

Next the macro-processor reads and processes the data in the input file. This file can contain the following types of lines:

1. control instructions
   The first character of a control instruction is '.'. The only recognized control instruction is:

   ```
   .TABS n [ , n ]
   ```

   N is a decimal number in the range 1 - 70 and it specifies a tab stop for the output lines. At most 7 tab stops are accepted. The default tab stops are 8,16,24,32,40,48,56,64.

2. comment lines
   The first character of a comment line is '*'. These lines are without any processing added to the output file.

3. data lines
   The word following the first blank is located and this word is compared with the component <type> of all prototypes of the macro-definitions. When this word is equal to a <type>, the corresponding macro-definition is interpreted; otherwise the data line is added to the output file.

When an end-of-file condition is detected in the input file the program is terminated.

The macro-processor can issue the following error-messages:

- ** ERR <xx> <id>FILE **

  An input-output error has occured. <xx> represents the SDOS SRB status; <id> the concerned file. The possible values of <id> are:

  I : input file;
  O : output file;
  M : macro-definition file.

- <c> <macrinstr>
  An error has being detected during the processing of an macro-instruction or control-instruction. <macrinstr> represents the concerned input line and <c> is the error indicator. The following error indicators exist:

C: invalid control instruction;
P: error in prototype;
D: error in macro-definition;
I: error in macro-instruction;
O: output line becomes too long.