# CALM
## **C**ommon **A**ssembly **L**anguage for **M**icroprocessors

## Macro


Patrick Fäh, La Colombière, CH-1783 Pensier, Suisse

## Contents

# 1 Introduction

Macros are treated by the macropreprocessor. The input filecontains program text and macro definitions which are to be expanded.The macropreprocessor is part of the actual CALM assembler and may beemulated by ASCALM file/R/V/C (eliminate macros).

## *1.1 Use of macros*

The assembly language programmer often finds himself repeatinggroups of instructions. By using a single instruction, called a macroinstruction, to represent each such group of assembly languageinstructions, the programmer is spared the tedious job of repetitivecoding. Moreover, the macro instructions can be thought of asstatements in a higher-level language which is particularly useful tothe programmer, because he himself has defined them.

## *1.2 Difference subroutine - macro*

A subroutine is characterized by the incorporation of only a singlecopy of its text in the complete program. The subroutine is executed,when the program statement, which calls the subroutine, is executed.Parameters passed to the subroutine tailor its function within limitsestablished when the subroutine was constructed. Subroutines reducethe overall size of a program, as well as the effort of writing it, bypermitting a call statement to serve as an abbreviation for an entiresubroutine.

The savings in space and effort cost time, for calling thesubroutine, for passing and accessing parameters, and for returningfrom the subroutine. The macro or "inline code" offers a fasteralternative at the cost of space and effort. A copy of the macroinstruction is written in the program instead of a call of asubroutine. Modifications are performed not at execution time butrather at programming time, allowing any function as desired by theprogrammer. Multiple copies of specialized program text are thusincorporated in the program, rather than a single copy of generalizedtext.

# 2 Command line

Please refer to the user manual of the CALM assembler.

# 3 Macro definition

The general structure of a macro definition is:

macro header define macro name and default parameters
macro body consists of skeleton source statements
macro terminator

There exist five pseudo-instructions for macros.

## *3.1 .MACRO*

.MACRO macroname.extension,parameter1,parameter2,...,parameter8

This pseudo-instruction starts a macro definition. The name of thatmacro is macroname. The allowed signs in macroname are the same as forlabels in the CALM assembler. It is not possible to define anothermacro within a macro definition. The significant length for themacroname is actually 32 characters.

Extension is any string terminated by a comma, which defines thedata specifier. Extension may be used in the macro body by thereserved symbol %0. Extension is the default value. It will bereplaced for %0, if no extension has been present in a correspondingmacro call. The dot is not included in %0. All characters are allowedexcept, commas, spaces, tabulators, semicolons and carriage returns.

Parameter1 to Parameter8 are the default parameters for that macroand are accessed in the macro body by %1 to %8. They are replaced forthe %1 to %8 if no parameters have been present in a correspondingmacro call. A comma separates the parameters. Spaces and tabulatorsbefore and after a parameter are ignored. But spaces and tabulatorswithin a parameter are accepted.

Example .MACRO TEST," ",D0

When a parameter contains commas, the parameter must be enclosedwithin angled brackets.
Example: MACRO TEST,<{R0}+DEPLACEMENT,R1>,<R0,R2>

To pass the left angled bracket as the first character, one mustuse <<string>. To pass the right angled bracket in such a parameter,it must be repeated twice (<string>>string>).

Parameter length is only limited by the length of the source line.

## *3.2 .LOCALMACRO*

.LOCALMACRO label1,...,label8

This pseudo-instruction defines up to 8 labels, which may be usedin the macro body. They are replaced by local labels of the typeM_0$..M_999$, when this macro is called (expanded). This avoidsmultiple defined labels when calling the macro more than once. Theallowed syntax for the labels is the same as for the labels in theCALM assembler. A comma separates the labels. Spaces and tabulatorsbefore and after the labels are ignored. This pseudo-instruction, ifused, must directly follow .MACRO. The significant length for thelabels is actually 32 characters.

## *3.3 .ENDMACRO*

.ENDMACRO

This pseudo-instruction terminates a macro definition.

### *3.4 .LAYOUTMACRO*

.LAYOUTMACRO COMMENT,COMPRESS,ERROR,LIST,REPLACE TRUE

This pseudoinstruction is valid anywhere in the program. Possibleparameters:

COMMENT saves the whole macro body (comments, empty lines, etc.).In the macro expansion all this information would appear. Default: nocomment: ignore empty lines and don't copy comments.

COMPRESS: the input text is compressed to minimize the size of theoutput file. All comments, empty lines and leading spaces areeliminated. Multiple separators are reduced to a single one. Noassembler source information is lost. Only the legibility of theoutput file is reduced. Default: do not compress input text.

ERROR copie les erreurs aussi dans le fichier de sortie. Par défaut: les erreurs ne sont pas copiées dans le fichier de sortie.

LIST includes the macro name with the parameters in the output fileat the moment of a macro call.

Example:

; MACRO TEST {A0}+10,{A0+}
... here follows the expanded macro body
Default: do not insert the original macro call.

REPLACE: a macro call, which recalls itself (directly or via othermacros), is recursive and leads to an error. With the command "REPLACETRUE" the recalled macro is not expanded but simply replaced by themacro call itself (source line is unchanged). Refer to the chapter"Using macros for assembly source translation" for more information.Default: REPLACE FALSE: gives an error if recursive macro call.

### *3.5 .EXITMACRO*

.EXITMACRO

This pseudo-instruction terminates the macro expansion. .EXITMACROhas the same effect as .ENDMACRO. The .IF/.ELSE/.ENDIFpseudo-instruction structures are resolved correctly.

### *3.6 Macro body*

The macro body contains all instructions to be expanded whencalling to that macro. Comments and empty lines are not taken perdefault (see .LAYOUTMACRO).

Example:

```
Macro definition:
        .MACRO        LDIR.8,D0,D0,#ERREUR#
        .LOCALMACRO   BOUCLE
LOOP:
        MOVE.%0       %1,%2
        DJ.16,NMO     %3,BOUCLE
        .ENDMACRO
with the macro call:
        LDIR.16       {A0+},{A1+},D0
will generate:
M_0$:
        MOVE.16       {A0+},{A1+}
        DJ.16,NMO     D0,M_0$
```

The data specifier (extension) is accessed with %0 and theparameters with %1 to %8. When %0 to %8 are enclosed by double quotesthey designate the values of the parameters at macro call time. Thedefault parameters are not included.

%0 replaced by the data specifier; specifier given in the macrocall or, if not present, by the default extension (if any).

"%0" replaced by the data specifier given in the macro call.

%1..%8 replaced by the nth parameter; parameter given in the macrocall or, if not present, by the default parameter (if any).

"%1".."%8" replaced by the nth parameter given in the macro call.

%MC is the macro counter and is replaced in a macro expansion to anumber, which represents the number of calls to that macro (1..999).

%ML is the current macro level at which the called macro is. Thisnumber may be used in a .IF pseudo-instruction to test, if this macrohas been called at the macro call level one. This is useful inassembly source translation.

### *3.7 Macro call*

The macro call statement, as already shown in the previousexamples, is made by three fields: the macro name, the extension andthe substitutable parameters. Each parameter corresponds one-to-onewith the parameter names used in the macro body: %1..%8. A parametercan be declared empty when calling a macro. A parameter inside themacro is empty when no parameter was specified at calling time and noparameter was defined as per default. No character will be substitutedin the generated statements for an empty parameter.

A macro may call another macro, but not itself (exception: see.LAYOUTMACRO). Up to 10 nested levels are possible.

The macro definition must always precede a call to that macro.

# 4 Macro extensions

The defined macro possibilities are the usual minimal macroelements. There exist additional macro functions. They allow toanalyse the passed parameters and to generate conditional assemblywithin macros.

## *4.1 Analyse de paramètre*

The following functions are only recognized within the macro bodyand are resolved at the moment of a call to that macro (macroexpansion). Abbreviations used:

| | |
|---|---|
| %Vj | global variables, set by .VARMACRO 0..9 {,Text} |
| par | %i ou "%i" ou %Vj 0 <= i <= 8, 0 <= j <= 9 |
| exp | numbers or LENGTH() operators: plus (+) and minus (-) |
| parcopy | par ou COPY() |
| parpos | parcopy or symbol name |

Functions:

| | |
|---|---|
| LENGTH(par) | returns the length of the parameter. |
| COPY(par,exp1,exp2) | copy from par a new parameter starting at theposition exp1 and with the length exp2. |
| DIGIT(parcopy) | returns TRUE if all characters in parcopy are digits["0".."9"] else FALSE. |
| LETTER(parcopy) | returns TRUE if all characters in parcopy areletters ["A".."Z","a"..."z"] else FALSE. |
| HEX(parcopy) | returns TRUE if all characters in parcopy are hexnumbers ["0".."9","A".."F","a"..."f"] else FALSE. |
| EMPTY(parcopy) | returns TRUE if the length of the parameter is zero.Hence one can test the presence of parameters at calling time. |
| UPCASE(parcopy) | converts parcopy to upper case letters. |
| POS(parpos1,parpos2) | returns position of parpos1 in parpos2 (0..) |

### *4.2 Conditional assembly in macros*

The additional functions become very powerful when combined withconditional assembly. In addition, a simple string comparison isimplemented, using the equal sign (=) within a .IF pseudo-instruction.

Examples: Test if a parameter is present at macro call time (68000):

```
(68000):   .MACRO       LDIR
           .LOCALMACRO  BOUCLE
   LOOP:
           MOVE.%0      %1,%2
           .IF          EMPTY("%3")
           *** Error: No counter!
           .ELSE
           DJ.16,NMO    %3,BOUCLE
           .ENDIF
           .ENDMACRO

   MACROGeneral macro testing if an operand is zero (68000):
           .MACRO       ZERO
           .IF          LENGTH("%1")=2
           .IF          COPY("%1",1,1)DIGIT(COPY("%1",2,1))=ATRUE
           COMP.%0      #0,%1       ; address register
           .ELSE
           TEST.%0      %1
           .ENDIF
           .ELSE
           TEST.%0      %1
           .ENDIF
           .ENDMACRO
```

## 5 Assembler pseudo-instructions which are interpreted

The CALM assembler interprets all pseudo-instructions. With theoptions /C/R/V one may generate an output file where all macropseudo-instructions and .IF/.ELSE/.ENDIF pseudo-instructions will beeliminated.

## 6 Macro examples

The use of macros is illustrated by the following examples.

a) urpose: insert a subroutine in a program only when used

```
Macro definition:
        .MACRO          SP_NAME
        .IF             %MC=1
        JUMP            SPNAMEE
SPNAME:
        ... here is the normal subroutine
        RET
SPNAMEE:
        .ENDIF
        CALL            SPNAME
        .ENDMACRO

First macro call:
        SP_NAME
will generate:
        JUMP            SPNAMEE
SPNAME:
        ... here is the normal subroutine
        RET
SPNAMEE:
        CALL            SPNAME
Subsequent macro calls will generate:
        CALL            SPNAME
```

b) Purpose: general move string macro, which takes into account thecounter length (68000). Optimise if the counter is a data register and.16. Define also a default one (like LDIR of Z80).

**Macro definition:**

```
     .MACRO       MOVESTRING.8,{A0+},{A1+},D0.32
     .LOCALMACRO  LOOP
     .IF          COPY(%3,LENGTH(%3)-1,2)=16
      .IF         DIGIT(COPY(%3,2,1))COPY(%3,1,1)LENGTH(%3)=TRUED5
     DEC.16       COPY(%3,1,2)
LOOP:
     MOVE.%0      %1,%2
     DJ.16,NMO    COPY(%3,1,2),BOUCLE
      .ELSE
LOOP:
     MOVE.%0      %1,%2
     DEC.16       COPY(%3,1,LENGTH(%3)-3)
     JUMP,NE      LOOP
      .ENDIF
     .ELSE
      .IF         COPY(%3,LENGTH(%3)-1,2)=32
LOOP:
     MOVE.%0      %1,%2
     DEC.32       COPY(%3,1,LENGTH(%3)-3)
     JUMP,NE      LOOP
      .ELSE
       .IF        COPY(%3,LENGTH(%3),1)=8
LOOP:
     MOVE.%0      %1,%2
     DEC.8        COPY(%3,1,LENGTH(%3)-2)
     JUMP,NE      LOOP
       .ENDIF
      .ENDIF
     .ENDIF
         .ENDMACRO
```

**The macro call:**

```
     MOVESTRING
```

**génère:**

```
M_0$:
     MOVE.8       {A0+},{A1+}
     DEC.32       D0
     JUMP,NE      M_0$
```

**The macro call:**

```
     MOVESTRING.16 {A4+},ADWINCH,{A6}+LEN.8
```

**will generate:**

```
M_1$:
     MOVE.16      {A4+},ADWINCH
     DEC.8        {A6}+LEN
     JUMP,NE      M_1$
```

**The macro call:**

```
     MOVESTRING.32 #0,{A3+},D2.16
```

**will generate:**

```
     DEC.16       D2
M_2$:
     MOVE.32      #0,{A3+}
     DJ.16,NMO    D2,M_2$
```

# 7 Using macros for assembly source translation

Very often the user wants to extend the basic instruction set byadding some instructions with additional operand possibilities. Thusthe operation code itself does not change. To avoid recursive macrocalls, the pseudo- instruction .LAYOUTMACRO with the operand REPLACEis used.

Example: Add to the instruction set of the Z80 the instructions:

```
        MOVE        DE,HL
        MOVE        BC,HL
```

**Macro definition:**

```
        .LAYOUTMACRO REPLACE TRUE ; remplace l'appel de macro recursif
        .MACRO       MOVE
        .IF          %1%2=DEHL
        MOVE         D,H
        MOVE         E,L
        .ELSE
         .IF         %1%2=BCHL
        MOVE         B,H
        MOVE         C,L
         .ELSE
        MOVE         %1,%2
         .ENDIF
        .ENDIF
        .ENDMACRO
        .LAYOUTMACRO REPLACE FALSE
```

**The macro call:**

```
        MOVE          {HL},A
```

**will generate:**

```
        MOVE          {HL},A
```

**The macro call:**

```
        MOVE          DE,HL
```

**will generate:**

```
        MOVE          D,H
        MOVE          E,L
```

The pseudo-instruction .LAYOUTMACRO with the REPLACE parameter setto TRUE will avoid a recursive macro call (or an error) when findingthe operation code MOVE again.

Another application is the translation from the manufacturer'snotation to the CALM notation. This is possible, when the generalstructure (labels, instructions, comment field, etc.) is the same asin CALM.

End of doc