CALM
Common **A**ssembly **L**anguage for **M**icroprocessors

Macro

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

Table des matières

1 Introduction

Les macros sont traitées par le macropréprocesseur. Le fichier source contient un texte de programme et des macros qu'il faut emplacer. Le macropréprocesseur fait partie de l'assembleur CALM actuel et peut être émulé par ASCALM fichier/R/V/C (éliminer macros).

1.1 Utilisation de macros

Le programmeur d'assembleur rencontre souvent des groupes d'instructions répétitifs. En utilisant une instruction unique, appelée instruction de macro, plusieurs instructions d'assembleur sont intégrées et le programmeur est libéré de répétitions embarrassantes. En outre, on peut comprendre une instruction de macro comme instruction d'un langage de haut niveau. Ceci est surtout intéressant pour le programmeur puisque lui-même a conçu ce langage.

1.2 Différence sous-programme - macro

Un sous-programme est caractérisé par le fait qu'il n'y a qu'une copie du texte de sous-programme dans le programme. Le sous-programme est exécuté si l'instruction de programme qui appelle le sous-programme, est exécutée. Les paramètres transmis au sous-programme déterminent sa fonction dans les limites établies au développement du sous-programme. Des sous-programmes diminuent la longueur totale d'un programme ainsi que l'effort pour les écrire puisqu'un appel remplace un sous-programme entier.

Cependant, les économies en mémoire et d'effort coûtent du temps, ar il faut appeler le sous-programme, passer les paramètres, accèder aux paramètres et quitter le sous-programme. La macro ou "le code de ligne" offre une solution plus rapide aux frais de la place mémoire et de l'effort. Une copie d'une instruction de macro au lieu d'un appel de sous-programme est placé dans le programme. Des modifications ne se font pas au moment de l'exécution mais lors de la programmation et permettent toutes les fonctions désirées par le programmeur. Ainsi on trouve plusieurs copies de textes spécialisés dans le programme au lieu d'une seule copie d'un texte général.

2 Ligne de commande

Veuillez consulter le manuel d'utilisation de l'assembleur CALM.

3 Définition d'une macro

La structure générale d'une macro a la forme suivante:

début	définit le nom de la macro, les paramètres prédéfinis
partie principale	définit le corps de la macro
fin	

Il y a cinq pseudo-instructions pour les macros.

3.1 .MACRO

.MACRO nom_de_la_macro.extension,paramètre1,paramètre2,..., paramètre8

Cette pseudo-instruction commence la définition d'une macro. Le nom de cette macro est nom_de_la_macro. Les caractères admis dans nom_de_la_macro sont les mêmes que pour les étiquettes dans l'assembleur CALM. Actuellement, 32 caractères sont significatifs dans

nom_de_la_macro. Une autre macro ne peut pas être définie à l'intérieur de la définition d'une macro.

Extension est une chaîne de caractères quelconque, terminée par une virgule, qui définit l'indicateur de données. On peut appeler l'extension dans la partie principale d'une macro par le symbole réservé %0. Extension est la valeur prédéfinie. Celle-ci est remplacée par %0 si l'on n'a pas spécifié une extension lors de l'appel à la macro correspondante. Le point ne fait pas partie de %0. Tous les caractères sont permis sauf virgule, espace, tabulateur, point-virgule et fin de ligne (<CR>).

Paramètre1 à paramètre8 sont les paramètres prédéfinis de cette macro et sont appelés dans la partie principale de la macro par les symboles %1 à %8. Ils sont remplacés par les %1 à %8 si le paramètre correspondant n'a pas été spécifié lors de l'appel à cette macro. Les espaces et les tabulateurs sont permis à l'intérieur d'un paramètre.

Exemple: .MACRO TEST," ",D0

Si un paramètre contient des virgules il faut placer le paramètre entre parenthèses pointues:

Exemple: MACRO TEST,<{R0}+DEPLACEMENT,R1>,<R0,R2>

Pour passer la parenthèse pointue gauche comme premier caractère, il faut utiliser <<chaîne de caractères>. Pour passer la parenthèse droite dans une telle chaîne, il faut la répéter deux fois (<chaîne de >> caractères).

La longueur d'un paramètre n'est limitée que par la longueur de la ligne de source.

3.2 .LOCALMACRO

.LOCALMACRO étiquette1,...,étiquette8

Cette pseudo-instruction définit jusqu'à huit étiquettes qui peuvent être utilisées dans la partie principale d'une macro. Ces étiquettes seront converties en étiquettes locales (M_0$..M_999$) si cette macro est appelée. Ceci évite des étiquettes à double si cette macro est appelée plusieurs fois. La syntaxe permise pour les étiquettes est la même que pour les étiquettes dans l'assembleur CALM. Une virgule sépare les étiquettes. Des espaces et des tabulateurs avant et après les étiquettes sont ignorés. Si cette pseudo-instruction est utilisée, elle doit directement suivre .MACRO. 32 caractères sont actuellement significatifs dans les noms des étiquettes.

3.3 .ENDMACRO

.ENDMACRO

Cette pseudo-instruction termine la définition d'une macro.

3.4 .LAYOUTMACRO

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

Cette pseudo-instruction est valable partout dans le programme. Les paramètres possibles:

COMMENT mémorise toute la partie principale d'une macro (commentaires, lignes vides, etc.). Ces informations apparaissent aussi dans les macros remplacées. Par défaut: pas de commentaires: les lignes vides et les commentaires sont ignorés.

COMPRESS: le texte d'entrée est comprimé pour minimiser la longueur du fichier de sortie. Tous les commentaires, les lignes vides et les espaces au début d'une ligne sont éliminés. Des espaces successifs sont réduits à un. Aucune information d'assembleur n'est perdue. Seulement la lisibilité est entravée. Par défaut: texte pas comprimé.

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

LIST copie le nom de la macro et les paramètres lors de l'appel de macro dans le fichier de sortie.

Exemple:

; MACRO TEST {A0}+10,{A0+}
... ici suit la partie principale de la macro
Par défaut: l'appel de la macro n'est pas copié.

REPLACE: un appel de macro qui s'appelle lui-même (directement ou par d'autres macros) est recursif et conduit à une erreur. Avec la commande "REPLACE TRUE" la macro de nouveau appelée ne sera pas interprétée mais simplement remplacée par l'appel de macro lui-même (la ligne d'entrée reste inchangée). Vous trouverez des informations plus détaillées dans le chapitre "traduction de texte d'assembleur avec des macros". Par défaut: REPLACE FALSE: produit une erreur s'il y a un appel de macro recursif.

3.5 .EXITMACRO

.EXITMACRO

Cette pseudo-instruction termine le remplacement d'une macro. L'effet correspond à .ENDMACRO. Les structures des pseudo-instructions IF/.ELSE/.ENDIF sont terminées correctement.

3.6 Partie principale d'une macro

La partie principale d'une macro contient toutes les instructions qui sont à remplacer lors d'un appel de macro. Des commentaires ou des lignes vides ne sont normalement pas mémorisés (voir .LAYOUTMACRO).

Exemple:

```
Macro definition:
        .MACRO        LDIR.8,D0,D0,#ERREUR#
        .LOCALMACRO   BOUCLE
LOOP:
        MOVE.%0       %1,%2
        DJ.16,NMO     %3,LOOP
        .ENDMACRO
L'appel de macro:
        LDIR.16       {A0+},{A1+},D0
produit:
M_0$:
        MOVE.16       {A0+},{A1+}
        DJ.16,NMO     D0,M_0$
```

On accède à l'indicateur de données (l'extension) par %0 et aux paramètres par %1 à %8. Si les %0 à %8 sont placés entre guillemets, ils se réfèrent aux paramètres de l'appel de macro, c.à.d. Les paramètres prédéfinis n'y sont pas inclus.

%0 est remplacé par l'indicateur de données; l'indicateur de données est défini à l'appel de macro, ou, si pas indiqué, par l'extension prédéfinie (si présent).

"%0" est remplacé par l'indicateur de données indiqué à l'appel de macro.

%1..%8 est remplacé par le n-ième paramètre; un paramètre est défini à l'appel de macro, ou, si pas indiqué, par les paramètres prédéfinis (si présent).

"%1".."%8" est remplacé par le n-ième paramètre indiquée à l'appel de macro.

%MC est le compteur de macro et est remplacé lors d'un remplacement d'une macro par un nombre qui indique le nombre d'appels à cette macro (1..999).

%ML est le niveau actuel d'imbrication auquel la macro appelée se trouve. On peut utiliser ce nombre avec la pseudo-instruction .IF pour, par exemple, tester si cette macro a été appelé au premier niveau d'imbrication. Ceci est utile dans les traductions de texte d'assembleur.

3.7 Appel de macro

Nous avons pu voir dans les exemples précédants qu'un appel de macro est composé de trois champs: le nom de la macro, l'extension et les paramètres. A chaque paramètre correspond exactement un nom de paramètre utilisé dans la partie principale d'une macro: %1..%8. Un paramètre à l'appel de macro peut être vide. Un paramètre à l'intérieur d'une macro est vide si aucun paramètre n'a été indiqué à l'appel de macro et aucune valeur n'a été prédéfini. Pour un paramètre vide, aucun caractère n'est inséré dans la macro remplacée.

Une macro peut appeler une autre macro mais pas elle-même (exception: voir .LAYOUTMACRO). Jusqu'à dix appels de macro imbriqués sont possible.

La définition d'une macro doit se trouver avant l'appel à cette macro.

4 Extensions de macro

Les possibilités de macro offertes correspondent aux performances de macro habituelles. Mais il y a des fonctions de macro additionnelles. Par exemple, on peut analyser les paramètres transmis et utiliser l'assemblage conditionnel dans la partie principale d'une macro.

4.1 Analyse de paramètre

Les fonctions suivantes ne sont reconnues qu'à l'intérieur de la partie principale d'une macro. Elles sont évaluées au moment de l'appel de macro (remplacement de la macro).
Abréviations utilisées:

%Vj	variables globales, modifiées par .VARMACRO 0..9 {,Text}
par	%i ou "%i" ou %Vj 0 <= i <= 8, 0 <= j <= 9
exp	nombres ou LENGTH() opérateurs plus (+) et moins (-)
parcopy	par ou COPY()
parpos	parcopy ou nom de symbole

Fonctions:

LENGTH(par)	retourne la longueur de par.
COPY(par,exp1,exp2)	génère de par un nouveau paramètre qui commence en position exp1 et qui a la longueur exp2.
DIGIT(parcopy)	donne TRUE, si tous les caractères dans parcopy sont des chiffres ["0".."9"], sinon FALSE.
LETTER(parcopy)	donne TRUE, si tous les caractères dans parcopy sont des lettres ["A".."Z","a"..."z"], sinon FALSE.
HEX(parcopy)	donne TRUE, si tous les caractères dans parcopy sont des caractères hexadécimales ["0".."9", "A".."F","a".."f"], sinon FALSE.
EMPTY(parcopy)	donne TRUE, si la longueur de parcopy est nulle. On peut vérifier la présence d'un paramètre lors d'un appel de macro.
UPCASE(parcopy)	convertit parcopy en majuscules.
POS(parpos1,parpos2)	retourne position de parpos1 dans parpos2 (0..)

4.2 Assemblage conditionnel dans les macros

Les fonctions additionnelles deviennent encore plus puissantes si l'on les combinent avec l'assemblage conditionnel. En plus, on peut comparer deux chaînes de caractères si elles sont placées avec un signe d'égalité dans la pseudo-instruction .IF.

Exemples: Teste si un paramètre a été défini à l'appel de macro

```
(68000):   .MACRO       LDIR
           .LOCALMACRO  BOUCLE
   LOOP:
           MOVE.%0      %1,%2
           .IF          EMPTY("%3")
           *** erreur: pas de compteur!
           .ELSE
           DJ.16,NMO    %3,LOOP
           .ENDIF
           .ENDMACRO

   Macro générale qui teste si une valeur est nulle (68000):
           .MACRO       ZERO
           .IF          LENGTH("%1")=2
           .IF          COPY("%1",1,1)DIGIT(COPY("%1",2,1))=ATRUE
           COMP.%0      #0,%1       ; registre d'adresse
           .ELSE
           TEST.%0      %1
           .ENDIF
           .ELSE
           TEST.%0      %1
           .ENDIF
           .ENDMACRO
```

5 Les pseudo-instructions qui sont interprétées

L'assembleur CALM interprète toutes les pseudo-instructions. Avec les options /C/R/V on peut générer un fichier de sortie qui ne contient plus les pseudo-instructions de macro et les pseudo-instructions .IF/.ELSE/.ENDIF.

6 Exemples de macro

L'utilisation de macros est illustrée dans les exemples suivants.

a) but: il faut insérer un sous-programme dans un programme uniquement si le sous-programme est appelé.

```
Macro definition:
        .MACRO          SP_NAME
        .IF             %MC=1
        JUMP            SPNAMEE
SPNAME:
        ... ici se trouve le sous-programme traditionnel
        RET
SPNAMEE:
        .ENDIF
        CALL            SPNAME
        .ENDMACRO

Premier appel de macro:
        SP_NAME
génère:
        JUMP            SPNAMEE
SPNAME:
        ... ici se trouve le sous-programme traditionnel
        RET
SPNAMEE:
        CALL            SPNAME
D'autres appels de macro génèrent:
        CALL            SPNAME
```

b) but: instruction de transfert de bloc général qui tient compte de la taille du compteur (68000). Si le compteur est un registre de données et d'une taille de .16, une optimisation est faite (DJ). De plus, une instruction de transfert par défaut doit être définie (comme LDIR dans le Z80).

Macro definition:

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

L'appel de macro:

```
     MOVESTRING
```

génère:

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

L'appel de macro:

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

génère:

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

L'appel de macro:

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

génère:

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

7 Traduction de texte d'assembleur avec des macros

L'utilisateur veut souvent étendre le jeu d'instructions en ajoutant quelques instructions avec des possibilités d'opérandes additonnelles. C.à.d. le code opératoire ne change pas. Pour éviter des appels de macro récursifs, la pseudo-instruction .LAYOUTMACRO avec l'opérande REPLACE est utilisée.

Exemple: Il faut ajouter les deux instructions suivantes au jeu d'instructions du Z80:

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

Définition de la macro:

```
        .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
```

L'appel de macro:

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

génère:

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

L'appel de macro:

```
        MOVE          DE,HL
```

génère:

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

La pseudo-instruction .LAYOUTMACRO avec le paramètre REPLACE TRUE évite qu'il y a un appel de macro récursif (c.à.d. une erreur) si l'on rencontre de nouveau le code opératoire MOVE.

Une autre utilisation est la traduction de la notation du fabricant en notation CALM. Ceci est possible si l'organisation générale (étiquettes, instructions, commentaires, etc.) ressemble à celle de CALM.

Fin du document.